Richard Hull
Jan Mendling
Stefan Tai (Eds.)

LNCS 6336

# Business Process Management

8th International Conference, BPM 2010
Hoboken, NJ, USA, September 2010
Proceedings

Springer

# Lecture Notes in Computer Science 6336

*Commenced Publication in 1973*
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Richard Hull   Jan Mendling   Stefan Tai (Eds.)

# Business Process Management

8th International Conference, BPM 2010
Hoboken, NJ, USA, September 13-16, 2010
Proceedings

Springer

Volume Editors

Richard Hull
IBM Research, Thomas J. Watson Research Center
19 Skyline Drive, Hawthorne, NY 10532, USA
E-mail: hull@us.ibm.com

Jan Mendling
Humboldt-Universität zu Berlin, Institut für Wirtschaftsinformatik
Unter den Linden 6, 10099 Berlin, Germany
E-mail: contact@mendling.com

Stefan Tai
Karlsruhe Institute of Technology (KIT)
Englerstraße 11, Gebäude 11.40, 76131 Karlsruhe, Germany
E-mail: stefan.tai@kit.edu

Library of Congress Control Number: 2010933361

CR Subject Classification (1998): D.2, F.3, D.3, D.1, D.2.4, F.2

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

| | |
|---|---|
| ISSN | 0302-9743 |
| ISBN-10 | 3-642-15617-7 Springer Berlin Heidelberg New York |
| ISBN-13 | 978-3-642-15617-5 Springer Berlin Heidelberg New York |

springer.com

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper        06/3180

# Preface

The BPM Conference series has established itself as the premier forum for researchers in the area of business process management and process-aware information systems. It has a record of attracting contributions of innovative research of the highest quality related to all aspects of business process management, including theory, frameworks, methods, techniques, architectures, systems, and empirical findings.

BPM 2010 was the 8th conference of the series. It took place September 14-16, 2010 on the campus of Stevens Institute of Technology in Hoboken, New Jersey, USA—with a great view of Manhattan, New York. This volume contains 21 contributed research papers that were selected from 151 submissions. The thorough reviewing process (each paper was reviewed by three to five Program Committee members followed in most cases by in-depth discussions) was extremely competitive with an acceptance rate of 14%. In addition to the contributed papers, these proceedings contain three short papers about the invited keynote talks.

In conjunction with the main conference, nine international workshops took place the day before the conference. These workshops fostered the exchange of fresh ideas and experiences between active BPM researchers, and stimulated discussions on new and emerging issues in line with the conference topics. The proceedings with the papers of all workshops will be published in a separate volume of Springer's *Lecture Notes in Business Information Processing* series. Beyond that, the conference also included a doctoral consortium, an industry program, fireside chats, tutorials, panels, and demonstrations.

We want to express our gratitude to all those who made BPM 2010 possible by generously and voluntarily sharing their knowledge, skills and time: the General Chair Michael zur Muehlen for providing an excellent environment for the conference, and all other colleagues holding offices. In particular, we thank the senior and regular Program Committee members as well as the additional reviewers for devoting their expertise and time to ensure the high quality of the conference scientific program through an extensive review and discussion process. Finally, we are grateful to all the authors who showed their appreciation and support for the conference by submitting their valuable work to it.

September 2010

Rick Hull
Jan Mendling
Stefan Tai

# Conference Organization

## General Chair

Michael zur Muehlen, USA

## Program Chairs

Rick Hull, USA
Jan Mendling, Germany
Stefan Tai, Germany

## Industry Chair

Michael Rosemann, Australia

## Local Organization

Michael zur Muehlen, USA

## Workshop Chair

Jianwen Su, USA

## Doctoral Consortium Chair

Ted Stohr, USA

## Demo Chair

Marcello La Rosa, Australia

## Publicity Chair

Marta Indulska, Australia

## Proceedings Chair

Robin Fischer, Germany

## Senior Program Committee

Serge Abiteboul, France
Gustavo Alonso, Switzerland
Boualem Benatallah, Australia
Marlon Dumas, Estonia
Schahram Dustdar, Austria
Claude Godart, France
Arthur ter Hofstefe, Australia
Stefan Jablonski, Germany
Frank Leymann, Germany
Tova Milo, Israel
Manfred Reichert, Germany
Hajo Reijers, The Netherlands
Michael Rosemann, Australia
Mathias Weske, Germany
Francisco Curbera, USA
Leon Zhao, China

## Program Committee

Wil van der Aalst, The Netherlands
Valeria de Antonellis, Italy
Pedro Antunes, Portugal
Djamal Benslimane, France
Shawn Bowers, USA
Christoph Bussler, USA
Fabio Casati, Italy
Malu Castellanos, USA
Peter Dadam, Germany
Umeshwar Dayal, USA
Alin Deutsch, USA
Remco Dijkman, The Netherlands
Asuman Dogac, Turkey
Boudewijn van Dongen, The Netherlands
Johann Eder, Austria
Gregor Engels, Germany
Avigdor Gal, Israel
Dimitrios Georgakopoulos, Australia
Guiseppe de Giacomo, Italy
Daniela Grigori, France
Paul Harmon, USA
Kees van Hee, The Netherlands
Willem-Jan van den Heuvel, The Netherlands
Marta Indulska, Australia
Leonid Kalinichenko, Russia
Gerti Kappel, Austria
Dimka Karastoyanna, Germany
Ekkart Kindler, Denmark
Jana Koehler, Switzerland
Agnes Koschmider, Germany
John Krogstie, Norway
Akhil Kumar, USA
Ana Liu, Australia
Heiko Ludwig, USA
Zakaria Maamar, UAE
Selma Limam Mansar, Qatar
Michael zur Muehlen, USA
Bela Mutschler, Germany
Prabir Nandi, USA
Andreas Oberweis, Germany
Yannis Papakonstantinou, USA
Theresa Pardo, USA
Oscar Pastor, Spain
Cesare Pautasso, Switzerland
Marco Pistore, Italy
Frank Puhlmann, Germany
Jan Recker, Australia
Wolfgang Reisig, Germany
Stefanie Rinderle-Ma, Austria
Roberto Rocha, USA
Domenico Sacca, Italy
Shazia Sadiq, Australia
Wasim Sadiq, Australia
Mohand Said-Hacid, France
Timos Sellis, Greece
Amit Sheth, USA
Farouk Toumani, France
Hagen Vlzer, Switzerland
Harry Wang, USA
Barbara Weber, Austria

Petia Wohed, Sweden
Eric Wohlstadter, Canada
Karsten Wolf, Germany
Andreas Wombacher, The Netherlands
Xiaohui Zhao, Australia
Christian Zirpins, Germany

## External Reviewers

Arya Adriansyah
Erdem Alpay
Maria Florencia Amitrano
Paul Bannerman
Henry Bi
Devis Bianchini
Petra Brosch
Mark Cameron
Shiping Chen
Soudip Roy Chowdhury
Fabian Christ
Claudio Di Ciccio
Amit Deokar
Felix Elliger
Dirk Fahland
Cédric Favre
Kathrin Figl
Robin Fischer
Sergio Flesca
Francesco Folino
Angelo Furfaro
Baris Güldali
Manolo Garcia
Ahmed Gater
Christian Gerth
Beat Gfeller
James Gibson
Christian Gierds
Karthik Gomadam
Suat Gonul
Antonella Guzzo
Hakim Hacid
Allel Hadjali
Armin Haller
Uwe Hartmann
Daning Hu
Christian Huemer
Zille Huma
Christian Janiesch
Sonja Kabicher
Udo Kannengiesser
Kathrin Kaschner
Nico Kerschbaumer
Markus Klems
Christian Koncilia
Kevin Lee
Maria Leitner
Domenico Lembo
Massimiliano de Leoni
Tammo van Lessen
Chen Li
Maya Lincoln
Mark Linehan
Manlu Liu
Rong Liu
Zheng Liu
Niels Lohmann
Ana Maria Lopez
Markus Luckey
Jürgen Mangler
Massimo Mecella
Michele Melchiori
Amin Mesmoudi
Kreshnik Musaraj
Alejandro Mussi
Benjamin Nagel
Tuncay Namli
Surya Nepal
Anil Nigam
Alexander Nowak
Olivia Oanea
Kian Win Ong
Riccardo Ortale
George Papastefanatos
Jarungjit Parnjai
Fabio Patrizi
Christian Pichler
Horst Pichler

Karsten Ploesser
Rodion Podorozhny
Luigi Pontieri
Senan Postaci
Francesco Pupo
Ella Rabinovich
Ajith Ranabahu
Marcello La Rosa
Christoph Rosenkranz
Pasquale Rullo
Jan Sürmeli
Juan Sañchez Díaz
Yacine Sam
Rob Schumaker
David Schumm
Nelly Schuster
Martina Seidl
Larisa Shwartz
Natalia Sidorova
Dimitrios Skoutas
Christian Soltenborn
Mirko Sonntag
Trent Spaulding
Giandomenico Spezzano
Christian Stahl
Michael Strommer
Sergey Stupnikov
Sherry Sun
Yehia Taher
Manolis Terrovitis
Fulya Tuncer
Jose Luis de la Vara
Eric Verbeek
Padmal Vitharana
Marc Voorhoeve
Hiroshi Wada
Sebastian Wagner
Matthias Weidlich
Daniela Weinberg
Edgar Weippl
Jan Martijn van der Werf
Branimir Wetzstein
Harro Wimmel
Fred Wu
Harris Wu
Moe Wynn
Marco Zapletal
Henry Zhang
Xuan Zhou
Liming Zhu

# Table of Contents

## Invited Talks

## BPM in Practice

## Correctness

## Design

## Distributed Processes

## Mining

## Semantics

## Processes and People

# The Next Decade of BPM

Phil Gilbert

IBM Corporation, Austin, Texas
pgilbert@us.ibm.com

**Abstract.** Business process management has been around for 20+ years. It can generally be described as having two distinct eras so far: in the 1990's BPM was led by process experts inside the business who transformed the focus from big-bang quality improvement initiatives (BPR-style) toward a focus on operational measurement and continuous improvement; in the 2000's BPM shifted to an IT-led competency, centering on the role technology played in process understanding and improvement. In 2010, BPM is still the province of experts from the business and IT. What's next? Two key changes will occur by 2020. First, notions of process will change from today's workflow-centric depictions to a more business-focused view of operations, transparency and measurement. Second, as we move from middleware-dependent systems into cloud-based software, BPM participation will spread throughout the organization, disintermediating the "experts," enabling entire cultures based on change and business improvement.

**Keywords:** BPM, governance, cloud, middleware, BPR, transparency, decentralization, measurement, globalization, process.

## BPM Is Not About Process

Many people attribute modern notions of process, leading up to today's business process management, to W. Edwards Deming and, to a lesser degree, George E. P. Box.[1] Deming is rightly credited with simplifying Walter A. Shewhart's work at Bell Laboratories into the Plan-Do-Check-Act cycle. There's no question that this notion of "continuous improvement" is dominant in today's discussions around BPM.

But for me the father of modern BPM is Alfred P. Sloan He invented the concepts of modern corporate organization in the 1920's at General Motors, and he was, somewhat ironically, a force for decentralization and transparency. Like Warren Buffett today, Sloan realized the best people to *operate* the business weren't the centralized executives at headquarters, but rather were the business people who knew the business, locally and intimately. He wasn't blind to the operational details, however. Sloan's genius was figuring out the set of detailed metrics his business units gathered, a level of operational transparency unprecedented in industry, and analyzing them in

[1] See, for example, venture capitalist J. William Gurley, Pay Attention To BPM, http://news.cnet.com/Pay-attention-to-BPM/2010-1071_3-994310.html

R. Hull, J. Mendling, and S. Tai (Eds.): BPM 2010, LNCS 6336, pp. 1–4, 2010.

the proper context. Decentralized operations, transparency into those operations, and centralized measurement and analysis.

As a young man he was an entrepreneur; he knew the operations of business. In middle years he was the leader of a large and growing enterprise; and he navigated these often turbulent times (early '20's; the Great Depression) by relentlessly focusing on the operational metrics combined with the financial results the company achieved – he married operational data (like production capacity and defects) with the business data (inventories, sell-through rates); he was always looking at the numbers and making adjustments to the business model. And to me his lasting legacy, even more than General Motors, is the best book on business yet written, My Years With General Motors.[2] It is here you really get to see how he thought, as he walks you through the data.

As we think about the future of business process management, we need to keep focused on the value this philosophy provides the business – the unique value – and make sure that our work drives toward unleashing that value. There are only two views that interest business people, the past and the future. Financial statements do a great job looking back; BPM is the technology that gives us a look into the future.

BPM should not be primarily about "process;" it should be about giving us a flashlight into the future. Light that is derived from a unique focus on operations, transparency and measurement. Process is only interesting because it gives us a way to represent business operations *in a structured way*. Process is the prism through which we can view operations in a structured way. For example, using this prism, I can now compare the operational qualities of, say, bank account opening vs. loan processing. If one group meets customer expectations (represented as SLAs) and another doesn't, I know where to focus my management time.

The goal isn't great process… the goal is great business outcomes! Process helps us structure the data so we can act on it. So BPM isn't about process so much as it is about using process as a proxy to unearth the operational plusses and minuses of your business.

Using BPM we can see the future and react to it. The better the BPM in your company, the better you can see the future. And the better you can see the future, the easier it is to be great.

## BPM Is Not About Experts

Resolved: Kill the "BPM Centers of Excellence!"

Imagine if I were to suggest "we don't need an informed citizenry to run a country, we only need a few informed people." I am pretty sure you wouldn't agree with me. I *hope* you wouldn't agree with me. In fact, this type of system has been tried (think "five year plans!" and, no, I'm not talking about ERP installations…)

I've talked about how BPM is actually about the combination of operations, transparency and measurement, which when done well results in a flashlight on the future. If so, then wouldn't the very best BPM come about as a result of *everyone* in operations participating in the conversation; being transparent or, more to the point, visible?

[2] Sloan, Alfred P (1964). <u>My Years at General Motors</u>. New York: Doubleday.

What if we could create entire populations at ease with change and familiar with this structured transparency? What if the goal of your BPM program were to empower everyone to participate instead of drawing and simulating process diagrams and models? That is, focus on the decentralization of actions and transparency as the end game. Which do you think would take longer: changing your culture or automating a given process? Which would have the largest impact?

In his 1999 book Business @ the Speed of Thought, Bill Gates said "[w]e always overestimate the change that will occur in the next two years and underestimate the change that will occur in the next ten." Why is this? Because real change rarely has to do with tools or even with creating a super class of experts using new tooling. Real change can only occur when culture shifts to reflect the possibilities of the new technologies. And changing culture takes time.

From 2000-2010, BPM's second decade began the exploration of creating tooling for real business people. Indeed, the defining, interesting attribute of BPM was the notion that business could regain control over its digital assets only if it participated directly in the definition, design, use and measurement of those assets. And so tooling began to be created on the backbone of standards emanating from the web revolution of the 1990's. Because this tooling could stand on the shoulders of IT-centric advances and standardized interfaces, vendors could concentrate on ease-of-use.

Strides were made but progress was insufficient for large-scale change. Today, most BPM "platforms" support only a few of a company's processes, even as many processes are presented as candidates for BPM, the number of qualified people and the ability to scale the platforms is in question. This isn't a result of inability of the platforms to scale, but the inability of organizations to scale yet another middleware platform.

Middleware itself is part of the bottleneck.

The other bottleneck is in our own failures of imagination. Because the history of computerized technology is one of leverage, we don't have the right mental model for creating the scale it takes to get everyone involved. Up until now, enterprise technology has been about scaling the very smart technologists by leveraging the technology they write. But the "new scale" is about inverting that pyramid, delivering software easy enough for the masses to use to communicate with, to be transparent with.

We have to make BPM scale not via the leverage of experts, but rather via the leverage of the masses.

To wit: Kill the centers of excellence; extol the hoi polloi!

We need to work on scaling the knowledge and ability to participate in business improvement conversations, not on scaling more technology and not solely on developing more experts and automating more processes.

In order to capture the possibilities of BPM, we have to create cultures of people who embrace change, who take control of the digital assets at their command, while still maintaining transparency into those activities and control over those assets. It's the assets that should be controlled; it's the actions on those assets that should be measure.

Software today can be delivered to these masses. In order to get to them in cost effective ways it imperative that we move to cloud-based, browser-based platforms. This will, in and of itself, require a cultural change.

BPM is not about experts. It’s about decentralization of operations, combined with transparency and accountability (or measurement). This is the best way for global organizations to continue to grow. Let’s focus on enabling this shift.

## This Keynote Address

Reflecting on the twin pillars of business empowerment and transparency, I’ll discuss how BPM is uniquely positioned to deliver the next great wave of value to companies; how the future can be seen not two seconds ahead of time, but weeks and months ahead of time.

In the next decade, the final maturation of BPM will be seen, leading to its becoming the dominant management paradigm for the next half century. But in order for this to happen, we need to change cultures globally.

I’ll give examples of real companies who are managing their future using BPM, and who have created cultures of thousands who embrace structured change around the world.

# BPM in Cloud Architectures: Business Process Management with SLAs and Events

Vinod Muthusamy and Hans-Arno Jacobsen

University of Toronto

## 1 Introduction

Applications are becoming increasingly distributed and loosely coupled in terms of their development processes, software architectures, deployment platforms, and other aspects. For example, in Web mashups [12,13], utility or cloud computing environments [2,3], and service-oriented architectures (SOA) [4,10] applications are developed by orchestrating reusable services using high level workflows or business processes. Application developers, however, must navigate a complex ecosystem that includes the services they depend on, the execution platforms of their applications and services, and the users of these applications, none of which they have much control over.

Business process management (BPM) practices address complexity in such environments with systematic development processes [1]. However, the development, administration and maintenance of a business process still requires much manual effort that can be automated. For example, non-functional goals, often expressed as Service Level Agreements (SLA) [7,11] defining a contract between a service provider and consumer, are specified during an early design stage but may need to be manually considered at each subsequent stage of development.

We present an SLA-driven approach to BPM for service-oriented applications in environments such as cloud computing platforms. The approach employs two key ideas: formally specified SLAs of applications, and event processing technologies. The SLAs are used to simplify tasks such as process deployment and monitoring, and an event paradigm is used to develop components, such as an event-based distributed process execution engine, that can exploit the characteristics of such distributed environments. These ideas complement one another in that the goals specified in the SLA can be used to automatically monitor the relevant parts of a process's execution in a loosely-coupled manner, or optimize the deployment of the process at runtime.

## 2 System Model and Architecture

Consider a typical SOA development cycle illustrated in Fig. 1 consisting of modelling, development, execution, and monitoring stages. Each stage differs in the level of abstraction considered and is performed by the indicated roles, each of whom have varying expertise and concerns.

R. Hull, J. Mendling, and S. Tai (Eds.): BPM 2010, LNCS 6336, pp. 5–10, 2010.

*SLA Model:* We make no major changes to the common development process outlined in Fig. 1 other than requiring a precise definition of the SLA requirements during the modelling stage. The details of our SLA model, which is designed to simplify the authoring of complex SLAs by composing and configuring existing SLA artifacts, is detailed in [5]. Capturing the SLAs early in a formal model has a number of benefits. For example, SLAs at the modelling stage can be mapped to lower level requirements on the services developed and resources provisioned, and translated to metrics that need to be monitored to observe SLAs violations. Furthermore, to ensure SLAs are satisfied, several runtime adaptations can be performed such as those discussed in Sec. 3.

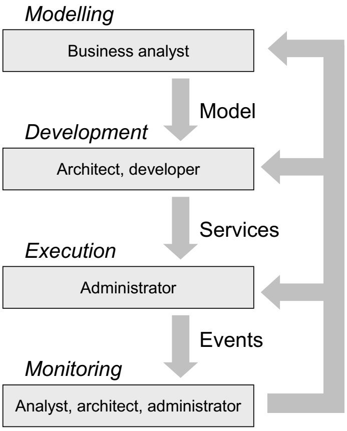


**Fig. 1.** Development cycle

*Cost Model:* Portions of the SLA are mapped to a cost model that captures various relevant factors. Some of these cost factors are shown in Table 1 grouped into cost components, such as the distribution cost which represents the overhead of distributing a process into fine-grained activities, the engine cost which captures the resource usage of an activity on the engine it is executing on, and the service cost which relates to the expense of calling external services.

A cost function based on a combination of cost components can flexibly express a variety of goals. The cost function specifies that an arbitrary weighting of the cost components should either meet a threshold or be minimized. In the former case, the process is adapted only when the threshold is violated, while in the latter, process adaptation occurs whenever a more optimal placement is found. For example, a cost function can constrain process response times to three seconds, or minimize the network traffic overhead of a process.

**Table 1.** Cost model components

| Component | Notation |
|---|---|
| *Distribution cost* | $C_{dist}$ |
| Message rate | $C_{d1}$ |
| Message size | $C_{d2}$ |
| Message latency | $C_{d3}$ |
| *Engine cost* | $C_{eng}$ |
| Load | $C_{e1}$ |
| Resources | $C_{e2}$ |
| Task complexity | $C_{e3}$ |
| *Service cost* | $C_{serv}$ |
| Service latency | $C_{s1}$ |
| Service execution | $C_{s2}$ |
| Marshalling | $C_{s3}$ |

*Distributed Process Execution:* Business process execution engines are typically centralized systems in which one node is provisioned to execute and manage all instances of one or more business processes. To address scalability, the centralized engine can be replicated and the process instances balanced among the replicas. We take a fundamentally different architectural approach whereby even individual process instances are executed in a distributed manner.

Our distributed execution engine decomposes a process, such as a BPEL process, into its individual activities and deploys these activities to any set of execution engines in the system. These activities then coordinate by emitting and consuming events over the PADRES[1] distributed publish/subscribe platform. For example, an activity that only executes after two other activities finish would subscribe to the composite event that indicates that both dependent activities have completed. Further details on how a BPEL process is mapped into this event-driven engine is presented in [8].

The execution engines in this architecture can be light-weight as they only execute fine-grained tasks, as opposed to complete processes. Another benefit of such an architecture is the ability to deploy *portions* of processes close to the data they operate on, thereby minimizing bandwidth and latency costs of a process. For example, for data intensive business processes, it is possible to deploy only those portions of the process that require access to large data sets close to their respective data sources. Different parts of the process that operate on different data sets can be independently deployed near their respective data sources. This is not possible in a clustered architecture since the entire process instance must be executed by a single engine.

*Execution Engine:* The internal components of an execution engine are presented in Fig. 2. Each engine is autonomous in deciding whether an activity should be redeployed to another engine. These decisions are based on the cost function associated with the process. Notably, there is no centralized component that is used to gather statistics, or to make activity placement decisions.

The distributed execution engine shown in Fig. 2 consists of a core Activity Manager that provides support services for the activities to collaborate among one another to execute a particular business process [8]. A Candidate Engine Discovery component is used to find other execution engines in the system [14], and these candidates are periodically probed by the Engine Profiler to gather various statistics. The Redeployment Manager computes the cost function for each activity executing in the engine, and determines if a more optimal placement of the activity is available among the known candidate engines. Finally, activities that are to be moved are redeployed using the Atomic Redeployer component which is responsible for ensuring that the movement of the activity does not affect the execution of the process [6].

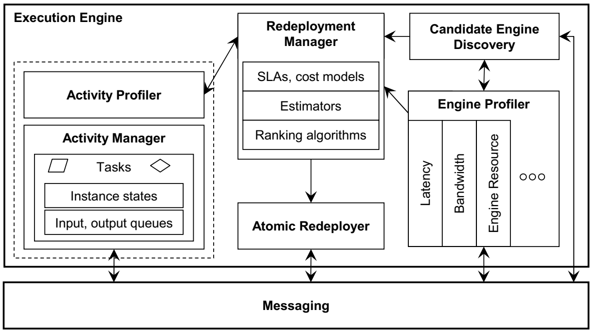


**Fig. 2.** Distributed execution engine

[1] Available for download at at http://padres.msrg.org

*Redeployment Manager:* The Redeployment Manager maintains for each activity $a_i$ the engine is currently hosting, the cost function $f(a_i)$ associated with the activity, a running average of the cost $c(a_i, e_j)$ imposed by the activity were it hosted by engine $e_j$, and the engines where activity $a_i$'s predecessors and successors are hosted. For convenience, the cost of deploying $a_i$ at the current engine is denoted as $c(a_i)$.

The running average of the cost $c(a_i, e_j)$ of an activity is computed and maintained based on information from various profilers. An update of the cost $c(a_i, e_j)$ may reveal a better placement for activity $a_i$. For example, if the cost function is to be minimized, the algorithm finds the engine $e_{min} \in E$ such that $c(a_i, e_i)$ is minimized across all $e_i \in E$ where $E$ is the set of known candidate engines. Activity $a_i$ is then moved to engine $e_{min}$. On the other hand, if the cost function associated with activity $a_i$ is a threshold function, a check is made to see if the accumulated cost $c(a_i)$ exceeds the threshold. If the cost is still within the threshold, nothing further is done. Otherwise, the system finds the engine $e_{min}$ that results in $\min\limits_{e_i \in E} c(a_i, e_i)$, and redeploys activity $a_i$ to engine $e_{min}$. Now it may be that $c(a_i, e_{min})$ still exceeds the cost function threshold, in which case the predecessors of activity $a_i$ are asked to redeploy themselves. This "back pressure" by activities to force a redeployment of their predecessors will occur repeatedly as long as the optimal placement of the activity is not sufficient to satisfy the cost function threshold. The redeployment procedure is further discussed in [9].

## 3 Benefits

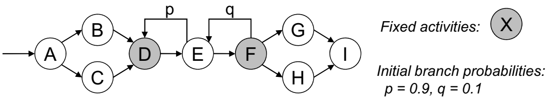
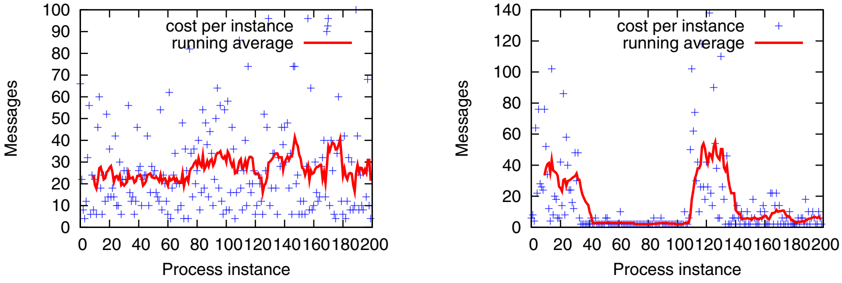


**Fig. 3.** Evaluated process

*Runtime Redeployment:* We present a sample of the benefits of the SLA-driven distributed process execution architecture. The process consisting of nine activities whose dependencies are depicted in Fig. 3 is deployed in a network of nine execution engines, with each engine running on a machine with a 1.6 GHz Xeon processor and 4 GB of memory. The execution engines utilize an overlay of PADRES publish/subscribe brokers communicating over a 1 Gbps switch.

To simplify the experiment, activities $D$ and $F$, as indicated in Fig. 3, are fixed to the initial engine they are deployed on, but the remaining activities can move. The process is invoked every second, and each process instance traverses the process graph according to the branch probabilities $p$ and $q$ as indicated in the figure. Notice that the 90% probability with which activity $E$ transitions to activity $D$ (as opposed to activity $F$) results in a process hotspot at activities $D$ and $E$. About halfway through the experiment, the transition probabilities of $p$ and $q$ are reversed, so that a hotspot now occurs at activities $E$ and $F$.

Fig. 4(a) presents the message overhead of executing the process in Fig. 3 under the case where activities remain in their initial deployment with each activity assigned to a different execution engine. The graph plots the number



(a) Static deployment (b) Dynamic deployment

**Fig. 4.** Message traffic for process in Fig. 3

of messages the engines exchange in order to coordinate the execution of each process instance. The average overhead computed over a sliding window of ten process instances shows that per-instance message traffic remains relatively stable at about thirty messages.

We expect, however, that due to the tight loop around activities $D$ and $E$, the message overhead can be reduced if these activities were deployed on the same engine. Indeed, when a cost function to minimize the message traffic is applied to the process, the system reconfigures itself by moving activities $A$, $B$, $C$, and $E$ to the same engine as activity $D$, and by redeploying activities $G$, $H$, and $I$ to the engine where $F$ resides. Fig. 4(b) shows that it takes about thirty seconds for the respective execution engines to determine the more optimal deployment and complete the movement of activities. Under the new deployment, the message traffic is only about 10% of the static case. After about 100 instances, the process traversal patterns are changed by swapping the initial branch probabilities $p$ and $q$. This causes a momentary increase in the traffic overhead before the system again stabilizes in about thirty seconds by this time placing activity $E$ together with activity $F$.

In more complex systems with multiple large processes and continuously changing process execution patterns and environmental conditions, it becomes increasingly infeasible to find an optimal static deployment, and the dynamic SLA-driven execution engine becomes more critical to ensure that the process SLA targets are achieved.

*Automated Monitoring:* A precisely specified SLA can be used to automatically instrument the process and generate monitoring code to detect SLA violations [5]. The monitoring subsystem is also event-driven and consumes the events emitted by the activities in our distributed process execution engine. This loose coupling allows the monitoring components to be deployed independently of the process, and can take advantage of the event processing optimizations of the underlying publish/subscribe system [8].

Furthermore, it turns out that the monitoring of a process can itself be modelled as a distributed process, thereby enabling the runtime process deployment optimizations above to be applied to the monitoring process as well [9]. For example, an SLA can be applied to the monitoring of the SLA to minimize the network overhead of monitoring, or to optimize the monitoring latency in order to detect SLA violations quickly.

*Service Selection:* The external services composed by an application can have significant effect on the execution of the application. In cases where there are a number of interchangeable external services that a process may use, such as a credit verification or weather reporting service, resource discovery techniques can be used to find the service instance that helps achieve the SLA, whether it be a low latency, high throughput, or cheap service instance [14].

## 4 Summary and Conclusions

BPM in cloud architectures give rise to challenging issues that stem, in part, from the conflicting demands of application developers seeking simplicity and flexibility, administrators focused on reducing costs, and users demanding performance. An event-based runtime architecture, coupled with formal SLA models, can address some of these challenges and afford a number of benefits to the end-to-end development of business processes including efficient resource utilization, dynamic process deployment, automated monitoring, and intelligent resource discovery.

## References

1. van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M.: Business process management: A survey. In: Business Process Management (2003)
2. Amazon Web Services, http://aws.amazon.com
3. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I., Zaharia, M.: Above the clouds: A Berkeley view of cloud computing. Tech. rep., Univ. of California, Berkeley (2009)
4. Channabasavaiah, K., Holley, K., Tuggle Jr., E.M.: Migrating to a service-oriented architecture. IBM developerWorks Technical Library (2003)
5. Chau, T., Muthusamy, V., Jacobsen, H.A., Litani, E., Chan, A., Coulthard, P.: Automating SLA modeling. In: CASCON, Toronto, Canada (2008)
6. Hu, S., Muthusamy, V., Li, G., Jacobsen, H.A.: Transactional mobility in distributed content-based publish/subscribe systems. In: ICDCS (2009)
7. IBM: Web service level agreements (WSLA) project, http://www.research.ibm.com/wsla/
8. Li, G., Muthusamy, V., Jacobsen, H.A.: A distributed service-oriented architecture for business process execution. ACM Trans. Web 4(1) (2010)
9. Muthusamy, V., Jacobsen, H.A., Chau, T., Chan, A., Coulthard, P.: SLA-driven business process management in SOA. In: CASCON, Toronto, Canada (2009)
10. Natis, Y.V.: Service-oriented architecture scenario. (April 2003), http://www.gartner.com/DisplayDocument?doc_cd=114358
11. Paschke, A., Schnappinger-Gerull, E.: A categorization scheme for SLA metrics. In: Service Oriented Electronic Commerce (2006)
12. Wang, H.J., Fan, X., Howell, J., Jackson, C.: Protection and communication abstractions for Web browsers in MashupOS. In: SOSP (2007)
13. Yahoo Pipes, http://pipes.yahoo.com
14. Yan, W., Hu, S., Muthusamy, V., Jacobsen, H.A., Zha, L.: Efficient event-based resource discovery. In: DEBS (2009)

# Warning: Don't Assume Your Business Processes Use Master Data

Clay Richardson

Forrester Research
crichardson@forrester.com

## Outline

It's an age old question: Which came first: data or process? — When this question is presented to IT professionals, the answer depends heavily on each person's individual perspective and role within IT. Ask most business process professionals, and the immediate response is: *"Without process, data does not exist."* Ask data management professionals, and the immediate response is: *"Without data, processes can't execute."*

In recent research, Forrester tackled this vexing question and concluded that *process and data are of course inseparable*. Unfortunately, most business process professionals, data management professionals, and vendors we interviewed couldn't see the forest for the trees. In our 2009 MDM & BPM Survey, *only 11% of respondents indicated that their MDM and BPM initiatives shared the same cost center* and that team members work together on a daily basis to develop solutions for the business.

In many ways, master data management (MDM) and business process management (BPM) represent two different sides of the same coin when it comes to business optimization and transformation. So, why is there such little collaboration and interaction between business process and data management professionals? Here are some of key findings our research uncovered:

- *Business process professionals assume process-related data is trustworthy.* While most process improvement initiatives pay lip service to data quality, only a small handful take data seriously and incorporate data modeling and data mapping into upfront discovery efforts. BPM teams typically focus on process first and user experience second, while data becomes an afterthought. Process improvement teams rarely ask whether data is clean until it's too late in the project.
- *Data management professionals value clean data over process context.* "If we create clean data, then they will come" is often the mantra of data management professionals. However, many executives still struggle to quantify the value of on-going data quality initiatives, with some executives going as far to ask: *"Why are we spending so much money on something when we can't articulate its business impact?"* In other words, data management professionals often fail to provide cross-enterprise context for MDM.

R. Hull, J. Mendling, and S. Tai (Eds.): BPM 2010, LNCS 6336, pp. 11–12, 2010.

- *BPM suite vendors offer few tools to connect process to data.* Technology vendors argue that service oriented architecture (SOA) is the silver bullet to bridge the gap between MDM and BPM initiatives. While SOA addresses component reuse and architecture challenges, BPM teams are provided very few tools to model and synchronize process-related data in upfront process discovery activities. Only a handful of BPM suite offerings provide business glossaries and business-oriented capabilities to connect process improvement and master data efforts.
- *MDM vendors focused on data governance and stewardship only.* The good news is these MDM vendors do in fact recognize that business process plays a major role in their success. The bad news is they are only actively addressing one side of the problem: the business processes and stewardship roles that govern the administrative workflows within the MDM tools themselves. MDM vendors are not actively influencing the business processes that consume and depend upon the master data these vendors provide.

Although business process and data management professionals may not see it, BPM and MDM are interconnected and one initiative cannot survive for long without the other. In this presentation, we highlight the fact that process improvement initiatives face a vicious cycle of deterioration and decline if master data issues are not addressed from the outset. We also highlight how MDM initiatives face an uphill battle and certain extinction if they're not connected to cross-cutting business processes that feed and consume master data from different upstream and downstream activities.

In 2009, Forrester Research highlighted *the emergence of "process data management"* in a comprehensive evaluation of the maturity of different components that make up the BPM ecosystem. This new category in the business process landscape organically merges BPM and MDM disciplines and capabilities to provide the enterprise with one version of "process and data truth."

Since outlining this new capability, Forrester has published several research docs and presentations that scope the process data management challenge, provide best practices for combining process improvement and data management initiatives, and present case studies highlighting teams that have successfully integrated these two critical disciplines.

# IT Requirements of Business Process Management in Practice – An Empirical Study

Susanne Patig, Vanessa Casanova-Brito, and Barbara Vögeli

University of Bern, Institute of Business Information Systems,
Engehaldenstrasse 8, CH-3012 Bern, Switzerland
{Susanne.Patig,Vanessa.Casanova-Brito,
Barbara.Voegeli}@iwi.unibe.ch

**Abstract.** Substantial use of dedicated software characterizes the highest level of Business Process Management (BPM) maturity. Currently, companies are far below this level. This situation is due to the fact that the existing BPM tools don't satisfy key requirements of BPM. We have conducted a worldwide survey of major public companies to elicit these requirements, which are grounded in the nature of processes and the usage of software. The analysis of 130 responses indicates that human-oriented process modeling languages and BPM tools as well as BPM tools with software integration capabilities are most urgently required.

**Keywords:** Adoption and Practice of BPM, Business Process Modeling, BPM maturity, Empirical Study.

## 1 Introduction

*Business process management (BPM)* deals with the design, administration, configuration, enactment and analysis of business processes [20]. *Business processes* are sets of linked activities or tasks that collectively realize a business objective or policy goal within the context of an organizational structure [22]. For brevity, we omit the term 'business' in the following and just speak about 'processes' wherever appropriate.

Older [15] and recent [23] empirical studies indicate that more and more companies adopt BPM to, e.g., improve customer satisfaction or reduce costs. However, empirical investigations also show that the level of BPM capabilities in the companies, i.e., the respective *BPM maturity,* is rather low. Typical *BPM maturity levels*[1] are [23], [10]:

- Level 1: No processes are defined, and the organization is functional.
- Level 2: The core and most commonly used processes are defined, and the representatives from functional areas meet regularly to coordinate with each other.
- Level 3: All processes are defined; BPM is employed with strategic intent; process-oriented organization (e.g., chief process officers (CPO) or central BPM teams) exist outside the functional organization to gather process data and optimize processes.

[1] For a history of distinct BPM maturity models and an alternative approach see [16].

R. Hull, J. Mendling, and S. Tai (Eds.): BPM 2010, LNCS 6336, pp. 13–28, 2010.

- Level 4: The coordination within the company and with its vendors and suppliers is process-oriented. The functional organization structure is subordinate to the process structure. Process performance measures and BPM software are extensively used.

Empirical evidence suggests that most companies in the USA and Europe are on BPM maturity Level 2 or between Level 2 and Level 3 [23], [10]. Level 4, which relies on the broad use of dedicated software, isn't reached yet. As a hypothesis, we attribute this situation to the fact that current software systems do not sufficiently satisfy the requirements of BPM. The (exploratory) *research goal* of our work is to elicit these requirements that must be met by tools that support BPM.

The term *BPM tool* denotes any software that can be used to manage business processes. According to the definition of BPM, a BPM tool must at least enable the *description* of processes (as explicit knowledge about processes is a prerequisite to their administration, configuration and enactment). We allow both modeling languages and text or tables for process descriptions (see Section 3.2). More progressive BPM tools support the *execution* of *workflows*, which are the computerized automation of business processes [21]. Thus, for the purpose of our requirements gathering, *BPM tools* comprise text processing systems, simple graphics tools, repository-based modeling tools as well as workflow engines. *BPM suites/systems* (i.e., TIBCO iProcess Suite[2]) combine several of the listed software categories [23].

The existing empirical investigations of BPM, which are summarized in Section 2, do not reveal in detail what companies expect from software that supports BPM. For that reason we have conducted a worldwide survey of major public companies and obtained 130 responses. The procedure and the results can be found in Section 3. Section 4 discusses our findings, relates them to the findings of other investigations and comments on their validity. Section 5 concludes our work and gives some outlook.

## 2 Related Work

During the last years, several empirical studies in the field of BPM have been published. This section classifies them (see Table 1). The main classification criteria are the *focus* of the study (management, BPM tools, process modeling languages), the *object* of research (company assessments, experts, BPM tools, business processes, process models) and the *research method* (cases, surveys, samples and a Delphi study). We assume studies that were conducted by vendors of BPM suites to be biased and, thus, exclude them here.

The oldest empirical study stems from 1999 and explores the importance, understanding and realization of business process *management* in European companies [15]. It focuses on the drivers and benefits of BPM as well as the degree of BPM implementation at that time; BPM tools were not considered.

Ten years later, based on assessments of hundreds of companies in the USA, Europe, China and Brazil, the key turning points in business process maturity were

[2] All names of products and services are trademarks, service marks or registered trademarks of the respective companies – even if not explicitly stated.

identified [10]: *Process documentation*, knowing the customer and endorsing teamwork are important at the BPM maturity Level 2. Process measurement and a *process language* (in the sense of corporation-wide process terms) are important to reach the BPM maturity Level 3. Finally, process-oriented jobs and the support by BPM tools are needed for the transition to the BPM maturity Level 4. Details about BPM languages or tools were not reported.

As BPM tools are important to reach higher levels or BPM maturity, some investigations (e.g., [11], [23]) combine questions on managerial issues of BPM and BPM tools: The study "Status Quo of BPM" [11] asked 185 companies from Germany, Austria and Switzerland (mainly IT-driven branches) about the alignment between BPM and business strategy, about process management methods, the organization of BPM and the role of IT. It showed that mainly proprietary systems and Enterprise Resource Planning (ERP) systems influence the companies' business processes; BPM suites are less important. If companies use a BPM suite, they mostly apply process modeling and process publication. Altogether, the tool-related questions are poorly reported, which affects their reliability.

The report "State of Business Process Management 2010" [23] focuses on the understanding of BPM in companies, the BPM drivers, spending, standardization and training. Only three questions deal with BPM tools and services: They show that companies mainly rely on simple graphics as well as repository-based modeling tools; only 26% use a BPM suite. The dominant BPM suite is IBM WebSphere BPM.

Most *tool-related studies* [18], [7], [12] deal with tool *evaluation*, i.e., they propose a set of criteria and calculate some quantitative measure that rates some BPM tool with respect to these criteria. Usually, the origin or distribution of the tools is not considered ('anywhere' in Table 1). Beside such evaluations, *cases of implementing business processe*s with BPM tools have been reported in conjunction with the realized benefits and the encountered challenges [8]. Finally, tools supporting the *Business Process Modeling Notation (BPMN*) [13] were investigated [17]: Among modelers from practice and academia, simple graphics tools (Microsoft Visio) are widespread, followed by repository-based modeling tools. These tools and the BPMN are mainly (51%) used to document processes and to support business process reengineering (BPR), but also for process simulation and workflow engineering. The most appreciated functionalities of the BPMN tools are navigation between process models, integrated repository and additional fields for attributes.

Empirical studies related to process modeling languages either concentrate on the BPMN [24], [25] or are language-neutral [4], [5]. For *BPMN*, two sets of sample processes have been analyzed to find out which BPMN constructs are really used in practice [24], [25]. The *language-neutral* investigations questioned experts from practice, academia and tool vendors on the benefits, issues and challenges of business process modeling. Process improvement, understanding and communication emerged as overall (across all three stakeholder groups) benefits of process modeling [4], whereas standardization, the value proposition of process modeling to business and model-driven process execution are currently seen as main overall issues and challenges [5].

Table 1 summarizes the studies just mentioned. None of them provides detailed information of the requirements of BPM that must be satisfied by BPM tools. These requirements mainly depend on the nature of processes and the kind of IT support aimed for. We investigated both aspects. The design and results of our investigation are described in the next section.

**Table 1.** Summary of relevant empirical studies in the field of BPM

| Ref. | Year | Focus | Type | Method | Object (Number) | Distribution of Objects | Findings |
|---|---|---|---|---|---|---|---|
| [15] | 1999 | Management | QN | Survey, Cases | EP (92) | Europe | BPM drivers, benefits, realization |
| [10] | 2009 | Management | QN | Cases | CA (hundreds) | USA, China, Europe, Brazil | Turning points in business process maturity |
| [11] | 2009 | Management (Tools) | QN | Survey | E [P] (185) | Austria, Germany, Switzerland | Alignment, BPM methods & organization, tools used |
| [23] | 2010 | Management (Tools) | QN | Survey | E [P] (264) | Worldwide | Understanding of BPM, BPM spending, tools used |
| [8] | 2007 | Tools (Management) | QL | Cases | P (4) | Switzerland | Benefits and challenges of BPM implementation |
| [18] | 2008 | Tools | QN/ QL | Cases | T (41) | Anywhere | Approach for tool evaluation |
| [7] | 2006 | Tools | QN | Cases | T (4) | Anywhere | Approach for tool evaluation |
| [12]¡ | 2006 | Tools (Languages) | QN | Cases | T (7) | Anywhere | Approach for tool evaluation |
| [17] | 2008 | BPMN Tools | QN | Survey | E (590) | Worldwide | Use of BPMN in practice |
| [4], [5] | 2009 | Languages | QN | Delphi Study | E [P/A/T] (62) | (Worldwide) | Benefits, issues and challenges of BP modeling |
| [25] | 2008 | Language (BPMN) | QN | Sample | PM (120) | not stated | Usage rates of BPMN constructs |
| [24] | 2007 | Language (BPMN) | QN | Cases | PM (19) | USA | Usage of BPMN constructs |

*Abbreviations*: CA: Company assessments, E [P/A/T] – Experts from Practice/Academia/ Vendors, P: Process, PM: Process models, QL: Qualitative, QN: Quantitative, T: Tools.

# 3 Empirical Investigation on the Requirements of Business Process Management

## 3.1 Method

*Participants*: The basic population for our investigation consisted of the companies from the list "Forbes Global 2000" [2]. This list is an annual ranking of the top 2,000 public companies in the world based on sales, profit, assets and market value. As we focus on IT support for BPM, 'e-readiness' is required. *E-Readiness* describes the ability of a country and its businesses to use information and communication technology to their benefits [1]. The 'e-readiness' of 70 countries is yearly assessed on a scale between 1 (lowest) and 10 [1]. We used the e-readiness rating of the year 2008, where the average e-readiness score amounted to 6.4. All companies that are headquartered in countries with below-average e-readiness were removed from the list "Forbes Global 2000". From the remaining 1,680 companies, we draw a sample of 1,172 companies (by random numbers), which were contacted as described below.

**Table 2.** Numbers of Responses per Country

| **Countries** | **Responses** | |
|---|---|---|
| (Listing according to equal count of responses) | *Per country* | *Total* |
| United States | 17 | 17 |
| Switzerland | 15 | 15 |
| Germany; Japan | 11 | 22 |
| UK; Canada | 9 | 18 |
| Australia; France; Spain | 5 | 15 |
| Hong Kong; Italy; Portugal; Sweden; Taiwan | 4 | 20 |
| Austria; Belgium | 3 | 6 |
| Netherlands; Singapore; Greece; Denmark; Ireland; Korea; Norway | 2 | 14 |
| New Zealand; Luxemburg; Iceland | 1 | 3 |
| Sum | | 130 |

In total, N = 130 companies responded; so, the response rate was 11%. Some questions were not answered by all companies; the resulting *missing values* are stated as '*na*' in this paper. By their headquarters, European companies account for the majority (58%) of responses, followed by companies from North America (20%), Asia (18%) and Oceania (5%). The numbers of responses per country are given in Table 2. All branches were represented (see Table 3); some of the companies operate in more than one branch. Most responses came from the banking sector (19%), followed by utilities (8%), transportation and insurance (7% in each case) as well as oil & gas, and technology hardware (6% in each case).

Half of the companies' representatives who answered the questionnaire work in IT departments (53%); other affiliations are departments for BPM (22.9%) or for company organization (13.3%), functional areas (9.6%) and product divisions (1.2%). In addition to knowledge about the processes of their own departments (29.4%), the participants stated to also know the processes of other departments (23.9%) or even have a company-wide picture (46.7%) of the processes.

*Materials*: The questionnaire consisted of 42 questions that were grouped in the several sections dealing with BPM (current status, tools), processes (characteristics, change, statistics), process modeling (procedure, languages) and socio-demographic information. Because of space limitations, we concentrate here on the questions related to IT-support for BPM; the results concerning process modeling are discussed in another paper.

**Table 3.** Numbers of Responses per Branch (multiple answers)

| **Branches** | **Responses** | |
|---|---|---|
| (Listing according to equal count of responses) | *Per branch* | *Total* |
| Banking | 30 | 30 |
| Utilities | 10 | 10 |
| Insurance; Transportation | 9 | 18 |
| Oil & Gas Operations; Technology Hardware & Equipment | 8 | 16 |
| Diversified Financials; Materials | 7 | 14 |
| Capital Goods; Retailing | 6 | 12 |
| Chemicals; Software & Services; Telecommunications Service | 5 | 15 |
| Consumer Durables; Food, Drink & Tobacco; Hotels, Restaurants & Leisure; Media; Semiconductors | 4 | 20 |
| Business Services & Supplies; Construction; Trading Companies Drugs & Biotechnology; Household & Personal Products | 3 | 12 |
| Health Care Equipment & Services; Conglomerates | 1 | 2 |

All 42 questions of the questionnaire were partially open-ended, i.e., they provided a list with alternatives and an alternative ‘other’ to enter free text for unanticipated answers. Some questions were optional. The data was collected on nominal scale (participants were asked to select *all* alternatives that apply) or on ordinal scale (participants were asked to make a ranking or to rate some alternative). Except for question Q19 (see Table 11 in Section 3.2), equal ratings were allowed. All rating scales had four levels (to avoid neutral answers) and an additional level (e.g., ‘Don’t know’, ‘Not applicable’, ’Not needed’, ‘None’) to avoid forced ratings [6]. The resulting total number of five rating levels is generally reckoned optimal since the ability to differentiate between ratings decreases with an increasing number of rating levels [6].

The questionnaire also contained explanations of key terms that were used in the questions. All material was written in English and implemented as an online form with the tool ‘Limesurvey’ [9]. To contact the companies, we sent letters by surface mail containing the link to the online form and explaining the goals and importance of our investigation. These letters were written in English, German, Spanish and Japanese – depending on the location of the company’s headquarter - to increase the understanding of our research.

*Procedure*: We conducted a pretest of the questionnaire with 10 BPM experts from practice and academia to check the questions for understandability and clarity; afterwards, the questionnaire was revised. As little is known about the internal organization of BPM in a company, we sent the letters with the link to the revised questionnaire to the CIOs or CEOs of the sampled companies and asked them to forward the letters to the persons responsible for BPM. After four to six weeks, we phoned the offices of the CIOs or CEOs to inquire after the status of our information request. After six to eight week we sent reminder letters by surface mail.

In total, the survey was carried out from *January to December 2009*. No incentives were given; the companies were only offered to obtain the results of the investigation free of charge. Some of these results can be found in the next section.

## 3.2 Results

*Reporting of the results*: In the following, we denote the questions on ordinal scale by the superscript '*'; all other questions are on nominal scale. As our questions on *nominal scale* ('choose all alternatives that apply …') usually allow more than one answer, the sum of counts usually exceeds the number N of companies' responses. In the tables, the column *'percentage'* always relates the count $c_i$ of each alternative to the sum $\Sigma c_i$ of counts of all alternatives, whereas the column *'percentage of responses'* relates the count $c_i$ of an alternative to the number N of responses ($N \leq 130$, depending on the number *na* of missing values). To save space, some nominal questions are reported by text only, where we give the count for each alternative answer followed (after a slash) by the percentage of responses. The alternative with the absolute highest *count* (absolute frequency) represents the *mode* [3].

*Processes:* First, our investigation gathered information about the nature of the managed processes. In addition to process statistics (see Table 4), we inquired about the scope (Q6) of processes (N=128, na=2), their organizational distribution (Q7*), average run time (Q8*) and execution frequency (Q9*) as well as about process change (Q10, Q11).

**Table 4.** Statistics for an average process

| *(Question)*<br>**Please estimate the …** | **Minimum** | **Maximum** | **Mean** | **Standard Deviation** | **N/na** |
|---|---|---|---|---|---|
| (Q1) Number of involved persons from the same department | 1 | 55 | 7.31 | 11.59 | 70/60 |
| (Q2) Number of involved departments | 1 | 18 | 3.70 | 2.56 | 73/57 |
| (Q3) Number of other companies involved | 0 | 8 | 1.60 | 1.39 | 58/72 |
| (Q4) Number of applications involved | 1 | 24 | 4.27 | 4.22 | 69/61 |
| (Q5) Number of tasks | 3 | 120 | 19.09 | 20.82 | 60/70 |

*Scope* (Q6): By absolute frequency, most processes are related to the companies' products (78/60.9%), immediately followed by administration (71/55.5%), customer contact (67/ 52.3%), system integration (66/51.6%), system development (65/50.8%), emergency procedures (33/25.8%) and other things (12/9.4%).

The questions Q7* to Q9* required the respondents to rate the portion of processes to which an answer alternative applies on a scale providing the levels 'all', 'most', 'some', 'a few' and 'none'. Table 5 assigns the answer alternatives of each question to their most frequent rating; the particular counts are given in brackets.

Concerning *process distribution* (Q7*; see also Table 4, Q2 and Q3), most processes stay within the company, but span several departments; some processes involve several persons from the *same* department, or they involve other companies (see also Q3 in Table 4). Only a few processes can be accomplished by a single person (see also Q1 in Table 4). The *average run time* (Q8*) of most processes is measured in days. Some processes have an average run time that is measured in weeks, and only a few processes run months or years. The majority of processes are *executed* (Q9*) several times a day, some are executed several times per week or per month, and a few processes are executed several times a year.

**Table 5**. Nature of processes according to modes

| Question | Rating |  |  | N/na |
|---|---|---|---|---|
|  | *Most* | *Some* | *A few* |  |
| Distribution of processes (Q7*) | 1 Company & >1 Department (83) | 1 Department & >1 Person (51);<br>>1 Company (55) | 1 Department & 1 Person (60) | 130/0 |
| Measure of average process run time (Q8*) | In days (54) | In weeks (49) | In month (34);<br>in years (30) | 87/43 |
| Execution frequencies of processes (Q9*) | Several times a day (59) | Several times a week (58) / a month (56) | Several times a year (50) | 102/28 |

*The frequency of process changes* (Q10; N=111, na=19) is low: Yearly process changes occur in 39/35.1% of the companies. Also rare (30/27%) or quarterly (29/26.1%) process changes are common, whereas short term process changes, i.e., monthly (8/7.2%) or weekly (4/3.6%), are rare. The most frequent *trigger of process* change (Q11; N=130, na=0) is the evolving internal business (115/88.5%), followed by forces from environment (92/70.8%), deviations from planned values (52/40%) and internal disruptions (28/21.5%).

*Process Description*: Process documentation is a key turning point to reach the BPM maturity Levels 2 and 3 [10]. Table 6 summarizes how the sampled companies currently *document* (Q12) their processes. Obviously, many companies combine text (55.9%) and some (modeling) language (altogether 55.9%), but also tables are widespread (31.5%). Among the languages, BPMN dominates, followed by the Unified Modeling Language (UML) and Event-driven Process Chains (EPC).

**Table 6**. Current documentation of processes (N = 127; na = 3)

| **Answers** |  | **Count** $c_i$ | **Percentage** ($c_i$ / $\Sigma c_i$) | **Percentage responses** ($c_i$ / N) |
|---|---|---|---|---|
| As text |  | 71 | 36.2% | 55.9% |
| As tables |  | 40 | 20.4% | 31.5% |
| As flow charts |  | 2 | 1.0% | 1.6% |
| With languages | BPMN | 27 | 13.8% | 21.3% |
|  | UML | 19 | 9.7% | 15.0% |
|  | EPC | 16 | 8.2% | 12.6% |
|  | BPEL | 5 | 2.6% | 3.9% |
|  | IDEF | 4 | 2.0% | 3.1% |
| Other |  | 12 | 6.1% | 9.4% |
| Total ($\Sigma c_i$) |  | 196 | 100.0% | 154.3% |

- *Organization* (Q13): In 42.6% of the companies, the processes are identified and described within the individual departments and then centrally aligned. However, the extreme positions – *decentralized* process modeling in the departments without alignment (30.8%) and a *central* BPM team (26.6%) - also exist.

- The *motives for describing processes* (Q14) partially build on each other (N=130, na=0). Ordered by decreasing frequency, they can be grouped as follows (the difference between the groups is 8 to 13 percentage points):
  - *Group 1*: Processes are described to prepare BPR, i.e., the optimization and reorganization of processes (92/70.8%) or to have precise guidelines ('to-be' processes) for the persons involved in process execution (89/68.5%).
  - *Group 2*: Companies describe 'as-is' processes to document what happens (78/60%) or to automate process execution (72/55.4%).
  - *Group 3*: Described 'to-be' processes serve as targets in monitoring process execution (58/44.6%) or to integrate software systems (57/43.8%).
  - *Group 4*: Processes are described for ISO certification (37/28.5 %) and to select between business applications by comparing the processes these systems support to the actual processes in the company (26/20%).
  - *Group 5*: This group comprises less important motives to describe processes, namely compliance (5/3.8%) and company-specific factors (8/6.2 %).
- More than half of the companies (70/58.8%) affirmed (Q15) that *process models* can be found in *non-BPM software* they use (N=119, na=11). In detail, process models exist within ERP software and alike (50/42%) – such as systems for HR or CRM-, integration software (39/32.8%), data warehouses (26/21.8%) and requirements engineering software (16/13.4%).

*Information systems*: Support by BPM tools is a prerequisite to reach the BPM maturity Level 4 [10]. Thus, we asked the participants to rate the *importance* (Q16*) of a list of *functionalities and qualities* for BPM tools on the following scale: 'very important' (1), 'important' (2), 'not so important' (3), and 4 ('not important at all') or 5 ('don't know'). The list was compiled based on BPM literature, e.g. [20], [12], [17], [7]. Fig. 1 shows the results: Usability is clearly the most important quality of BPM tools (78/60%). Among the important requirements, the following functionalities or qualities appear (ordered by decreasing frequency): alignment to standards (71/55%), modeling capabilities (64/49%), simulation capabilities (63/48%) and workflow enactment (63/48%). To get a clearer picture of the overall importance of each requirement, we calculated the mean rating for each functionality and quality (see Table 7): Usability stays in the first place, followed by modeling capabilities, software integration, report generation, alignment to standards and process execution.

**Table 7**. Important functionalities and qualities for BPM tools (N= 130, na=0)

| Answer | Mean | Std. Dev. | Rank | Answer | Mean | Std. Dev. | Rank |
|---|---|---|---|---|---|---|---|
| Usability | 1.49 | .760 | 1 | Process Monitoring | 2.25 | 1.027 | 6 |
| Modeling Capabilities | 1.81 | .924 | 2 | Process Simulation | 2.45 | .957 | 7 |
| Software Integration | 2.12 | .996 | 3 | Language Adaptability | 2.52 | .998 | 8 |
| Report Generation | 2.13 | .976 | 4 | Correctness Proofs | 2.55 | 1.028 | 9 |
| Alignment to Standards | 2.18 | .947 | 5 | Syntax Checks | 2.65 | 1.009 | 10 |
| Process Execution | 2.18 | .979 | 5 | Different Notations/ Standards | 2.72 | 1.019 | 11 |

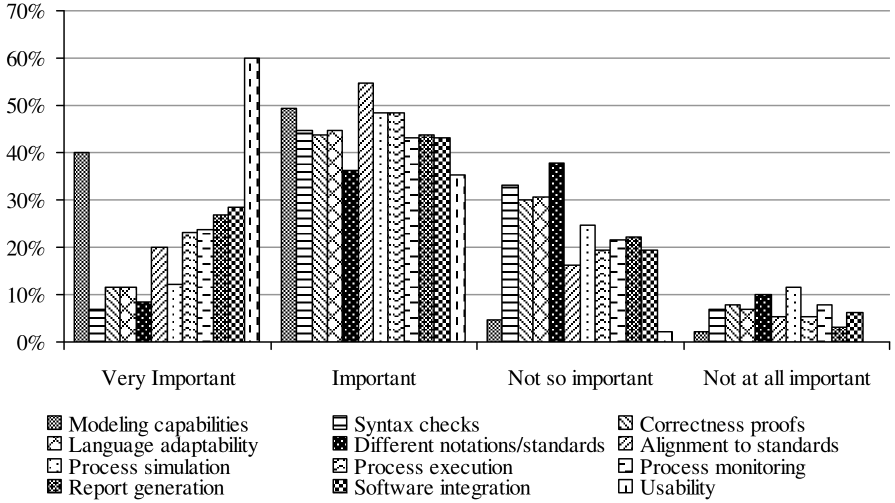


**Fig. 1.** Important functionalities and qualities for BPM tools

Software integration capabilities come third among the requirements of BPM tools. This finding becomes plausible when analyzing the *applications involved* (Q17) in the execution of business processes (see Table 8): Mainly databases, ERP systems and office software are needed. Obviously, the implementation of business processes does not necessarily require BPM suites, which rank 10th in the list.

**Table 8**. Applications involved in the execution of business processes (N=130; na=0)

| **Answers** | **Count** | **Percentage** | **Percentage responses** |
|---|---|---|---|
| Database(s), data warehouse(s) | 87 | 11.5% | 66.9% |
| Enterprise resource planning system(s) | 79 | 10.4% | 60.8% |
| Office software | 79 | 10.4% | 60.8% |
| Customer relationship management system(s) | 76 | 10.1% | 58.5% |
| Integration software, middleware | 66 | 8.7% | 50.8% |
| Procurement system(s) | 63 | 8.3% | 48.5% |
| Content management system(s) | 56 | 7.4% | 43.1% |
| Supply chain management system(s) | 55 | 7.3% | 42.3% |
| Product data management system(s) | 50 | 6.6% | 38.5% |
| BPM suites | 41 | 5.4% | 31.5% |
| Application software at business partners | 38 | 5.0% | 29.2% |
| Production data acquisition | 29 | 3.8% | 22.3% |
| Educational software | 18 | 2.4% | 13.8% |
| Other | 10 | 1.3% | 7.7% |
| No software | 9 | 1.2% | 6.9% |
| Total | 756 | 100.0% | 581.5% |

Asked for the *kind of support* (Q18*) required from software during process execution, the average rankings – on the scale 1 ('very important'), 2 ('important'), 3 ('not so important'), 4 ('not important at all') or 5 ('don't know') - reveal that the execution of tasks is most appreciated, followed by task routing, information providing and automated process execution (see Table 9). This overall ranking is consistent with the answers to the previous question (Q17).

**Table 9.** Required kind of support for process execution

| | | Execution of Tasks | Routing of Tasks | Information Providing | Automated Process Execution |
|---|---|---|---|---|---|
| N | Counts | 120 | 112 | 113 | 117 |
| na | Missing Values | 10 | 18 | 17 | 13 |
| Mean | | 2.11 | 2.43 | 2.54 | 2.83 |
| Standard Deviation | | 0.797 | 0.824 | 0.991 | 0.854 |
| Rank | | 1 | 2 | 3 | 4 |

The *software currently used* (Q19) in the companies to describe or manage business processes was also gathered; see Table 10. In line with the preferred ways of describing processes (Q12; Table 6), drawing and writing tools come first and second place, followed by a particular ERP system. The first BPM suite in the list is the ARIS toolset; in our sample, its prevalence slightly outperforms the one of in-house solutions and IBM Websphere BPM.

**Table 10.** Software used to describe business processes (N=129; na=1)

| Answers | Count | Percentage | Percentage responses each |
|---|---|---|---|
| MS Visio | 78 | 25.4% | 60.5% |
| MS Word | 60 | 19.5% | 46.5% |
| SAP R/3 or mySAP ECC | 39 | 12.7% | 30.2% |
| ARIS Toolset | 29 | 9.4% | 22.5% |
| In-house solution | 24 | 7.8% | 18.6% |
| IBM Websphere BPM | 23 | 7.5% | 17.8% |
| TIBCO iProcess Suite | 5 | 1.6% | 3.9% |
| iGrafx Suite, Oracle | 4 | 1.3% | 3.1% |
| ADONIS | 3 | 1.0% | 2.3% |
| Intalio BPMS, MS Excel, MS Powerpoint, MS Sharepoint, Nimbus | 2 | .7% | 1.6% |
| Prometheus Suite, Semtalk | 1 | .3% | .8% |
| Other | 26 | 8.5% | 20.2% |
| Total | 307 | 100.0% | 238.0% |

To get information on the distinct *criteria* influencing the *selection of BPM tools* (Q20*), we asked the companies to assign one of the ranks 1 (highest rank) to 5 (lowest rank) to each of the criteria listed in Table 11. In the overall order resulting from the mean ranks, functionalities and qualities are in first position, followed by pricing, support, the positive image and experience of the vendor as well as the prior availability of some tool in a company.

**Table 11.** Decision criteria for BPM tool selection (N=130, na=0)

| | Functionalities and Qualities | Pricing | Support | Vendor Image | Availability |
|---|---|---|---|---|---|
| Mean | 1.77 | 2.99 | 3.03 | 3.43 | 3.78 |
| Standard Deviation | 1.450 | 1.350 | 1.207 | 1.011 | 1.163 |
| Rank | 1 | 2 | 3 | 4 | 5 |

## 4 Discussion

In this section we interpret our results, relate them to the results of other studies (see Table 12) and comment on the validity of our findings.

To the best of our knowledge, we are the first to inquire into the characteristics of business processes on a global scale. The current picture is as follows: Processes tend to stay within a company (Q7*), which is typical for the BPM maturity Levels 1-3 [10]. Our assessment of current BPM maturity is supported by other studies that locate the companies worldwide on BPM maturity Level 2 [23]. In the companies of our sample, the average processes (Q1-Q5; Table 4) span 3-4 departments and involve around 7 persons as well as 20 tasks and 4 applications. The latter number also means that each process potentially (in the case of automation) requires substantial software integration. The main time unit for process duration and execution frequency is a day (Q8*, Q9*; Table 5); so, processes run fast and frequently – and they are stable (most changes occur yearly; Q10). Finally, processes are product-, administration- and customer-driven (Q6); the first and the latter driver conform to ‘increased customer satisfaction’ as a well-known, historical motive for BPR and BPM [15], [23].

The BPM maturity Levels 2 and 3 are characterized by documented processes and a process language [10], and our study suggests that both should be *‘human-oriented’*: First, ‘BPR’ as an organizational goal, ‘guidelines for humans’ and ‘documentation’ form the three most important motives to describe processes; process automation follows only in the fourth place (Q14). These findings are completely [4] or at least concerning BPR [23] and documentation [17] confirmed by other studies. Secondly, apposite to these motives, text in natural language (Q12; Table 6) and the corresponding tools (Q17; Table 8) are as popular to describe processes as modeling languages. Thirdly, usability is the by far most important requirement concerning BPM tools (Q16*; Fig. 1). Finally, the software involved in the execution of business processes (Q17; Table 8) and the kind of software support required for process execution (Q18*; Table 9) indicate that task routing is currently often in the hand of humans, who use the description of the processes as a guideline (Q14). The usage of text and tables to describe processes, the importance of usability and the required software support during process execution are original results of our study. The low prevalence of BPM suites (around 30% of all companies) other studies report [23], [11] affirm our finding that task routing is often not automated, but done by employees.

According to question Q18* (Table 9), the companies want software to undertake the routing of tasks. Our survey makes clear that a central requirement for this support is *software integration*: It is the third most important requirement (Q16*; Table 7), which is explained by the distribution of processes within and between companies (Q7*; Table 5) as well as the number (Q4; Table 4) and the kind (Q16*; Table 8) of applications involved in the execution of average processes.

The important role of ERP and CRM systems during process execution (Q16*; Table 8) was also observed by the study “Status Quo of BPM” [11]. Based on our results, this observation can be interpreted as follows: First, ERP systems fulfill the requirement of ‘task execution’ (Q18*; Table 9). Secondly, they are the most important ‘non-BPM’ software that incorporates process models (Q15). Having CRM systems in fourth position (Q17, Table 8 and [11]) among the software that is involved in process execution agrees with the process scope (Q6: a product or customer contact) and the BPM motive ‘customer satisfaction’ [23].

**Table 12.** Results of this survey and relations

| Findings of our survey | | Relation to other surveys | | | |
|---|---|---|---|---|---|
| | | [11] | [23] | [17] | [4] |
| **Processes are currently:** | | | | | |
| Mainly company-internal (Q3, Q7*) | ++ | ○ | ++ | ○ | ○ |
| Medium-sized (Q1, Q2, Q5), fast (Q8*), frequent (Q9*), stable (Q10), substantially applications involving (Q4), mainly product-driven (Q6) | ++ | ○ | ○ | ○ | ○ |
| **Processes descriptions are:** | | | | | |
| Mainly created decentralized and centrally aligned (Q13) | ++ | + | + | ○ | ○ |
| Needed for BPR, documentation, automated process execution (Q14) | ++ | ○ | + | + | ++ |
| Text, models (BPMN) or tables (Q12) | ++ | ○ | ○ | (BPMN) | ○ |
| **Software relevant to BPM:** | | | | | |
| Business processes rely on databases, ERP, office software, CRM software, EAI …. WfMS (Q17) | ++ | + | ○ | ○ | ○ |
| To describe business processes, graphics tools, text processing software, ERP systems and BPM suites are used (Q19) | ++ | ○ | + | + | ○ |
| The most important requirements concerning BPM tools are usability, modeling and software integration capabilities (Q16*) | ++ | ○/+/○ | ○/+/○ | - (BPMN tools) | ○ |
| During process enactment, software is needed to execute tasks, to route tasks or to provide information (Q18*) | ++ | ○ | ○ | ○ | ○ |

*Symbols*:
++ Completely confirmed, + Partially confirmed, —: Contradiction, ○: Not investigated

The importance of software integration for BPM is not visible from other empirical investigations. If they ask for requirements at all, they focus on requirements concerning

BPM suites and neglect more or less the overall IT context of BPM. However, it is generally confirmed that modeling capabilities are important for BPM tools [11], [23].

Our study shows, moreover, that process documentation is *opportunistic*, i.e., adapted to (1) the purposes of BPM, (2) the available standards and (3) the used tools: First, the above-mentioned human-oriented documentation can be conveniently realized by text and tables (Q12), whereas workflow execution or non-BPM software (Q15) need process modeling. Secondly, there seems to be some satisfaction with both the BPMN as the current process modeling standard – it is the prevalent modeling language (Q12; Table 6) – and the resulting modeling capabilities provided by tools: Language adaptability (rank 8) as well as support for different notations and standards (rank 11) are no longer important requirements of BPM tools (Q16*; Table 7). The worldwide acceptance of the BPMN as a process modeling standard is confirmed by other studies [23], [17]. Contrary to our results, for BPMN tools some extensibility at the level of attributes seems to be required [17].

Finally, the current focus on company-internal processes (Q7*; Table 5) and the just beginning integration of suppliers and vendors (Q3; Table 4) indicate that the companies of the sample are mainly on BPM maturity Level 2. The organization of process modeling (Q13), however, shows evidence of the next higher level of BPM maturity: Over two thirds of the companies either centrally align process modeling or even have a central process modeling team. This result of our investigation strengthens the observations of the report "State of BPM 2010" [23].

Because of the sampling procedure (see Section 3.1), our results can be assumed to be representative for large companies that operate worldwide, independent of their branch. However, the focus on large companies is accompanied by some lack of selectivity: Most of the investigated companies consist of more or less independent suborganizations that often use distinct process modeling methods or tools. As the majority of our participants had a company-wide picture of the processes (see Section 3.1), the impression about BPM on the company level often looks like 'anything goes'. This problem affects all investigations of large companies. However, the resulting impression of BPM in practice is very realistic.

The list 'Forbes Global 2000' we used suffers from three limitations: First, it disregards large non-American companies that don't have commercial relations with the USA, and, secondly, ranking companies based on sales, profit and market value favors branches where borrowed capital is important (e.g., banking and insurance companies). Thirdly, non-profit organizations (i.e., public administration, universities) are completely excluded. These limitations might well affect the validity of our results.

Half of our participants work for IT departments. The resulting IT focus is in line with our research goal and, at the same time, not restrictive: The majority of the participants had a company-wide picture of the processes, and the findings concerning human-orientation show that the role of IT was not overemphasized. Finally, by comparison with other investigations, this section has proven the plausibility of our results.

The ratings we have obtained from the companies reflect the subjective experiences of our participants, which is a common limitation of such surveys. From a methodical point of view, the calculation of mean ratings is only valid if the ratings of the underlying scale are equidistant (interval scale). This assumption is generally made [6]. Additionally, our conclusions rely on the *orders* of alternative answers resulting from the calculated means and, thus, remain on the ordinal scale of the ratings. Moreover, wherever appropriate we referred to the modes as they are valid descriptive statistics for nominal and ordinal data and insensitive to outliers [3].

## 5 Conclusions and Outlook

We have conducted an empirical investigation of 130 major public companies from all over the world about the requirements of BPM, which must be satisfied by the used BPM tools. Our investigation confirms that the majority of companies are on the BPM maturity Level 2, where the core business processes are defined and documented. The soon transition to the BPM maturity Level 3 is possible in two-thirds of the inquired companies, since they already centrally align process modeling or even have a central process modeling team.

The transition to the highest BPM maturity level seems to be hindered by the fact that current BPM tools do not satisfy two requirements: First, BPM requires process modeling languages and BPM tools that are designed for the *use by humans* (currently the focus is on automation). Secondly, BPM tools must be able to *integrate* the heterogeneous applications involved in the execution of business processes. Both requirements hint at necessary improvements of process modeling languages and BPM tools.

Usually, requirements engineering is not done by surveys, but for particular cases and organizational settings. For that reason we asked the companies that participated in our survey whether they would allow a more in-depth investigation of their processes. Twenty-five companies agreed. We will continue our requirements engineering research by an analysis of these cases to obtain detailed, qualitative data.

## References

1. Economist Intelligence Unit (in co-operation with the IBM Institute for Business Value): The 2006 e-readiness rankings. A White Paper from the Economist Intelligence Unit. London et al. (2006)
2. Forbes: The Forbes Global (2000), http://www.forbes.com/2005/03/30/05f2000land.html
3. Gravetter, F.J., Wallnau, L.B.: Statistics for the Behavioral Sciences, 8th edn. Wadsworth, Belmont (2009)
4. Indulska, M., Recker, J., Rosemann, M., Green, P.: Business process modeling: Perceived Benefits. In: Laender, A.H.F. (ed.) ER 2009. LNCS, vol. 5829, pp. 458–471. Springer, Heidelberg (2009)
5. 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)
6. Jackson, S.L.: Research Methods and Statistics: A Critical Thinking Approach, 3rd edn. Wadsworth, Belmont (2009)
7. Jaklic, J., Bosilj-Vuksic, V., Stemberger, M.I.: Business Process Oriented Tool Selection Model – A Case Study. In: Hlupic, V., et al. (eds.) 1st Int. Conf. Future Challenges and Current Issues in Business Information, Organization and Process Management 2006, pp. 94–102. Westminster Business School, London (2006)

8. Künger, P., Hagen, C.: The fruits of Business Process Management: an experience report from a Swiss bank. Business Process Management Journal 13, 477–487 (2007)
9. Limesurvey, Version 1.85, http://www.limesurvey.org/
10. McCormack, K., et al.: A global investigation of key turning points in business process maturity. Business Process Management Journal 15, 792–815 (2009)
11. Neubauer, T.: An empirical study about the status of business process management. Business Process Management Journal 15, 166–183 (2009)
12. Nietro-Ariza, E.M., Rodriguez-Ortiz, G., Ortiz-Hermández, J.: An empirical evaluation for business process tools. In: Ochoa, S.F., Roman, G.-C. (eds.) Advanced Software Engineering: Expanding the Frontiers of Software Technology, pp. 77–84. Springer, Boston (2006)
13. Object Management Group (OMG): Business Process Model and Notation (BPMN), Version 1.2. OMG Document Number: formal/2009-01-03, http://www.omg.org/spec/BPMN/1.2/
14. Pernici, B., Weske, M.: Business process management. Data & Knowledge Engineering 56, 1–3 (2007)
15. Pritchard, J.-P., Armisted, C.: Business process management – Lessons from European Business. Business Process Management Journal 5, 10–35 (1999)
16. Rosemann, M., de Bruin, T., Hueffner, T.: A Model for Business Process Management Maturity. In: Proc. ACIS 2004, Paper 6 (2004), http://aisel.aisnet.org/acis2004/6
17. Recker, J.: BPMN Modeling – Who, where, how and why. BPTrends (March 2008) http://www.sparxsystems.com/press/articles/pdf/bpmn_survey.pdf
18. Schmietendorf, A.: Assessment of Business Process Modeling Tools under Consideration of Business Process Management Activities. In: Dumke, R., et al. (eds.) IWSM 2008. LNCS, vol. 5338, pp. 141–154. Springer, Heidelberg (2008)
19. van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F., Verbeek, E.: Business process management: Where business processes and web services meet. Data & Knowledge Engineering 61, 1–5 (2007)
20. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer, Berlin (2007)
21. The Workflow Management Coalition (WfMC): The Workflow Reference Model, Document Number TC00-1003, Issue 1.1 (November 19 1995), http://www.wfmc.org/standards/docs/tc003v11.pdf
22. The Workflow Management Coalition (WfMC): Terminology & Glossary, Document Number WFMC-TC-1011, Issue 3.0 (February 1999), http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf
23. Wolf, C., Harmon, P.: The State of Business Process Management 2010. BPTrends Reports (February 2010), http://www.bptrends.com/surveys_landing.cfm
24. zur Muehlen, M., Recker, J., Indulska, M.: Sometimes less is more: Are process modeling languages overly complex? In: Proc. EDOC Conference Workshop, EDOC. Eleventh International IEEE, Annapolis, MD, pp. 197–204 (2007)
25. zur Muehlen, M., Recker, J.: How much language is enough? Theoretical and practical use of the Business Process Modeling Notation. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 465–479. Springer, Heidelberg (2008)

# How Novices Model Business Processes

Jan Recker, Niz Safrudin, and Michael Rosemann

Queensland University of Technology 126 Margaret Street, Brisbane QLD 4000, Australia
{j.recker,norizan.safrudin,m.rosemann}@qut.edu.au

**Abstract.** In this paper, we examine the design of business process diagrams in contexts where novice analysts only have basic design tools such as paper and pencils available, and little to no understanding of formalized modeling approaches. Based on a quasi-experimental study with 89 BPM students, we identify five distinct process design archetypes ranging from textual to hybrid, and graphical representation forms. We also examine the quality of the designs and identify which representation formats enable an analyst to articulate business rules, states, events, activities, temporal and geospatial information in a process model. We found that the quality of the process designs decreases with the increased use of graphics and that hybrid designs featuring appropriate text labels and abstract graphical forms are well-suited to describe business processes. Our research has implications for practical process design work in industry as well as for academic curricula on process design.

**Keywords:** Design skills, process modeling, design quality, experiment.

## 1 Introduction

When seeking to (re-) design business processes to organizations increasingly use graphical documentations of their business processes – so called process models [1]. These models act as blueprints of organizational processes, and are a key tool for making re-design decisions, i.e., decisions about where, how and why changes to the processes should be enacted to warrant improved operational efficiency, cost reductions, increased compliance or better IT-based systems.

Essentially, a process model is a cognitive design tool allowing the process analyst to offload memory and information processing, and to promote discovery and inferences about the process at hand [2]. When the process design activity is not computer-supported (e.g., through a modeling tool), analysts use basic tools such as pencil and paper to illustrate how a business operates at present (as-is process design) or in the future (to-be process design).

Our interest in this paper is in the way analysts use the affordances offered by paper and pencil to create diagrammatic representations of business process designs. Specifically, we seek to understand how novice analysts create business process design representations when they are uninformed of any process design method (such as a process modeling notation like BPMN [3]). We have several reasons for this specific focus of our study. First, in organizations, the share of employees equipped with method knowledge about process design methods is typically radically low. Domain

R. Hull, J. Mendling, and S. Tai (Eds.): BPM 2010, LNCS 6336, pp. 29–44, 2010.

experts involved in process (re-) design work are often unable to review a (semi-) formalized process model or to provide meaningful feedback. In some cases domain experts even reject process models because of a lack of exposure and training in process modeling methods [4]. Second, process design artifacts (e.g., the process models) are meant to facilitate a shared understanding in the organization, which therefore includes employees unfamiliar with the chosen process design method. Third, studies of process design in industry practice [5] still report on the widespread use of 'butcher paper' process design work. Typical workshops on process design employ design tools such as whiteboards, flip charts and post-its to capture knowledge about a current or future process [6]. Fourth, informal sketches and diagrammatic drawings were found to be key to any design activity, as they serve as an externalization of one's internal thoughts, and assist in idea creation and problem-solving [7, 8], two key skills to support business process re-design. Deriving insights on these external representations may therefore promote an understanding of how individuals form their own cognitive framework in process design work [9]. In conclusion, understanding how uninformed analysts externalize their conceptions of a business process design using a basic cognitive tool such as paper and pencil is an important object of study.

When given basic cognitive design tools without the use of a (semi-) formalized design method, individuals have numerous ways to illustrate a business process design. For instance, their design diagrams may entail the use of textual descriptions, graphical icons, geometric shapes, or even cartoon sketches, to name just a few. An example for such an informal design diagram, representing an airport check-in and boarding process, is given in Fig. 1.

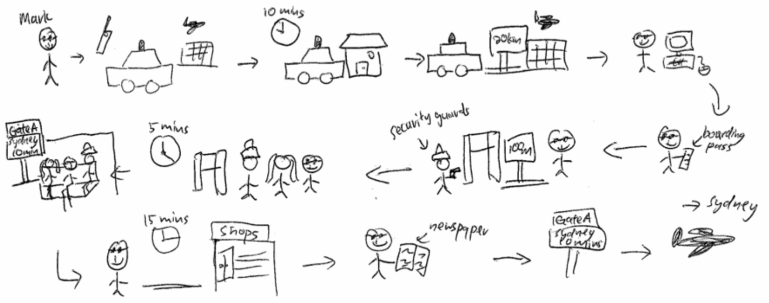


**Fig. 1.** Example of an informal business process diagram

The aim of our research is two-fold. First, we seek to understand which design forms novice analysts choose when conceiving business process diagrams with paper and pencil. Second, we seek to establish differences between these process design types in terms of their ability to convey relevant information about the business process represented. To that end, in this paper we report on an empirical analysis of process design work carried out by a team of student analysts as part of their university coursework. We state the following research questions:

RQ1 How can process design representations chosen by novice analysts be characterized?

RQ2 How good are different types of process designs in describing important elements of a business process?

We proceed in the following manner. First, we review prior work on process modeling as a design activity, and related work from design disciplines that provide an understanding of the design process as such. We then discuss our research model. Next, we discuss how we collected data on informal business process designs by novice analysts, and how we prepared this data for analysis. In section 4 we give the results from our study, and present a discussion of these results in section 5. We conclude this paper in section 6 by reviewing contributions, implications and limitations.

## 2 Background

### 2.1 Prior Work

The common aim of process design representations such as process models is to facilitate a shared understanding and to increase knowledge about a business process, so as to support problem solving for making (re-) design decisions, a task performed by business analysts and systems designers, for instance, in the context of organizational re-structuring, compliance management or workflow implementations. Following Simon [10], we can classify process modeling as a design activity because process models are used to represent the (process) problem so as to make potential solutions apparent. Being the most commonly employed cognitive vehicle in process (re-) design work, process models are therefore asked to be readily and intuitively understandable by the various stakeholder group engaged in this work [11].

Various approaches have been suggested to measure the quality of a process model (e.g., [12, 13]). Yet, these only apply to formalized process modeling methods such as Petri Nets, EPCs or BPMN only. However, these approaches are not applicable to informal design representations such as sketches, diagrams or text that do not follow an explicit meta model and well-defined syntactical rules. For us to be able to judge the quality of informal business process design representations, we turn to diagram correctness criteria suggested by Yang et al. [14], and the quality of a process design as its ability to accurately represent all the important constituent factors of a business process in context, i.e., the *activities*, *events*, *states*, and *business rule* logic that constitute a business process [15]. We complement these process-specific correctness criteria with two criteria found to be important in general design work, viz., *temporal* and *geospatial* design information [2, 16]. These two criteria, in a process design, relate to where (geographical location) and when (temporal location) work tasks in a business process have to be carried out.

Fig. 2 illustrates how typical cognitive design vehicles, in this case a BPMN process model, meet these criteria. Specifically, it shows that *temporal* and *geospatial* design information is normally absent from these design representations.

Process modeling, as any design work, is a cognitive activity [17]. Regardless of the work discipline, designs bear similarities, particularly in terms of the cognitive approach taken by the designer. For instance, an architectural student is more likely to generate multiple solutions to a problem before arriving at a final design, whereas a science student is more likely to analyze a problem thoroughly before drawing out only one design solution [17].

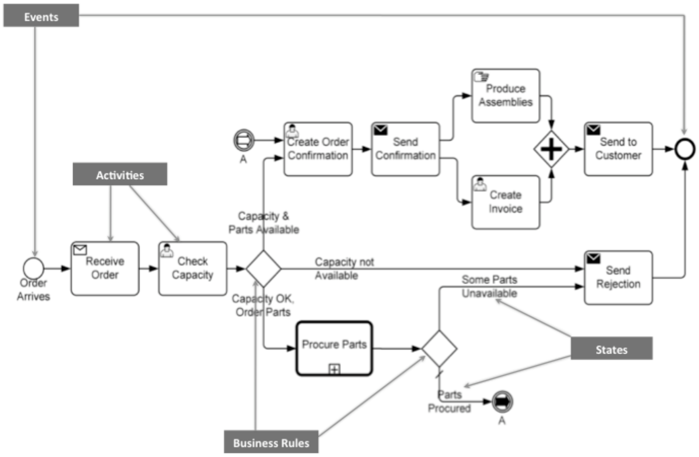


**Fig. 2.** Important Constituent Process Elements in a BPMN Diagram

Viewing process modeling as a design activity suggests the importance of prior experience in design approaches (e.g., experience in process modeling methods) to this activity. For instance, Wang and Brooks [18] found that novice modelers conceptualize important domain elements in a fairly linear process in contrast to experts, who were found to have better analysis and critical evaluation skills.

Looking at the artifacts created in process design work, business process diagrams, at a very simple level, typically entail the use of graphic icons, basic geometric shapes, and textual information [11]. Several studies highlight the importance of visual means to aid understanding of the design outcome – which is the key premise underlying process modeling [19]. Visual attributes function as an aid for the human mind to recognize and group objects in diagrams [20]. Work on imagery have shown how images have particular properties [21] that can affect interpretations. These findings suggest that different types of visual aids used in business process design will affect interpretation and understandability of the created process models.

Often, conceptual design work is carried out using informal sketching, a process of mental imagery [22], with the purpose of identifying properties of imaged elements to enable the retrieval of information from memory. Like drawings, sketching across multiple disciplines plays a consistent role in the generation, development, evaluation, and communication of ideas [9], which suggests their applicability to process (re-) design activities.

Within sketches as well as more formal process diagrams, the use of graphical icons, in addition to geometric shapes, is often prevalent. This is because graphic icons are quicker and easier to recognize than text [23]. The two types of graphic icons typically used in process diagrams can be categorized as *Concrete* and *Abstract*. *Concrete*, high-imagery and high frequency graphics, are often represented with freehand sketches of objects such as stickman figures and telephone icons (see Fig. 5c-Fig. 5e), while *Abstract* are low-imagery, low-frequency graphics that entail geometric shapes and arrows [24] (see Fig. 5b). Also, process diagrams typically feature textual information in the form of labels attributed to geometric shapes (like activity boxes) or additional free-text descriptions. Textual information plays a vital role in ensuring proper interpretation and association, as well as to enhance the building of a cognitive model [20]. Textual information further enhances the graphical information in a process diagram, because textual and graphical information can be processed in parallel through the complementary receptor channels of the human brain [25].

In conclusion, we assert that a study of process design work with informal representation forms should consider, at least:

- which representation aids are used in the process design (e.g., the use of textual means, geometric shapes, iconic imagery, and the like);
- to what extent process design means enable a reader to receive all relevant information about a business process (such as important events, activities, states, or business rules);
- whether and how temporal or geospatial information about the business process is conveyed; and
- how individual experience levels, specifically with design work, with modeling approaches or with the process itself, contribute to the design work.

## 2.2 Research Model

Based on our review of relevant work, we conceptualize the above research objectives that we attempt to address in this study in the research model shown in Fig. 3.

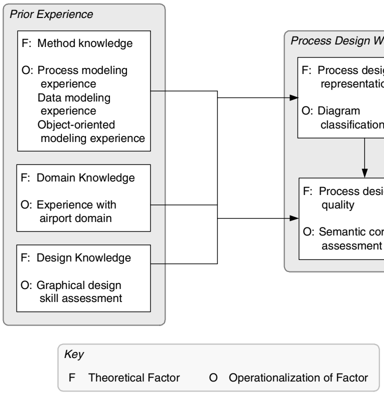

Prior Experience
F: Method knowledge
O: Process modeling experience
Data modeling experience
Object-oriented modeling experience
F: Domain Knowledge
O: Experience with airport domain
F: Design Knowledge
O: Graphical design skill assessment
Process Design Work
F: Process design representation type
O: Diagram classification
F: Process design quality
O: Semantic correctness assessment
Key
F Theoretical Factor
O Operationalization of Factor

**Fig. 3.** Research Model

In line with our research questions, first, we seek to understand the types of process design representations chosen by novice analysts. To that end, we seek to ascertain to which extent prior experience determines the type of process design representation used. As per Fig. 3, we distinguish two forms of experience: Following Khatri et al.

[26] we differentiate (a) *experience with a method* (a modeling approach) from (b) *experience with a process* (knowledge of the process domain). We anticipate that novice analyst with an educational or working background in any formalized modeling approach (data-, process- or object-oriented) would have a predisposition towards the diagramming representation typically associated with the modeling approach, which can be expected to affect their preference for such a process design representation type. Domain knowledge has been shown to affect modeling processes and outcomes [26], and may thus influence both the type and quality of the process design conceived. Given the importance of graphical and visual cues in conceptual design work [20, 21] we further expect that novice analysts with *experience in graphical design work* may choose a design representation format that is more graphically than textually oriented.

Second, we seek to examine the outcome of the process design work. Following Fig. 3, our interest in the outcome of the design process is two-fold, namely the *type of process design representation* chosen by the novice analysts, and the *quality of the designs created*. In the following, we describe how we collected data to examine our research model.

## 3 Method

Data collection was conducted using a three-part quasi-experiment conducted with a group of Information Systems students enrolled in a Business Process Modeling subject unit as part of their university Information Systems course. The experiment took place during opening minutes of the very first lecture in the subject in a lecture hall, consuming approximately 25 minutes.

The first part of the experiment captured demographic information about the students, viz., their level of education (under-graduate or post-graduate), gender, English Language as their arterial language, their experience in formalized modeling methods (process-, data- and/or object-oriented), and their familiarity with the procedures at an airport, which was the process domain selected for our study.

The second part of the experiment aimed at assessing the students' ability to draw graphical diagrams, as a proxy measure for graphical design skills. To that end, a picture of the Sydney Opera House was projected to the participants, who were to draw an accurate sketch of the image on a blank piece of paper. The rationale behind the Sydney Opera House image was based on the assumption that the majority of the participants would be familiar with the landmark, as it represents one of Australia's most prominent features. Students were given ten minutes to complete this task but task times were not recorded.

The third part of the experiment was to examine the students' ability to create a business process design representation. A specific process scenario was portrayed in textual format to the participants as a narrative of an actor seeking to travel to Sydney. This included a detailed account of the arrival at the airport, followed by check-in and boarding procedures and leisurely activities taken in between. The rationale behind this activity was to provide a business process with which both domestic and international students would have some level of familiarity with (as opposed to a business process in a specific industry vertical – for instance, insurance – where results could have been significantly biased due to non-existence of any domain knowledge). Students were asked to draw a model that represents the airport process scenario as accurately and completely as possible, within ten minutes, using only a blank piece of paper.

Overall, 89 students participated voluntarily in the study. Complete data about all three parts of the experiment were provided by 75 students (84%).

# 4 Analysis and Results

Data analysis proceeded in several steps. First, we coded the demographic information obtained. Our specific interest was in students' experience of airport processes (domain knowledge), as well as experience in formal modeling methods – process modeling knowledge (PMK), data modeling knowledge (DMK), and object modeling knowledge (OMK).

Second, we assessed the quality of the Opera house drawings, to create a measure of graphical design skills. To that end, all drawings were provided to a professional artist, who judged each drawing using a six-item drawing quality measure that assessed *composition* (COM), *proportions* (PROP), *perspectives* (PERS), *shading* (SHAD), *drawing style* (STY) and *overall impression* (IMP) of the drawings on a 7-point scale (1 = very bad, 4 = neutral, 7 = very good).

Third, to distinguish different process design representation types, we categorized the various types of process design representations created in the third part of the experiment, in accordance with their aesthetic design properties. This assessment included the examination of the relative use of graphical icons, textual information, and sequential flow or structure of the process diagram. To ensure coding reliability, all diagrams were assessed separately by three research assistants, who then, iteratively, met to discuss, defend and revise their coding work until consensus was reached.

Fourth, we attempted to measure the quality of each process design representation. To that end, we adapted the semantic correctness criteria suggested by Yang et al. [14] to the constituent elements of business process models (activities, events, states, business rules, see [11]) and other design artifacts (temporal and geospatial information, see [2]), in a six-item 5-point scale (1 = aspect not at all represented, 5 – aspect fully represented). Again, we used a three-member coding team and an iterative consensus-building process to ensure validity and reliability of our assessment.

Using this data, the following sections report on the analyses carried out to address the two research questions as per our research model (see Fig. 3).

## 4.1 Identifying Process Design Types

Our coding of the 75 process diagrams resulted in the identification of five process design archetypes. This assessment was based on the aesthetic representation of the process diagrams, such as frequency of graphic use, textual information, and the sequential flow of the process structured within the Euclidean space afforded by the piece of paper. Similar to the Physics of Notations suggested by Moody [27], we found that the archetypes could be differentiated based on their use of text and graphics.

Fig. 4 positions the five identified archetypes along a continuous scale from dominantly textual (type I) to dominantly graphical (type V) representation formats, and describes key traits of each design type. Fig. 5a-5e provide examples for each design archetype.

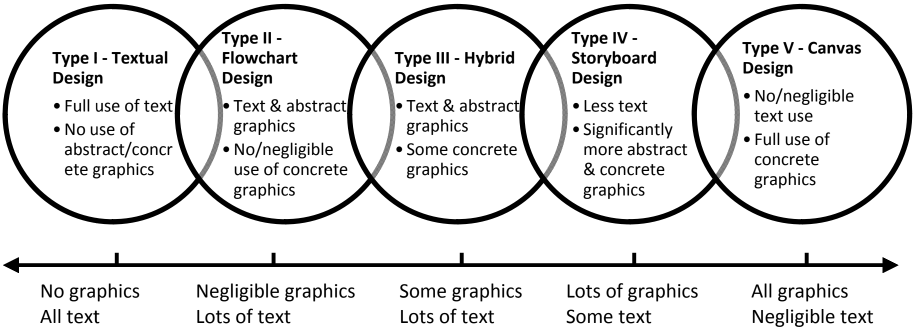


**Fig. 4.** Process Design Archetypes

The first type, *Textual design*, resembles very closely that of an algorithm pattern. This design type does not utilize any form of graphical illustration but uses lines of words as the primary representation of process information. The second type, *Flowchart design*, contains textual information embedded within graphical shapes that are of abstract nature, i.e., lines/arrows and/or boxes and borderlines around captions, and generally have a sequential flow that, to some extent, resembles more formal modeling techniques used for process-, data-, or object-modeling, and of course, the classical flowchart. The third type, *Hybrid designs,* uses concrete graphics (such as stickman figures, telephone icons and the like) to supplement the textual labels and descriptions in the presence of abstract graphics (shapes and boxes). The Hybrid design also follows a structured process flow of information. The forth and fifth design types are notable due to the distinctively dominant use of concrete graphics over and above textual representations. The *Storyboard design* uses a great variety of concrete graphics such as icons, complemented with brief textual descriptions, typically in the form of verbs and nouns.

Resembling a real “Storyboard”, this design type further features segmented pieces of information, some partitioned as objects within rectangular boxes (abstract graphics) or swim-lanes, and were structured in a flowing manner to accommodate the Euclidean space and orientation of the paper. As for the *Canvas design*, the entire process is illustrated with concrete graphics without any meaningful use of textual information, occupying the entire page of the paper to provide a picturesque view of the scenario. Due to the “picture-painting” nature of this design, the diagram lacks any precise representation of the process flow, or detailed textual information.

Having distinguished the five different process design representation types, we examined whether any of the experience factors we considered (method, domain or graphical design experience) was significantly associated with any of the design representation types chosen by the participants. To that end, we ran logistic regression analyses [28] for each design type (DT1-5), using as independent factors three binary variables PMK, DMK, OMK capturing respondents' prior experience with modeling methods, and six factor scores (IMP, COM, PROP, PERS, SHAD, STY) describing the graphical design skills as per the evaluation from a professional artist. Last, for the factor domain knowledge we created a binary variable groupDK that grouped respondents into two groups (high/low) as per their self-perceived rating of familiarity with airport procedures.

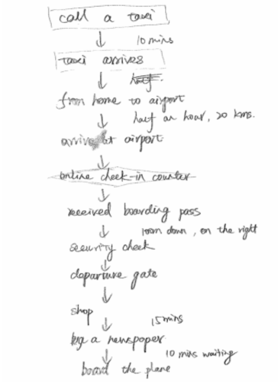


**Fig. 5a.** Process Design Type 1: Textual (1 diagram)

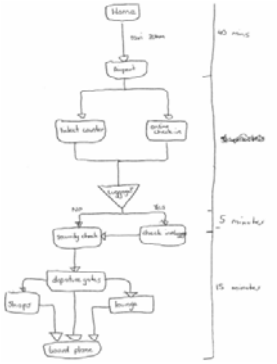


**Fig. 5b.** Process Design Type 2: Flowchart (54 diagrams)

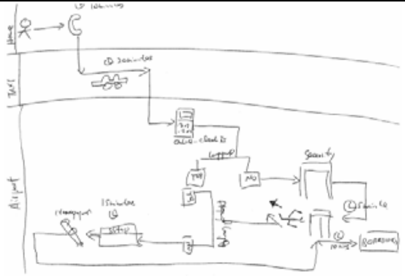


**Fig. 5c.** Process Design Type 3: Hybrid (6 diagrams)

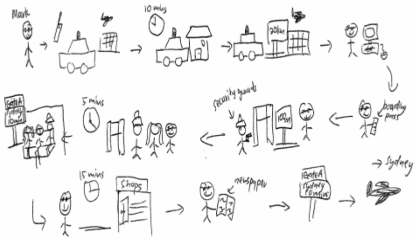


**Fig. 5d.** Process Design Type 4: Storyboard (11 diagrams)

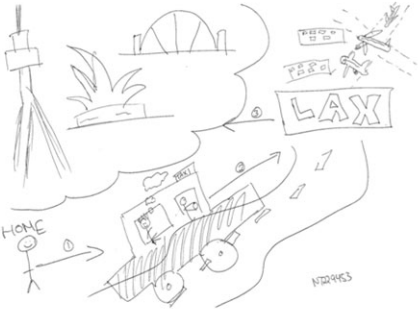


**Fig. 5e.** Process Design Type 5: Canvas (3 diagrams)

We omit a complete description and discussion of the results. The results from the logistic regression analysis showed that there are no significant relationships between the independent variables considered with DT1 (textual design), DT3 (hybrid design) and DT5 (canvas design). It may well be that the non-significance of the results for DT1 and DT5 is due to the limited sample size.

For DT2 (flowchart design) however, we found a significant association with previous domain knowledge (Beta = 1.465, p = 0.039). This result suggest that people highly familiar with airport procedures tend to prefer a flowchart-based representation of airport processes, unlike the representation format of current process modeling methods. Interestingly, for this design, process modeling method knowledge was a largely insignificant predictor (Beta = -0.444, p = 0.534). This finding suggests that domain expertise dominates method expertise as a predictive factor. It might well be that the thorough understanding of the domain facilitates the capability to abstract the process into the form of flowcharts while pure method expertise is not sufficient to clearly identify and isolate the individual steps of this process.

For DT4 (storyboard design), we found a significant association with object-oriented modeling method knowledge (Beta = -3.619, p = 0.009). Note that participants unfamiliar with object-oriented modeling methods showed a significant association with predominantly graphic storyboard process designs, whereas those with object-oriented modeling method knowledge did not choose this design type. This could indicate that the loose and creative structure of storyboard forms a contrast to the conceptually advanced ideas of object-orientation and its paradigms such as coupling and decomposition.

### 4.2 Evaluating Process Design Quality

Next, we examine the data collected about the quality of the process designs, as per our six-item semantic correctness measure adapted from [14]. We proceeded in two steps.

First, we ran a Univariate Analysis of Variance (ANOVA, [28]), with *Design Quality* (DQ) as an aggregate dependent variable, computed as the average total factor score of the six semantic correctness scale items. As independent factors we used *design type* (DT), the binary grouping variable *domain knowledge* (groupDK), the three measures for previous modeling *method knowledge* (PMK, DMK and OMK) and the *graphic design* score overall impression (IMP). The ANOVA results showed that design type (F = 12.459, df = 4, p = 0.000) and previous domain knowledge (F = 9.569, df = 1, p = 0.005) are significant predictors of the aggregate design quality measure, whilst the other independent factors as well as all interaction effects were insignificant. The results from the ANOVA specifically showed that higher levels of domain knowledge results in higher quality designs, and that more textually oriented process design representation types achieved higher quality scores than the graphically oriented process design representation types (as per the classification in Fig. 4).

To examine these results in more detail, we then ran a Multivariate Analysis of Variance (MANOVA), with the six semantic correctness measures as dependent variables, and the same input factors as above. Table 1 gives selected descriptive results from the MANOVA about the impact of the design type, and Table 2 displays corresponding significance levels.

**Table 1.** Multivariate ANOVA: Selected Descriptive Results

| DT with highest *mean* results | State | Task | Event | Business Rules | Time | Distance |
|---|---|---|---|---|---|---|
| **DT1** | 5.00 | 5.00 | 1.00 | 4.00 | 4.00 | 5.00 |
| **DT2** | 2.98 | 3.81 | 2.81 | 4.06 | 3.15 | 3.07 |
| **DT3** | 2.50 | 3.00 | 1.33 | 3.17 | 3.00 | 3.67 |
| **DT4** | 2.73 | 2.82 | 1.27 | 3.09 | 2.91 | 3.73 |
| **DT5** | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |

The results from Table 1 and Table 2 suggest that there is relationship between the type of design employed by the students to represent the business processes and the different dimensions of the quality of these designs. Specifically, Table 1 suggests that more textually oriented design types are better in representing the *State*, *Task*, *Event*, and *Business Rules* aspects (under elimination of DT1 – which only featured one case). The purely graphical design, DT5 Canvas, scored the lowest aggregate in representing all six factors that entail the design quality. We note specifically that DT2 (Flowchart) scored the highest aggregate in all aspects of quality, except for *Distance*, which is best represented with DT4 (Storyboard). Table 2 shows that these score differences were significant, except for the quality dimension *Business Rules*, where we did not identify a significant association with the type of design used. These findings suggest that the use of graphical shapes in combination with textual encoding leads to superior design representations, and offer some empirical evidence for the theory of effective visual notations offered by Moody [27].

**Table 2.** Multivariate ANOVA: Significant Results of design type and interaction effects

| Independent variables with *significant* results | Significance levels | | | | | |
|---|---|---|---|---|---|---|
| | State | Task | Event | Business Rules | Time | Distance |
| DT | 0.005 | 0.002 | 0.011 | - | 0.003 | 0.007 |
| DT & PMK | - | - | - | - | 0.002 | 0.001 |
| DT & OMK | - | - | - | - | 0.017 | - |
| DT & groupDK | - | - | - | - | - | 0.016 |

Table 2 further suggests important interaction effects stemming from the type of knowledge possessed by the participants. We note that participants with prior process modeling knowledge, when exercised with their choice of design, achieved higher quality scores for their representations of *Time* and *Distance*. Subjects with object modeling knowledge were found to be better in representing *Time* with their design type, while those with previous domain knowledge were found to be better in representing *Distance*.

Perusing MANOVA we further found a number of interesting effects on design quality stemming from prior experience of the subjects. Table 3 summarizes the significance levels for the different types of prior experience captured.

**Table 3.** Multivariate ANOVA: Significant Results of prior experience

| Aspects with *significant* results | **State** | **Task** | **Event** | **Business Rule** | **Time** | **Distance** |
|---|---|---|---|---|---|---|
| **PMK** | - | - | - | 0.037 | - | - |
| PMK & DMK | 0.023 | - | - | - | - | 0.010 |
| **DMK** | - | - | - | - | 0.023 | - |
| DMK & GroupDK | - | 0.032 | - | - | - | - |
| **OMK** | - | - | - | - | 0.018 | - |
| OMK & GroupDK | - | - | - | 0.047 | - | - |
| OMK & PMK | 0.006 | - | - | - | - | - |

Examination of the data displayed in Table 4 shows that those participants with knowledge of process modeling methods achieved higher scores for representing *Business Rules* (p = 0.037). Students with both process and data modeling knowledge achieved higher scores for representing *States* (p = 0.023) and *Distance* (p = 0.010). *Time* was well represented by students with data modeling knowledge (p = 0.023) and those with object modeling knowledge (p = 0.018). The data also showed the existence of an interaction effect between students with both data modeling and domain knowledge in representing *Tasks* (p = 0.032), while those with object modeling and domain knowledge represented *Business Rules* well (p = 0.047). Last, we found an interaction effect concerning the representation of States (p = 0.006), for those participants with both object and process modeling knowledge. These findings suggest that different method knowledge, solely or when combined with other method or domain knowledge, can increase the specific level of quality in a business process diagram.

## 5 Discussion

The finding that design representation forms chosen to conceptualize business processes range from predominantly textual, to hybrid, to predominantly graphical types, and the finding that some of the design types, more notably the combined graphical and textual types, achieve higher quality scores, extend our understanding on the use of conceptual design tools and the quality traits of the design outcomes.

We turn to Dual Coding theory [29] to discuss our results. This theory stipulates that text and graphics together can provide a more effective conveyance of information than using either on their own. We find that design types two (Flowchart) and three (Hybrid) both fall under this banner. Yet, the results regarding the relative superiority of the Flowchart over the Hybrid design type provides an important extension of Dual Coding theory, by suggesting that text and *abstract graphics* (shapes such as boxes, circles and arrows) apparently are more effective in displaying important domain semantic elements than the combined use of text and *concrete graphics* (icons such as stickman figures – as found in design type 3, hybrid).

And indeed, during the three-member evaluation of the process diagrams, it was reported that certain concrete icons, when unfamiliar with the given context, tended to create a certain level of ambiguity towards the end-users. For instance, one of the coders mis-interpreted a sketched icon representing the utility of an online check-in facility (as per context scenario), as a public restroom. This anecdotal evidence further corroborates our findings on the Hybrid design.

Moody's [27] theory of effective visual notations provides a rationale for this finding. The use of concrete graphics such as icons can in some instances violate the notion of monosemy whereby a symbol should have one predefined and independent meaning. This is not to say that all concrete graphics used in diagrams are undefined. For instance, the use of concrete graphics such as stickman figures, which clearly represent the main actor in a process, or a combination of a stickman with a telephone icon, followed by a taxi vehicle, can clearly indicate the representation of the actor calling a taxi as described in the process scenario. Such icons are of a semantically immediate nature, which allows novices to establish its meaning based on their appearance alone [27]. Still, the only partial and inconsistent use of semantically immediate concrete graphics in more graphically oriented diagrams (types 3, 4, or 5) may explain why the more textually-oriented process diagrams, such as the Flowchart design, which employ abstract graphics such as geometric shapes and arrows, appear to provide more clarity in conveying process information. Moody [27] highlights such symbols as being semantically opaque, in which the relationship between a symbol's appearance and connotation is merely arbitrary. Note that we found that predominantly students with notably high levels of domain knowledge tended to employ this design type with increased use of text and semantically opaque symbols. This finding would suggest specifically that geometric shapes can faithfully be used to describe different constituent elements of a process such as activities (typically rectangles), events (typically circles) or business rules (typically diamond-shaped gateways). It also highlights the important role of appropriate textual labels and the importance of conventions to guide the textual semantic specification of these labels.

Further note that the Flowchart design was also found to be the most favored type of design by the majority of students (72%), which may not only indicate preference, but perhaps also the novice's default way of conveying process information (using bare minimum concrete graphics).

Turning to what appears to be the second most used type of design (15% of students), the Storyboard design, we note that the simultaneous use of both graphics and text, plus a structured flow of process, may imply intuitiveness of graphical use to emphasize representation. And indeed, the theory of spatial contiguity [30] suggests that inclusion rather than segregation of both text and images can be more effective towards the end-user in terms of comprehension, regardless of spatial and verbal abilities. This theory may also contribute to explaining why we found only one case of design type 1, Textual design, as, per theory, such diagrams lack the intuitiveness of graphics for end-users.

Therefore, we posit that concrete graphic icons, in certain instances, enable a reader to receive and understand relevant information. They are aesthetically pleasing as people generally have a preference on real objects rather than abstract shapes. However, our study shows that abstract icons, in conjunction with the use of textual information, are beneficial for those who lack designing skills or diagramming expertise. It is also important to note that while graphics may be attributed a more readily intuitive appearance, an overuse of concrete graphics over and above textual or abstract graphical shapes can also be detrimental, as we have seen in the case of design type 5, Canvas design, which has the lowest design quality in conveying semantic correctness. Do et al. [7] studied how verbal protocols and reasoning account for inaccurate designing processes. Their findings suggest an impact of the verbal instructions (to draw a model of the airport scenario) given to the individuals who adopted the Canvas design. The novices interpreted the word "draw" literally, resulting in a strongly picturesque design of the process, thus signifying the imagery's congruence to one's perception and various psychological phenomena [31].

As a last item of discussion, we turn to the representation of the "non-standard" contextual process elements temporal and geospatial information.

We found that distances appeared to be best represented through the Storyboard design, whose dominant representation comprises of graphics, both abstract and concrete, with little textual annotation. Notably, we found the most prominent representation to be a signboard graphic icon with the unit of measure (e.g., 3 km).

Temporal information, on the other hand, was found to be best conveyed again through Flowchart designs. In this style, we found that temporal information was generally conveyed using text labels and abstract shapes such as additional timeline arrows complementary to the process flow. This finding could suggest that it is deemed more accurate for both the illustrator and the reader to use textual descriptions of time periods, as opposed to drawing a clock icon (a concrete graphic) to indicate a particular time or duration.

## 6 Conclusion

In this paper we reported on an experimental study carried out to examine how novices conceptualize their understanding of a business process using paper and pencil. We considered three main factors, namely, drawing skills, formal modeling method knowledge, and domain experience, to determine the impact on the quality of the process design against the resulting design types. Our findings reveal that the five types of design range from being dominantly textual, to a hybrid of text and graphics (both abstract and concrete), and to being dominantly graphical.

We acknowledge that our study bears certain limitations. First, the subjects observed were students and not business analysts. As such, our findings may only hold for novice analysts, which, however, was the desired cohort for our study. Second, there could be some subjectivity in our coding of data analysis. We attempted to mitigate potential bias through a multiple coder approach. Third, our attempt to ascertain the designing skills of the students could be seen as an assessment of their *drawing* but not their *design* skills. Another limitation is the potentially limited explanatory power of the statistical analysis due to the non-normal distribution of the design categories, and their relative sample size. For some design types we received only few data points, which renders some conclusions about these types difficult to make. Yet, our selected data analyses do not require normal data distribution, which increases our confidence in the results obtained. Still, an identified opportunity for lies in the recoding of the process models by a professional process modeler to ensure integrity in representing process information.

Our findings on the various types of design generated by students have provided insights on how individuals without experience in formal modeling method(s) conceptualize and externalize business processes. Specifically, the moderate use of graphics and abstract shapes to illustrate a process is more intuitive and would aid the understanding on the concept of process modeling. This would benefit the teaching aspect of business process modeling subjects, or any process-oriented disciplines, by introducing an informal approach before applying formal modeling methods. This is due to the nature of graphical illustrations being intuitive, such as that of concrete icons and abstract symbols used in the Flowchart, Hybrid and Storyboard designs. However, there is also a trade-off in the quality of process design when graphics are fully incorporated which suggests that while graphics can, to a certain extent, aid the understanding and communication of a business process, it could also result in a loss of information due to ambiguity and/or misinterpretation. On the other hand, process designs that fully utilize textual labels and descriptions, such as that in Textual design, may be useful in representing certain process information such as Business Rules, but are not entirely intuitive. We believe that our study provides some valuable insights on the cognitive aspects of novice process designers, which can be the basis for further cognitive studies in the field of business process design.

## References

1. Davies, I., Green, P., Rosemann, M., Indulska, M., Gallo, S.: How do Practitioners Use Conceptual Modeling in Practice? Data & Knowledge Engineering 58, 358–380 (2006)
2. Nickerson, J.V., Corter, J.E., Tversky, B., Zahner, D., Rho, Y.J.: The Spatial Nature of Thought. In: Boland, R.J., Limayem, M., Pentland, B.T. (eds.) Proceedings of the 29th International Conference on Information Systems. Association for Information Systems, Paris, France (2008)
3. BPMI.org, OMG: Business Process Modeling Notation Specification. Final Adopted Specification. Object Management Group (2006), `http://www.bpmn.org`
4. Grosskopf, A., Edelman, J., Weske, M.: Tangible Business Process Modeling - Methodology and Experiment Design. In: Mutschler, B., Recker, J., Wieringa, R. (eds.) Proceedings of the 1st International Workshop on Empirical Research in Business Process Management. LNBIP, vol. 1. Springer, Heidelberg (2009)
5. Rosemann, M.: Potential Pitfalls of Process Modeling: Part A. Business Process Management Journal 12, 249–254 (2006)
6. Edelman, J., Grosskopf, A., Weske, M.: Tangible Business Process Modeling: A New Approach. In: Proceedings of the 17th International Conference on Engineering Design. Stanford University, Stanford (2009)
7. Do, E.Y.-L., Gross, M.D., Neiman, B., Zimring, C.: Intentions in and Relations Among Design Drawings. Design Studies 21, 483–503 (2000)
8. Eisentraut, R., Günther, J.: Individual Styles of Problem Solving and their Relation to Representations in the Design Process. Design Studies 18, 369–383 (1997)
9. Prats, M., Lim, S., Jowers, I., Garner, S.W., Chase, S.: Transforming Shape in Design: Observations from Studies of Sketching. Design Studies 30, 503–520 (2009)
10. Simon, H.A.: The Sciences of the Artificial, 3rd edn. MIT Press, Cambridge (1996)
11. Mendling, J., Reijers, H.A., Recker, J.: Activity Labeling in Process Modeling: Empirical Insights and Recommendations. Information Systems 35, 467–482 (2010)
12. Recker, J., Rosemann, M., Indulska, M., Green, P.: Business Process Modeling: A Comparative Analysis. Journal of the Association for Information Systems 10, 333–363 (2009)

13. Krogstie, J., Sindre, G., Jørgensen, H.D.: Process Models Representing Knowledge for Action: a Revised Quality Framework. European Journal of Information Systems 15, 91–102 (2006)
14. Yang, Y., Tan, Q., Xiao, Y.: Verifying Web Services Composition Based on Hierarchical Colored Petri Nets. In: Hahn, A., Abels, S., Haak, L. (eds.) Proceedings of the 1st International Workshop on Interoperability of Heterogeneous Information Systems, pp. 47–54. ACM, Bremen (2005)
15. Curtis, B., Kellner, M.I., Over, J.: Process Modeling. Communications of the ACM 35, 75–90 (1992)
16. Boroditsky, L.: Metaphoric Structuring: Understanding Time through Spatial Metaphors. Cognition 75, 1–28 (2000)
17. Visser, W.: Design: One, but in Different Forms. Design Studies 30, 187–223 (2009)
18. Wang, W., Brooks, R.J.: Empirical Investigations of Conceptual Modeling and the Modeling Process. In: Henderson, S.G., Biller, B., Hsieh, M.-h. (eds.) Proceedings of the 39th Conference on Winter Simulation, pp. 762–770. IEEE, Washinton (2007)
19. Larkin, J.H., Simon, H.A.: Why a Diagram Is (Sometimes) Worth Ten Thousand Words. Cognitive Science 11, 65–100 (1987)
20. Koning, H., Dormann, C., van Vliet, H.: Practical Guidelines for the Readability of IT-architecture Diagrams. In: Haramundanis, K., Priestley, M. (eds.) Proceedings of the 20th Annual International Conference on Computer Documentation, pp. 90–99. ACM, Ontario (2002)
21. Purcell, A.T., Gero, J.S.: Drawings and the Design Process: A Review of Protocol Studies in Design and Other Disciplines and Related Research in Cognitive Psychology. Design Studies 19, 389–430 (1998)
22. Kavakli, M., Gero, J.S.: Sketching as Mental Imagery 22(4) (2001)
23. Ferreira, J., Noble, J., Biddle, R.: A Case for Iconic Icons. In: Piekarski, W. (ed.) Proceedings of the 7th Australasian User Interface Conference, pp. 64–100. CRPIT, Hobart (2006)
24. Rogers, Y.: Pictorial Representations of Abstract Concepts Relating to Human-Computer Interaction. ACM SIGCHI Bulletin 18, 43–44 (1986)
25. Mayer, R.E.: Multimedia Learning. Cambridge University Press, Cambridge (2001)
26. Khatri, V., Vessey, I., Ramesh, V., Clay, P., Sung-Jin, P.: Understanding Conceptual Schemas: Exploring the Role of Application and IS Domain Knowledge. Information Systems Research 17, 81–99 (2006)
27. Moody, D.L.: The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. IEEE Transactions on Software Engineering 35, 756–779 (2009)
28. Tabachnick, B.G., Fidell, L.S.: Using Multivariate Statistics, 4th edn. Allyn & Bacon, Boston (2001)
29. Paivio, A.: Mental Representations: A Dual Coding Approach. Oxford University Press, New York (1990)
30. Mayer, R.E., Moreno, R.: Nine Ways to Reduce Cognitive Load in Multimedia Learning. Educational Psychologist 38, 43–51 (2003)
31. Kavlaki, E., Loucopoulos, P.: Experiences With Goal-Oriented Modeling of Organizational Change. IEEE Transactions on Systems, Man and Cybernetics - Part C 36, 221–235 (2006)

# BPM in Practice: Who Is Doing What?

Hajo A. Reijers[1], Sander van Wijk[2], Bela Mutschler[3], and Maarten Leurs[2]

[1] Eindhoven University of Technology, The Netherlands
h.a.reijers@tue.nl
[2] Deloitte Consulting, The Netherlands
{svanwijk,mleurs}@deloitte.nl
[3] Ravensburg-Weingarten University of Applied Sciences, Germany
bela.mutschler@hs-weingarten.de

**Abstract.** This paper investigates the adoption of BPM, i.e., the use and deployment of BPM concepts in different kinds of organizations. A set of 33 completed, industrial BPM projects is analyzed based on project documentation and interviews with involved project members. In addition to the main study, which is conducted in the Netherlands, the paper also presents results of a replication study in Germany comprising six interview-based case studies and an international survey among 77 BPM experts. Thereby, various characteristics of BPM projects (such as a project's objective, strategic orientation or focus area) are analyzed to derive valuable insights both for practitioners performing BPM projects and for academics facing the challenge to support practitioners with innovative solutions in the field of BPM.

**Keywords:** Business process management, BPM projects, case study analysis.

## 1 Introduction

Considerable confusion exists about what Business Process Management (BPM) entails, as it blends paradigms and methodologies from organization management theory, computer science, mathematics, linguistics, semiotics, and philosophy. It is this versatility and interdisciplinary setting that also characterizes BPM as a true "theory in practice" subject [1]. In fact, BPM is pre-eminently a field where theoretical and technological developments are directly motivated by industrial application.

An industrial motivation calls for an understanding of industrial practice. For the progress of BPM – a management discipline that is closely linked to and enabled by various technologies – it is highly relevant to understand which *organizations* are embracing it in practice and which *activities* they undertake. Only by knowing more about the "consumers" of BPM methods, tools, and techniques, it becomes possible for its "producers" to properly position their research activities, engage in meaningful cooperation with industrial partners, and develop a meaningful research agenda.

In this light, it may come as a surprise that contemporary insights are missing into which categories of organizations are adopting BPM and which type of BPM projects they are carrying out. Most reflections on this subject are anecdotic or presented without any empirical evidence. For example, Gartner recently stated that "BPM

R. Hull, J. Mendling, and S. Tai (Eds.): BPM 2010, LNCS 6336, pp. 45–60, 2010.

adoption has been strongest in the service industries (such as banking, insurance, telephone companies and other utilities)" [2]. For those who recall that BPM initially was most popular in and emerged from manufacturing settings [3], such a statement craves for empirical back-up.

The objective of this paper is to provide a contemporary, empirically informed insight into the types of organizations that are adopting BPM and the type of projects they are carrying out. More specifically, we want to know whether any organizational watersheds in the practicing of BPM can be established. For this purpose, we investigated a range of organizational dimensions (*size*, *profit motive*, *sector*, and *strategic orientation*) as well as various characteristics of the BPM projects organizations are involved in (e.g., which phases of the BPM life-cycle are covered). The criteria we used to analyze both organizational dimensions and BPM project characteristics have been selected on the basis of a literature review and an exploration of industrial cases.

To gain access to a large set of actual BPM projects, this research has been carried out in close cooperation with the Dutch practice of one of the world's largest consulting firms. This cooperation provided us with the unique opportunity to examine in detail all the BPM projects of recent years that this firm has been involved in: all project documentation could be referred to, and the involved consultants were available for consultation. Since this specific cooperation potentially also introduces bias, specifically with respect to the geographic area of the projects and the client base, we have taken various measures to counter this threat, including a replication of our investigation through an international survey and case studies in Germany.

The structure of this paper is as follows. Section 2 provides the background for this paper, by providing a short reflection on the BPM concept. It also examines which indications of organizational differentiators are known to play a role in BPM adoption. In Section 3, the design of the research is addressed. Section 4 presents the results of our research. The paper concludes with a discussion of the results and our conclusions.

## 2 Background Information

### 2.1 Views on BPM

BPM is a difficult concept to pin down, as has been noted before, e.g., in [4-6]. On the one hand, it is strongly associated with process-oriented techniques and tools, in particular with process modeling and workflow management technology [7-10]. On the other hand, BPM can be also considered as a management concept [3;11;12], which draws from predecessors such as Total Quality Management and Business Process Reengineering. Since industrial projects are often labeled as BPM projects without specific technologies playing an important role, we will follow the latter, more holistic, interpretation. In line with [11], we see BPM as an integrated management philosophy and set of practices that includes incremental change and radical change in business process, and emphasizes continuous improvement, customer satisfaction, and employee involvement. An earlier study that has studied the characteristics of industrial BPM projects is [13], but its focus is mostly on process modeling.

Various authors have been using the notion of a life-cycle to distinguish between the various phases that a BPM initiative can go through, see e.g. [8;14;15]. All these life-cycles have in common that a business process is seen as the object that is continuously improved. One of the most elaborate life-cycles is presented in [16], which distinguishes between the following phases (see Fig. 1).

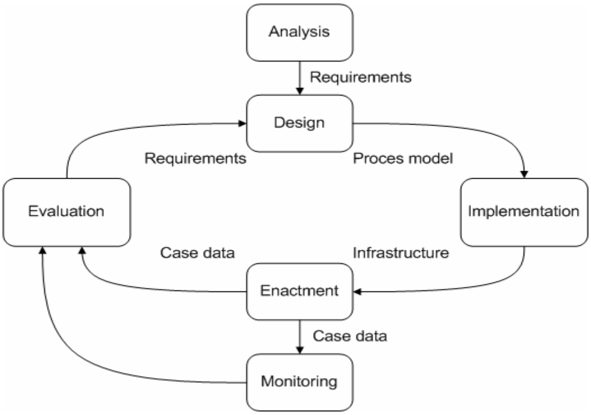


**Fig. 1.** The BPM life-cycle, cf. [16]

In the *analysis* phase of this life-cycle, a set of requirements is developed for the business process in question such as performance goals or intentions. During the *design*, the process activities, their order, the assignment of resources to activities and the organization structure are defined. The infrastructure for the business process is set up during the *implementation*, which includes training of staff, provision of a dedicated work infrastructure or the technical implementation and configuration of software. In the *enactment* phase, the dedicated infrastructure is used to handle individual cases covered by the business process. Depending on process metrics, counteractions are taken to deal with problematic situations in the *monitoring* phase. The *evaluation* phase leads to new requirements that are taken as input in the next turn of the business process management life-cycle.

In the remainder of this paper, these phases will be used as one of the aspects to characterize the type and sophistication of the analyzed BPM projects.

## 2.2 BPM Adoption

Despite the benefits that are often associated with BPM initiatives, little research has been conducted on the considerations of, or types of, organizations adopting BPM. The earliest report on the industrial application of BPM that is known to us [3], specifically ties BPM to manufacturing settings and the goal to better handle product quality. The authors also call for the wider adoption of BPM in non-manufacturing settings, implying that BPM is applicable there, but hardly popular at the time of the survey. The main survey in [4] is accompanied by a brief discussion of possible differences in adoption rate of BPM between small and large organizations on the one hand and between public- and private organizations on the other. However, no significant differences with respect to organizational BPM adoption on these dimensions were established within the European organizations that were examined.

In [17] a method is described to measure the extent to which an organization is process-oriented. The application of this instrument to a large set of American companies suggests that smaller companies tend to score better than larger ones, implying that those are more process-oriented. Service companies also seemed to score better than manufacturing companies. While this work did not focus on the adoption of BPM as a management concept or on actual BPM projects, it clearly hints at different BPM needs within organizations.

Considering the further literature, a potential link between BPM adoption and strategic management can also be distinguished [18]. Several authors [11;19] explicitly mention the type of strategic alignment as a crucial part of BPM and it is included in Rosemann & De Bruin's recent *BPM Maturity Model* [12]. Moreover, in [4;20] it is claimed that a "true process enterprise" should be connected to an overarching strategic initiative. In this context, it is important to note that the most common way to characterize an organization's strategic orientation is provided in [21], where a distinction is made between a strategy of operational excellence, customer intimacy or product leadership.

In summary, while insights into the kind of organizations adopting BPM are virtually absent, the literature hints at the following dimensions that are of potential importance: *size* (small vs. large), *profit motive* (public vs. private), *sector* (manufacturing vs. non-manufacturing), and *strategic orientation* (operational excellence, customer intimacy, product leadership). These dimensions will be used as the basis for our study, as will be outlined in the following section.

# 3 Research Design

## 3.1 Research Method

Given the empirical nature of the targeted research, as well as the lack of contemporary studies into the topic, the applied research method is based on a method as proposed in [22]. This approach is highly iterative and explorative without requirements on explicit hypotheses as a starting point. In such an approach the various phases characteristically overlap, but in this case the main flow of the study can be said to have involved the initial phases of *exploration* and *conceptualization*, which were followed by iterations of *data collection* and *data analysis*. In addition, a *replication study* was carried out to deal with the bias resulting from our approach. These phases are shown in Fig. 2 and further discussed in the following sections.

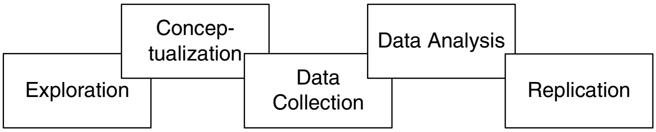


**Fig. 2.** Research Phases

### 3.2 Exploration and Conceptualization

The *exploration phase* incorporated a literature study to determine relevant organizational characteristics (see Section 2.2.), as well as a first round of interviews with BPM consultants. These consultants were targeted based on "snowball sampling" and by an analysis of their resumes, as available within the consulting firm. Additionally, project documentation was studied to get a first impression of BPM encounters.

To arrive at a proper conceptualization, i.e. decide which concepts to study and how they are related, it is important to determine at what level BPM activities are studied. We decided to focus on the project level, hence, BPM projects represent the unit of analysis. In contrast to what can be found in the professional literature [23], we do not see the start of a BPM *project* as a necessary indication of a low level of BPM maturity of the organization it takes place in. Rather, in our view BPM projects can be encountered in organizations that find themselves in any phase of the BPM life-cycle.

As a preparatory step for the subsequent phases of data gathering and analysis, we decided on the criteria for the selection of cases. First of all, we decided to only consider projects that are characterized by a focus on a *business process* as the object of study. After all, the notion of an end-to-end sequence of business tasks is fundamental to what BPM techniques and technologies entail. This criterion ruled out projects, for example, that are carried out within a single department or target stand-alone IT systems. Secondly, we excluded cases for which it was certain that their only aim was to document business activities. The motivation here is that a BPM project in some way aims to facilitate or assist organizational change in the foreseeable future. Thirdly, we focused on recent projects because of our goal to derive a contemporary state of the art, effectively only considering projects that were completed in 2005 or later. Finally, we only considered projects where a complete set of project documentation existed and for which consultants are still present within the consulting organization who were actually involved in the project.

While the literature review led to the identification of important organizational characteristics, the further identification of the relevant concepts to be studied at the BPM project level was done as part of the data collection and analysis phase, as will be explained below.

### 3.3 Data Collection and Analysis

The collection and analysis of data has been carried out in two iterations. The first iteration served to complete the conceptualization phase by (1) determining the characteristics that can be used to describe a *BPM project* on the one hand (in addition to the set of *organizational* characteristics determined before) (2) developing a *codification framework*, which covers both the characteristics of the *BPM project* and the *organization* in which the project was carried out. We determined the characteristics and developed the framework by selecting and analyzing a sub-set of the overall number of projects that were available to us. We selected these on the basis of their diversity, and iteratively improved the framework to maximize the applicability of the codes on a diverse set of cases. For the sake of illustration, the nine cases that were used for this purpose are shortly described in Table 1. Because the codification framework is highly relevant for the justification and interpretation of our results, we will discuss it as part of the results section (see Section 4.1).

**Table 1.** Case Descriptions Used for the Development of the Codification Framework

| Organization | Project |
|---|---|
| *Municipal authority* | To become compliant to new regulations, the processes concerning permits - for among others building and demolition - are redesigned and implemented including supporting IT. |
| *Market supervision institute* | Quality management including the identification, description and implementation of processes. |
| *Financial service provider #1* | An optimization and implementation of mortgage sales processes based on Lean Six Sigma methodology. |
| *Private equity firm* | The identification, modeling and improvement of most basic processes to facilitate a lagging ERP system implementation. |
| *Energy solutions provider* | A Lean Six Sigma project to improve processes in order to become more cost effective and best in class. |
| *Entertainment producer* | Process fit-gap-analysis to move from a local to a more central distribution model and achieve vertical integration of production activities. |
| *Financial service provider #2* | A Business Process Redesign to achieve compliance with new regulations. The project included a full implementation and processing of a due diligence of all existing clients. |
| *Soft drink producer* | Analysis, harmonization and modeling of over 300 processes to uniformize SAP implementation and facilitate best practice transfer. |
| *Insurance company* | Development and implementation of new processes, organizational structure and some supporting IT applications to enable the introduction of new product in the market through a new channel. |

The second iteration of the data collection and analysis phase aimed at a characterization of all those BPM projects that satisfied the selection criteria we established before. For example, for each project it was determined whether it was part of an overarching initiative (e.g., a merger) or carried out as an independent project. The coding allowed for a subsequent clustering of the projects in the analysis phase.

In order to reduce bias and to increase replicability, for each project one of the participating consultants was asked to code the project as well. This was done through a semi-structured interview which in all but two cases was conducted face-to-face or through telephone. In the two exceptional cases the questions were answered via e-mail. All interviewees were invited to answer the questions, shaped as a questionnaire[1], and comment on their answers to obtain more qualitative data. Those comments were documented. This approach mitigates the weakness of having just one investigator and allowed us to ensure that understanding of the concepts exists with the interviewee. The outcomes of the interviews were used during a further statistical analysis.

[1] The questionnaire can be downloaded from `http://tinyurl.com/2uqkm7p`

### 3.4 Replication

After having carried out the previously described phases as part of the main study, we additionally performed six interview-based case studies in southern Germany and an additional international (online) survey. Analyzing and comparing results of both studies, we had two goals: (1) fight geographical bias and (2) avoid potential bias resulting from working with a consultancy company. Selected results from both the survey and the six German case studies are also described in Section 4.

## 4 Results

### 4.1 Codification Framework

As mentioned in the design of our research, we coded BPM projects on both the *organizational characteristics* of the organization in which these are carried out, and *BPM project characteristics*. These two sets of characteristics were both split into different sub-characteristics, which were one-by-one tested for correlations with each other in order to find possible relationships between the characteristics of organizations and the BPM projects carried out within these. We aimed for at most four or five categories (or code values) per characteristic in order to obtain both enough contrast between the various categories and have sufficiently large numbers of cases in each category. We will discuss both categories of characteristics in more detail before presenting the results.

**Organizational characteristics.** Several distinctions between organizations can be made concerning *organization size*. Research on ERP adoption [24], for example, takes the number of employees as a measure. We followed this approach.

Based on the sample of cases, we defined four categories of organization size. The smallest category consists of organizations employing less than 250 employees while organizations in the category 'very large' employ over 10.000 employees. Annual reports and organizational databases were searched to obtain the required numbers. For projects conducted within multi-divisional organizations, the size of the entity in which the project was conducted is taken as the size of the organization.

We coded the *profit motive* of each organization as either being public or private. Due to the limited number of cases, we limited the number of options for an organization's *sector* at two, and determined whether the main value proposition of the organization is of a manufacturing, or non-manufacturing (e.g. service) nature.

Finally, the *strategic orientation* is based on [21] and thus brings three options: operational excellence, customer intimacy and product leadership. We explicitly considered strategic orientation as a *predominant* concept for an entire organization, following [25]. An assumption with this classification is that none of the organizations is "stuck in the middle", i.e. does not have a predominant strategic orientation. The classification on this characteristic was done on the basis of documentation on the organization as, for example, annual reports and organizational websites.

**BPM project characteristics.** We derived BPM project characteristics from literature. First of all, an initiative of any kind is generally triggered by some event or need.

In [20] it is mentioned that an *overarching strategic initiative* can trigger a move towards BPM or a BPM project. One can think of business events like a merger, supply chain integration, an ERP implementation or a move to e-commerce. All of these examples are also mentioned in [23]. Clearly, not each BPM project is associated with such an initiative. Therefore, we have chosen to classify a BPM project as either triggered by (or being part of) an *overarching (strategic) initiative* or as an *independent project*.

Secondly, we examined the objectives of the BPM project. Based on the first set of nine cases, we decided to split the objectives in two different characteristics: *business objectives* and *technical objectives*. While the business objective can either be an aim for *business performance* improvement, another option is to aim for *business conformance*. The latter is commonly observed when a BPM project is aimed at ensuring compliance with the rules and regulations as issued by relevant authorities.

The technical objective also consists of two categories. In many cases, implementation of technology (IT) is the main goal, for example an ERP implementation or the development of a mid-office solution. In those cases, the BPM project is classified as having a *technical objective*. If IT is not of any great relevance for the project, or is only used in a supportive way, (for instance a process modeling tool), the BPM project is classified as *not* pursuing a technical objective.

Apart from the reasons for initiation and its objectives, a BPM project is scoped in terms of the area in which the project is carried out. The *focus area* of a BPM project refers to the type of business processes that are in scope. We used the classification of [26], which comprises three types of processes: *primary processes* (which are value-creating processes), *secondary processes* (which support primary processes), and *managerial processes*. To arrive at a somewhat even distribution of cases over the classes, we decided to merge the support and managerial processes to *support processes*. This yielded the following options; pure creating processes (*core processes)*, pure *support processes*, or a mix of both.

Finally, the most extensive project characteristic is formed by the *type of BPM*. Here, we applied the BPM life cycle of [16] (cf. Section 2.1). For each phase of this life-cycle (*analysis, design, implementation, enactment, monitoring, evaluation)* we defined whether this phase was in- or outside of the scope of the BPM projects under study. Arguably, advanced phases in the BPM life-cycle point at a higher level of sophistication of the BPM project.

### 4.2 General Demographic Data

In total, we identified 67 BPM projects that complied with all criteria (cf. Section 3.2). Of this initial set, 33 cases turned out to be sufficiently documented to allow for a classification on all the organizational and BPM project characteristics we described. Those cases were used for the main data analysis. This set thus includes the nine cases described in Table 1 and used to develop the framework. Documentation of the omitted cases mostly lacked a clear description of the background of the project, hampering an assessment of the trigger of the project. The organizations in the sample vary greatly in size, ranging from about 25 to up over 40.000 employees. 13 of them are active in the financial service industry, eight cases represent public sector organizations. Also, energy & utilities (3), manufacturing (3), consumer business (2), technology media & telecommunications (2), aviation & transportation (1) and health care (1) organizations are represented, leading to 24 service (non-manufacturing) organizations and nine manufacturing organizations in the sample. With this division, the sample shows great resemblance to the total client base of the consulting firm involved. The strategic orientation of the organizations is rather evenly distributed over operational excellence (11), customer intimacy (14) and product leadership (8). Given this composition of the sample, there is no apparent bias that may influence the BPM project characteristics.

The characteristics of BPM projects show a great diversity and are rather evenly distributed over the various categories. In 18 out of the total of 33 cases, an overarching (strategic) initiative is identified to be the trigger. Business performance improvement is the main objective in a large majority of the projects (25); in the remaining eight projects, business conformance was the objective.

Moreover, almost half of the projects (16) had a technical objective. Note that when a technical objective was not dominant, IT might still be used as a solution in the project. In effect, in three cases the implementation of an IT system was a part of a project that was classified as having a non-technical objective. Here, IT served as an enabler to support the main objective of the BPM project: business performance improvement through better business processes.

A focus on core processes was found in as many as 15 BPM projects. Another 15 projects had a focus on both core and support processes. In only three cases, support processes are found to be the exclusive focus.

Almost all projects (31) go through the design phase of the BPM life-cycle, in most cases preceded by an analysis phase. The analysis phase is part of the project for 29 cases and in two cases it is the only phase that is performed. The implementation phase is part of 12 BPM projects. Only for eight BPM projects, additional phases are carried out after the implementation phase. On average, the BPM projects in our sample touch three phases of the overall life-cycle.

### 4.3 Inter-rater Agreement

To assess the reliability of this data, the codes assigned by the research team were compared to the codes by the involved consultants. This inter-rater reliability is measured through the Cohen's Kappa ($\kappa$) statistic [27]. Its value ranges from -1 (no agreement) and 0 (no agreement above chance) to 1 (representing perfect agreement). This metric can be used in case of more than two categories, but does lead to a lower $\kappa$ value [28]. Hence, the $\kappa$ values for the organizational size, predominant strategic orientation and focus area are potentially an underestimation.

Cohen's Kappa is calculated for every organizational and BPM project characteristic. For two characteristics, the agreement is 100% leading to a $\kappa$ of 1.000. Even the lowest value obtained ($\kappa$ = 0.472 for the strategic orientation) is still classified as moderate agreement [29].

A suitable method to assess the overall $\kappa$ of the entire dataset is proposed by Fleiss *et al.* [30], whose formula is based on the standard errors of the individual $\kappa$ values. As this method cannot cope with complete agreement, the two instances where $\kappa$ equals 1.000 are not included in the calculation of $\kappa_{\text{overall}}$. $\kappa_{\text{overall}}$ is determined at 0.701.

This demonstrates that the inter-rater agreement is substantial, which provides support for the reliability of the data.

### 4.4 Correlation Tests

With the validated data set as an input, we conducted a statistical analysis aimed to identify correlations between organizational and BPM project characteristics. Note that in cases where a disagreement existed between the coding of the research team and the involved consultants, the latter coding prevailed.

All characteristics are coded on a categorical, mainly nominal scale, except for organizational size which has an underlying rank order and is thus measured on an ordinal scale. The categorical nature of the data excludes the possibility of applying non-parametric tests like rank sum tests. In such cases, specifically Chi-square ($\chi^2$) contingency tests and Fisher's exact tests are useful to uncover correlations between variables measured on a nominal scale. Basically, those tests calculate the likelihood that a correlation exists between the rows and the columns of a contingency table.

The statistical package SPSS is used, which automatically conducts the $\chi^2$ test, both with and without the continuity adjustment for 2x2 contingency tables. The Fisher's exact test (or Fisher-Irwin test) is capable of handling small sample sizes, and can be considered as an alternative. We opted for methodological triangulation and conducted both tests. For all cases, the null hypothesis that the row-and-column classifications are independent is tested. Hence, in case the *P*-value falls below the desired value of α, the null hypothesis is rejected.

**Table 2.** Fisher's exact test results (P-values) * = significant at α = 0.05

| | **Organization size** | **Profit motive** | **Manufacturing / non- manufacturing** | **Strtegic orientation** |
|---|---|---|---|---|
| *Trigger* | 0.680 | 0.722 | 1.000 | **0.017** * |
| *Business objective* | 1.000 | 0.164 | 0.394 | 0.768 |
| *Technical objective* | 0.569 | 0.161 | 0.708 | **0.038** * |
| *Focus area* | 0.299 | 0.138 | 0.855 | 0.053 |
| *BPM Phase: Analysis* | **0.029*** | 0.289 | 1.000 | 0.328 |
| *BPM Phase: Design* | 0.330 | 1.000 | 0.477 | 0.324 |
| *BPM Phase: Implementation* | **0.050*** | 0.461 | 0.425 | 0.346 |
| *BPM Phase: Enactment* | 0.446 | 0.640 | 1.000 | 0.729 |
| *BPM Phase: Monitoring* | 0.062 | 0.646 | 0.642 | 0.417 |
| *BPM Phase: Evaluation* | **0.025*** | 0.397 | 1.000 | 0.653 |
| *BPM Phase: No. of phases* | 0.596 | 0.705 | 0.321 | 0.619 |

Both types of tests yielded largely similar *P*-values and led to the exact same decisions with respect to the null hypotheses. The single exception was the correlation between organizational size and the implementation life cycle phase, which yielded a *P*-value of 0.060 in the $\chi^2$test and a *P*-value of 0.050 for the Fisher exact test. We decided to reject the null hypothesis at the $\alpha = 0.05$ level, although the exact *P*-value remains questionable.

An additional Kruskal-Wallis test was performed for the tests involving organization size. This test is specifically aimed at contingency tables composed of an ordinal and a nominal variable. Again, the resulting *P*-values are only slightly different from the ones obtained from other tests. This agreement between several statistical methods strengthens the confidence in the outcomes. A summary of the outcomes of the Fisher's exact test is depicted in Table 2.

Based on the correlations found, we can present four main findings. First of all, we can establish on the basis of our sample that organization size correlates significantly with the type of BPM activities. Hence, the null hypothesis is rejected. After all, for three phases of the BPM life cycle, the correlation found is significant at $\alpha = 0.05$:

*Finding #1*: An organization's size relates with the phases of the BPM life cycle that a BPM project goes through within such an organization.

Interestingly, in the case of the profit motive and manufacturing or non-manufacturing, we could not nearly reject the null hypothesis. This null hypothesis states that the profit motive and the distinction between manufacturing and non-manufacturing do not have a correlation with any of the BPM project characteristics studied. Hence, the failure to reject the null hypothesis leads to the conclusion that a statistically significant correlation between those organizational characteristics and the BPM projects could not be found. This contrasts the existing literature (cf. section 2.2, among others [3], [4], [17]). Even though this literature hints upon differences, our results lead to the following findings #2 and #3:

*Finding #2*: The profit motive of an organization does not relate to the characteristics of the BPM projects carried out within such an organization.

*Finding #3*: Whether an organization is of a manufacturing type or not does not matter for the characteristics of the BPM projects carried out within such an organization.

The strategic orientation correlates in a highly significant manner with the trigger and technical objective of the BPM projects. This leads to our fourth and final finding:

*Finding #4*: An organization's strategic orientation shows a relation with the characteristics of the BPM projects carried out within such an organization.

In the remainder of this paper, we will shed light on our findings and provide more context, both by relating our findings to existing literature and by presenting a secondary study testing whether our results can be generalized.

### 4.5 Results from the Replication Study

Can the results from the Dutch main study be generalized? In order to answer this question, we performed an international (online) survey as well as six interview-based (mini) case studies in southern Germany. Thereby, we followed the research design specified by the Dutch main study (cf. Section 3). This section shortly summarizes the main results from both the survey and the case studies.

**Online Survey.** The survey involves BPM experts from manufacturing and non-manufacturing organizations from both the private and the public sector. 77 participants from more than 70 organizations across the world participated. Specifically, 58.44% of the responses stem from European participants, 20.78% from North American ones, 11.69% from South American ones, 2.6% from Asian ones and 1.6% from Australian and African ones. The questionnaire (which is exactly the one which was also used in the main Dutch study) has been distributed via a Web-based delivery platform and comprised 14 questions. Most questions are structured, i.e., provide a predefined set of possible answers. Some questions additionally allow for giving other than predefined answers. Known international mailing lists as well as personal contacts were used to promote the online survey.

Taking a look at the survey data, we see that only two of the four main findings are directly confirmed, namely finding #2, i.e., profit motive of the client organization does not relate with BPM project characteristics, and finding #3, i.e., manufacturing/non-manufacturing organizations do not differ with respect to the BPM project characteristics. By contrast, survey data do not show a statistically significant correlation between organization size and the support of BPM life-cycle phases (finding #1). Also, data do not show a statistically significant correlation between an organization's strategic orientation and the characteristics of its BPM projects (finding #4).

However, please note that, despite statistically significant correlations cannot be found for findings #1 and #4, collected data regarding findings #1 and #4 is not conflictive. Indeed, the data also points into the expected direction, only a statistically significant correlation cannot be determined. A bigger sample may well result in a confirmation of the hypotheses #1 and #4 after all.

**Case Studies.** The case studies are based on interviews at six organizations that comprise four manufacturing organizations and two non-manufacturing ones. As manufacturing organizations, we selected two large suppliers from the automotive industry, one mid-tier organization from the metal-working industry, and one mid-sized organization from the logistics domain. As non-manufacturing organizations, we included one public organization (a mid-size municipal administration department specifically) and one franchise company providing haircutting services.

Interviews were based on a semi-structured interview guideline. Each interview lasted between 90 and 180 minutes. At each organization, one interview session was performed, whereas these interviews were actually group interviews in most cases, i.e., typically several BPM key players joined the interviews in order to take the different perspectives on BPM activities into account.

Most importantly, our interview results confirm all findings from the Dutch main study. Generally, we can see that BPM projects are performed in a more systematic manner in larger and more mature organizations (echoing findings #1 and #4). An additional distinction between private sector and public sector organizations is not necessary as both kinds of organizations are generally faced with the challenge to work as efficiently as possible (echoing findings #2). The reason is that larger and more mature organizations deal with the challenge to optimize their overall way of working already for a long time. Thereby, especially manufacturing organizations already pick up many of the typical BPM activities (such as process modeling, process automation, process monitoring). Notwithstanding, our interviews also showed that non-manufacturing organizations adopt many basic themes underlying BPM initiatives as well, such as process thinking, adoption of a life-cycle-oriented way of working, process analysis – although the respective activities are sometimes not explicitly considered or recognized as “BPM activities” (which, however, seems more of a wording problem). This is in line with finding #3.

## 5 Summary, Conclusions and Future Work

The goal of this paper is to derive valuable insights both for practitioners performing BPM projects and for academics facing the challenge to support practitioners with innovative solutions in the field of BPM. In order to achieve this goal, we have investigated the adoption of BPM, i.e., the use and deployment of BPM techniques and tools in different kinds of organizations. Specifically, a large set of 33 BPM projects has been analyzed based on interviews, available project documentation, and project data. To improve the validity of the results, we have taken various measures, including a replication of our investigation through an international survey among 77 BPM experts and six case studies in Germany. The findings have been described in Section 4. But what are now the specific implications of the presented work for both research and practice?

Adopting the academic perspective, we have to recapitulate that our study is one of the first reports dealing with considerations of, or types of, organizations adopting BPM. As mentioned in Section 2, one of the rare studies is described in [3]. In this study, BPM was tied to manufacturing settings. At the same time, the study requested a wider adoption of BPM in non-manufacturing settings. Picking up this request, our study clearly shows that significant progress has been made in non-manufacturing settings so far (finding #2), whereas the profit-motive underlying a BPM initiative is thereby not relevant (finding #3).

Surprisingly, adopting typical BPM concepts mainly depends on the size of an organization (finding #1). In fact, finding #4 states that also an organization’s strategic orientation shows a relation with the characteristics of the BPM projects carried out within such an organization. However, as strategic considerations are often only discussed in larger enterprises (where objectives such as cost reductions or process optimizations play a more important role), this finding can be considered as a consequence of finding #1. This simple, but interesting conclusion (which also contrasts existing literature) needs to be picked up both by researchers and practitioners and has some significant impact on the way BPM solutions have to be discussed.

A consequence for researchers is that they also have to address “smaller” BPM use cases emerging in small and medium-sized enterprises. Such enterprises are not that interested in the top-of-the-notch process technology, but do certainly aim at the application of BPM concepts for which the gateway hurdle is significantly lower. Academic research has to pick up this demand and has to develop innovative, but less complex and therefore BPM solutions that are easier to apply. Respective solutions can also make use of (simple) process technology, but have to focus more on organizational issues at the beginning. Technology seems to be the second step only.

Adopting the practitioners' perspective, we see – as aforementioned – that BPM projects are performed in a much more systematic manner in larger and more mature organizations (cf. finding #1). The reason is that these kinds of organizations deal with the continuous need to optimize their work processes for a longer time. Consequently, small and medium-sized enterprises need to be supported in learning from larger companies. Thereby, it is not necessary to distinguish between manufacturing and non-manufacturing enterprises (finding #3), i.e., the occurring and emerging problems are more or less the same. Also, and this is confirmed by our study as well (cf. finding #2), the profit motive does not significantly influence specific kinds of BPM activities. What practitioners in small and medium-sized enterprises need is therefore more and better methodological support enabling them to profit from experiences in larger enterprises. This requires, however, that available BPM solutions are scaled down, so that small and medium-sized enterprises are able to adopt innovative, but less complex BPM solutions.

From an overall perspective, we can conclude that the size of an enterprise is the most important dimension determining the use and deployment of BPM concepts in practice. Neither the specific setting (manufacturing vs. non-manufacturing) nor the profit motive influences the characteristics of BPM projects. Summarizing, we strongly believe that more empirical insights into the application of BPM concepts is needed. Hence, future work will include additional empirical research activities to investigate the presented findings and observed phenomena in more detail. Respective activities will include additional online surveys, case studies, and the performance of controlled experiments (involving both students and practitioners). Special attention will be paid on extending the international scope of BPM projects and client bases.

**Acknowledgments.** Special thanks go to Edgar Rot and Christian Waibel for accomplishing the international survey and their support on performing the six German case studies. We further acknowledge the generous help of all consultants involved in the analysis of the case studies. This research is supported by the Technology Foundation STW, applied science division of NWO, and the technology programme of the Dutch Ministry of Economic Affairs.

## References

[1] Ko, R.K.L., Lee, S.S.G., Lee, E.W.: Business Process Management (BPM) Standards: a Survey. BPM Journal 15(5) (2009)

[2] Hill, J.B., Cantara, M., Kerremans, M., Plummer, D.C.: Magic Quadrant for Business Process Management Suites. Gartner Research, 164–485 (2009)

[3] Elzinga, D.J., Horak, T., Lee, C.Y., Bruner, C.: Business Process Management: Survey and Methodology. IEEE Transactions on Engineering Management 42(2) (1995)

[4] Pritchard, J.P., Armistead, C.: Business Process Management: Lessons From European Business. BPM Journal 5(1), 10–32 (1999)

[5] Vergidis, K., Turner, C.J., Tiwari, A.: Business Process Perspectives: Theoretical Developments Vs. Real-World Practice. International Journal of Production Economics 114(1), 91–104 (2008)

[6] Ko, R.K.L.: A Computer Scientist's Introductory Guide to Business Process Management (BPM). Crossroads 15(4), 4 (2009)

[7] Georgakopoulos, D., Hornick, M., Sheth, A.: An Overview of Workflow Management. Distributed and Parallel Databases 3(2), 119–153 (1995)

[8] 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)

[9] Leymann, F., Roller, D., Schmidt, M.T.: Web Services and Business Process Management. IBM Systems Journal 41(2), 198–211 (2002)

[10] Mutschler, B., Reichert, M., Bumiller, J.: Unleashing the Effectiveness of Process-Oriented Information Systems: Problem Analysis, Critical Success Factors and Implications. IEEE Transactions on Systems, Man, and Cybernetics (Part C) 38(3), 280–291 (2008)

[11] Hung, R.Y.Y.: Business Process Management As Competitive Advantage. TQM & Business Excellence 17(1), 21–40 (2006)

[12] Rosemann, M., de Bruin, T.: Towards a Business Process Management Maturity Model. In: Proceedings of the 13th European Conference on Information Systems, ECIS 2005 (2005)

[13] Iden, J., Opdahl, A.L., Eikebrokk, T.R., Olsen, D.H.: What Makes Process Modelling Effective: Modelling or Project Factors. In: Proceedings of the 1st International Working Conference on Business Process and Services Computing (BPSC 2007), pp. 78–92 (2007)

[14] Zur Muehlen, M.: Workflow-Based Process Controlling. Logos (2004)

[15] Reijers, H.A.: Design and Control of Workflow Processes: Business Process Management for the Service Industry. Springer, Heidelberg (2003)

[16] Mendling, J.: Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness. Springer, Heidelberg (2008)

[17] McCormack, K.: Business Process Orientation: Do You Have It? Quality Progress 34(1), 51–60 (2001)

[18] Kiraka, R.N., Manning, K.: Managing Organisations Through a Process-Based Perspective. Business Change and Re-engineering 12(4), 288–298

[19] Lee, R.G., Dale, B.G.: Business Process Management: a Review and Evaluation. BPM Journal 4(3), 214–225 (1998)

[20] Hammer, M., Stanton, S.: How Process Enterprises Really Work. Harvard Business Review 77, 108–120 (1999)

[21] Treacy, M., Wiersema, F.: The Discipline of Market Leaders. Addison-Wesley, Reading (1995)

[22] Eisenhardt, K.M.: Building Theories From Case Study Research. The Academy of Management Review 14(4), 532–550 (1989)

[23] Jeston, J., Nelis, J.: Management by Process: A Practical Road-Map to Sustainable Business Process Management. Butterworth-Heinemann, Butterworths (2008)

[24] Laukkanen, S., Sarpola, S., Hallikainen, P.: Enterprise Size Matters: Objectives and Constraints of ERP Adoption. Journal of Enterprise Information Management 20(3), 319–334 (2007)

[25] Kaplan, R.S., Norton, D.P.: Having Trouble With Your Strategy? Then Map It. Harvard Business Review 78(5), 167–176 (2000)

[26] van der Aalst, W.M.P., Hee, K.M.: Workflow Management: Models, Methods, and Systems. MIT Press, Cambridge (2002)

[27] Cohen, J.: A Coefficient of Agreement for Nominal Scales. Educational and Psychological Measurement 20(1), 37–46 (1960)

[28] Sim, J., Wright, C.C.: The Kappa Statistic in Reliability Studies. Physical Therapy 85(3), 257 (2005)

[29] Landis, J.R., Koch, G.G.: The Measurement of Observer Agreement for Categorical Data. Biometrics 33(1), 159–174 (1977)

[30] Fleiss, J.L., Levin, B., Paik, M.C.: Statistical Methods for Rates and Proportions (1981)

# How to Implement a Theory of Correctness in the Area of Business Processes and Services

Niels Lohmann and Karsten Wolf

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

**Abstract.** During the previous years, we presented several results concerned with various issues related to the correctness of models for business processes and services (i. e., interorganizational business processes). For most of the results, we presented tools and experimental evidence for the computational capabilities of our approaches. Over the time, the implementations grew to a consistent and interoperable family of tools, which we call SERVICE-TECHNOLOGY.ORG.

This paper aims at presenting this tool family SERVICE-TECHNOLOGY.ORG as a whole. We briefly sketch the underlying formalisms and covered problem settings and describe the functionality of the participating tools. Furthermore, we discuss several lessons that we learned from the development and use of this tool family. We believe that the lessons are interesting for other academic tool development.

## 1 Introduction

It is a phenomenon which is common to several technologies based on formal methods that they suffer from a devastating worst-case complexity, but perform surprisingly well on real-world instances. Most prominent examples are *model checking* [10] and *SAT checking* [48]. Consequently, any approach based on such technologies needs to be complemented with a prototypical implementation, which provides evidence for the difference between theoretical complexity and actual observed run time. At the same time, such implementations are an important milestone in the technology transfer between academia and industry as they can be tried out and evaluated in realistic case studies such as [42, 19].

In this paper, we aim at introducing and discussing a family of tools, SERVICE-TECHNOLOGY.ORG, which has been developed over the previous couple of years and is, in different aspects, related to correctness of business processes and such services which internally run nontrivial processes as well. Whereas various members of the family have already been used to provide experimental data in several articles, for instance [26, 36, 34, 31, 60, 63, 19, 30, 39], this is the first occasion to introduce and discuss the common principles and lessons learned from SERVICE-TECHNOLOGY.ORG as a whole.

Central artifacts in the tool family are formal models for services and business processes (based on Petri nets and automata-based formalisms) and formal models for *sets of* services (based on annotated automata). The member tools of SERVICE-TECHNOLOGY.ORG[1]

[1] Available for download at http://service-technology.org/tools.

R. Hull, J. Mendling, and S. Tai (Eds.): BPM 2010, LNCS 6336, pp. 61–77, 2010.

are concerned with importing and exporting models from and to more established languages, with analyzing artifacts, and with synthesizing artifacts. The distinguishing feature of the tool family is that each basic functionality is provided by its own command-line based tool. Additional functionality can be derived by combining tools by connecting their input and output data streams. This way, our tool family is similar to the family of basic UNIX tools such as grep, awk, sed, and the like.

Many tools in the tool family implement algorithms that involve exponential worst-case complexity and complex data structures. To this end, *an efficient implementation is the key factor to apply these algorithms to realistic case studies.*

In Sect. 2, we discuss related tools and approaches which as well aim at demonstrating practical applicability of formal methods. Then, in Sect. 3, we give a walk-through introducing the current members of the tool family and the recurring concepts. In the remaining sections, we discuss several observations that we made concerning the integration of the tool family into its environment (Sect. 4), the formal models used (Sect. 5), and the chosen architecture (Sect. 6). Section 7 summarizes the lessons we learned during the development which may as well be applicable to other domains.

## 2 Related Work and Tools for Process Correctness

Errors in business process models obstruct correct simulation, code generation, and execution of these models. Consequently, business process verification techniques received much attention from industry and academia which is reflected by a constant and large number of tools and research papers on this topic. Interestingly, general-purpose model checking tools are hardly used. This could be explained, on the one hand, by the lacking tool support and efficiency in the early days of BPM research. On the other hand, the properties a business process needs to satisfy are typically much simpler than what temporal logics such as CTL or LTL offer. For over one decade, *soundness* [1] is now the established correctness notion. Soundness ensures proper termination (i. e., the absence of deadlocks and livelocks in the control flow) while excluding dead code (i. e., tasks that can never be executed).

Soundness can be naturally expressed in terms of simple Petri net properties, and tools such as *Woflan* [58] exploit efficient Petri net algorithms to verify soundness. Because of a close relationship [37] of many business process modeling languages to Petri nets, soundness analysis techniques are effortlessly applicable to other formalisms. For instance, Woflan is embedded as plugin into the *ProM framework* [2], and with *WofYAWL* [57], there also exists an extension to verify YAWL [3] models.

A common assumption for business process is that they can be modeled by *workflow nets* [1]; that is, models with a single distinguished initial and final state, respectively. This structural restriction is heavily exploited by state-of-the-art tools such as Woflan and may help to avoid expensive state space exploration whenever possible. The *SESE decomposition approach* [56] which is used by the *IBM Websphere Business Modeler* to decompositionally check soundness further assumes the free-choice property [15]. Similarly, the *ADEPT framework* [12] employs a block-structured language and offers efficient algorithms to ensure soundness in adaptive systems [49].

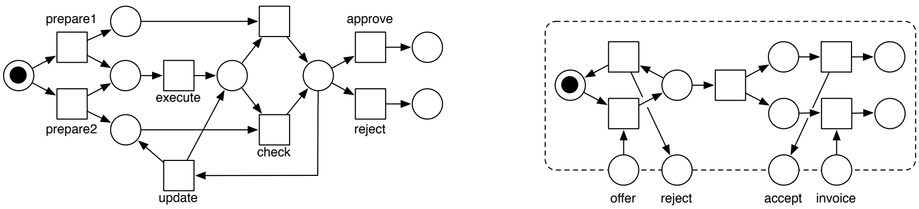


**Fig. 1.** Petri net models for processes (left) and services (right)

Structural techniques such as Woflan's heuristics or the SESE decomposition are state of the art as a recent case study [19] reports. However, not all features of current languages, such as BPMN or BPEL, can be expressed in terms of the mentioned structural restrictions. At the same time, the case study [19] reveals that state space verification became likewise efficient. Consequently, only state space verification tools offer the necessary flexibility to verify business processes with complex structures.

## 3 Overview of the Tool Family

The tools available in SERVICE-TECHNOLOGY.ORG are all concerned with control flow models of business processes or services. These models are typically given as Petri nets. The only distinction between processes and services is the absence (or presence) of distinguished interface places and transitions; see Fig. 1 for simple examples. We use a simple file format for Petri nets and have compilers available which manage the import and export into standard formats such as PNML [8].

Most tools in our tool family read or write such Petri net models; Figure 2 provides an overview of the whole tool family. To avoid repetition of programming efforts, we created a Petri net API which provides an internal representation of all basic Petri net elements and provides standard access methods (e. g., iterating on the set of nodes).

One way of creating a Petri net model to use a plain text editor or to import it from modeling tools that offer PNML export such as *Oryx* [14] or *Yasper* [25]. For obtaining realistic Petri net models, we use two different approaches. First, we have several compilers which translate specifications of practically relevant languages into Petri net models:

- *BPEL2oWFN* [32] for translating WS-BPEL [4] specifications into Petri nets;
- *UML2oWFN* [18] for translating business process models from the IBM Websphere Business Modeler into Petri nets.

From UML2oWFN, we obtain a business process model. For BPEL2oWFN, the user may chose whether to create a business process model from a single WS-BPEL process specification (its internal control flow) or the control flow of a service collaboration specified in BPEL4Chor [13, 34], or whether to obtain a service model from a WS-BPEL process specification.

More models can be obtained using distinguished SERVICE-TECHNOLOGY.ORG synthesis tools (see below). These tools typically produce automata-like models for the control flow of a business process or a service. Such a model can, however, be translated into

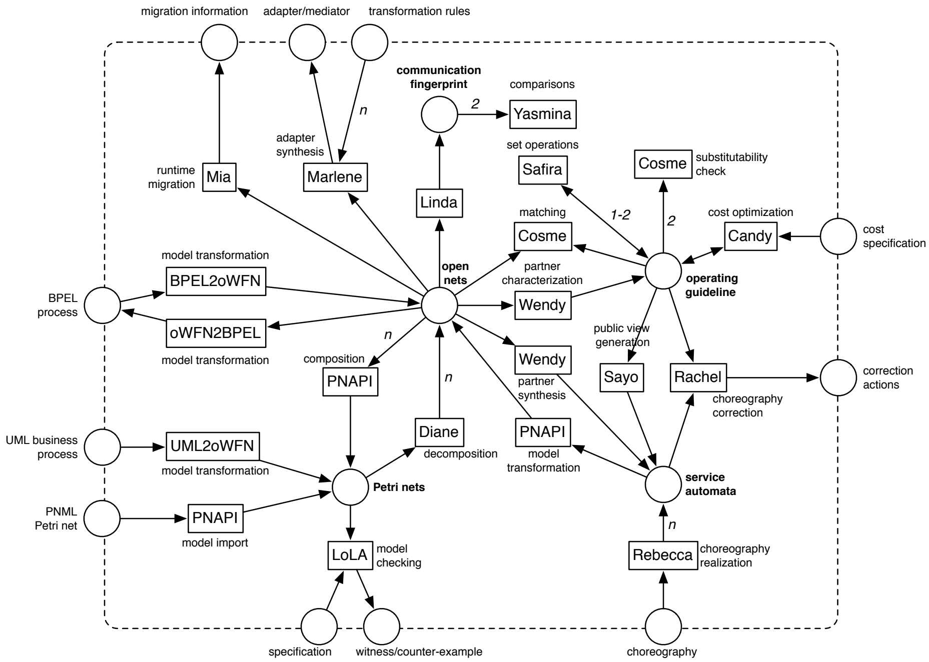


**Fig. 2.** Illustration of the interdependencies of the SERVICE-TECHNOLOGY.ORG tools

a Petri net model by either using a brute-force translation where each state of the automaton is translated into a separate place of the Petri net, or by using region theory [5] which leads to much more compact Petri net models. Both methods are available in our Petri net API. For the latter method, the API calls the external tools *Petrify* [11] or *Genet* [9] which, consequently, interoperate seamlessly with our tool family.

Our tool family can as well be plugged into existing tools and frameworks. We would like to mention ongoing integration efforts into the ProM framework [2], Oryx [14], and the *YAWL editor* [3]. In these cases, we chose to build plugins to these tools which translate the native modeling language into a representation of our Petri net API. It is then close to trivial to incorporate any subset of the tools mentioned next.

A Petri net model of a business process can be verified. Within the tool family, the Petri net-based model checker *LoLA* [61] is available which has proven to be powerful enough for the investigation of business process models [26, 19] or service collaborations [34]. LoLA verifies properties by explicitly investigating the state space of a Petri net using several of state-of-the-art state space reduction techniques.

For a model of a single service, we can not only verify its internal control flow. With the concept of *controllability* ("does the service have at least one correctly interacting partner?") [62], verification covers the interaction of the service with its environment. We decide the controllability problem by trying to construct a canonical ("most permissive") partner. This is one of the tasks of our tool *Wendy* [39]. For constructing the partner service, Wendy explores the state space of the given service. The distinguishing feature of Wendy with respect to a prior implementation is that it employs the state

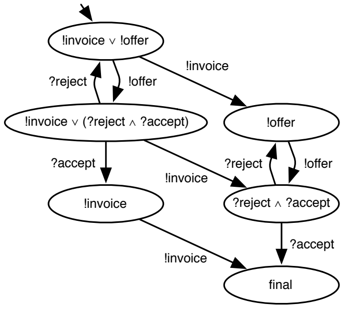


**Fig. 3.** Operating guideline [36]

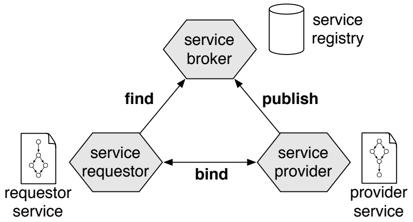


**Fig. 4.** The SOA triangle [23]

space exploration power of LoLA rather than traversing the state space using its own implementation. Surprisingly, the gain of using the sophisticated data structures and algorithms in LoLA outweighs the loss caused by transferring Petri net models and ASCII representations of computed state spaces between LoLA and Wendy.

A partner as constructed by Wendy is useful for a several applications. It can be used as a communication skeleton for actually programming such a partner. To this end, our compiler *oWFN2BPEL* [33] is quite useful which maps a Petri net to an abstract WS-BPEL process thus closing the loop between our formal realm and reality.

A related application for the partner created by Wendy is the synthesis of an *adapter* [16] service that mediates between otherwise incorrectly interacting services. This is the task of the tool *Marlene*. Marlene reads service models and a specification of permitted adapter activities (which messages can be created, transformed, deleted by the adapter?) and produces an adapter. As the adapter can be roughly seen as a correctly interacting partner to the given services, it is no surprise that large parts of the synthesis task are left to a call of Wendy.

Instead of inserting an additional component (the adapter) into the service collaboration, it is also possible to replace one of the participating services. This task is supported by the tool *Rachel* [31]. Given a service collaboration which does not interact correctly, Rachel can replace one of its participants (selected by the user) by another service which interacts as similar as possible but establishes correct interaction.

The second central artifact in the SERVICE-TECHNOLOGY.ORG tool family is a finite representation of possibly infinite *sets* of service models. Syntactically, we use *annotated automata* [36]. An annotated automaton $A$ represents the set of all those service models which can establish a simulation relation [44] with the automaton $A$ such that the constraints represented in the annotations of $A$ get satisfied. The most prominent example for a useful set of services is the set of all correctly interacting partners of a given service that we called *operating guidelines* in several publications [35,36,52]. Figure 3 shows an example of such an operating guideline.

Consequently, the transformation of a service model into one or all correctly interacting partner services is a core element of our theory. This is a nontrivial task as obvious ideas like just copying the control flow and reversing the direction of message transfer do not work in general. Our solution [36] annotates the most permissive partner. The corresponding implementation is thus done within Wendy.

In several settings, it is also desirable to replace a service with a new one such that the new one interacts correctly in any collaboration where the old one was participating (be it for technical update or changes in the context like new legal requirements). In this case, the tool *Cosme* can verify the underlying *substitutability* property [52]. Substitutability means that every correct partner of the old service interacts correctly with the new one. The key to deciding this inclusion on (typically infinite) sets of services is their finite representation as operating guidelines.

For collaborations which are already running, *Mia* [30] suggests transitions from states of the old service to states of the new service such that the old service can be migrated to the new service and the latter can take over the collaboration at run time.

One of the core concepts in the way services are expected to collaborate is the *service-oriented architecture* (SOA) [23] with its eye-catching SOA triangle (see Fig. 4). Some of the SERVICE-TECHNOLOGY.ORG members aim at supporting the procedures visualized in the triangle. The tool *Linda* turns a service model $S$ into what we call a *communication fingerprint* [54]. This is an abstract pattern covering the correct interactions that $S$ is able to produce. In the current version Linda provides lower and upper bounds for the occurrence of messages in correctly terminating runs. The idea of fingerprints is that it is much easier to select a correctly interacting partner from a repository if most entries in the repository can be ruled out by just finding incompatibilities in the communication fingerprint than by performing model checking on the complete service models. The task of comparing communication fingerprints is implemented in the tool *Yasmina*.

Another opportunity for organizing a service repository is to store operating guidelines of services instead of the services themselves. In this case, a SOA requester needs to be compared (matched) with the operating guidelines of registered services. Whereas the generation of operating guidelines is done by Wendy, it is Cosme to perform the actual matching. With the tool *Candy*, nonfunctional properties such as costs can be added to service models to refine the analysis.

In [28], we argued that it would be beneficial to aggregate operating guidelines for speeding up the selection of registered services. At the same time, more abstract *find* requests can be translated into annotated automata; that is, the formalism used for representing operating guidelines. Basic operations to be performed in a registry then boil down to basic set operations (union, intersection, complement, membership test, emptiness test). Our tool *Safira* [27] implements many of these operations by manipulating annotated automata.

If, for some reason, operating guidelines are not suitable for some purpose, our tool *Sayo* transforms the operating guideline of a given service $S$ into some service model $S'$ — a *public view* of $S$ — which is behaviorally equivalent to $S$. We believe that this procedure could be used as an obfuscator for $S$; that is, for publishing all relevant information about $S$ without disclosing its control flow. Unfortunately, we still lack some information-theoretic foundation of this obfuscation property.

Beyond SOA, *choreographies* are a promising technology for creating service collaborations. For choreographies, *realizability* [21] is a fundamental correctness property comparable to soundness of workflows. We observed that the realizability problem is conceptually very close to a certain variant of the controllability problem for services [40]. As a result, the tool *Rebecca* which implements the algorithm to check

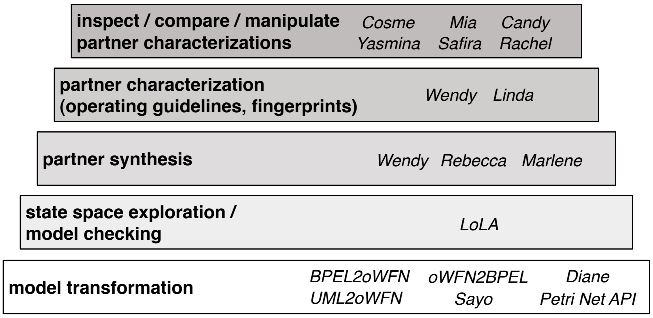


**Fig. 5.** Our technology stack

realizability for a choreography and to synthesize realizing services can also be used by Wendy to investigate that variant of controllability.

Finally, we would like to mention our tool *Diane* [43] which provides another connection between business processes and services. Diane takes a Petri net model of a business process $N$ (in fact an arbitrary Petri net) and produces a set of service models $S_1, \ldots, S_n$ such that their composition is equivalent to $N$. This procedure can be used for divide-and-conquer approaches for the verification of Petri nets in general [46]. We are also working on an extension of this idea leading to suggestions for spinning off services from existing business process models.

To summarize, we have a large set of single purpose tools that exchange information by means of files or data streams. Single-purposeness is the main distinction between SERVICE-TECHNOLOGY.ORG and previous implementations in our group. The tools can be grouped into compilers (BPEL2oWFN, oWFN2BPEL, UML2oWFN), the two core technology providers LoLA and Wendy, a large set of application-oriented tools (Marlene, Rachel, Safira, Rebecca, Linda, Yasmina, Cosme, Candy), and a few model manipulation tools (Sayo, Diane, Petri Net API). Virtually, all tools use the Petri net API for the internal representation of Petri nets. Some external tools are very useful in our family. Apart from the already mentioned tools Petrify and Genet, we rely on powerful libraries for solving linear constraint problems (*lpsolve* [20]), for manipulating Boolean functions using Binary Decision Diagrams (*CUDD* [51]), and for solving the satisfiability problem for propositional Boolean formulae (*MiniSat* [17]).

The application-oriented tools import critical calculations (such as state space exploration or partner synthesis) from the core technology providers LoLA and Wendy. Hence, optimizations done to the data structures and algorithms in these tools directly transfer to the performance of the application-oriented tools. This led us to view our tool family in terms of a technology stack (see Fig. 5).

We already discussed the model transformations which realize the import and export of models, as well as conversions between different file formats. On top of this layer, the bottom technology is state space exploration as provided in LoLA. The next two layers use state space exploration to produce a partner service to a given service which in turn is used as a central element in operating guidelines. Most other applications inherit their computational power at least partly from LoLA or Wendy.

In the remainder of this article, we discuss several lessons we learned from the availability of the tools listed so far and from the way we implemented them. We discuss the lessons in the light of academic tool development which differs in several respects from industrial tool development.

## 4 Link to Reality

This section is devoted to lessons related to the link between our family of tools and reality. The following lessons are not necessarily surprising. We believe, however, that our experience with the SERVICE-TECHNOLOGY.ORG tool family adds some evidence to these lessons.

One of the most apparent purposes for implementing theoretical approaches is to prove applicability in nontrivial context. This is necessary as theoretical results on worst-case complexity typically do not clearly distinguish between working and non-working approaches. In [19], for example, we demonstrated that — different to common belief — state space methods can very well be competitive for the formal verification of business processes. Only because of the convincing results in this study we have been to raise the issue of using verification even as part of actual modeling process ("verify each time you save or load models").

**Lesson 1.** *An actual prototype implementation propels transfer of technology from theory to practice.*

Another advantage of the prototype implementation is that experimental data presented in papers become more serious. Over the previous few years, we chose to make tools and experimental input data available over the Internet. The page[2] is an example related to the paper [31]. Because of the simple architecture of our tool family, it is possible to provide just the relevant part of our tool family, and to freeze the state of implementation at the point in time where the experiments have actually been carried out. By making experimental results repeatable by independent instances and by disclosing the source code of the tools, we address recent discussions on fraud and bad scientific practice.

**Lesson 2.** *Prototype implementations help in making experimental evidence transparent.*

Our implementations also provided valuable feedback to the improvement of tools. In the case study [19], we compared various combinations of soundness checking techniques. Among them were preprocessing approaches based on Petri net structural reduction rules [7,45]. Such a rule locally replaces a Petri net pattern with a simpler one yet preserving given properties (such as soundness). By the experiments on a large sample set, we found that the effect of applying these reductions is marginal if the subsequent state space verification is using the *partial order reduction*, one of the most effective state space reduction methods. Consequently, we were able to shift efforts from further implementation of structural reduction to other, more beneficial tasks.

[2] http://service-technology.org/publications/lohmann_2008_bpm

**Lesson 3.** *Large case studies help to detect bottlenecks early.*

We observed anecdotal evidence for this during the evaluation of the results of a case study. With our earlier tool Fiona, we calculated the operating guidelines of several industrial service orchestrators we translated from WS-BPEL processes. Both runtime and memory consumption were larger than we expected, compared to the relatively small sizes of the resulting operating guidelines. Fiona faithfully implemented the construction algorithm from [36] and first calculated an overapproximation of the final annotated automaton from which it iteratively removed states that could not belong to the final result. For the concrete examples, however, it turned out that over 90 % of the generated states were eventually removed. We did not observe such a large overhead during the analysis of academic examples. Consequently, any previous optimization effort was put into an *efficient generation* of this overapproximation. With the experience of this case study, we reimplemented the algorithm in the new tool Wendy and implemented a static analysis of the model to *avoid the generation* of spurious states in the first place. Even though this preprocessing employs assumed expensive state space verification, it allowed for dramatic speedups. Wendy is typically 10 to 100 times faster than Fiona, because we could reduce the number of spurious states to less than 1 %.

## 5 Choice of Formalism

In this section, we report on observations that we made concerning the choice of formalism. During the development of some of the tools, we discussed several variations, most notably the use of a workflow net structure [1]. In the end, we found that we would benefit most from the most liberal formalism that is still analyzable: bounded (finite state) Petri nets and finite automata.

Again, the study [19] yields an excellent example. For comparing tools, we translated various business processes not only into the LoLA format, but also into Woflan format which uses the aforementioned workflow nets both in modeling and verification. Sticking to this workflow structure has a couple of advantages in verification, for instance through a simple connection between soundness on one hand and well investigated Petri net properties [1] on the other hand. By the workflow net structure, each model has a single distinguished end state. Unfortunately this was not the case for our models stemming from the IBM Websphere Business Modeler. These models inherently have multiple end states. Using the algorithm of Kiepuszewski et al. [29], we were able to extend the models such that all end states merged into one. This construction, however, is only applicable to nets with *free choice* structure [15]. For free choice nets, many efficient verification techniques are available which do not work at all, or at least not efficiently, for arbitrary Petri nets. So whereas Woflan forced us into a workflow net structure for reasons of efficiency in one regard, it forced us into less structured models, with more serious runtime penalties in other regards.

**Lesson 4.** *The input formalism of a verification tool should not contain any structural restrictions beyond those required by the implemented technique.*

With a liberal formalism, the resolution of the tradeoff between different structural features can be left to the generation of the model. Whether a model satisfies a certain structural constraint (such as workflow net structure or free choice nature) can typically be easily decided.

Admittedly, the situation in our tool family is easier than in Woflan. Our models are typically generated by the compilers of our tool family, whereas Woflan models can be directly drawn using a graphical editor. Not forcing any structural restrictions in Woflan would mean that users tend to produce spaghetti-like models which are then rather error-prone and hard to verify. Hence, the previous lesson only works in combination with another one.

**Lesson 5.** *It is beneficial to separate the formalism used for modeling from the one used for verification.*

Then, the generation of a "verification-friendly" model is enforced in the translation between the formalisms. Decoupling the modeling from verification has another advantage. Whereas modeling tools inherently need to address domain-specific notations, structuring mechanisms, and procedures, this is not necessarily the case for verification technology. Actually, when we tuned our tool LoLA for the verification of business processes, we ended up with more or less the same combination of reduction techniques which were also found optimal in the verification of asynchronous hardware circuits [53] or even biochemical reaction chains [55]. In turn, having implemented a verification tool for some purpose, a domain-unspecific formalism helps to conquer new application areas and to transfer experience from one domain to another. A further good example in this regard is the tool Petrify. Although its designers focused on applications in hardware synthesis, they kept their basic formalism sufficiently general and hence allowed us to use their tool in a services context. According to our experience, the ability to reach out for new application domains is much more significant than potential performance gains by domain-specific heuristics. Last but not least, domain-unspecific formalisms tend to be much simpler with respect to basic concepts than domain-specific ones (compare Petri nets or automata with notations such as EPC or BPMN) thus simplifying data structures and algorithms in the tools.

**Lesson 6.** *Keep verification technology domain-unspecific.*

As our internal representation of business processes and services did not reflect any restrictions from the modeling domain, the only tool affected by the shift from the Petri net semantics for BPEL4WS 1.1 [26] to the one for WS-BPEL 2.0 [32] was the corresponding compiler. In both semantics, we were able to represent complex mechanisms such as fault handlers, compensation handlers, or termination handlers. Likewise, the link to new languages (such as BPMN 2.0) will only involve building a new compiler whereas we will not need to touch any technology provider or application-oriented tool in our family.

As a side-effect of domain-unspecific technology, it is easy to integrate our tools into other frameworks. LoLA is already integrated into several Petri net frameworks including *CPN-AMI* [24], the *Petri Net Kernel* [59], the *Model Checking Kit* [50], and the *Pathway Logic Assistant* [55]. Integrations into other frameworks are progressing, such as into ProM, Oryx, or YAWL. This way, we benefit from functionality of the mentioned tools. In particular, there is no reason for us to invest resources into graphical user interfaces or simulation engines. Instead, we can focus all available resources on our core competencies which are verification techniques.

Providing domain-unspecific verification technology does not necessarily mean that, through a sufficiently general formalism used for verification, the development of compilers into the formalism can be done completely independent of the subsequent technology providers. Compare, for example, the different Petri net semantics given to WS-BPEL [32, 47]. The proposal of [47] employs *cancellation regions*, a feature which is supported by YAWL. These cancellation regions can be transformed to ordinary Petri nets. This translation, however, introduces several global status places which introduce dependencies between Petri net transitions. These dependencies, in turn, spoil the partial order reduction, LoLA's most powerful state space reduction technique. Consequently, the translation [32] carefully avoids the introduction of global status places. A detailed discussion of the different modeling approaches is provided in [38].

**Lesson 7.** *Despite a domain-unspecific formalism, compilers need to fit to the subsequent technology providing tools.*

## 6 Architecture

Several lessons can be learned from the architecture of SERVICE-TECHNOLOGY.ORG. Two years ago, most of our results were implemented within a single tool, *Fiona* [41]. As more and more results emerged, the tool constantly grew and eventually we ran into severe maintenance problems.

With Wendy, we completely rethought our tool development process. We came to the conclusion that it would be beneficial to switch from a single multipurpose tool to the family of single-purpose tools which we sketched in Sect. 3. The advantages of the single-purpose paradigm became such evident that we managed to copy most functionality of Fiona in the new tool family in less than a year.

The main lesson of this section is:

**Lesson 8.** *Separate functionality into a multitude of single-purpose tools rather than integrating it into a single tool.*

The decisive advantage of the single-purpose approach is that data structures are significantly simpler than in a multipurpose tool. In our case, many algorithms roughly shared the same core data structures (in particular, annotated automata). However, each of the algorithms required some subtle changes in the general scheme, or required certain constraints. In effect, we dived into an increasingly deep class hierarchy. As this hierarchy became less and less transparent, programmers did no longer oversee which invariants they could rely on for certain data structures and so a number of redundant computations occurred. It become increasingly difficult to familiarize students with the existing code and to add functionality. Cross-dependencies caused severe problems for maintenance and testing.

Wendy was an attempt to flee this self-magnifying loop. It reimplements the core tasks of Fiona: computation of a correctly interacting partner and computation of an annotated automaton which represents all correctly interacting partners. We achieved the following results:

- Wendy's core functionality could be implemented within 1,500 lines of code instead of more than 30,000 lines of code in Fiona.
- Wendy was coded within two weeks compared to several years of developing Fiona.
- Wendy solves problems 10 to 100 times faster than Fiona although it is used on the same basic algorithmic idea. At the same time, the memory consumption could be decreased by a similar factor.
- Wendy has shallow data structures and its code is easy to understand. For instance, the whole functionality is implemented in six classes and there was no need for any class hierarchies (viz. inheritance). In contrast, Fiona consists of over 50 classes with an up to 5-level class inheritance.

The data structures could be kept shallow as they no longer needed to fit the needs of many tool components at once.

**Lesson 9.** *Single-purpose tools yield simpler data structures which in turn simplifies maintenance and testability.*

The clearer structure of the tool helped us to design a set of tests with more than 95 % code coverage. A similar endeavor would have been hopeless for Fiona as the various interfering features would have caused an astronomical number of test cases. For the same lack of interference with other features, it is much easier to repeat bugs on simple samples rather than big input files thus speeding up debugging.

With Wendy's 1,500 lines of code, it was quite easy to delegate further implementation tasks to students. Excited by the efficiency gains of Wendy, we span off more functionality from Fiona. The pattern described above repeated to all the tools which now form the SERVICE-TECHNOLOGY.ORG family. In addition, we observed that the separation into many autonomous tools actually led to tremendous speed-ups in the response to errors. As we assigned whole tools to students (what we could do because of the light weight of each tool), responsibilities for errors were fully transparent. Moreover, we recognized that the students committed much more to "their tool" than they used to commit to "their changes in the code" of the big tool. It was easier to get them to work. In the old tool, they hesitated to touch code as they feared "to break everything".

These observations may be typical for an academic context. Here, much of the programming work is carried out by students. On the one hand, their commitment to tool development is only part of their duties and only for a very limited time interval. Programming is often just a side-product of writing a thesis or a paper. This way, skills and experience do not grow over time and senior software architects are rare. On the other hand, the algorithmic solutions to be implemented tend to be rather involved as they stem from theoretically challenging problems. Hence, for successful tool development it is critical to provide an architectural environment that is as simple as possible.

**Lesson 10.** *Single purpose tools have just the right granularity for tool development in an academic context.*

Apparently, other groups have come to similar conclusions. At least, the plugin concepts in ProM and Oryx also seem to follow the idea that the unit of code to be delegated to a single student should be extremely well encapsulated. However, their plugin concept seems to require a very well designed and mature core unit which does not exist in our family. Furthermore, these tools offer very sophisticated graphical user interfaces which not only justifies the need of a central framework, but also a higher level of abstraction in the interface between the plugins and this framework. In our tool family, however, performance is the key factor and we decided to keep user interaction and abstraction layers to a minimum.

**Lesson 11.** *Single purpose tools permit tool development at varying speed and sophistication.*

In our case, the transition from a monolithic tool to a family of tools led to smaller release cycles. We were able to implement concepts such as code reviews, four-eyes programming and other *extreme programming* [6] techniques. In particular, the core control logic of each individual tool tends to be rather simple and linear.

Of course, one may object that we just shifted complexity from a single tool to the interplay between the tools. One issue could be the definition of formats for exchanging definitions. Surprisingly, this was a minor issue so far in our family, for the following simple reason:

**Lesson 12.** *Data exchanged between tools of the family are typically tied to precisely defined core concepts of the underlying theory.*

As the theoretical concepts are only to a small degree subject to continuous improvements, the exchange formats remain rather stable. Another issue with single purpose tools could be to find out what the single purpose of such a tool would be. Again the answer might be surprising.

**Lesson 13.** *It is not overly important to precisely understand the purpose of a single-purpose tool in early development stages.*

If, after some time of development, it turns out that the functionality of a tool should have been split into separate tools, just split then! This is what we have exercised with Fiona. Originally thought as a tool to decide controllability, Fiona grew over time into a tool for synthesizing partners and operating guidelines, for verifying substitutability, for synthesizing adapters, and other tasks. Only then we realized that this functionality would be better provided by more than one tool and only then could we plan these new tools. Only with the experience from Fiona we have been able to define the functionality that should go into the Petri net API. The costs for refactoring are affordable and provide a canonical opportunity for revising design decisions.

In addition, a certain modularity in the tool family is suggested by a corresponding modularity in the underlying theory. In the single-purpose paradigm, a cited result in the theory is often copied by a call of the related tool. Here, Marlene is an excellent example. Marlene is devoted to the computation of an adapter between otherwise incompatible services $P$ and $R$. Theory suggests [22] that the adapter $A$ should be composed of two parts, $E$ and $C$. $E$ implements the individual adaptation activities that can be performed *in principle* by the adapter, based on a specification which is part of the input to Marlene. $C$ controls, based on the actual exchanged messages, when to execute the available activities in $E$. According to the theory, $C$ is just a correctly interacting partner of the composition $P \oplus E \oplus R$. Hence, Marlene transforms the input into a Petri net $E$, calls Wendy on $P \oplus E \oplus R$ to synthesize $C$ and compiles the resulting $C$ to the final adapter $A = E \oplus C$. Every optimization or extension that is done to Wendy is thus immediately inherited by Marlene.

A third concern of the single-purpose paradigm could be the fear of divergence between the tools which could compromise interoperability. However, despite available tools for software engineering, a whole tool will always be more encapsulated than a single method.

**Lesson 14.** *It is easier to let different versions of a tool coexist than to maintain different versions of a function or method within a single tool.*

We think that this is an advantage of a tool family even compared to a plugin based architecture where it is nontrivial to run various versions of a plugin concurrently.

## 7 Conclusion

We introduced SERVICE-TECHNOLOGY.ORG as a tool family which provides prototype implementations for several theoretical results on correctness of business processes and services. The core technology providers in this family have been successfully exposed to real-world problems. These real-world problems were accessible, because we have several compilers which mediate between theoretical core formalisms and industrial languages. In this paper, we shared several insight that we observed upon tool development. As our development included a significant refactoring from a monolithic tool toward an interoperable tool family, we are able to directly compare different solutions. Our findings particularly concern the specifics of academic tool development.

## 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. van der Aalst, W.M.P., van Dongen, B.F., Günther, C.W., Mans, R.S., de Medeiros, A.K.A., Rozinat, A., Rubin, V., Song, M., Verbeek, H.M.W.E., Weijters, A.J.M.M.T.: ProM 4.0: Comprehensive support for *real* process analysis. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 484–494. Springer, Heidelberg (2007)
3. van der Aalst, W.M.P., Hofstede, A.H.M.t.: YAWL: yet another workflow language. Inf. Syst. 30(4), 245–275 (2005)
4. Alves, A., et al.: Web Services Business Process Execution Language Version 2.0. OASIS Standard, OASIS (2007)
5. Badouel, E., Darondeau, P.: Theory of regions. In: Reisig, W., Rozenberg, G. (eds.) APN 1998. LNCS, vol. 1491, pp. 529–586. Springer, Heidelberg (1998)
6. Beck, K.: Extreme Programming Explained: Embrace Change, 2nd edn. Addison-Wesley, Reading (2005)
7. Berthelot, G., Lri-Iie: Checking properties of nets using transformation. In: Rozenberg, G. (ed.) APN 1985. LNCS, vol. 222, pp. 19–40. Springer, Heidelberg (1986)

8. Billington, J., Christensen, S., van Hee, K.M., Kindler, E., Kummer, O., Petrucci, L., Post, R., Stehno, C., Weber, M.: The Petri net markup language: Concepts, technology, and tools. In: van der Aalst, W.M.P., Best, E. (eds.) ICATPN 2003. LNCS, vol. 2679, pp. 483–505. Springer, Heidelberg (2003)
9. Carmona, J., Cortadella, J., Kishinevsky, M.: Genet: a tool for the synthesis and mining of Petri nets. In: ACSD 2009, pp. 181–185. IEEE, Los Alamitos (2009)
10. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (1999)
11. Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A.: Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers. Trans. Inf. and Syst. E80-D(3), 315–325 (1997)
12. Dadam, P., Reichert, M.: The ADEPT project: a decade of research and development for robust and flexible process support. Computer Science - R&D 23(2), 81–97 (2009)
13. Decker, G., Kopp, O., Leymann, F., Weske, M.: BPEL4Chor: Extending BPEL for modeling choreographies. In: ICWS 2007, pp. 296–303. IEEE, Los Alamitos (2007)
14. Decker, G., Overdick, H., Weske, M.: Oryx - an open modeling platform for the BPM community. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 382–385. Springer, Heidelberg (2008)
15. Desel, J., Esparza, J.: Free Choice Petri Nets. Cambridge University Press, Cambridge (1995)
16. Dumas, M., Spork, M., Wang, K.: Adapt or perish: Algebra and visual notation for service interface adaptation. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 65–80. Springer, Heidelberg (2006)
17. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
18. Fahland, D.: Translating UML2 Activity Diagrams Petri nets for analyzing IBM WebSphere Business Modeler process models. Informatik-Berichte 226, Humboldt-Universität zu Berlin, Berlin, Germany (2008)
19. 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.) Business Process Management. LNCS, vol. 5701, pp. 278–293. Springer, Heidelberg (2009)
20. Berkelaar, M., et al.: Free Software Foundation: lpsolve: Mixed Integer Linear Programming (MILP) Solver, http://lpsolve.sourceforge.net
21. Fu, X., Bultan, T., Su, J.: Conversation protocols: a formalism for specification and verification of reactive electronic services. Theor. Comput. Sci. 328(1-2), 19–37 (2004)
22. Gierds, C., Mooij, A.J., Wolf, K.: Specifying and generating behavioral service adapter based on transformation rules. Preprint CS-02-08, Universität Rostock, Rostock, Germany (2008)
23. Gottschalk, K.: Web Services Architecture Overview. IBM whitepaper, IBM developerWorks (2000), http://ibm.com/developerWorks/web/library/w-ovr
24. Hamez, A., Hillah, L., Kordon, F., Linard, A., Paviot-Adet, E., Renault, X., Thierry-Mieg, Y.: New features in CPN-AMI 3: focusing on the analysis of complex distributed systems. In: ACSD, pp. 273–275. IEEE, Los Alamitos (2006)
25. van Hee, K.M., Oanea, O., Post, R., Somers, L.J., van der Werf, J.M.E.M.: Yasper: a tool for workflow modeling and analysis. In: ACSD 2006, pp. 279–282. IEEE, Los Alamitos (2006)
26. Hinz, S., Schmidt, K., Stahl, C.: Transforming BPEL to Petri nets. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 220–235. Springer, Heidelberg (2005)
27. Kaschner, K.: Safira: Implementing set algebra for service behavior. In: CEUR Workshop Proceedings, ZEUS 2010, vol. 563, pp. 49–56. CEUR-WS.org (2010)
28. Kaschner, K., Wolf, K.: Set algebra for service behavior: Applications and constructions. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 193–210. Springer, Heidelberg (2009)

29. Kiepuszewski, B., Hofstede, A.H.M.t., van der Aalst, W.M.P.: Fundamentals of control flow in workflows. Acta Inf. 39(3), 143–209 (2003)
30. Liske, N., Lohmann, N., Stahl, C., Wolf, K.: Another approach to service instance migration. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC 2009. LNCS, vol. 5900, pp. 607–621. Springer, Heidelberg (2009)
31. 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)
32. Lohmann, N.: A feature-complete Petri net semantics for WS-BPEL 2.0. In: Dumas, M., Heckel, R. (eds.) WS-FM 2007. LNCS, vol. 4937, pp. 77–91. Springer, Heidelberg (2008)
33. Lohmann, N., Kleine, J.: Fully-automatic translation of open workflow net models into simple abstract BPEL processes. In: Modellierung 2008. LNI, vol. 127, pp. 57–72. GI (2008)
34. Lohmann, N., Kopp, O., Leymann, F., Reisig, W.: Analyzing BPEL4Chor: Verification and participant synthesis. In: Dumas, M., Heckel, R. (eds.) WS-FM 2007. LNCS, vol. 4937, pp. 46–60. Springer, Heidelberg (2008)
35. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing interacting BPEL processes. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 17–32. Springer, Heidelberg (2006)
36. Lohmann, N., Massuthe, P., Wolf, K.: Operating guidelines for finite-state services. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 321–341. Springer, Heidelberg (2007)
37. Lohmann, N., Verbeek, H., Dijkman, R.M.: 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); Special Issue on Concurrency in Process-Aware Information Systems
38. Lohmann, N., Verbeek, H., Ouyang, C., Stahl, C.: Comparing and evaluating Petri net semantics for BPEL. Int. J. Business Process Integration and Management 4(1), 60–73 (2009)
39. Lohmann, N., Weinberg, D.: Wendy: A tool to synthesize partners for services. In: Lilius, J., Penczek, W. (eds.) PETRI NETS 2010. LNCS, vol. 6128, pp. 297–307. Springer, Heidelberg (2010)
40. Lohmann, N., Wolf, K.: Realizability is controllability. In: Laneve, C. (ed.) WS-FM 2010. LNCS, vol. 6194, pp. 110–127. Springer, Heidelberg (2010)
41. Massuthe, P., Weinberg, D.: Fiona: A tool to analyze interacting open nets. In: CEUR Workshop Proceedings, AWPN 2008, vol. 380, pp. 99–104. CEUR-WS.org. (2008)
42. Mendling, J., Moser, M., Neumann, G., Verbeek, H.M.W., van Dongen, B.F., van der Aalst, W.M.P.: Faulty EPCs in the SAP reference model. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 451–457. Springer, Heidelberg (2006)
43. Mennicke, S., Oanea, O., Wolf, K.: Decomposition into open nets. In: CEUR Workshop Proceedings, AWPN 2009, vol. 501, pp. 29–34. CEUR-WS.org (2009)
44. Milner, R.: Communication and concurrency. Prentice-Hall, Inc., Englewood Cliffs (1989)
45. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77(4), 541–580 (1989)
46. Oanea, O., Wimmel, H., Wolf, K.: New algorithms for deciding the siphon-trap property. In: Lilius, J., Penczek, W. (eds.) PETRI NETS 2010. LNCS, vol. 6128, pp. 267–286. Springer, Heidelberg (2010)
47. Ouyang, C., Verbeek, E., van der Aalst, W.M.P., Breutel, S., Dumas, M., Hofstede, A.H.M.t.: Formal semantics and analysis of control flow in WS-BPEL. Sci. Comput. Program. 67(2-3), 162–198 (2007)
48. Prasad, M.R., Biere, A., Gupta, A.: A survey of recent advances in SAT-based formal verification. STTT 7(2), 156–173 (2005)

49. Reichert, M., Rinderle-Ma, S., Dadam, P.: Flexibility in process-aware information systems. In: Jensen, K., van der Aalst, W.M.P. (eds.) ToPNoC II. LNCS, vol. 5460, pp. 115–135. Springer, Heidelberg (2009); Special Issue on Concurrency in Process-Aware Information Systems
50. Schröter, C., Schwoon, S., Esparza, J.: The Model-Checking Kit. In: van der Aalst, W.M.P., Best, E. (eds.) ICATPN 2003. LNCS, vol. 2679, pp. 463–472. Springer, Heidelberg (2003)
51. Somenzi, F.: CUDD: CU Decision Diagram Package, http://vlsi.colorado.edu/~fabio/CUDD/
52. Stahl, C., Massuthe, P., Bretschneider, J.: Deciding substitutability of services with operating guidelines. In: Jensen, K., van der Aalst, W.M.P. (eds.) ToPNoC II. LNCS, vol. 5460, pp. 172–191. Springer, Heidelberg (2009); Special Issue on Concurrency in Process-Aware Information Systems
53. Stahl, C., Reisig, W., Krstic, M.: Hazard detection in a GALS wrapper: A case study. In: ACSD 2005, pp. 234–243. IEEE, Los Alamitos (2005)
54. Sürmeli, J.: Profiling services with static analysis. In: CEUR Workshop Proceedings, AWPN 2009, vol. 501, pp. 35–40. CEUR-WS.org (2009)
55. Talcott, C.L., Dill, D.L.: Multiple representations of biological processes. In: Priami, C., Plotkin, G. (eds.) Transactions on Computational Systems Biology VI. LNCS (LNBI), vol. 4220, pp. 221–245. Springer, Heidelberg (2006)
56. 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)
57. Verbeek, H.M.W., van der Aalst, W.M.P., Hofstede, A.H.M.t.: Verifying workflows with cancellation regions and OR-joins: An approach based on relaxed soundness and invariants. Comput. J. 50(3), 294–314 (2007)
58. Verbeek, H.M.W., Basten, T., van der Aalst, W.M.P.: Diagnosing workflow processes using Woflan. Comput. J. 44(4), 246–279 (2001)
59. Weber, M., Kindler, E.: The Petri Net Kernel. In: Ehrig, H., Reisig, W., Rozenberg, G., Weber, H. (eds.) Petri Net Technology for Communication-Based Systems. LNCS, vol. 2472, pp. 109–124. Springer, Heidelberg (2003)
60. Weinberg, D.: Efficient controllability analysis of open nets. In: Bruni, R., Wolf, K. (eds.) WS-FM 2008. LNCS, vol. 5387, pp. 224–239. Springer, Heidelberg (2009)
61. Wolf, K.: Generating Petri net state spaces. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 29–42. Springer, Heidelberg (2007)
62. Wolf, K.: Does my service have partners? In: Jensen, K., van der Aalst, W.M.P. (eds.) Transactions on Petri Nets. LNCS, vol. 5460, pp. 152–171. Springer, Heidelberg (2009)
63. Wolf, K., Stahl, C., Ott, J., Danitz, R.: Verifying livelock freedom in an SOA scenario. In: ACSD 2009, pp. 168–177. IEEE, Los Alamitos (2009)

# Deciding Behaviour Compatibility of Complex Correspondences between Process Models

Matthias Weidlich[1], Remco Dijkman[2], and Mathias Weske[1]

[1] Hasso-Plattner-Institute, University of Potsdam, Germany
{matthias.weidlich,weske}@hpi.uni-potsdam.de
[2] Eindhoven University of Technology, The Netherlands
r.m.dijkman@tue.nl

**Abstract.** Compatibility of two process models can be verified using common notions of behaviour inheritance. However, these notions postulate 1:1 correspondences between activities of both models. This assumption is violated once activities from one model are refined or collapsed in the other model or in case there are groups of corresponding activities. Therefore, our work lifts the work on behaviour inheritance to the level of complex 1:n and n:m correspondences. Our contribution is (1) the definition of notions of behaviour compatibility for models that have complex correspondences and (2) a structural characterisation of these notions for sound free-choice process models that allows for computationally efficient reasoning. We show the applicability of our technique, by applying it in a case study in which we determine the compatibility between a set of reference process models and models that implement them.

## 1 Introduction

For two process models the compatibility of their behaviour can be verified, by determining that their behaviour is equivalent, modulo activities that have been added, removed, or refined. Compatibility verification is, for example, applied to determine whether a business process correctly implements the service that an organization provides to its clients, as it is specified by another (abstract) process (cf., [1]). As another example, compatibility verification is used to check whether a business process correctly implements a reference process (cf., [2,3]).

Compatibility verification is based on correspondences that are defined between activities that are considered to be equivalent. While we assume these correspondences to be given, we discuss techniques for identifying correspondences when reviewing related work. For the case of elementary 1:1 correspondences between activities, common notions of behaviour inheritance [4,5] can be applied to check for the absence of behavioural contradictions. These notions differ with respect to the treatment of activities that are without counterpart in the other model (i.e., added or removed). In the behavioural analysis, these transitions might either be *hidden* or *blocked*. If both models satisfy a certain behaviour equivalence, e.g., branching bisimulation or trace equivalence, once activities that are without any correspondence are hidden (blocked), we conclude on projection inheritance (protocol inheritance) [4].

In this paper, we solve the problem of checking compatibility in the presence of complex 1:n and n:m correspondences between two process models. We build upon the

R. Hull, J. Mendling, and S. Tai (Eds.): BPM 2010, LNCS 6336, pp. 78–94, 2010.

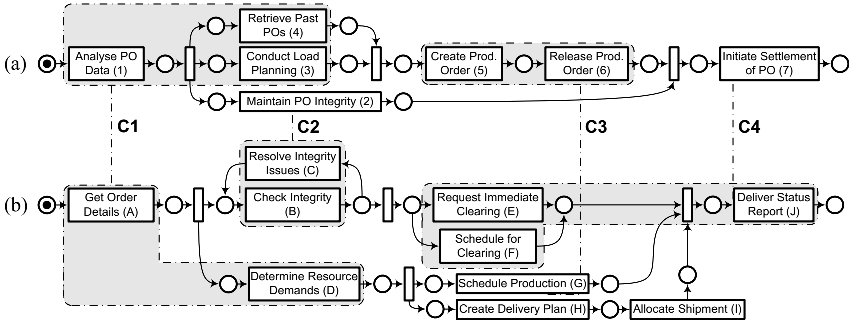

(a)
Analyse PO Data (1)
Retrieve Past POs (4)
Conduct Load Planning (3)
Maintain PO Integrity (2)
Create Prod. Order (5)
Release Prod. Order (6)
Initiate Settlement of PO (7)
C1
C2
C3
C4
(b)
Get Order Details (A)
Resolve Integrity Issues (C)
Check Integrity (B)
Request Immediate Clearing (E)
Schedule for Clearing (F)
Deliver Status Report (J)
Determine Resource Demands (D)
Schedule Production (G)
Create Delivery Plan (H)
Allocate Shipment (I)

**Fig. 1.** Two process models that illustrate an order processing, $(a)$ is a reference model, $(b)$ is a model customised for a specific organisation

existing work on behaviour inheritance and lift it to the level of complex correspondences. Here, 1:n correspondences stem from activities from one model that are refined or collapsed in the other model. Moreover, n:m correspondences represent a relation between sets activities, for which there are no correspondences between one of their activity subsets. That is due to differences in modularisation of functionality between two process models. Fig. 1 illustrates our setting by a reference model $(a)$ and a customised process model $(b)$, along with four correspondences. Apparently, model $(b)$ is not a hierarchical refinement of model $(a)$, such that we observe a non-trivial relation between both models. For instance, activities $A$ and $D$ of model $(b)$ have been identified to correspond to activities $1$, $3$, and $4$ in the reference model $(a)$. We answer the question, whether these correspondences are compatible.

The contribution of this paper is twofold. First, we introduce the notions of projection and protocol compatibility of correspondences and, therefore, process models. To this end, we use trace equivalence as the underlying equivalence criterion. Albeit based on the ideas of behaviour inheritance, we speak of compatibility as the notions are not directed. Second, we show that for the class of sound free-choice process models, these notions of compatibility can be characterised structurally. Thus, our notions can be decided efficiently based on structural analysis. We also report on findings from a case study. Due to space limitations, all proofs can be found in [6].

The remainder of this paper is structured as follows. Section 2 gives preliminaries for our work in terms of a formal model. Section 3 elaborates on our notions of behaviour compatibility of correspondences. Subsequently, their structural characterisation is addressed in Section 4. Section 5 introduces our case study. Finally, we review related work in Section 6 and conclude in Section 7.

## 2 Preliminaries

Our investigations are based on workflow (WF-) nets [7], a class of Petri nets used for process modelling and analysis. Petri net based formalisations have been presented for (parts of) common process modelling languages, such as BPEL, BPMN, and UML activity diagrams (e.g., [8,9,10]).

We recall basic definitions according to [7,11]. A *net* is a tuple $N = (P, T, F)$ with $P$ and $T$ as finite disjoint sets of places and transitions, and $F \subseteq (P \times T) \cup (T \times P)$ as the flow relation. Without stating it explicitly, we assume a net to be always defined as $N = (P, T, F)$. We write $X = (P \cup T)$ for all nodes. The transitive closure of $F$ is denoted by $F^+$. For a node $x \in X$, its preset and postset are defined as $\bullet x := \{y \in X \mid (y, x) \in F\}$ and $x\bullet := \{y \in X \mid (x, y) \in F\}$, respectively. A tuple $N' = (P', T', F')$ is a *subnet* for a net $N = (P, T, F)$, if $P' \subseteq P$, $T' \subseteq T$, and $F' = F \cap ((P' \times T') \cup (T' \times P'))$. Note that a subnet is induced by a given subset of places or transitions, respectively. A net $N$ is *free-choice*, iff $\forall\, p \in P$ with $|p\bullet| > 1$ holds $\bullet(p\bullet) = \{p\}$. A *workflow (WF-) net* is a net $N = (P, T, F)$, such that there is exactly one place $i \in P$ with $\bullet i = \emptyset$, exactly one place $o \in P$ with $o\bullet = \emptyset$, and $\forall\, x \in X\ [\, iF^+x \wedge xF^+o\,]$. A *path* of length $n \in \mathbb{N}$, $n > 1$, is a sequence $\pi : \{1, \ldots, n\} \mapsto X$, denoted by $\pi(x_1, x_n)$ or $\pi = x_1, \ldots, x_n$, which satisfies $((1, x_1), (2, x_2)), \ldots, ((n-1, x_{n-1}), (n, x_n)) \in F$. We write $t \in \pi$ if $(i, t) \in \pi$ for some $i \in \mathbb{N}$. A *subpath* $\pi'$ of a path $\pi$ is a subsequence. The set $\mathcal{P}_N$ contains all complete paths $\pi(i, o)$ of a WF-net $N$. A path $\pi = x_1, \ldots, x_n$ in a net $N = (P, T, F)$ can be restricted to its transitions yielding the path $\pi^T = x_1, x_3, \ldots, x_m$ (if $x_1 \in T$) or $\pi^T = x_2, x_4, \ldots, x_m$ (otherwise) with $m \in \{n-1, n\}$ and $x_m \in T$. The set $\mathcal{P}_N^T$ contains all complete paths restricted to their transitions of $N$. We write $\pi^T \subseteq T'$ if for all $(i, t) \in \pi$ it holds $t \in T' \subseteq T$.

We define semantics for a WF-net $N = (P, T, F)$ with initial place $i$ and final place $o$ according to [7]. $M : P \mapsto \mathbb{N}$ is a *marking* of $N$, $\mathbb{M}$ is the set of all markings. For a place $p \in P$, $M_p$ is the marking that puts a token on $p$ and no token elsewhere. For a transition $t \in T$, $M_t$ is the marking that puts a token on every place $p \in \bullet t$ and no token elsewhere. For a WF-net, $M_i$ is the initial, $M_o$ the final marking. $M(p)$ returns the number of tokens in $p$, if $p \in dom(M)$. Moreover, for two markings $M, M' \in \mathbb{M}$, $M \geq M'$ if $M(p) \geq M'(p)$ for all $p \in P$. For any transition $t \in T$ and any marking $M \in \mathbb{M}$, $t$ is *enabled* in $M$, denoted by $(N, M)[t\rangle$, iff $\forall\, p \in \bullet t\ [\, M(p) \geq 1\,]$. Marking $M'$ is reached from $M$ in $N$ by *firing* of $t$, denoted by $(N, M)[t\rangle(N, M')$, such that $M' = M - \bullet t + t\bullet$. A *firing sequence* of length $n \in \mathbb{N}$ is a sequence $\sigma : \{1, \ldots, n\} \mapsto T$. For $\sigma = \{(1, t_x), \ldots, (n, t_y)\}$, we also write $\sigma = t_1, \ldots, t_n$. We write $t \in \sigma$ if $(i, t) \in \sigma$ for some $i \in \mathbb{N}$, and $\sigma \subseteq T'$ if for all $(i, t) \in \sigma$ it holds $t \in T' \subseteq T$. Any firing sequence $\sigma$ with $(N, M_i)[\sigma\rangle(N, M_o)$ is a *complete trace* (or trace). The set of all complete traces of $N$ is the *language* of $N$, denoted by $\mathcal{L}_N$. A *subtrace* $\sigma'$ of a trace $\sigma$ is a subsequence of $\sigma$. A marking $M' \in \mathbb{M}$ is reachable from $M \in \mathbb{M}$ in $N$, denoted by $M' \in [N, M\rangle$, if there exists a firing sequence $\sigma$, such that $(N, M)[\sigma\rangle(N, M')$.

We also recall the *soundness* criterion, which requires WF-nets (1) to always terminate, and (2) to have no dead transitions (proper termination is implied for WF-nets) [12]. A WF-net $N$ is *live*, if for every marking $M \in [N, M_i\rangle$ and $t \in T$, there is a marking $M' \in [N, M\rangle$ such that $(N, M')[t\rangle$. A WF-net $N$ is *bounded*, iff the set $[N, M_i\rangle$ is finite. A WF-net $N$ with $N = (P, T, F)$ is *sound*, iff the short-circuit net $N'$, $N' = (P, T \cup \{t_c\}, F \cup \{(o, t_c), (t_c, i)\})$, is live and bounded.

## 3 Behaviour Compatibility of Correspondences

This section introduces behaviour compatibility for correspondences between WF-nets. We use WF-nets as behavioural models due to our focus on process models. It is worth to mention though, that all concepts can be lifted to general Petri nets or even state transitions systems in a straightforward manner. First, Section 3.1 clarifies the notion of a correspondence. Second, Section 3.2 elaborates on a partitioning of traces that is imposed by these correspondences. Third, Section 3.3 introduces two kinds of behaviour compatibility for a pair of correspondences. Finally, Section 3.4 elaborates on how to decide behaviour compatibility.

### 3.1 Correspondences between WF-Nets

In general, a correspondence between two WF-nets is defined by two sets of transitions of the WF-nets. Following on the classification of correspondences between data schemata or ontologies [13], we speak of elementary or complex correspondences depending on the cardinality of the associated set of transitions.

**Definition 1 (Correspondence).** *Let $N = (P, T, F)$ and $N' = (P', T', F')$ be two WF-nets. The* correspondence relation $\equiv \subseteq \wp(T) \times \wp(T')$ *associates corresponding sets of transitions of both nets to each other. Let $T_1 \subseteq T$ and $T_2 \subseteq T'$. If $T_1 \equiv T_2$ then $(T_1, T_2)$ is referred to as a* correspondence. *$(T_1, T_2)$ is called* elementary*, iff $|T_1| = |T_2| = 1$, and* complex *otherwise.*

In the remainder of this paper, we assume all correspondences to be non-overlapping. That is, two correspondences $C = (T_1, T_3)$ and $C' = (T_2, T_4)$ must not share any transition in any of the WF-nets, i.e., $T_1 \cap T_2 = \emptyset$ and $T_3 \cap T_4 = \emptyset$. Overlapping correspondences raise various questions regarding their intended meaning. Assume two correspondences are defined as $C = (\{a\}, \{x, y\})$ and $C' = (\{b\}, \{y\})$. Then, the occurrence of the two transitions $x$ and $y$ in one model might correspond to both, the occurrence of transition $a$ only, or the occurrence of both transitions, $a$ and $b$, in the other model. Hence, the inherent semantic ambiguity of overlapping correspondences has to be addressed as a prerequisite for any behavioural analysis.

In our example in Fig. 1, for instance, $C1$ would be classified as a complex 3:2 correspondence, while $C3$ is a 2:1 correspondence. Note that the correspondences depicted in Fig. 1 are all non-overlapping.

After having defined the notion of a correspondence, it is worth to mention that such a correspondence induces certain semantics. In terms of trace semantics, the transitions that are part of a correspondence occur in dedicated subtraces of the net. Thus, a correspondence induces a relation between subtraces of the one net and subtraces of the other net. For instance, correspondence $C1$ in Fig. 1 relates the subtraces $\langle 1, 3, 4\rangle$ and $\langle 1, 4, 3\rangle$ in model $(a)$ to the subtrace $\langle A, D\rangle$ in model $(b)$. For $C2$, in turn, there is a relation between the subtrace $\langle 2\rangle$ in model $(a)$ and an infinite set of traces in model $(b)$, e.g., $\langle B\rangle$ and $\langle B, C, B\rangle$.

### 3.2 Trace Partitioning Based on Correspondences

In the previous section, we argued that a correspondence between sets of transitions induces semantics in terms of subtraces of two nets that are considered to correspond to each other against the background of the alignment. Therefore, for two correspondences, the constraints between the respective subtraces of both correspondences imposed by one net, should hold for the respective subtraces in the other net as well. Here, constraints refer to the observable order and number of occurrences of such subtraces in all complete traces of a net.

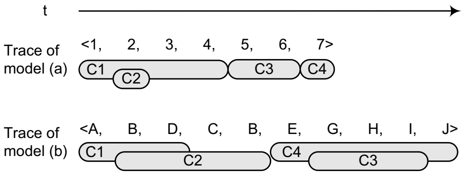


**Fig. 2.** Exemplary traces of the models of Fig. 1 along with their relation to correspondences

We illustrate the relation between subtraces by means of the WF-nets of Fig. 1. Here, we see that the constraints for the subtraces relating to the correspondences $C1$ and $C3$ are equal. That is, any subtrace of model $(a)$ built of transitions of $C1$, is followed by a subtrace comprising transitions of $C3$. In addition, both subtraces occur at most once. This also holds for the respective subtraces in model $(b)$, as exemplified for a pair of traces in Fig. 2. For correspondences $C1$ and $C2$, the constraints imposed by both models are equal either. That is, in any trace of both nets, a transition belonging to $C1$ is observed first and might potentially be followed by transitions of $C1$ and $C2$. Note that the specific order of interleaving transitions of both correspondences is different though. For instance, in the subtrace $\langle A, B, D, C, B\rangle$ of model $(b)$, two transitions of $C1$ are followed by a transition of $C2$. This is not possible in any subtrace of model $(a)$, due to the different number of interleaving transitions of $C1$ and $C3$ in both nets, cf., Fig. 2. When focussing on correspondences $C3$ and $C4$, however, we detect differences in the imposed constraints. For instance, there is a trace in model $(b)$, in which a transition of $C4$ occurs before any transition of $C3$, which yields a contradiction with the semantics of model $(a)$.

These examples illustrate that the interleaving of transitions belonging to different correspondences has to be taken into account when assessing behaviour compatibility. We speak of two interleaving transitions, if for every trace in which they directly occur together, there is another trace that is equal to the first one besides a switched order of the interleaving transitions.

**Definition 2 (Interleaving Transitions).** *Let $N = (P, T, F)$ be a WF-net. Two transitions $(t_1, t_2) \in (T \times T)$ are* interleaving, *iff for each trace $\sigma_1 \in \mathcal{L}_N$ with $(i, t_1), (i+1, t_2) \in \sigma_1$ for some $i \in \mathbb{N}$, there is a trace $\sigma_2 \in \mathcal{L}_N$ with $\sigma_2 = \{(i, t_2), (i+1, t_1)\} \cup \{(j, t) \mid (j, t) \in \sigma_1 \ \wedge \ j \neq i \ \wedge \ j \neq i+1\}$. Given two disjoint sets of transitions $T_1, T_2 \subseteq T$, the set $\iota_{(T_1,T_2)}(N) \subseteq T_1 \cup T_2$ contains all transitions that are part of an interleaving transition pair $(t_1, t_2) \in (T_1 \times T_2)$.*

For our example in Fig. 1, the set of interleaving transitions for the correspondences $C_1 = (T_1, T_3)$ and $C_2 = (T_2, T_4)$ with $T_1 = \{1, 3, 4\}$, $T_2 = \{2\}$, $T_3 = \{A, D\}$, and $T_4 = \{B, C\}$ are defined as $\iota_{(T_1,T_2)} = \{2, 3, 4\}$ for model $(a)$ and $\iota_{(T_3,T_4)} = \{B, C, D\}$ for model $(b)$.

Given two correspondences for a net, their dependencies can be assessed in a certain trace by partitioning the trace into subtraces that represent interleaving and non-interleaving parts of the correspondences.

**Definition 3 (Partitioning of a Trace).** *Let $N = (P, T, F)$ be a WF-net and $T_1, T_2 \subseteq T$ two disjoint sets of transitions. For any trace $\sigma \in \mathcal{L}_N$ the partitioning $\rho_{(T_1,T_2)}(\sigma)$ induced by $T_1$ and $T_2$ is a sequence of subtraces of maximal length $\rho_{(T_1,T_2)}(\sigma) = \sigma_1, \ldots, \sigma_n$ such that for any $i \in \mathbb{N}$ with $1 \leq i \leq n$ it holds either $\sigma_i \subseteq T_1 \setminus \iota_{(T_1,T_2)}(N)$, $\sigma_i \subseteq T_2 \setminus \iota_{(T_1,T_2)}(N)$, $\sigma_i \subseteq \iota_{(T_1,T_2)}(N)$, or $\sigma_i \subseteq T \setminus (T_1 \cup T_2)$.*

According to this definition, any transition that is part of a trace belongs to one of the four classes w.r.t. two sets of transitions. It is an interleaving transition, it belongs to one of the sets of transitions without being an interleaving transition, or it is not part of the two sets at all. The partitioning of traces for the models of our example is illustrated in Fig. 3 for three exemplary pairs of transition sets. As these sets are induced by the respective correspondences, we also speak of *transitions of correspondences* and name the sets accordingly. Note that all transitions that do not relate to the respective correspondences have been neglected. For correspondences $C1$ and $C3$, in all traces a subtrace comprising non-interleaving transitions of $C1$ is followed by a subtrace comprising non-interleaving transitions of $C3$ in both models, $(a)$ and $(b)$. Similarly, for correspondences $C1$ and $C2$, non-interleaving transitions of $C1$ are followed by a subtrace consisting of interleaving transitions of both correspondences in both models. In contrast, for correspondences $C3$ and $C4$, the contradicting constraints as discussed above are visible in the trace partitioning. Non-interleaving transitions of $C3$ are followed by non-interleaving transitions of $C4$ in model $(a)$, whereas interleaving transitions of $C3$ and $C4$ are followed by non-interleaving transitions of $C4$ in model $(b)$.

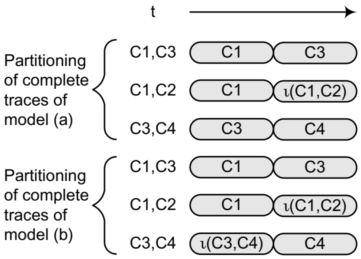

t
Partitioning of complete traces of model (a)
C1,C3
C1
C3
C1,C2
C1
ι(C1,C2)
C3,C4
C3
C4
Partitioning of complete traces of model (b)
C1,C3
C1
C3
C1,C2
C1
ι(C1,C2)
C3,C4
ι(C3,C4)
C4

**Fig. 3.** Partitioning of traces of the models in Fig. 1

### 3.3 Notions of Behaviour Compatibility

Based on a trace partitioning induced by the transitions of two correspondences, we define two notions of compatibility. Informally, both notions require that for each trace of the one net, there is a trace in the other net that shows the same partitioning in interleaving and non-interleaving subtraces of the transitions of the respective correspondences. Still, we distinguish two ways of coping with transitions that are not part of any correspondence. They might be hidden or blocked, cf., [4,5]. Following on the notions of projection inheritance (hiding) and protocol inheritance (blocking), this distinction leads to the notions of projection and protocol compatibility of correspondences. First and foremost, we define projection compatibility for correspondences. It uses the notion of a trace projection. Given a WF-net $N = (P, T, F)$, a set of transitions $H \subseteq T$, and a trace $\sigma \in \mathcal{L}_N$, the set $H_{\sigma|j} = \{(x, t) \in \sigma \mid x < j \wedge t \in H\}$ denotes the occurrences of transitions of $H$ in $\sigma$ up to index $j \in \mathbb{N}$. Based thereon, we define the projection $\tau_H(\sigma)$ for a trace $\sigma \in \mathcal{L}_N$ of length $n$ induced by $H$ as $\tau_H(\sigma) = \bigcup_{i=0}^{|H_{\sigma|n}|}(i, t_i)$ with $t_i \in H$, such that $\exists\, j \in \mathbb{N}\ [\ (j, t_i) \in \sigma \wedge i = |H_{\sigma|j}|\ ]$. Informally, the projected trace $\tau_H(\sigma)$ is derived by taking all transitions in $H$ from $\sigma$.

**Definition 4 (Projection Compatibility).** *Let $N = (P, T, F)$ and $N' = (P', T', F')$ be WF-nets, and $C_1 = (T_1, T_3)$, $C_2 = (T_2, T_4)$ two correspondences.*

- ○ *$C_1$ and $C_2$ are* projection compatible from $N$ to $N'$, *iff for any trace $\sigma \in \mathcal{L}_N$, there is a trace $\sigma' \in \mathcal{L}_{N'}$, such that for the partitioned projections $\rho_{(T_1,T_2)}(\tau_{(T_1 \cup T_2)}(\sigma)) = \sigma_1, \ldots, \sigma_n$ and $\rho_{(T_3,T_4)}(\tau_{(T_3 \cup T_4)}(\sigma')) = \sigma'_1, \ldots, \sigma'_m$ it holds $n = m$ and for all $i \in \mathbb{N}$ with $0 \leq i \leq n$:*
  - $\sigma_i \subseteq (T_1 \setminus \iota_{(T_1,T_2)}(N)) \Rightarrow \sigma'_i \subseteq (T_3 \setminus \iota_{(T_3,T_4)}(N'))$.
  - $\sigma_i \subseteq (T_2 \setminus \iota_{(T_1,T_2)}(N)) \Rightarrow \sigma'_i \subseteq (T_4 \setminus \iota_{(T_3,T_4)}(N'))$.
  - $\sigma_i \subseteq \iota_{(T_1,T_2)}(N) \Rightarrow \sigma'_i \subseteq \iota_{(T_3,T_4)}(N')$.
- ○ *$C_1$ and $C_2$ are* projection compatible, *iff they are projection compatible in either direction.*

Projection compatibility of two correspondences between two nets means that every complete trace in one net has a corresponding complete trace in the other net, which shows the same partitioning w.r.t. the two correspondences. We see that projection compatibility can be decided for two correspondences in isolation, i.e., independent of other correspondences. That is due to the fact that any transitions not belonging to the respective correspondences are projected before comparing the partitioning of traces. In contrast, protocol compatibility of two correspondences has to be decided always against the background of an alignment, i.e., a set of correspondences. Following on the approach introduced for protocol inheritance in [4], we use an encapsulation operator $\delta_H$ that creates the subnet induced by a set of transitions $H \subseteq T$ from a net $N = (P, T, F)$, such that $\delta_H(N) = (P, H, F_H)$. Encapsulation of a WF-net might yield a net that is not a WF-net anymore. Therefore, we also define the normalisation operator $\eta_N$ that creates the workflow subnet of a subnet $N_1$ of a WF-net $N$, such that $\eta_N(N_1) = (P_\eta, T_\eta, F_\eta)$ with $P_\eta = P_1 \setminus X_r$ and $T_\eta = T_1 \setminus X_r$ and $X_r = \{x \in X_1 \mid i \cancel{F_1^+} x \ \vee \ x \cancel{F_1^+} o\}$ ($i$ and $o$ being the initial and final place of $N$). Normalisation yields the empty net, if there is no workflow subnet.

**Definition 5 (Protocol Compatibility).** *Let $N = (P, T, F)$ and $N' = (P', T', F')$ be WF-nets and $\equiv$ a correspondence relation between them. Let $T_\equiv \subseteq T$ and $T'_\equiv \subseteq T'$ be the transitions of both nets that are part of any correspondence, and $E = \eta_N(\delta_{T_\equiv}(N))$ and $E' = \eta_{N'}(\delta_{T'_\equiv}(N'))$ the normalised encapsulated nets. Let $C_1 = (T_1, T_3)$ and $C_2 = (T_2, T_4)$ be two correspondences.*

- ○ *$C_1$ and $C_2$ are* protocol compatible from $N$ to $N'$, *iff $E$ and $E'$ are WF-nets and $C1$ and $C2$ are projection compatible from $E$ to $E'$.*
- ○ *$C_1$ and $C_2$ are* protocol compatible, *iff they are protocol compatible in either direction.*

Protocol compatibility of correspondences between two nets captures that every complete trace of the one net has a corresponding complete trace in the other net, which shows the same partitioning w.r.t. the two correspondences once transitions that are not aligned are removed. Thus, protocol compatibility is traced back to projection compatibility of the normalised encapsulated nets that contain only aligned transitions. However, it is important to notice that both notions are orthogonal. That is, correspondences between two nets might show solely projection compatibility, but not protocol compatibility, and vice versa.

Regarding the WF-nets in Fig. 1, we conclude that, for instance, correspondences $C1$ and $C2$ are projection compatible, whereas correspondences $C1$ and $C4$ are not due to the interleaving of their transitions in model $(b)$ (transitions $D$, $E$, and $F$), which is not possible in model $(a)$, cf., Fig. 3. Direct application of the protocol compatibility criterion to our example yields a negative result, as both models contain no-operation (NOP) transitions that only realise the splitting and merging of control flow. These transitions are not part of the encapsulated nets, such that normalisation yields an empty net and there is no completed trace from the initial to the final marking in both nets. Still, these NOP transitions can be neglected in a way that they are part of the encapsulated nets. Then, for instance, encapsulation removes transitions $H$ and $I$ of model $(b)$. As this does not change the observable behaviour of the remaining transitions, correspondences $C1$ and $C2$ are also protocol compatible, whereas correspondences $C1$ and $C4$ are not, owing to the aforementioned issues.

So far, we discussed the compatibility of a pair of correspondences in isolation. Both notions can be lifted from a pair of correspondences to a set of correspondences between two nets in a straightforward manner.

**Definition 6 (Projection & Protocol Compatibility (Correspondence Relation)).** *A correspondence relation between two nets is* projection (protocol) compatible*, iff all correspondences are pairwise projection (protocol) compatible.*

### 3.4 Decidability of Behaviour Compatibility

In the general case, behaviour compatibility of two correspondences between two WF-nets can be decided by state space exploration. Under the assumption of a finite state space, all state transitions relate to none of the correspondences, one of the correspondences, or the interleaving of both correspondences, respectively. Therefore, the trace partitioning (cf., Definition 3) is directly visible in the respective labelled transition system (LTS). Fig. 4 illustrates this dependency by the LTS of model (a) of Fig. 1. In the lower system, we highlighted the transitions that are related to correspondences $C_1 = (T_1, T_3)$ and $C_2 = (T_2, T_4)$. Each state transition that is part of $T_1 \cup T_2$ can be classified as belonging to one of the three sets, $\iota_{(T_1,T_2)}$, $T_1 \setminus \iota_{(T_1,T_2)}$, or $T_2 \setminus \iota_{(T_1,T_2)}$. Based thereon, complex state transitions are derived that represent the transition sequences of maximal length comprising solely transitions of one of the aforementioned three sets and transitions that are not in $T_1 \cup T_2$. The latter is illustrated in the lower LTS in Fig. 4, in which the transition $\iota_{(C_1,C_2)}$ contains solely interleaving transitions and transitions that are not related to $C1$ and $C2$.

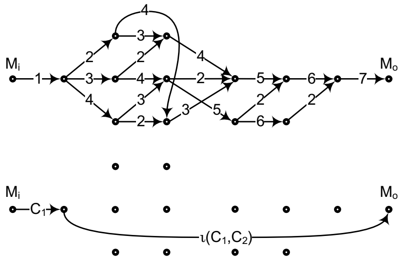


**Fig. 4.** LTS of model (a) of Fig. 1 along with transitions related to correspondences $C1$ and $C2$

Moreover, interleaving transitions can be characterised as being enabled concurrently in some marking or as not changing the marking when being fired.

**Lemma 1.** *Let $N = (P, T, F)$ be a WF-net. A pair of transitions $(t_1, t_2) \in (T \times T)$ is interleaving, iff there is a marking $M \in [N, M_i\rangle$ such that (1) $M \geq M_{t_1} + M_{t_2}$ and with $(N, M)[t_1\rangle(N, M_1)$ and $(N, M)[t_2\rangle(N, M_2)$ it holds $M_o \in [N, M_1\rangle$ and $M_o \in [N, M_2\rangle$, or (2) $(N, M)[t_1\rangle(N, M)$, $(N, M)[t_2\rangle(N, M)$, and $M_o \in [N, M\rangle$.*

Based thereon, we conclude decidability of behaviour compatibility of correspondences for nets with a finite state space. The proofs can be found in [6].

**Theorem 1.** *Given two bounded WF-nets and a set of correspondences, it is decidable whether two correspondences are projection or protocol compatible.*

Apparently, any approach of deciding behaviour compatibility based on state space exploration is computationally hard in the general case, due to the state explosion problem. The problem of whether two LTS show the same trace semantics is PSPACE-complete [15]. Hence, structural characterisations of behaviour compatibility for certain classes of nets are crucial for any real-world application.

## 4 A Structural Characterisation of Compatibility

This section shows that projection compatibility and, therefore, also protocol compatibility are decided efficiently for correspondences between sound free-choice WF-nets. That is due to the fact that for sound free-choice WF-nets, there is a tight coupling of syntax and semantics. First, Section 4.1 discusses the properties of sound free-choice WF-nets that are used in our approach. Second, Section 4.2 introduces the notion of path consistency of two correspondences between two WF-nets. Finally, Section 4.3 elaborates on how this structural characterisation is used to decide behaviour compatibility.

### 4.1 Properties of Sound Free-Choice WF-Nets

As mentioned above, sound free-choice WF-nets show a tight coupling of syntax and semantics. In particular, if $N$ is sound and free-choice, the existence of a path $\pi(x, y)$ between places $x$ and $y$ implies the existence of a firing sequence containing all transitions on $\pi(x, y)$ (cf., Lemma 4.2 in [16]). Actually, this implication requires the marking $M_y$ to be a home marking (a marking reachable from every marking that is reachable from the initial state). Still, the implication might be lifted to all home markings $M_1$ with $M_1(y) > 0$. Due to soundness of the net $N$, the short-circuit net $N'$ is live and bounded, such that all markings $M \in [N, M_i\rangle$ are home markings in $N'$. Thus, all markings $M_1(y) > 0$ are reachable from markings $M_2(x) > 0$, if $M_1, M_2 \in [N', M_i\rangle$.

Another important property of sound free-choice nets is the possibility to compute the following two relations efficiently.

**Concurrency Relation.** The concurrency relation $|| \subseteq X \times X$ for the nodes $X$ of a net $N$ contains all pairs $(x_1, x_2)$ such that $M \geq M_{x_1} + M_{x_2}$ for some reachable marking $M$. Thus, the concurrency relation identifies concurrently enabled transitions or marked places, respectively. Note that any sound free-choice net is also safe (cf., Lemma 1 in [17]). Thus, a single transition cannot be enabled concurrently with itself.

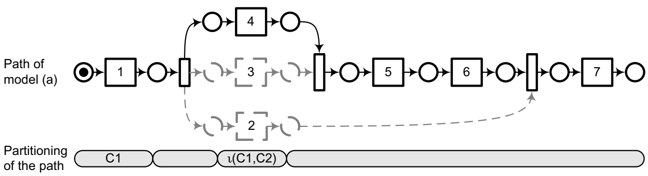


**Fig. 5.** A path of model $(a)$ of Fig. 1 along with the partitioning induced by the non-interleaving and interleaving transitions of correspondences $C1$ and $C2$

According to [18], the concurrency relation can be determined in $O(n^3)$ time for live and bounded free-choice nets with $n$ as the number of nodes of the net.

**Exclusiveness Relation.** The exclusiveness relation $+ \subseteq T \times T$ for the transitions of a net $N$ contains all pairs $(t_1, t_2)$ that never occur together in a complete trace, i.e., for all complete traces $\sigma \in \mathcal{L}_N$ it holds $t_1 \in \sigma \Rightarrow t_2 \notin \sigma$ and $t_2 \in \sigma \Rightarrow t_1 \notin \sigma$. According to [19] (Lemma 3), the exclusiveness relation can be deduced from the concurrency relation and the transitive closure of the flow relation for sound free-choice nets. Based thereon, the exclusiveness relation can also be computed in $O(n^3)$ time with $n$ as the number of nodes as detailed in [19]. The exclusiveness relation can be lifted from the transitions to all nodes of a net. Two places $p_1$ and $p_2$ are exclusive if there is no complete trace that visits two markings $M_1$ and $M_2$ with $M_1(p_1) > 0$ and $M_2(p_2) > 0$. Obviously, this information can be deduced directly from the exclusiveness of transitions.

In our example in Fig. 1, for instance, transitions $D$ and $E$ of model $(b)$ are in the concurrency relation, while transitions $E$ and $F$ are exclusive.

## 4.2 Path Consistency of Correspondences

In order to reason on behaviour compatibility of two correspondences between two sound free-choice WF-nets, we assess their structural consistency. That is, the existence of certain paths in two process models is evaluated with respect to the correspondences. To this end, we define the partitioning of a path that is induced by two sets of transitions, similar to the partitioning of a trace presented in Section 3.2. Here, we consider solely the transitions of a path and neglect all places. Note that such a partitioning is grounded on the interleaving transitions of both sets. However, according to Lemma 1, the notion of two interleaving transitions can be traced back to their concurrent enabling (or a structural analysis of their pre- and postset, respectively), which, in turn, can be decided structurally for sound free-choice WF-nets, cf., Section 4.1.

**Definition 7 (Partitioning of a Path).** *Let $N = (P, T, F)$ be a WF-net and $T_1, T_2 \subseteq T$ two disjoint sets of transitions. For any path $\pi \in \mathcal{P}_N^T$ the partitioning $\rho_{(T_1,T_2)}(\pi)$ induced by $T_1$ and $T_2$ is a sequence of subpaths of maximal length $\rho_{(T_1,T_2)}(\pi) = \pi_1, \ldots, \pi_n$ such that for any $i \in \mathbb{N}$ with $1 \leq i \leq n$ it holds either $\pi_i \subseteq T_1 \setminus \iota_{(T_1,T_2)}(N)$, $\pi_i \subseteq T_2 \setminus \iota_{(T_1,T_2)}(N)$, $\pi_i \subseteq \iota_{(T_1,T_2)}(N)$, or $\pi_i \subseteq T \setminus (T_1 \cup T_2)$.*

Fig. 5 Illustrates the partitioning of an exemplary path of model $(a)$ of our example with respect to correspondences $C1$ and $C2$. As mentioned before, transition 1 is a

non-interleaving transition related to correspondence $C1$. Transition $4$ is in the set of interleaving transitions of both correspondences. All other transitions on the highlighted path do not relate to any of the correspondences.

When comparing the partitioning of paths induced by two correspondences between two WF-nets, certain subpaths have to be neglected. That is, subpaths that represent a detour of a transition that is part of a correspondence are identified and removed from the net. Apparently, this reduction has to happen solely in case there is another transition of the correspondence that might be enabled concurrently. We illustrate the need for this kind of preprocessing with Fig. 6. It shows an excerpt of model $(b)$ of our example. Assume that we investigate correspondences $C3$ and $C4$. Then, a path comprising transitions $H$ and $I$ would suggest that a non-interleaving transition related to $C4$ (transition $J$) can occur without any occurrence of an interleaving transition of both correspondences (transitions $E$, $F$, and $G$). Hence, the subpath comprising transitions $H$ and $I$ is removed by the preprocessing. Note that the preprocessing uses the concurrency relation and the exclusiveness relation, which can be derived from the net structure as discussed in Section 4.1.

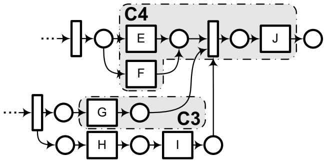


**Fig. 6.** Excerpt of model $(b)$

**Definition 8 (Preprocessing).** *Let* $N = (P, T, F)$ *be a WF-net and* $T_1, T_2 \subseteq T$ *two disjoint sets of transitions. Let* $X_{pp} \subseteq (X \setminus (T_1 \cup T_2))$ *contain all nodes* $x$ *for which there is a transition* $t_1 \in T_1 \cup T_2$*, such that* $x || t_1$ *and for all* $t_2 \in T_1 \cup T_2$ *with* $t_1 || t_2$ *it holds either* $x || t_2$ *or* $x + t_2$*. The* preprocessed *WF-net for* $N$ *is a subnet* $N' = (P', T', F')$ *with* $P' = P \setminus X_{pp}$ *and* $T' = T \setminus X_{pp}$*.*

Once two WF-nets are preprocessed with respect to a pair of correspondences, their path consistency is assessed. Loosely spoken, path consistency implies that both nets show equal partitionings of paths from the initial to the final place regarding the correspondences when transitions not related to the correspondences are neglected. Similar to the projection for a trace (cf., Section 3.3), we define the projection of path as follows. Given a WF-net $N = (P, T, F)$, a set of transitions $H \subseteq T$, and a path $\pi \in \mathcal{P}_N^T$, the set $H_{\pi|j} = \{(x, t) \in \pi \mid x < j \land t \in H\}$ denotes the containment of transitions of $H$ in $\pi$ up to index $j \in \mathbb{N}$. Then, the projection $\tau_H(\pi)$ for a path $\pi \in \mathcal{P}_N^T$ of length $n$ induced by $H$ is defined as $\tau_H(\pi) = \bigcup_{i=0}^{|H_{\pi|n}|} (i, t)$ with $t \in H$, such that $\exists\, j \in \mathbb{N}\; [\, (j, t) \in \pi \land i = |H_{\pi|j}|\, ]$.

**Definition 9 (Path Consistency of Correspondences).** *Let* $N = (P, T, F)$ *and* $N' = (P', T', F')$ *be two WF-nets preprocessed with respect to two correspondences* $C_1 = (T_1, T_3)$ *and* $C_2 = (T_2, T_4)$*.*

- $C_1$ *and* $C_2$ *are* path consistent *from* $N$ *to* $N'$*, iff for any path of transitions* $\pi \in \mathcal{P}_N^T$*, there is a path* $\pi' \in \mathcal{P}_{N'}^{T'}$*, such that for the partitioned projections* $\rho_{(T_1,T_2)}(\tau_{(T_1 \cup T_2)}(\pi)) = \pi_1, \ldots, \pi_n$ *and* $\rho_{(T_3,T_4)}(\tau_{(T_3 \cup T_4)}(\pi')) = \pi'_1, \ldots, \pi'_m$ *it holds* $n = m$ *and for all* $i \in \mathbb{N}$ *with* $0 \leq i \leq n$*:*
  - $\pi_i \subseteq (T_1 \setminus \iota_{(T_1,T_2)}(N)) \Rightarrow \pi'_i \subseteq (T_3 \setminus \iota_{(T_3,T_4)}(N'))$.
  - $\pi_i \subseteq (T_2 \setminus \iota_{(T_1,T_2)}(N)) \Rightarrow \pi'_i \subseteq (T_4 \setminus \iota_{(T_3,T_4)}(N'))$.

  - $\pi_i \subseteq \iota_{(T_1,T_2)}(N) \Rightarrow \pi'_i \subseteq \iota_{(T_3,T_4)}(N')$.
- ○ *$C_1$ and $C_2$ are* path consistent*, iff they are path consistent in either direction.*

For our example setting in Fig. 1 and correspondences $C1$ and $C2$, we see that both models are path consistent. All paths from the initial to the final place in both models shows the same (projected) partitionings, i.e., a non-interleaving transition related to $C1$ is followed by an interleaving transition of both correspondences. In contrast, correspondences $C3$ and $C4$ are not path consistent. Even if both nets are preprocessed, for instance, model $(a)$ contains a path in which non-interleaving transitions related to correspondence $C3$ (transitions 5 and 6) are followed by a non-interleaving transition related to $C2$ (transition 7). Such a path does not exit in model $(b)$ as transitions $E$, $F$, and $G$ are interleaving.

### 4.3 Reasoning on Behaviour Compatibility

We illustrated the dependency between path consistency of a pair of correspondences and their behaviour compatibility using our example. In fact, we show that both notions coincide for sound free-choice WF-nets, see [6] for the proofs.

**Theorem 2.** *Let $N = (P, T, F)$ and $N' = (P', T', F')$ be two preprocessed sound free-choice WF-nets, and $C_1 = (T_1, T_3)$, $C_2 = (T_2, T_4)$ two correspondences. Then, path consistency and projection compatibility of $C_1$ and $C_2$ coincide.*

Based thereon, behaviour compatibility of correspondences between sound free-choice WF-nets can be decided efficiently.

**Corollary 1.** *The following problem can be solved in $O(n^3)$ time with $n$ as the maximum of the number of nodes of both nets.*
*For two correspondences between two sound free-choice WF-nets, to decide projection and protocol compatibility.*

## 5 Evaluation

We evaluated our techniques for deciding behaviour compatibility, by applying them to a collection of similar model pairs, between which correspondences were already identified. For both notions of behaviour compatibility, we first identified incompatibilities for all pairs of models with respect to their correspondences. Second, we investigated the resulting incompatibilities, to determine whether they represent information that is useful to the designer.

The collection consisted of 10 pairs that were taken from Dutch municipalities. Each of these pairs represents a standard process [20] and an implementation of this standard process by a municipality. Each process model from the collection has, on average, 17.9 nodes, with a minimum of 11 nodes and a maximum of 69 nodes for a single process model. The average number of arcs pointing into or out of a single node is 1.2. In total there were 190 correspondences between the model pairs, 31 of which were complex. All models were available as (or could be transformed into) free-choice WF-nets. In addition, we verified that all models are sound, such that the structural characterisation of

**Table 1.** Overview on compatible and incompatible correspondences

| Type of Comp. | Type of Correspondence Pair | Compatible | Incompatible |
|---|---|---|---|
| Projection Comp. | Elementary | 83% (1732) | 17% (354) |
| | Complex | 30% (115) | 70% (274) |
| Protocol Comp. | Elementary | 38% (86) | 62% (143) |
| | Complex | 8% (7) | 92% (86) |

behaviour compatibility as introduced in Section 4 could be leveraged. With our implementation of this approach, the analysis of both, projection and protocol compatibility, was done in milliseconds for most model pairs. For two pairs of models, it took up to several seconds to decide compatibility.

**Projection Compatibility.** For all 190 correspondences in total 2475 combinations of correspondences were to be investigated for projection compatibility. Table 1 shows the number of projection compatible and incompatible pairs for the correspondences. We say that a pair of correspondences is elementary, if both correspondences are elementary; if one is complex, we say that the pair is complex. The table shows that most elementary correspondences are projection compatible, while most complex correspondences are not. This result is not surprising, because complex correspondences are more complicated than elementary correspondences and, therefore, it is harder to make them compatible.

As a second step, we randomly selected 25 elementary and 25 complex incompatibilities to investigate whether they represented information that is useful to the designer. This was indeed the case. However, when studying these incompatibilites in detail, we established that there existed overlap between them in the sense that they could be traced back to the same cause for incompatiblity. Specifically, there were 26 pairs that had overlap with another pair (of the 26). If we considered each correspondence pair only once, there were only 8 cases of incompatibility; the 'common' pairs caused 3.25 incompatibilities on average. This kind of redundancy in the information that is presented to the designer is undesirable and leads to the conclusion that incompatibilities can be presented to the designer in a more compact manner.

**Protocol Compatibility.** For all pairs of models, we also derived the encapsulated models in order to assess protocol compatibility. To this end, we removed solely transitions representing activities that are not part of any correspondence and neglected additional NOP transitions realising the splitting and merging of control flow. However, for four out of our 10 pairs of models, we observed that at least for one model encapsulation led to a net that could not be normalised into a WF-net. In these cases, encapsulation led to a disconnect of the initial and the final place, such that both places were no longer connected by any path. As these models describe processes that are bound to failure (they cannot complete properly), they could not be investigated any further. For the remaining four model pairs, the normalised encapsulated nets were sound, such that our structural characterisation of behaviour compatibility could be exploited. As illustrated in Table 1, the amount of compatible correspondences is much lower than for the case of projection compatibility. This is mainly due to activities that have been introduced as intermediate steps when implementing the standard process. Apparently, such deviations are not in line with protocol compatibility. Due to the freedom of the municipalities to deviate from the reference process in such a way, the notion of projection compatibility seems to be more appropriate than protocol compatibility for this use case.

## 6 Related Work

Our work is related to three streams of research, *matching of process models*, *model specialisation*, and *process model similarity*.

In order to assess behaviour consistency, we postulate the existence of correspondences between activities of two process models. In some use cases, these correspondences are given implicitly, e.g., when deriving a custom process model from a reference model. Still, other use cases might require the explicit definition of correspondences, such that automatic support for suggesting correspondences is needed. To this end, techniques based on structural analysis and natural language processing have been proposed in order to identify correspondences between single activities [21,22]. Recently, the ICoP framework has been introduced, which aims also at the detection of complex correspondences [23]. In addition, techniques known from the field of schema and ontology matching [13,24] can be applied to detect correspondences between process model elements.

A behavioural model can be specialised by *refinement* and *extension* [5]. Refinement refers to the definition of an activity or a set thereof in more detail. Extension, in turn, refers to the act of adding new activities. Both transformations might or might not preserve one of the well-known behaviour equivalences, see [25]. Behaviour consistent refinements have been investigated in detail for many formal models, such as process algebras and Petri nets [25,26,27,28]. See [29] for a thorough survey on Petri net refinements. Obviously, the work on model refinement and extension has a different focus than our work. We target at an assessment of correspondences between models for which the concrete specialisation relation is not known. Still, transformations that preserve the introduced notions of projection and protocol compatibility need to be investigated. For the existing notions of behaviour inheritance, projection and protocol inheritance, a set of four inheritance preserving model transformations has been presented in [4].

Behaviour compatibility is a boolean criterion based on a behaviour equivalence. Process models that are related by correspondences might also be analysed regarding their behavioural similarity. Recently, the question of how to quantify behavioural similarity has received much attention [30]. Process similarity can be assessed by using behavioural abstractions [31], relating similar (sub-) traces of two models to each other [32], or quantifying the degree of state-based simulation [33]. These approaches typically focus on the complete behaviour of two models. Therefore, additional effort might be required to give diagnostic information with respect to the correspondences for a similarity value below one.

## 7 Conclusion

In this paper, we addressed the question of how to decide on the compatibility of two business process models. To this end, correspondences between both models are assumed to exist, whereas we do not impose any restrictions on the type of correspondences that can exist. In particular, there might be complex 1:n and or even n:m correspondences between activities of both models. Building upon the existing work on behaviour inheritance, we introduced the notions of projection and protocol compatibility of correspondences between process models. They guarantee that correspondences do not induce behavioural contradictions in terms of trace semantics, once activities that are not part of any correspondence are hidden or blocked. Besides the definition of these notions, our contribution is a structural characterisation of both notions for a pair of correspondences between sound free-choice WF-nets. Based thereon, behaviour compatibility is decided in $O(n^3)$ time with $n$ as the maximum of the number of nodes of both nets. As a proof of concept, we applied our technique to determine the compatibility between 10 reference process models and 10 models that implement them.

Clearly, our contribution is of relevance not only for the use case of customising reference models. The application of behaviour inheritance has been advocated to solve other problems, such as those related to *dynamic change*, *information management* [2], and *service-oriented design* [1]. Dynamic change addresses the question how to ensure behavioural consistency for running process instance once the respective process model is adapted. Information management refers to an aggregated view on multiple variants of a process model. Service-oriented design addresses the issue of designing a business process that correctly implements a service that an organisation provides to its clients, as it is specified by another (abstract) process. Our notions of behaviour compatibility allow for tackling these problems in a broader context by going beyond elementary 1:1 relations between activities when comparing model behaviour. Still, this requires further investigations on model transformations that preserve behaviour compatibility. Although a formal discussion is beyond the scope of this paper, the aforementioned four transformation rules that preserve projection (and partly protocol) inheritance [4] can be assumed to preserve our notions of behaviour compatibility as well. The presence of complex correspondences, however, opens the space for investigations on transformation rules that consider the partitioning of activities induced by such correspondences. In future work, stricter notions of behaviour equivalence, such as branching bisimulation [25], might also be applied as the grounding for behaviour compatibility. Finally, the application of our technique in a case study shows that many redundant incompatibilities are notified to the process designer. Consequently, in future work we aim at developing a technique that presents incompatibilities in a compact manner.

## References

1. Dijkman, R., Dumas, M.: Service-oriented design: a multi-viewpoint approach. IJCIS 13(4), 337–368 (2004)
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. Guth, V., Oberweis, A.: Delta-analysis of petri net based models for business processes. In: Proc. of the 3rd Int. Conf. on Applied Informatics, pp. 23–32 (1997)
4. Basten, T., Aalst, W.: Inheritance of Behavior. JLAP 47(2), 47–145 (2001)
5. Schrefl, M., Stumptner, M.: Behavior-consistent specialization of object life cycles. ACM Trans. Softw. Eng. Methodol. 11(1), 92–148 (2002)
6. Weidlich, M., Dijkman, R., Weske, M.: Deciding Behaviour Compatibility of Complex Correspondences between Process Models. Technical report 11-2010, Hasso Plattner Institute, http://bpt.hpi.uni-potsdam.de/pub/Public/MatthiasWeidlich/bc_r.pdf
7. Aalst, W.: The application of Petri nets to workflow management. Journal of Circuits, Systems, and Computers 8(1), 21–66 (1998)
8. Lohmann, N.: A feature-complete Petri net semantics for WS-BPEL 2.0. In: Dumas, M., Heckel, R. (eds.) WS-FM 2007. LNCS, vol. 4937, pp. 77–91. Springer, Heidelberg (2008)
9. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Information & Software Technology 50(12), 1281–1294 (2008)
10. Eshuis, R., Wieringa, R.: Tool support for verifying UML activity diagrams. IEEE Trans. Software Eng. 30(7), 437–447 (2004)
11. Desel, J., Esparza, J.: Free Choice Petri Nets. Cambridge University Press, Cambridge (1995)
12. Aalst, W.: Verification of workflow nets. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 407–426. Springer, Heidelberg (1997)
13. Euzenat, J., Shvaiko, P.: Ontology Matching. Springer, Heidelberg (2007)
14. Hack, M.: Decidability Questions for Petri Nets. PhD thesis, M.I.T. (1976)
15. Valmari, A.: The state explosion problem. In: Reisig, W., Rozenberg, G. (eds.) APN 1998. LNCS, vol. 1491, pp. 429–528. Springer, Heidelberg (1998)
16. Kiepuszewski, B., Hofstede, A., Aalst, W.: Fundamentals of control flow in workflows. Acta Inf. 39(3), 143–209 (2003)
17. Aalst, W.: Workflow verification: Finding control-flow errors using petri-net-based techniques. In: van der Aalst, W.M.P., Desel, J., Oberweis, A. (eds.) BPM 2000. LNCS, vol. 1806, pp. 161–183. Springer, Heidelberg (2000)
18. Kovalyov, A., Esparza, J.: A polynomial algorithm to compute the concurrency relation of free-choice signal transition graphs. In: WODES. The Institution of Electrical Engineers, pp. 1–6 (1996)
19. Weidlich, M., Mendling, J., Weske, M.: Efficient Consistency Measurement based on Behavioural Profiles of Process Models. IEEE Trans. on Software Engineering (2010) (to appear)
20. Documentair structuurplan, http://model-dsp.nl/ (accessed: February 20, 2009)
21. Nejati, S., Sabetzadeh, M., Chechik, M., Easterbrook, S.M., Zave, P.: Matching and merging of statecharts specifications. In: ICSE, pp. 54–64. IEEE CS (2007)
22. Dijkman, R., Dumas, M., García-Bañuelos, L., Käärik, R.: Aligning business process models. In: EDOC, pp. 45–53 (2009)
23. 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)
24. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB Journal 10(4), 334–350 (2001)
25. van Glabbeek, R.J., Goltz, U.: Refinement of actions and equivalence notions for concurrent systems. Acta Inf. 37(4/5), 229–327 (2001)
26. Aceto, L., Hennessy, M.: Adding action refinement to a finite process algebra. Inf. Comput. 115(2), 179–247 (1994)

27. Quartel, D., Pires, L.F., van Sinderen, M.: On architectural support for behavior refinement in distributed systems design. Journal of Integrated Design and Process Science 6(1), 1–30 (2002)
28. Vogler, W.: Behaviour preserving refinement of petri nets. In: Tinhofer, G., Schmidt, G. (eds.) WG 1986. LNCS, vol. 246, pp. 82–93. Springer, Heidelberg (1987)
29. 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)
30. Dumas, M., García-Bañuelos, L., Dijkman, R.M.: Similarity search of business process models. IEEE Data Eng. Bull. 32(3), 23–28 (2009)
31. van Dongen, B.F., Dijkman, R.M., Mendling, J.: Measuring Similarity between Business Process Models. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 450–464. Springer, Heidelberg (2008)
32. Wombacher, A.: Evaluation of technical measures for workflow similarity based on a pilot study. In: Meersman, R., Tari, Z. (eds.) OTM 2006. LNCS, vol. 4275, pp. 255–272. Springer, Heidelberg (2006)
33. Sokolsky, O., Kannan, S., Lee, I.: Simulation-based graph similarity. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 426–440. Springer, Heidelberg (2006)

# Correctness Ensuring Process Configuration: An Approach Based on Partner Synthesis

Wil van der Aalst[1,3], Niels Lohmann[1,2], Marcello La Rosa[3], and Jingxin Xu[3]

[1] Eindhoven University of Technology, The Netherlands
w.m.p.v.d.aalst@tue.nl
[2] Universität Rostock, Germany
niels.lohmann@uni-rostock.de
[3] Queensland University of Technology, Australia
m.larosa@qut.edu.au, jingxin.xu@connect.qut.edu.au

**Abstract.** A configurable process model describes a family of similar process models in a given domain. Such a model can be configured to obtain a *specific* process model that is subsequently used to handle individual cases, for instance, to process customer orders. *Process configuration is notoriously difficult as there may be all kinds of interdependencies between configuration decisions.* In fact, an incorrect configuration may lead to behavioral issues such as deadlocks and livelocks. To address this problem, we present a novel verification approach inspired by the "operating guidelines" used for partner synthesis. We view the configuration process as an external service, and compute a characterization of all such services which meet particular requirements using the notion of *configuration guideline*. As a result, we can characterize all feasible configurations (i. e., configurations without behavioral problems) at design time, instead of repeatedly checking each individual configuration while configuring a process model.

**Keywords:** Configurable process model, operating guideline, Petri nets.

## 1 Introduction and Background

Although large organizations support their processes using a wide variety of process-aware information systems, the majority of business processes are still not directly driven by process models. Despite the success of Business Process Management (BPM) thinking in organizations, Workflow Management (WfM) systems — today often referred to as *BPM systems* — are not widely used. One of the main problems of BPM technology is the "lack of content", that is, providing just a generic infrastructure to build process-aware information systems is insufficient as organizations need to support specific processes. Organizations want to have "out-of-the-box" support for standard processes and are only willing to design and develop system support for organization-specific processes. Yet most BPM systems expect users to model basic processes from scratch. Enterprise Resource Planning (ERP) systems such as SAP and Oracle, on the other hand, focus on the support of these common processes. Although all ERP systems have workflow engines comparable to the engines of BPM systems, the majority of processes are not supported by software which is driven by models. For example, most of

R. Hull, J. Mendling, and S. Tai (Eds.): BPM 2010, LNCS 6336, pp. 95–111, 2010.

SAP's functionality is not grounded in their workflow component, but hard-coded in application software. ERP vendors try to capture "best practices" in dedicated applications designed for a particular purpose. Such systems can be configured by setting parameters. System configuration can be a time consuming and complex process. Moreover, configuration parameters are exposed as "switches in the application software", thus making it difficult to see the intricate dependencies among certain settings.

A model-driven process-oriented approach toward supporting business processes has all kinds of benefits ranging from improved analysis possibilities (verification, simulation, etc.) and better insights, to maintainability and ability to rapidly develop organization-specific solutions. Although obvious, this approach has not been adopted thus far, because BPM vendors have failed to provide content and ERP vendors suffer from the "Law of the handicap of a head start". ERP vendors manage to effectively build data-centric solutions to support particular tasks. However, the complexity and large installed base of their products makes it impossible to refactor their software and make it process-centric.

Based on the limitations of existing BPM and ERP systems, we propose to use *configurable process models*. A configurable process model represents a *family of process models*, that is, a model that through configuration can be customized for a particular setting. Configuration is achieved by *hiding* (i. e., bypassing) or *blocking* (i. e., inhibiting) certain fragments of the configurable process model [12]. In this way, the desired behavior is selected. From the viewpoint of generic BPM software, configurable process models can be seen as a mechanism to add content to these systems. By developing comprehensive collections of configurable models, particular domains can be supported. From the viewpoint of ERP software, configurable process models can be seen as a means to make these systems more process-centric, although in the latter case quite some refactoring is needed as processes are hidden in table structures and application code.

Various configurable languages have been proposed as extensions of existing languages (e. g., C-EPCs [22], C-iEPCs [17], C-WF-nets [3], C-SAP, C-BPEL) but few are actually supported by enactment software (e. g., C-YAWL [13]). In this paper, we are interested in models in the latter class of languages, which, unlike traditional reference models [9,8,11], are executable after they have been configured. Specifically, we focus on the *verification of configurable executable process models*. In fact, because of hiding and/or blocking selected fragments, the instances of a configured model may suffer from behavioral anomalies such as deadlocks and livelocks. This problem is exacerbated by the total number of possible configurations a model may have, and by the complex domain dependencies which may exist between various configuration options. For example, the configurable process model we constructed from the VICS documentation — an industry standard for logistics and supply chain management — comprises 50 activities. Each of these activities may be "blocked", "hidden", or "allowed", depending on the configuration requirements. This results in $3^{50} \approx 7.18e{+}23$ possible configurations. Clearly, checking the *feasibility* of each single configuration can be time consuming as this would typically require to perform state-space analysis. Moreover, characterizing the "family of correct models" for a particular configurable process model is even more difficult and time-consuming as a naive approach would require to solve an exponential number of state-space problems.

As far as we know, our earlier approach [3] is the only one focusing on the verification of configurable process models which takes into account behavioral correctness and avoids the state-space explosion problem. Other approaches either only discuss syntactical correctness related to configuration [22,10,8], or deal with behavioral correctness but run into the state-space problem [14]. In this paper, we propose a completely novel verification approach where we consider the configuration process as an "external service" and then synthesize a "most permissive partner" using the approach described by Wolf [24] and implemented in the tool Wendy [21]. This most permissive partner is closely linked to the notion of *operating guidelines* for service behavior [20]. In this paper, we define for any configurable model a so-called *configuration guideline* to characterize all correct process configurations. This approach provides the following advantages over our previous approach [3]:

- We provide a *complete characterization of all possible configurations at design time*, that is, the *configuration guideline*.
- Computation time is moved*from configuration time to design time* and results can be reused more easily.
- *No restrictions are put on the class of models* which can be analyzed. The previous approach [3] was limited to sound free-choice WF-nets. Our new approach can be applied to models which do not need to be sound, which can have complex (non-free choice) dependencies, and which can have multiple end states.

To prove the practical feasibility of this new approach, we have implemented it as a component of the toolset supporting C-YAWL.

The remainder of this paper is organized as follows. Section 2 introduces basic concepts such as open nets and weak termination. These concepts are used in Section 3 to formalize the notion of process configuration. Section 4 presents the solution approach for correctness ensuring configuration. Section 5 discusses tool support and Section 6 concludes the paper.

## 2 Business Process Models

For the formalization of the problem we use Petri nets which offer a formal model of concurrent systems. However, the same ideas can be applied to other languages (e. g., C-YAWL, C-BPEL).

**Definition 1 (Petri net).** *A* marked Petri net *is a tuple $N = (P, T, F, m_0)$ such that: $P$ and $T$ ($P \cap T = \emptyset$) are finite sets of places and transitions, respectively, $F \subseteq (P \times T) \cup (T \times P)$ is a flow relation, and $m_0 : P \to \mathbb{N}$ is an initial marking.*

A Petri net is a directed graph with two types of nodes: places and transitions, which are be connected by arcs as specified in the flow relation. If $p \in P$, $t \in T$, and $(p, t) \in F$, then place $p$ is an input place of $t$. Similarly, $(t, p) \in F$ means that $p$ is an output place of $t$.

The *marking* of a Petri net describes the distribution of tokens over places and is represented by a *multiset of places*. For example, the marking $m = [a^2, b, c^4]$ indicates

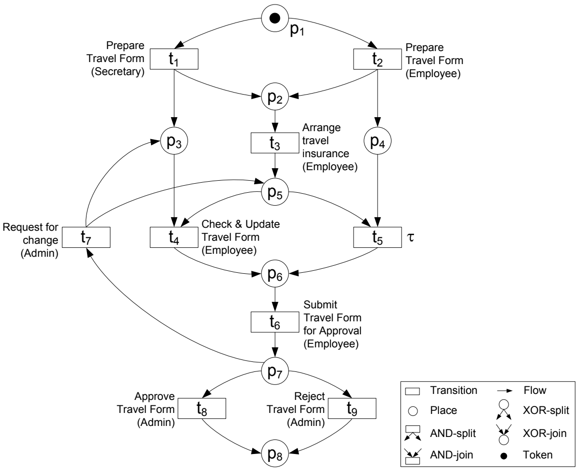


**Fig. 1.** The open net for travel request approval ($\Omega = \{[p_8]\}$)

that there are two tokens in place $a$, one token in $b$, and four tokens in $c$. Formally $m$ is a function such that $m(a) = 2$, $m(b) = 1$, and $m(c) = 4$. We use $\oplus$ to compose multisets; for instance, $[a^2, b, c^4] \oplus [a^2, b, d^2, e] = [a^4, b^2, c^4, d^2, e]$.

A transition is *enabled* and can *fire* if all its input places contain at least one token. Firing is atomic and consumes one token from each of the input places and produces one token on each of the output places. $m_0 \xrightarrow{t} m$ means that $t$ is enabled in marking $m_0$ and the firing of $t$ in $m_0$ results in marking $m$. We use $m_0 \xrightarrow{*} m$ to denote that $m$ is reachable from $m_0$, that is, there exists a (possibly empty) sequence of enabled transitions leading from $m_0$ to $m$.

For our configuration approach, we use *open nets*. Open nets extend classical Petri nets with the identification of final markings $\Omega$ and a labeling function $\ell$.

**Definition 2 (Open net).** *A tuple $N = (P, T, F, m_0, \Omega, L, \ell)$ is an* open net *if*

- $(P, T, F, m_0)$ *is a marked Petri net (called the* inner net *of $N$),*
- $\Omega \subset P \to \mathbb{N}$ *is a finite set of* final markings*,*
- $L$ *is a finite set of* labels*,*
- $\tau \notin L$ *is a label representing* invisible *(also called silent)* steps*, and*
- $\ell : T \to L \cup \{\tau\}$ *is a* labeling function.

We use *transition labels* to represent the activity corresponding to the execution of a particular transition. Moreover, if an activity appears multiple times in a model, we use the same label to identify all the occurrences of that activity. The special label $\tau$ refers to an invisible step, sometimes referred to as "silent". Invisible transitions are typically used to represent internal actions which do not mean anything at the business level (cf. the "inheritance of dynamic behavior" framework [2,7]). We use visible labels to denote activities that may be configured while in Section 4 we use these labels to synchronize two open nets.

Figure 1 shows an example open net which models a typical travel request approval. The process starts with the preparation of the travel form. This can either be done by an employee or be delegated to a secretary. In both cases, the employee personally needs to arrange the travel insurance. If the travel form has been prepared by the secretary, the employee needs to check it before submitting it for approval. An administrator can then approve or reject the request, or make a request for change. Now, the employee can update the form according to the administrator's suggestions and resubmit it. In Fig. 1 all transitions bear a visible label, except for $t_5$ which bears a $\tau$-label as it has only been added for routing purposes.

Unlike our previous approach [3] based on WF-nets [1] and hence limited to a single final place, we allow for *multiple final markings* here. Good runs of an open net end in a marking in set $\Omega$. Therefore, an open net is considered to be erroneous if it can reach a marking from which no final marking can be reached any more. An open net *weakly terminates* if a final marking is reachable from every reachable marking.

**Definition 3 (Weak termination).** *An open net* $N = (P, T, F, m_0, \Omega, L, \ell)$ weakly terminates *if and only if (iff) for any marking* $m$ *with* $m_0 \overset{*}{\rightarrow} m$ *there exists a final marking* $m_f \in \Omega$ *such that* $m \overset{*}{\rightarrow} m_f$.

The net in Fig. 1 is weakly terminating. Weak termination is a weaker notion than soundness, as it does not require transitions to be quasi-live [1]. This correctness notion is more suitable as parts of a correctly configured net may be left dead intentionally.

## 3 Process Model Configuration

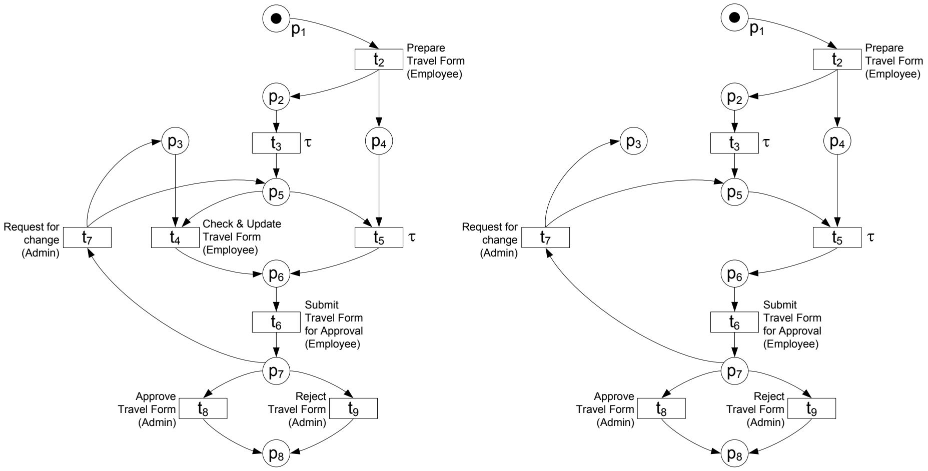
We use open nets to model configurable process models. An open net can be configured by blocking or hiding transitions which bear a visible label. Blocking a transition means that the corresponding activity is no longer available and none of the paths with that transition cannot be taken any more. Hiding a transition means that the corresponding activity is bypassed, but paths with that transition can still be taken. If a transition is neither blocked nor hidden, we say it is allowed, meaning it remains in the model. Configuration is achieved by setting visible labels to *allow*, *hide* or *block*.

**Definition 4 (Open net configuration).** *Let* $N$ *be an open net with label set* $L$. *A mapping* $C_N : L \rightarrow \{allow, hide, block\}$ *is a* configuration for $N$. *We define:* $A_N^C = \{t \in T \mid \ell(t) \neq \tau \wedge C_N(\ell(t)) = allow\}$, $H_N^C = \{t \in T \mid \ell(t) = \tau \vee C_N(\ell(t)) = hide\}$, *and* $B_N^C = \{t \in T \mid \ell(t) \neq \tau \wedge C_N(\ell(t)) = block\}$.

An open net configuration implicitly defines an open net, called *configured net*, where the blocked transitions are removed and the hidden transitions are given a $\tau$-label.

**Definition 5 (Configured net).** *Let* $N = (P, T, F, m_0, \Omega, L, \ell)$ *be an open net and* $C_N$ *a configuration of* $N$. *The resulting* configured net $\beta_N^C = (P, T^C, F^C, m_0, \Omega, L, \ell^C)$ *is defined as follows:* $T^C = T \setminus (B_N^C)$, $F^C = F \cap ((P \cup T_C) \times (P \cup T_C))$, *and* $\ell^C(t) = \ell(t)$ *for* $t \in A_N^C$ *and* $\ell^C(t) = \tau$ *for* $t \in H_N^C$.

As an example, Fig. 2(a) shows the configured net derived from the open net in Fig. 1 and the configuration $C_N$(*Prepare Travel Form* (*Secretary*)) = *block* (to allow only



(a) A weakly terminating configured net

(b) An incorrectly configured net

**Fig. 2.** Two possible configured nets based on the model in Fig. 1

employees to prepare travel forms), $C_N$(*Arrange Travel Insurance* (*Employee*)) = *hide* (to skip arranging the travel insurance), and $C_N(x)$ = *allow* for all other labels $x$.

Typically, configurable process models cannot be freely configured, because the use of hiding and blocking has to comply with the application domain in which the model has been constructed. For instance, in the travel request example we cannot hide the labels of both $t_1$ and $t_2$, because all the other activities depend on the preparation of the travel form, nor block the label of $t_8$, because there must be an option to approve the travel request. The link between configurable process models and domain decisions was explored in [18] and can be incorporated easily (see Sect. 6)

A configured net may have disconnected nodes and some parts may be dead (i. e., can never become active). Such parts can easily be removed. However, as we impose no requirements on the structure of configurable models, these disconnected or dead parts are irrelevant with respect to weak termination. For example, if we block the label of $t_2$ in Fig. 1, transition $t_5$ becomes dead as it cannot be enabled any more, and hence can also be removed without causing any behavioral issues. Nonetheless, not every configuration of an open net results in a weakly terminating configured net. For example, by blocking the label of $t_4$ in the configured net of Fig. 2(a), we obtain the configured net in Fig. 2(b). This net is not weakly terminating because after firing $t_7$ tokens will get stuck in $p_3$ (as this place does not have any successor) and in $p_5$ (as $t_5$ can no longer fire).

Blocking can cause behavioral anomalies such as the deadlock in Fig. 2(b). However, hiding cannot cause such issues, because it merely changes the labels of an open net. Hence, we shall focus on blocking rather than hiding. In this paper we are interested in all configurations which yield weakly terminating configured nets. We use the term *feasibility* to refer to such configured nets.

**Definition 6 (Feasible configuration).** *Let $N$ be an open net and $C_N$ a configuration of $N$. $C_N$ is* feasible *iff the configured net $\beta_N^C$ weakly terminates.*

Given a configurable process model $N$, we are interested in the following two questions: i) Is a *particular* configuration $C_N$ feasible? ii) How to characterize the set of *all* feasible configurations?

The remainder of this paper is devoted to a new verification approach answering these questions. This approach extends the work in [3] in two directions: (i) it imposes *no unnecessary requirements* on the configurable process model (allowing for non-free-choice nets and nets with multiple end places/markings), and (ii) it checks a *weaker correctness* notion (i. e., weak termination instead of soundness). For instance, the net in Fig. 1 is not free-choice because $t_4$ and $t_5$ share an input place, but their sets of input places are not identical. The non-free-choice construct is needed to model that after firing $t_1$ or $t_7$, $t_5$ cannot be fired, and similarly, after firing $t_2$, $t_4$ cannot be fired.

## 4 Correctness Ensuring Configuration

To address the two main questions posed in the previous section, we could use a direct approach by enumerating all possible configurations and simply checking whether each of the configured nets $\beta_N^C$ weakly terminates. As indicated before, the number of possible configurations is exponential in the number of configurable activities. Moreover, most techniques for checking weak termination typically require the construction of the state space. Hence, traditional approaches are computationally expensive and do not yield a useful characterization of the set of all feasible configuration. Consequently, we propose a completely different approach using the synthesis technique described in [24]. *The core idea is to see the configuration as an "external service" and then synthesize a "most permissive partner".* This most permissive partner represents all possible "external configuration services" which yield a feasible configuration. The idea is closely linked to the notion of *operating guidelines* for service behavior [20]. An operating guideline is a finite representation of all possible partners. Similarly, our *configuration guideline characterizes all feasible process configurations*. This configuration guideline can also be used to *efficiently check the feasibility of a particular configuration without exploring the state space of the configured net*. Our approach consists of three steps:

1. Transform the configurable process model $N$ into a *configuration interface* $N^{CI}$.
2. Synthesize the "most permissive partner" (our *configuration guideline*) $Q^{C_N}$ for the configuration interface $N^{CI}$.
3. Study the composition of $N^{CI}$ with $Q^{C_N}$.

We first introduce the notion of *composition*. Open nets can be composed by synchronizing transitions according to their visible labels. In the resulting net, all transitions bear a $\tau$-label and labeled transitions without counterpart in the other net disappear.

**Definition 7 (Composition).** *For $i \in \{1, 2\}$, let $N_i = (P_i, T_i, F_i, m_{0_i}, \Omega_i, L_i, \ell_i)$ be open nets. $N_1$ and $N_2$ are* composable *iff the inner nets of $N_1$ and $N_2$ are pairwise disjoint. The* composition of two composable open nets *is the open net $N_1 \oplus N_2 = (P, T, F, m_0, \Omega, L, \ell)$ with:*

- $P = P_1 \cup P_2$,
- $T = \{t \in T_1 \cup T_2 \mid \ell(t) = \tau\} \cup \{(t_1, t_2) \in T_1 \times T_2 \mid \ell(t_1) = \ell(t_2) \neq \tau\}$,
- $F = (F_1 \cup F_2) \cap ((P \times T) \cup (T \times P)) \cup \{(p, (t_1, t_2)) \in P \times T \mid (p, t_1) \in F_1 \vee (p, t_2) \in F_2\} \cup \{((t_1, t_2), p) \in T \times P \mid (t_1, p) \in F_1 \vee (t_2, p) \in F_2\}$,
- $m_0 = m_{0_1} \oplus m_{0_2}$, $\quad \Omega = \{m_1 \oplus m_2 \mid m_1 \in \Omega_1 \wedge m_2 \in \Omega_2\}$,
- $L = \emptyset$, *and* $\ell(t) = \tau$ *for* $t \in T$.

Composition can limit the behavior of each original net; for instance, transitions may no longer be available or may be blocked by one of the two original nets. Hence, it is possible that $N_1$ and $N_2$ are weakly terminating, but $N_1 \oplus N_2$ is not. Similarly, $N_1 \oplus N_2$ may be weakly terminating, but $N_1$ and $N_2$ are not. The labels of the two open nets in Def. 7 serve now a different purpose: they are not used for configuration, but to synchronize the two nets as described in [24].

With the notions of composition and weak termination, we define the concept of *controllability*, which we need to reason about the existence of feasible configurations.

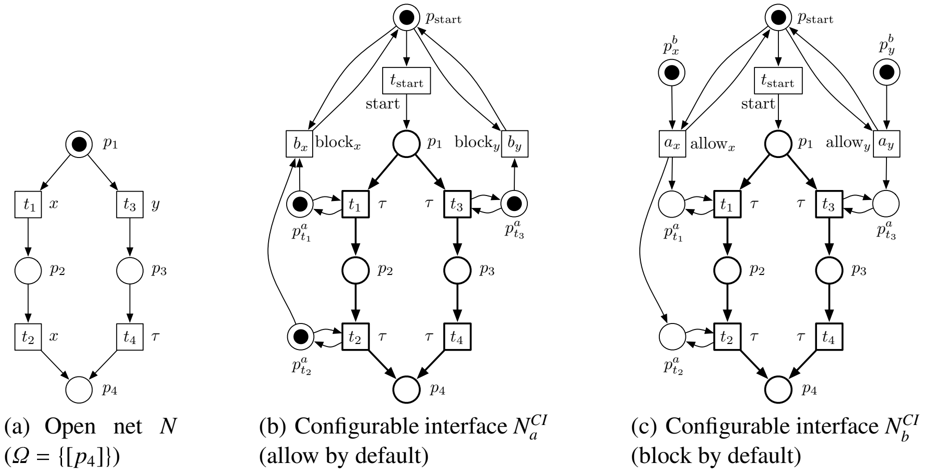
**Definition 8 (Controllability).** *An open net $N$ is* controllable *iff there exists an open net $N'$ (called* partner*) such that $N \oplus N'$ is weakly terminating.*

In [24], Wolf presents an algorithm to check controllability: if an open net is controllable, this algorithm can synthesize a partner.

After these preliminaries, we define the notion of a *configuration interface*. One of the objectives of this paper was to characterize the set of all feasible configurations by synthesizing a "most permissive partner". To do this, we transform a configurable process model (i. e., an open net $N$) into an open net $N^{CI}$, called the configuration interface, which can communicate with services which configure the original model. In fact, we shall provide two configuration interfaces: one where everything is *allowed by default* and the external configuration service can block labels, and the other where everything is *blocked by default* and the external configuration service can allow labels. These two interfaces allow us to configure both nets where all transitions are initially allowed (and configuration is done by blocking transitions) and nets where all transitions are initially blocked (and configuration is done by allowing transitions). In either case, the resulting open net $N^{CI}$ is controllable iff there exists a feasible configuration $C_N$ of $N$. Without loss of generality, we assume a 1-safe initial marking, that is, $m_0(p) > 0$ implies $m_0(p) = 1$. This assumption helps to simplify the configuration interface.

**Definition 9 (Configuration interface – allow by default).** *Let $N = (P, T, F, m_0, \Omega, L, \ell)$ be an open net. We define the open net with configuration interface $N^{CI}_a = (P^C, T^C, F^C, m^C_0, \Omega^C, L^C, \ell^C)$ with:*

- $T^V = \{t \in T \mid \ell(t) \neq \tau\}$,
- $P^C = P \cup \{p_{start}\} \cup \{p^a_t \mid t \in T^V\}$, $\quad T^C = T \cup \{t_{start}\} \cup \{b_x \mid x \in L\}$,
- $F^C = F \cup \{(p_{start}, t_{start})\} \cup \{(t_{start}, p) \mid p \in P \wedge m_0(p) = 1\} \cup \{(t, p^a_t) \mid t \in T^V\} \cup \{(p^a_t, t) \mid t \in T^V\} \cup \{(b_x, p_{start}) \mid x \in L\} \cup \{(p_{start}, b_x) \mid x \in L\} \cup \{(p^a_t, b_{\ell(t)}) \mid t \in T^V\}$,
- $m^C_0 = [p^1 \mid p \in \{p_{start}\} \cup \{p^a_t \mid t \in T^V\}]$,
- $\Omega^C = \{m \oplus \bigoplus_{t \in T} m^*_t \mid m \in \Omega \ \wedge \ \forall_{t \in T}\ m^*_t \in \{[\ ], [p^a_t]\}\ \}$,
- $L^C = \{start\} \cup \{block_x \mid x \in L\}$
- $\ell^C(t_{start}) = start$, $\ell^C(b_x) = block_x$ *for* $x \in L$, *and* $\ell^C(t) = \tau$ *for* $t \in T$.



(a) Open net $N$ ($\Omega = \{[p_4]\}$)

(b) Configurable interface $N^{CI}_a$ (allow by default)

(c) Configurable interface $N^{CI}_b$ (block by default)

**Fig. 3.** An example open net (a) and its two configuration interfaces (b,c)

Figure 3 illustrates the two configuration interfaces for a simple open net $N$. In both interfaces, the original net $N$ consisting of places $\{p_1, p_2, p_3, p_4\}$ and transitions $\{t_1, t_2, t_3, t_4\}$ is retained, but all transition labels are set to $\tau$. Let us focus on the configuration interface where all activities are allowed by default (Fig. 3(b)). Here transitions $b_x$ and $b_y$ are added to model the blocking of labels $x$ and $y$, respectively. Places $p^a_{t_1}$, $p^a_{t_2}$, and $p^a_{t_3}$ are also added to connect the new transitions to the existing ones, and are initially marked as all configurable transitions are allowed by default. Firing $b_x$ will block $t_1$ and $t_2$ by removing the tokens from $p^a_{t_1}$ and $p^a_{t_2}$. These two transitions are blocked at the same time as both bear the same label $x$ in $N$. Firing $b_y$ will block $t_3$. Transitions $b_x$ and $b_y$ are labeled respectively block$_x$ and block$_y$. This means that in the composition with a partner they can only fire if a corresponding transition in the partner can fire. Transition *start* has been added to ensure configuration actions take place *before* the original net is activated. In this way, we avoid "configuration on the fly". Figure 3(c) shows the construction of the configuration interface where all activities are blocked and is discussed later.

Consider now a configuration service represented as an open net $Q$. $N^{CI}_a \oplus Q$ is the composition of the original open net ($N$) extended with a configuration interface ($N^{CI}_a$), and the configuration service $Q$. First, blocking transitions such as $b_x$ and $b_y$ can fire (apart from unlabeled transitions in $Q$). Next, transition *start* fires after which blocking transitions such as $b_x$ and $b_y$ can no longer fire. Hence, only the original transitions in $N^{CI}_a$ can fire in the composition after firing *start*. The configuration service $Q$ may still execute transitions, but these cannot influence $N^{CI}_a$ any more. Hence, $Q$ represents a feasible configuration iff $N^{CI}_a$ can reach one of its final markings from any reachable marking in the composition. So $Q$ corresponds to a feasible configuration iff $N^{CI}_a \oplus Q$ is weakly terminating, that is, $Q$ is a partner of $N^{CI}_a$.

To illustrate the basic idea, we introduce the notion of a *canonical configuration partner*, that is, the representation of a configuration $C_N : L \rightarrow \{allow, hide, block\}$

in terms of an open net which synchronizes with the original model extended with a configuration interface.

**Definition 10 (Canonical configuration partner – allow by default).** *Let $N$ be an open net and let $C_N : L \to \{allow, hide, block\}$ be a configuration for $N$. $Q_a^{C_N} = (P, T, F, m_0, \Omega, L^Q, \ell)$ is the* canonical configuration partner *with:*

- $B = \{x \in L \mid C_N(x) = block\}$ *is the set of blocked labels,*
- $P = \{p_x^0 \mid x \in B\} \cup \{p_x^\omega \mid x \in B\}$, $T = \{t_x \mid x \in B\} \cup \{t_{start}\}$,
- $F = \{(p_x^0, t_x) \mid x \in B\} \cup \{(t_x, p_x^\omega) \mid x \in B\} \cup \{(p_x^\omega, t_{start}) \mid x \in B\}$,
- $m_0 = [(p_x^0)^1 \mid x \in B]$, $\Omega = \{\, [\,]\, \}$,[1]
- $L^Q = \{block_x \mid x \in B\} \cup \{start\}$, *and* $\ell(t_x) = block_x$ *for* $x \in B$, $\ell(t_{start}) = start$.

The set of labels which need to be blocked to mimic configuration $C_N$ is denoted by $B$. The canonical configuration partner $Q_a^{C_N}$ has a transition for each of these labels. These transitions may fire in any order after which the transition with label *start* fires. We observe that in the composition $N_a^{CI} \oplus Q_a^{C_N}$ first all transitions with a label in $\{\text{block}_x \mid x \in B\}$ fire in a synchronous manner, followed by the transition with label *start* (in both nets). After this, the net is configured and $Q_a^{C_N}$ plays no role in the composition $N_a^{CI} \oplus Q_a^{C_N}$ any more.

The following lemma formalizes the relation between the composition $N_a^{CI} \oplus Q_a^{C_N}$ and feasibility.

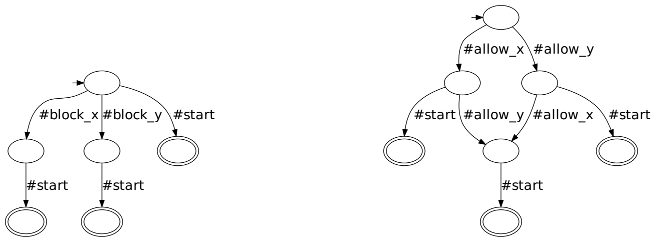
**Lemma 1.** *Let $N$ be an open net and let $C_N$ be a configuration for $N$. $C_N$ is a feasible configuration iff $N_a^{CI} \oplus Q_a^{C_N}$ is weakly terminating.*

*Proof.* ($\Rightarrow$) Let $C_N$ be a feasible configuration for $N$ and let $N_a^{CI}$ be as defined in Def. 9. Consider the composition $N_a^{CI} \oplus Q_a^{C_N}$ after the synchronization via label *start* has occurred. By construction, (1) $N_a^{CI} \oplus Q_a^{C_N}$ reached the marking $m = m_0 \oplus m_1 \oplus m_2$ such that $m_0$ is the initial marking of $N$, $m_1$ marks all places $p_t^a$ of transitions $t \in A_N^C \cup H_N^C$, and $m_2$ is the empty marking of $Q^{C_N}$. Furthermore, (2) all transitions which bear a synchronization label (i. e., $t_{\text{start}}$ and all $b_x$ transitions) and all $t \in B_N^C$ are dead in $m$ and cannot become enabled any more. From $N_a^{CI}$, construct the net $N^*$ by removing these transitions and their adjacent arcs, as well as the places $p_{\text{start}}$ and $p_t^a$ for all $t \in T^V$. The resulting net $N^*$ coincides with $\beta_N^C$ (modulo renaming). Hence, $N_a^{CI} \oplus Q_a^{C_N}$ weakly terminates.

($\Leftarrow$) Assume $N_a^{CI} \oplus Q_a^{C_N}$ weakly terminates. From $Q_a^{C_N}$, we can straightforwardly derive a configuration $C$ for $N$ in which all labels are blocked which occur in $N_a^{CI} \oplus Q_a^{C_N}$. With the same observation as before, we can conclude that $\beta_N^C$ coincides with the net $N^*$ constructed from $N_a^{CI}$ after the removal the described nodes. Hence, $\beta_N^C$ weakly terminates and $C$ is a feasible configuration for $N$. □

Lemma 1 states that checking the feasibility of a particular configuration can be reduced to checking for weak termination of the composition. However, the reason for modeling configurations as partners is that we can synthesize partners and test for the existence of feasible configurations.

[1] $[x^k \mid x \in X]$ denotes the multiset where each element of $X$ appears $k$ times. $[\,]$ denotes the empty multiset.



(a) $CG^a$ Allow by default

(b) $CG^b$ Block by default

**Fig. 4.** Two configuration guidelines characterizing all possible configurations

**Theorem 1 (Feasibility coincides with controllability).** *Let $N$ be an open net. $N_a^{CI}$ is controllable iff there exists a feasible configuration $C_N$ of $N$.*

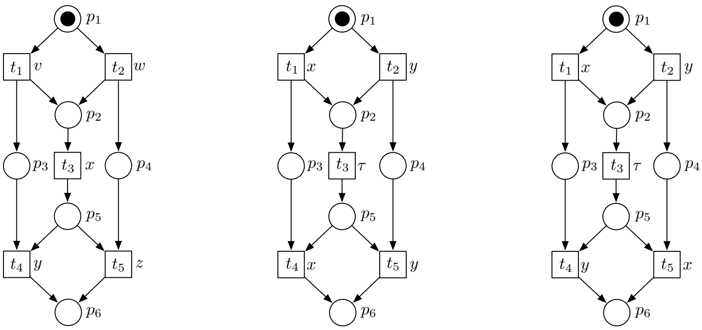
*Proof.* ($\Rightarrow$) If $N_a^{CI}$ is controllable, then there exists a partner $N'$ of $N_a^{CI}$ such that $N_a^{CI} \oplus N'$ is weakly terminating. Consider a marking $m$ of the composition reached by a run $\sigma$ from the initial marking of $N_a^{CI} \oplus N'$ to the synchronization via label *start*. Using the construction from the proof of Lemma 1, we can derive a net $N^*$ from $N_a^{CI}$ which coincides with a configured net $\beta_N^C$ for a configuration $C_N$. As $N_a^{CI} \oplus N'$ is weakly terminating, $C_N$ is feasible.

($\Leftarrow$) If $C_N$ is a feasible configuration of $N$, then by Lemma 1, $N_a^{CI} \oplus Q_a^{C_N}$ weakly terminates and by Def. 8, $N_a^{CI}$ is controllable. □

As shown in [24], it is possible to synthesize a partner which is *most-permissive*. This partner simulates any other partner and thus characterizes all possible feasible configurations. In previous papers on partner synthesis in the context of service oriented computing, the notion of an *operating guideline* was used to create a finite representation capturing all possible partners [20]. Consequently, we use the term *Configuration Guideline* (CG) to denote the most-permissive partner of a configuration interface. Fig. 4(a) shows the configuration guideline $CG^a$ for the configurable model in Fig. 3(a), computed from the configuration interface $N_a^{CI}$ in Fig. 3(b).

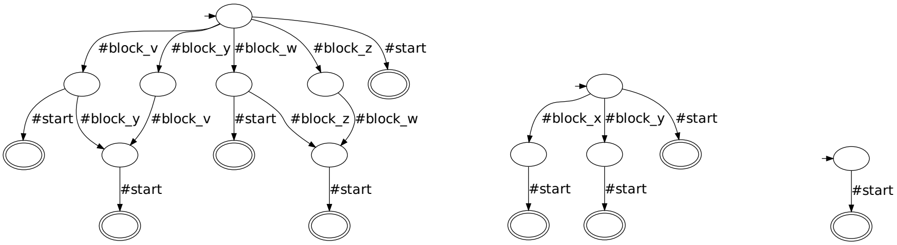
A configuration guideline is an automaton with one start state and one or more final states. *Any path in the configuration guideline starting in the initial state and ending in a final state corresponds to a feasible configuration.* The initial state in Fig. 4(a) is denoted by a small arrow and the final states are denoted by double circles. The leftmost path in Fig. 4(a) (i. e., $\langle$block$_x$, start$\rangle$), corresponds to the configuration which blocks label $x$. Path $\langle$block$_y$, start$\rangle$ corresponds to the configuration which blocks label $y$. The rightmost path (i. e., $\langle$start$\rangle$) does not block any label. The three paths capture all three feasible configurations. For example, blocking both labels is not feasible. Figure 4(a) is trivial because there are only two labels and three feasible configurations.

Thus far, we used a configuration interface that allows all configurable activities by default, that is, blocking is an explicit action of the partner. It is also possible to use a completely different starting point and initially block all activities. As this “block by default” strategy is analogous to the “allow by default” approach we discussed before, we



(a) $N_1$ (b) $N_2$ (c) $N_3$

**Fig. 5.** Three open nets ($\Omega = \{[p_6]\}$)



(a) $CG_1^a$ (b) $CG_2^a$ (c) $CG_3^a$

**Fig. 6.** The configuration guidelines (allow by default) for $N_1$ (a), $N_2$ (b) and $N_3$ (c)

refer to [5] for formal definitions and proofs. Figure 3(c) depicts the "block by default" configuration interface for the net $N_1$ (Fig. 3(a)) and Fig. 4(b) shows the respective configuration guideline.

Let us now consider a more elaborated example to see how configuration guidelines can be used to rule out unfeasible configurations. Figure 5 shows three open nets. The structures are identical, only the labels are different. For example, blocking $x$ in $N_2$ corresponds to removing both $t_1$ and $t_4$ as both transitions bear the same label, while blocking $x$ in $N_3$ corresponds to removing $t_1$ and $t_5$. For these three nets, we can construct the configuration interfaces using Def. 9 and then synthesize the configuration guidelines, as shown in Fig. 6.

Figure 6(a) reveals all feasible configurations for $N_1$ in Fig. 5(a). From the initial state in the configuration guideline $CG_1^a$, we can immediately reach a final state by following the rightmost path ⟨start⟩. This indicates that all configurations which block nothing (i. e., only allow or hide activities) are feasible. It is possible to just block $v$ (cf. path $\langle \text{block}_v, \text{start} \rangle$) or block both $v$ and $y$ (cf. paths $\langle \text{block}_v, \text{block}_y, \text{start} \rangle$ and $\langle \text{block}_y, \text{block}_v, \text{start} \rangle$). However, it is not allowed to block $y$ only, otherwise a token would deadlock in $p_3$. For the same reasons, one can block $w$ only or $w$ and $z$, but not $z$ only. Moreover, it is not possible to combine the blocking of $w$ and/or $z$ on the one hand and $v$ and/or $y$ on the other hand, otherwise no final marking can be reached. Also $x$

can never be blocked, otherwise both $v$ and $w$ would also need to be blocked (to avoid a token to deadlock in $p_2$) which is not possible. There are $3^5 = 243$ configurations for $N_1$. If we abstract from hiding as this does not influence feasibility, there remain $2^5 = 32$ possible configurations. Of these only 5 are feasible configurations which correspond to the final states in Fig. 6(a). This illustrates that the configuration guideline can indeed represent all feasible configurations in an intuitive manner.

Figure 6(b) shows the three feasible configurations for $N_2$ in Fig. 5(b). Again all final states correspond to feasible configurations. Here one can block the two leftmost transitions (labeled $x$) or the two rightmost transitions (labeled $y$), but not both.

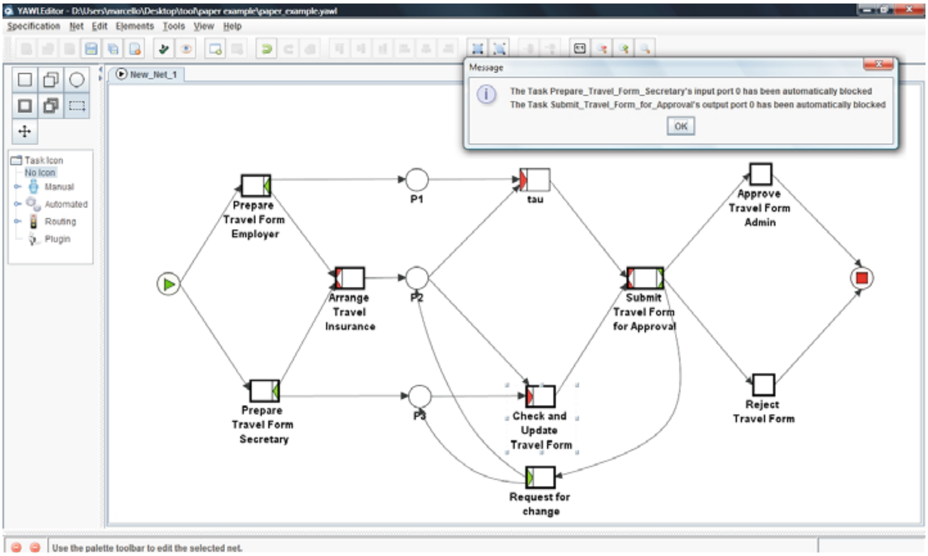
The configuration guideline in Fig. 6(c) shows that nothing can be blocked for $N_3$ (Fig. 5(c)). Blocking $x$ or $y$ will yield an unfeasible configuration as a token will get stuck in $p_4$ (when blocking $x$) or $p_3$ (when blocking $y$). If both labels are blocked, none of the transitions can fire and thus no final marking can be reached.

## 5 Tool Support

To prove the feasibility of our approach, we applied it to the configuration of C-YAWL models [13] and extended the YAWL system accordingly. YAWL is based on the well-know workflow patterns [4] and is one of the most widely used open source workflow systems [15]. For configuration we restrict ourselves to the basic control-flow patterns and do not use YAWL's cancelation sets, multiple instance tasks and OR-joins.

A C-YAWL model is a YAWL model where some tasks are annotated as *configurable*. Configuration is achieved by restricting the routing behavior of configurable tasks via the notion of *ports*. A configurable task's joining behavior is identified by one or more *inflow* ports, whereas its splitting behavior is identified by one or more *outflow* ports. The number of ports for a configurable task depends on the task's routing behavior. For example, an AND-split/join and an OR-join are each identified by a single port, whereas an XOR-split/join is identified by one port for each outgoing/incoming flow. An OR-split is identified by a port for each combination of outgoing flows. To restrict a configurable task's routing behavior, inflow ports can be hidden (thus the corresponding task will be skipped) or blocked (no control will be passed to the corresponding task via that port), whereas outflow ports can only be blocked (the outgoing paths from that task via that port are disabled). For instance, Fig. 7 shows the C-YAWL model for the travel request approval in the YAWL Editor, where configurable tasks are marked with a ticker border.

The new *YAWL Editor* can be downloaded from `www.yawlfoundation.org`. It provides a graphical interface to conveniently configure and check C-YAWL models and subsequently generate configured models. The *C-YAWL Correctness Checker* [5] which is embedded in the editor converts C-YAWL models into open nets and passes these on to the tool *Wendy* [21] to produce configuration guidelines. Wendy implements the algorithms to synthesize partners [24] and calculates operating guidelines [20]. The complexity of the partner synthesis is exponential in the size of the Petri net with the configuration interface (the reachability graph needs to be generated) and the size of the interface. However, practical experiences show that Wendy is able to analyze industrial models with up to 5 million states and to synthesize partners of about the same size [21].


YAWLEditor - D:\Users\marcello\Desktop\tool\paper example\paper_example.yawl
Specification Net Edit Elements Tools View Help
New_Net_1
Task Icon
No Icon
Manual
Automated
Routing
Plugin
Message
OK
Prepare Travel Form Employer
P1
tau
Approve Travel Form Admin
Arrange Travel Insurance
P2
Submit Travel Form for Approval
Prepare Travel Form Secretary
P3
Check and Update Travel Form
Reject Travel Form
Request for change

The Task Prepare_Travel_Form_Secretary's input port 0 has been automatically blocked
The Task Submit_Travel_Form_for_Approval's output port 0 has been automatically blocked

Use the palette toolbar to edit the selected net.

**Fig. 7.** The C-YAWL model for travel request approval

At each configuration step, the Correctness Checker scans the set of outgoing edges of the current state in the configuration guideline, and prevents users from blocking those ports not included in this set. This is done by disabling the block button for those ports. As users block a valid port, the Correctness Checker traverses the configuration guideline through the corresponding edge and updates the current state. If this is not a *consistent* state, that is, a state with an outgoing edge labeled "start", further ports need to be blocked, because the current configuration is unfeasible. In this case the component provides an *"auto complete" option*. This is achieved by traversing the shortest path from the current state to a consistent state and automatically blocking all ports in that path. After this, the component updates the current state and notifies the user with the list of ports that have been automatically blocked. For example, Fig. 7 shows that after blocking the input port of task *Check and Update Travel Form*, the component notifies the user that the input port of task *Prepare Travel Form for Approval (Secretary)* and the output port of task *Submit Travel Form for Approval* to task *Request for Change* have also been blocked. Similarly, the component maintains a consistent state in case users decide to allow a previously blocked port. In this case it traverses the shortest backward path to a consistent state and allows all ports in that path. By traversing the shortest path we ensure that the number of ports being automatically blocked or allowed is minimal.

The C-YAWL example of Fig. 7 comprises ten inflow ports and nine outflow ports. In total more than 30 million configurations are potentially possible. If we abstract from hiding we obtain 524,288 possible configurations, of which only 1,593 are feasible according to the configuration guideline. Wendy took an average of 336 seconds (on a 2.4 GHz processor with 2GB of RAM) to generate this configuration guideline which consumes 3.37 MB of disk space. Nonetheless, the shortest path computation is a simple depth-first search which is linear on the number of nodes in the configuration guideline. Thus, once the configuration guideline has been generated, the component's response time at each user interaction is instantaneous.

## 6 Conclusion

Configurable process models are a means to compactly represent families of process models. However, the verification of such models is difficult as the number of possible configurations grows exponentially in the number of configurable elements. Due to concurrency and branching structures, configuration decisions may interfere with each other and thus introduce deadlocks, livelocks and other anomalies. The verification of configurable process models is challenging and only few researchers have worked on this. Moreover, existing results impose restrictions on the structure of the configurable process model and fail to provide insights into the complex dependencies among different process model configuration decisions.

The main contribution of this paper is an innovative approach for ensuring correctness during process configuration. Using partner synthesis we compute the configuration guideline — a compact characterization of all feasible configurations, which allows us to rule out configurations that lead to behavioral issues. The approach is highly generic and imposes no constraints on the configurable process models that can be analyzed. Moreover, all computations are done at design time and not at configuration time. Thus, once the configuration guideline has been generated, the response time is instantaneous thus stimulating the practical (re-)use of configurable process models. The approach is implemented in a checker integrated in the YAWL Editor. This checker uses the Wendy tool to ensure correctness while users configure C-YAWL models.

Several interesting extensions are possible. First, the partner synthesis could be further refined using *behavioral constraints* [19] in order to rule out specific partners. This could be used to encode knowledge about a process' application domain [16] in the configuration interface. For example, domain knowledge may state that two activities cannot be blocked or allowed at the same time. Similarly, one could study techniques to identify semantic inconsistencies between control-flow and data-flow that can arise from configuration, and use behavioral constraints to encode these inconsistencies (e.g., extend the approach in [23]). Second, one could consider configuration at run-time, that is, while instances are running, configurations can be set or modified. This can be easily embedded in the current approach. Finally, one could devise more compact representations of configuration guidelines (e.g. exploiting concurrency [6]).

## 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., Basten, T.: Inheritance of Workflows: An Approach to Tackling Problems Related to Change. Theoretical Computer Science 270(1-2), 125–203 (2002)
3. van der Aalst, W.M.P., Dumas, M., Gottschalk, F., ter Hofstede, A.H.M., La Rosa, M., Mendling, J.: Preserving Correctness During Business Process Model Configuration. Formal Aspects of Computing 22(3), 459–482 (2010)

4. 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)
5. van der Aalst, W.M.P., Lohmann, N., La Rosa, M., Xu, J.: Correctness Ensuring Process Configuration: An Approach Based on Partner Synthesis (extended version). BPM Center Report BPM-10-02, BPMcenter.org (2010)
6. Badouel, E., Darondeau, P.: Theory of regions. In: Reisig, W., Rozenberg, G. (eds.) APN 1998. LNCS, vol. 1491, pp. 529–586. Springer, Heidelberg (1998)
7. Basten, T., van der Aalst, W.M.P.: Inheritance of Behavior. Journal of Logic and Algebraic Programming 47(2), 47–145 (2001)
8. Becker, J., Delfmann, P., Knackstedt, R.: Adaptive Reference Modeling: Integrating Configurative and Generic Adaptation Techniques for Information Models. In: Reference Modeling: Efficient Information Systems Design Through Reuse of Information Models, pp. 27–58. Physica-Verlag, Springer, Heidelberg (2007)
9. Curran, T., Keller, G.: SAP R/3 Business Blueprint: Understanding the Business Process Reference Model. Upper Saddle River (1997)
10. Czarnecki, K., Antkiewicz, M.: Mapping Features to Models: A Template Approach Based on Superimposed Variants. In: Glück, R., Lowry, M. (eds.) GPCE 2005. LNCS, vol. 3676, pp. 422–437. Springer, Heidelberg (2005)
11. Fettke, P., Loos, P.: Classification of Reference Models - A Methodology and its Application. Information Systems and e-Business Management 1(1), 35–53 (2003)
12. Gottschalk, F., van der Aalst, W.M.P., Jansen-Vullers, H.M.: Configurable Process Models: A Foundational Approach. In: Reference Modeling: Efficient Information Systems Design Through Reuse of Information Models, pp. 59–78. Physica-Verlag, Springer, Heidelberg (2007)
13. Gottschalk, F., van der Aalst, W.M.P., Jansen-Vullers, M.H., Rosa, M.L.: Configurable Workflow Models. Int. J. Cooperative Inf. Syst. 17(2), 177–221 (2008)
14. Hallerbach, A., Bauer, T., Reichert, M.: Guaranteeing Soundness of Configurable Process Variants in Provop. In: CEC, pp. 98–105. IEEE, Los Alamitos (2009)
15. ter Hofstede, A.H.M., van der Aalst, W.M.P., Adams, M., Russell, N.: Modern Business Process Automation: YAWL and its Support Environment. Springer, Heidelberg (2010)
16. La Rosa, M., van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M.: Questionnaire-based Variability Modeling for System Configuration. Software and Systems Modeling 8(2), 251–274 (2009)
17. La Rosa, M., Dumas, M., ter Hofstede, A.H.M., Mendling, J., Gottschalk, F.: Beyond Control-Flow: Extending Business Process Configuration to Roles and Objects. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (eds.) ER 2008. LNCS, vol. 5231, pp. 199–215. Springer, Heidelberg (2008)
18. La Rosa, M., Lux, J., Seidel, S., Dumas, M., ter Hofstede, A.H.M.: Questionnaire-driven Configuration of Reference Process Models. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 424–438. Springer, Heidelberg (2007)
19. Lohmann, N., Massuthe, P., Wolf, K.: Behavioral Constraints for Services. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 271–287. Springer, Heidelberg (2007)
20. Lohmann, N., Massuthe, P., Wolf, K.: Operating Guidelines for Finite-State Services. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 321–341. Springer, Heidelberg (2007)
21. Lohmann, N., Weinberg, D.: Wendy: A tool to synthesize partners for services. In: Lilius, J., Penczek, W. (eds.) PETRI NETS 2010. LNCS, vol. 6128, pp. 297–307. Springer, Heidelberg (2010)

22. Rosemann, M., van der Aalst, W.M.P.: A Configurable Reference Modelling Language. Information Systems 32(1), 1–23 (2007)
23. Trcka, N., van der Aalst, W.M.P., Sidorova, N.: Data-Flow Anti-Patterns: Discovering Data-Flow Errors in Workflows. In: van Eck, P., Gordijn, J., Wieringa, R. (eds.) CAiSE 2009. LNCS, vol. 5565, pp. 425–439. Springer, Heidelberg (2009)
24. Wolf, K.: Does my service have partners? In: Jensen, K., van der Aalst, W.M.P. (eds.) ToPNoC II. LNCS, vol. 5460, pp. 152–171. Springer, Heidelberg (2009)

# Impact of Granularity on Adjustment Behavior in Adaptive Reuse of Business Process Models

Oliver Holschke

Technische Universität Berlin, Fachgebiet Systemanalyse und EDV
FR 6-7, Franklinstr. 28-29, 10587 Berlin, Germany
oliver.holschke@sysedv.tu-berlin.de

**Abstract.** Business process diagrams as exteriorized forms of distributed organizational knowledge can be valuable assets when shared and reused in similar process design tasks. However, little empirical research has been conducted to shed light on the cognitive processes involved during the adaptation of retrieved process models. We hypothesize that model granularity has significant effects on human adjustment behavior irrespective of the editing distances between reuse and solution models. The results of our laboratory experiment, which is dimensioned according to real-world cases, contribute to a more specific classification of adaptation operations and their cognitive efforts, and refine the notion of process similarity. This study follows up on our former research work by amending minor flaws in the experiment setup; it now provides a comprehensive analytical apparatus for further replicated tests as the predictive power of our explorative study, regarding e.g. varied business contexts and task dimensions, remains limited.

**Keywords:** Design by Reuse, Adaptive Reuse, Process Granularity, Cognitive Heuristics, Knowledge Reuse, Modeling Assistance, Cognitive Ergonomics, Reference Process Models, Experimental Study.

## 1 Introduction

Nowadays graphical descriptions of processes are a common extension of thought in organizations, ranging from private enterprises to public administrations. Through the lens of Activity Theory [1] we can regard diagrams as "exteriorized" forms of mental processes, and as these mental processes are manifested in diagrammatic *tools*, they become more readily accessible and communicable to other people, thereafter becoming useful for social interaction.

The sharing and the reuse of various kinds of knowledge within organizations, including process diagrams, is an attractive form of organized activity because of the potential economic benefits conveyed as time-savings, qualitative improvements and economies of scale. The effective management of knowledge reuse may even be seen as the successful *exploitation* of existing diverse ideas, and poses an important ingredient for radical innovation [2, 3].

R. Hull, J. Mendling, and S. Tai (Eds.): BPM 2010, LNCS 6336, pp. 112–127, 2010.

There are many examples in system design and development that focus on reuse drawing upon various artifacts settled in different enterprise contexts [4, 5], including business process models [e.g. 6, 7].

It is important to investigate these ergonomics related aspects to determine whether the utility of visual process diagrams as language-anchored knowledge can actually prevail in process-oriented organizations and what conditions must be established to improve information diffusion and specifications of reuse-based design environments.

Former experimental setups lacked well-grounded values in dimensions essentially important to larger organizations, such as complexity of the task and placement in a specific application domain. We therefore followed up our own research work [8] and corrected some minor flaws in the experimental setup to pinpoint effects of granularity variation on design adjustments. These corrections include, among others, the sizing of the process design task, the precise dimensioning of the editing distances between reuse models and solutions, and better control of individual characteristics of the participants. We also had the possibility to base the task and reuse models on real world dimensions from an industry case study.

The rest of this article is organized as follows. Section II presents the related research work and links the question at hand. Section III introduces the concept of granularity, its quantification and application to process models. In Section IV the granularity concept and adjustment bias effects are considered in the design process under reuse and hypotheses are presented. Section V describes the experimental study conducted. Section VI summarizes the results and section VII concludes with the limitations of this study and a brief outlook toward further research activities.

## 2 Related Work

While extensive research work has been conducted on the strategies, organization and adaptation techniques of reusing process models [9, 10], still little is known about the *cognitive processes* involved when adjusting process diagrams (and diagrams in general) and what psychological effects may be involved during the adaptation process [11]. Only for specials tasks the benefits of diagrams were formally shown [12]. The adaptation phase [13] is of crucial importance as simple analogous reuse in dynamic environments can hardly be expected. We therefore focus on the phase after the decision of searching for a reusable process model has been made (and a model found), because an *insurmountable* performance gap had been perceived [2].

Substantial work has been done in the area of model-reading and the factors influencing e.g. better comprehension and/or understanding, among them *modularity*, which is highly related to the concept of granularity [14, 15]. However, model-reading is just one part in the problem-solving process; using a diagram and actually re-applying its content in *writing* a new model bears further challenges (such as making correspondences and adaptation). In [16] groups were trained with a rule-based and pattern-based approach respectively and the effect on model-writing (data model) performance was tested. While the study explores model-writing and different forms of assistance, the domain was different from process models and the help provided was not represented as a solution estimate of the same notation.

Among many known cognitive biases in problem-solving using assistant artifacts, the *anchoring* heuristic is of high interest as it has shown to have specific influences on the adjustment behavior of humans [17, 18]. We explore the anchoring heuristic in a reuse-based process design task, in which we vary the *granularity* of the reusable artifact, because our assumption is that the anchoring effect will have different effects on the designer's adjustment behavior depending on reuse model granularity.

## 3 Granularity: Concept and Quantification

### 3.1 Granularity as a Concept of Human Cognition

The term granularity has been discussed in various research areas such as Granular Computing, Cognitive Informatics, Pattern Classification, and Conceptual Modeling. Granularity is a fundamental concept in human cognition and deals with the construction, interpretation, and representation of *granules*. A granule is a clump of points (objects) drawn together by indistinguishability, similarity, proximity or functionality [19]. Granules are the result of a granulation process - the process that involves dividing some *universe* into subsets or the grouping of individual subjects into clusters. Granules can be viewed as subsets of the universe, which may be either fuzzy or crisp [19, 20]. Once granulation has been performed, it is necessary to label granules. This can be done by classification, i.e. assigning a name to a granule such that an element in the granule is an instance of the named category.

A *partition* of a universe *U* is a collection of non-empty pair-wise disjoint subsets of *U* whose union is *U*. Each subset in a partition is also called a block. In the granulated view, *partitions*, being elements of the partition of *U*, are the basic building blocks and are called elementary granules. They are the smallest nonempty subsets that can be defined, observed or measured. From elementary granules, larger granules can be constructed by taking unions of elementary granules [19]. Since partitions are nonempty, they may have a cardinality bigger than 1. The parts or blocks of the partitions are countable, but not observable because they cannot be differentiated. A conceptual visualization of different granularities of partitions $\pi_1$ and $\pi_2$ is presented in Fig. 1. The granularity of different partitions is an important characteristic of design tasks that reside on a specific level of granularity as it may affect how designs are planned and developed, and how efficiently available design artifacts may be reused.

### 3.2 Measuring Granularity and Application to Process Models

According to [19] the following function *G* is used as a measure of granularity for a partition $\pi$, which has been applied in [8]:

$$G(\pi) = \sum_{i=1}^{m} \frac{|A_i|}{|U|} \log |A_i| \qquad (1)$$

We apply the granularity concept now to process models. The granularity concept requires a perspective on the cardinality of the universe, i.e. some sort of "baseline" of the world that is perceived must be known. Any partitions that are then made will be

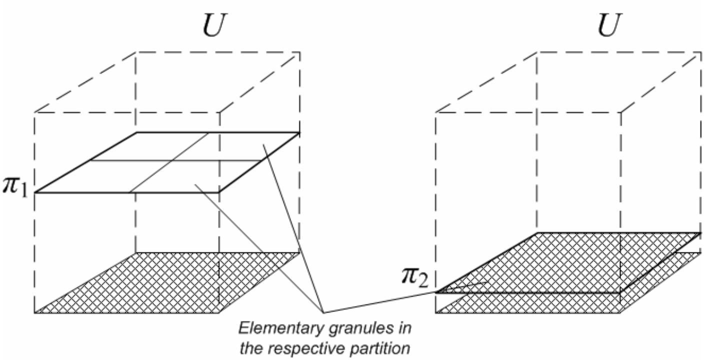


**Fig. 1.** Granularity of partition $\pi_1$ (left) and partition $\pi_2$ (right) of a universe $U$

based on that baseline and regarded in relation to it. The perception of the universe in the case of process modeling tasks to a large extent stems from the description of the specific task. Since we are constrained to a laboratory experimental setup, the textual description of the task provided by the experimenter to test subjects is used as the baseline to determine the universe cardinality. In Fig. 2 and 3 distinct granularities of two process models (high and low) are presented. It is shown how the textual description of the task can be related to a very fine baseline view represented by atomic tasks and other BPMN symbols. Based on that, two different partitions, i.e. two differently granular process models can be defined (cf. Fig. 2 and 3). The focus is on grouping atomic tasks to coarser sub-processes (though not indicated as sub- processes (though not indicated as sub-processes); the grouping of other BPMN constructs is thinkable in principal, but the abstracting relationships are not as straight-forward as it is for tasks, e.g. the grouping of several gateways to a parent gateway is harder in terms of abstracting the information. The focus on activities with regard to the granularity concept is therefore directly related to the concept of *modularity* that has been reviewed an investigated regarding effects on model understanding in [14]. Applying equation 3 to the partitions in Fig. 2 and Fig. 3 we get $G(\pi_1) = 0{,}773$ and $G(\pi_2) = 0{,}516$.

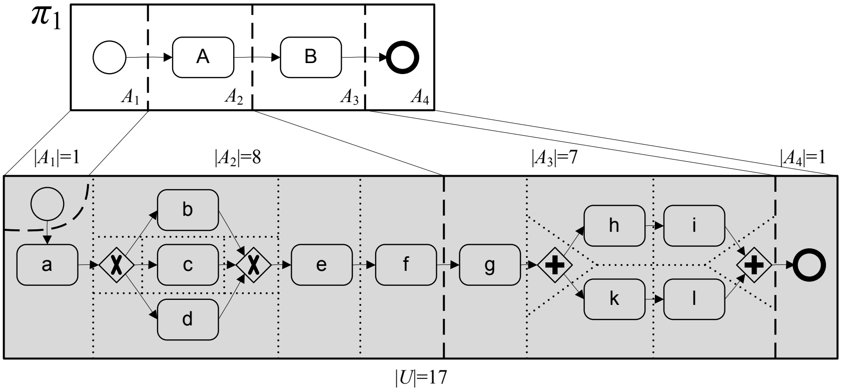


**Fig. 2.** Process model having high granularity with regard to the cardinality of the task

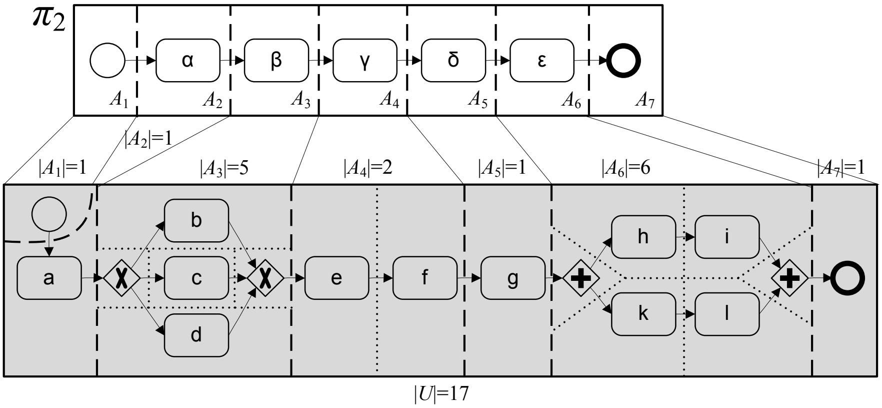


**Fig. 3.** Process model having low granularity with regard to the cardinality of the task

## 4 Granularity and Adjustment Behavior in Reuse-Based Design

Evidence from cognitive psychology suggests that making judgments about the adaptations required to meet specific application needs can be difficult, thereby adding to the challenge of achieving effective reuse of system designs. Reference [18] draws upon the *anchoring* heuristic and applies it to the domain of software design. Anchoring (or focalism) is a cognitive bias which describes a frequent human tendency to rely too heavily, or “anchor”, on one or few cues when making decisions [17]. In [18] it is shown for a reuse-based Entity-Relationship modeling task from the aviation domain, that anchoring to extraneous functionality is significant. Considering extraneous information in solution models as a severe impairment – because parsimony is violated – specific reuse-oriented organizational setups have to be seriously questioned. Assuming that, despite these explorative findings, a real benefit particularly for complex design tasks could indeed be obtained when designs are reused, we extend the former research work in two ways to gain deeper understanding. Firstly, while significant work has been done in the domain of database models and conceptual schemas, we apply the above mentioned cognitive heuristics to the *process modeling* domain, opening the discussion about how process knowledge shall be prepared to effectively diffuse within organizations. Secondly, while acknowledging that anchoring is an important cognitive bias to consider, we are interested in the exact conditions that prevail when potential reuse benefits are either enabled or impaired.

We draw upon granularity as a factor to be explored more deeply in the context of anchoring [8]. Varying granularity of a reuse artifact in a process design task corresponds in practice with differently distributed adaptation operations required to arrive at the solution. This can be seen in approaches where the “dilution” of reuse artifacts, i.e. the number of extraneous information elements, progresses with decreasing granularity. Compare for instance those initiatives capturing fine-level data for broad application domains considering many different contingencies (cf. e.g. [21]). The individual organization that reuses the artifacts will in most cases require only a fraction of the available information. All extraneous information then must be *deleted* – one specific type of adaptation. Since the share of extraneous information may be substantial, high search costs are imposed that countervail the reuse benefits.

In the case of high granular reuse models, i.e. bigger "chunks" (see [6] as an example), however, the adaptation requirements usually turn out to be *inclusion* activities (regarding the solution) rather than deletion. Information elements are too coarse to reflect the task requirements therefore the respective element has to be replaced by finer constructs, i.e. more precise elements have to be included in the reuse model. Because the inclusion of new elements is required, left out information in the reuse model (but needed) is a case of *omission* (cf. [18]). As both cases – low and high granular – could be constructed in the way that both would contain the same proportions of extraneous and omitted information (against the solution), practice suggests that provided reuse artifacts of high granularity will require more adaptive *inclusion*, and those of low granularity will require more adaptive *deletion* of elements. Implying the effects of anchoring that have been shown in other application domains, we expedite this finding and presume that the extent of the anchoring effect significantly differs depending on the type of adaptation operation being carried out on the reuse model during the design task (which relates to the reuse model granularity as explained above). Not only do we think that the difference is significant – we further assume that deleting extraneous elements imposes the significantly heavier cognitive burden, leading to the following null (directed difference) hypothesis to be falsified:

**H0a:** The percentage of required adjustments actually performed in the high granular case are *not* significantly higher than in the low granular treatment.

Further, we assume that – for the size and complexity of the provided task – the model quality of solutions created with the help of a reuse model will be superior to those created from scratch, leading to our second null (directed difference) hypothesis:

**H0b:** The quality of solution models created in the reuse cases (high and low granular) will *not* be higher compared with the model qualities achieved in the from-scratch case.

## 5 Research Design and Experimental Setup and Procedure

We applied to large extent the research framework for experiment design to evaluate conceptual modeling techniques by [22]. As the scope of our evaluation is confined to the granularity characteristic of a reusable process model and its impact on modeling performance, we controlled for the other two major factors, i.e. individual characteristics of the designers and task characteristics, as is detailed in the controlled variable section below. We adopted the laboratory-based approach, which has been employed in data modeling studies comparing user performance [e.g. 16, 23], and in recent years has been applied increasingly in studies that investigate process modeling in more detail [14, 24]. Laboratory experiments allow for precision, objectiveness, strong control, and relatively high internal validity [25]. Our complete research design is depicted in Fig. 4.

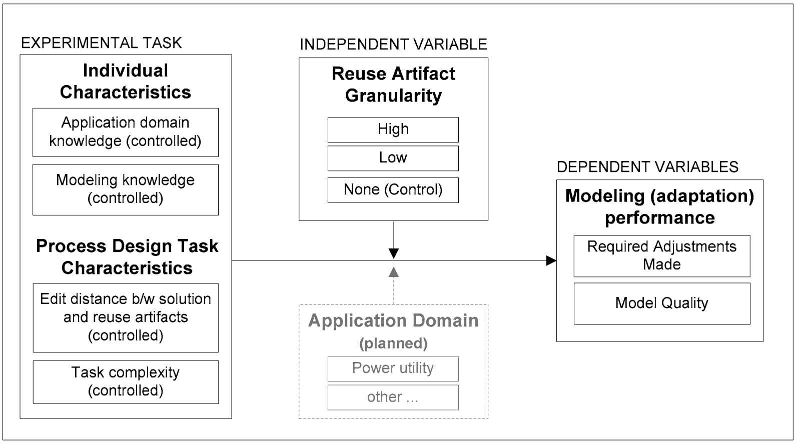

EXPERIMENTAL TASK
Individual Characteristics
Application domain knowledge (controlled)
Modeling knowledge (controlled)
Process Design Task Characteristics
Edit distance b/w solution and reuse artifacts (controlled)
Task complexity (controlled)
INDEPENDENT VARIABLE
Reuse Artifact Granularity
High
Low
None (Control)
Application Domain (planned)
Power utility
other ...
DEPENDENT VARIABLES
Modeling (adaptation) performance
Required Adjustments Made
Model Quality

**Fig. 4.** Research model

## 5.1 Independent Variable

The independent variable consists of a reusable process model at different granularity levels that is given to participants as an assistant artifact. We treat three groups with different artifacts, i.e. group 1 receives a coarse granular reuse model (high G), group 2 receives a fine granular reuse model (low G), and group 3 does not receive an assistant artifact to reuse, but has to solve the modeling task from scratch (this is the control group). Analyzing group 3 will allow us to determine whether there are any significant benefits at all when solving the task of the given complexity and difficulty by reusing a provided process model (groups 2 and 3).

The granularities of the reuse models had to be specifically constructed. Since in this study the independent variable is regarded as categorical, it was adequate to select two granularity levels sufficiently apart from another. In a pretesting phase the process models of differing granularity were shown to several individuals at the department to get feedback on the apparentness of the granularity distinction. The authors made minor adjustments to the reuse models according to the feedback and ultimately specified the granularities of the a) coarse reuse model at $G = 0.700$ (high G) and the b) fine reuse model at $G = 0.316$ (low G) respectively. Excerpts of the constructed reusable process models can be seen in Appendix A, full reuse models are available at http://sites.google.com/site/holschke/taskdescription

## 5.2 Dependent Variables

**Required Adjustments Made:** The main dependent variable consisted of a specific aspect of modeling outcome which reflects certain human adjustment behavior during reuse, i.e. the adjustment biases that occur in the case of required changes. We operationalized this by counting those adjustments made (to the reuse model) that were actually required and label it “Required Adjustments Made”. In order to make the ‘required adjustments made’ comparable, we present them as a relative share in percentage (the absolute number could have been taken as well though, as the edit distances in both treatments are nearly equal). Two sub-cases of adjustments have to be distinguished to understand the different characters of adaptation that were required.

Required Adjustments Made:

- The sub-case of *extraneous* elements in the reuse model: extraneous elements are not required by the original design task and therefore had to be *deleted* from the reuse model. Too many carried over extraneous elements distort the solution process model leading to inferior quality. Extraneous elements were the main manipulation introduced in the *low granularity* treatment.
- The sub-case of *omitted* elements in the reuse model: omitted elements are missing or too coarse elements in the reuse model – although they are required by the design task. These had to be *added* in the solution model. Omitted elements were the main manipulation introduced in the *high granularity* treatment.

**Model quality:** While the overall quality of a process model remains an important goal in order to improve shared mental models within and across organizational units and thereby facilitate effective communication (cf. [26-28]), our main focus was the adaptive behavior of human modelers. Still, we were interested whether the solutions of reuse groups had any significant qualitative advantages over from-scratch solutions (modeled by control group). A grading scheme along the dimensions validity and completeness [26] was developed which was applied by the authors to rate the solution models. The scheme was based on 7-Point Likert scales. The average per model ratings provided a rough quality estimate that was regarded sufficient to show the potential differences in quality between reuse-based and from-scratch approaches. We refrained from measuring syntax and comprehension based quality dimensions [26] as syntax-checking facilities are available in modern editors, and layout development hinges on tool ergonomics respectively. Our main interest was directed at whether the solution model was valid and complete with regard to the task.

### 5.3 Controlled Variables

Participant and task characteristics were controlled for in this modeling experiment. We focused on advanced end users who played the role of novice process designers with basic computer and process modeling training, and limited application domain knowledge. The 39 participants of the experiment were mostly graduate students of computer science or business & information system engineering of the Technische Universität Berlin, Freie Universität Berlin, Humboldt Universität zu Berlin, Hasso-Plattner-Institute (HPI) Potsdam, and KTH Stockholm. Beside these, participants were taken from the group of research assistants of these institutions. In addition to these, some participants were business professionals that work for IT-related companies. All participants were selected based on the contact databases of our department containing several hundreds of students, research assistants and business professionals respectively.

**Application domain knowledge:** The process design task was selected from a specific domain, i.e. the internal power supplier switch process that appeared unfamiliar to the participants with high probability. We took the task from a real case of a large European power utility provider to which the switch process is also new because of recent legislative demands the participants' application knowledge could therefore be classified as beginner's level. Significant impacts stemming from this factor were therefore not expected (cf. [29] on the role of application knowledge in schema understanding).

**Modeling language and knowledge:** The participants could be classified as moderate to advanced modelers as they all have received formal training on various process modeling notations. For instance all academic participants have completed courses that involved formal modeling notations and methods, such as Business Process Modeling Notation (BPMN), Event-driven Process Chains (EPC) and Unified Modeling Language (UML) that were applied to different business contexts. Business expert participants have had similar education and training. We opted for BPMN in this study because of its wide-spread use in process modeling projects, intensive standardization efforts and support through many tools offered by global vendors.

**Edit distance between reuse model and solution:** We controlled for edit distance between the two reuse models and the solution, as a reuse model that is very different from a to-be designed solution is assumedly much harder to be reused and adapted to the actual solution model as it would be in case of a reuse model very similar to the solution. Edit distance can be regarded as an operationalization of business process similarity (cf. [30-32] on similarity and for further references). Because we want to focus on the effects of granularity only, we kept the edit distances between high G-reuse model and solution and low G-reuse model and solution, respectively, on an equal level. Applying the graph edit distance measure from [31] we set the distance to 35, using *skipn* (node insertion and deletion) and *skipe* (edge insertion and deletion). We relaxed the requirement of string edit distance for this first exploratory approach. Taking into account the notion of graph edit similarity [31], our study will contribute to the specific costs that are involved in different adaptation operations and thereby render the concept of business process similarity more precisely with regard to cognitive biases.

**Task complexity:** Various measures have been proposed to describe the complexity of tasks [33-35]. As the selected task was presented as a textual description and as several solution paths exist to arrive at a solution model, we drew upon constructs from applied linguistics and problem-solving. We controlled e.g. the number of information elements [33], degree of abstraction [36], degree of "here-and-now" [33] and syntactic features such as number of words and sentences, average sentence length and others [35]. As in this exploratory study a single task only was used, task complexity was not of prime importance – but in the event of testing design tasks from other domains, as is planned by the researcher team (see Fig. 4.), the control of task complexity will be crucial. Having tackled the research on task complexity measures here, adequate constructs are readily available.

### 5.4 Experiment Procedure

Pilot sessions were conducted to test the experimental procedure and evaluate the clarity, length, and format of the task description and the adaptability of the provided reuse models. Based on the feedback, minor adjustments were made to the textual task description to eliminate instances of major imprecision. As pilot users showed some technical difficulties regarding the limited space available within reuse models, the canvas size was evenly increased to reduce any adaptation bias caused by space limitations as much as possible.

The overall experimental sessions were conducted individually with people (some in parallel) at various appointments over the time period Oct. ‘09 to Feb. ‘10. The subjects were given the same process design task and were instructed to read the case and design the process model with the web-based Signavio-Oryx process model editor [37]. Solutions were modeled and saved therein only.

The subjects had overall 60 minutes for reading the case and modeling their solution. By doing this we indirectly measure how quickly the task was captured by evaluating how far the subjects would come within the given timeframe depending on the treatment they received. The time restriction can also be seen as a proxy for conditions in project work in real enterprises which are usually subject to time pressure. The reuse models are provided in Appendix A; the task description can be found in the online Appendix at: http://sites.google.com/site/taskdescription.

The 39 participants were randomly assigned to the three treatment groups: high granularity reuse model, low granularity reuse model, and no reuse treatment. Group size is comparable to other studies on modeling techniques, such as in [14] (n = 14) or [16] (n = 13).

## 6 Results and Discussion

The performances regarding “Required Adjustments Made” of treatment groups 1 and 2 showed obvious differences (see Table 1). The analysis of variance (one-way ANOVA) test was used to detect whether this difference was significant (at $\alpha = 0.05$ level of significance). Because we had two independent populations for the dependent variable “Required Adjustments Made” ANOVA leads to the same result as a two independent sample t-test. We used ANOVA because a third control population was introduced for the subsequent model quality evaluation (for prevention of alpha error accumulation). The variation of reuse model granularity had a significant effect on the required adjustments actually made. We thus can reject the null hypothesis H0a as the probability of making a Type I error is $p < 0.05$ (see Table 2).

**Table 1.** Group statistics of collected data: Required Adjustments Made

| Treatment group | Mean | N | Std. deviation |
|---|---|---|---|
| High G | 0,57 | 13 | 0,165 |
| Low G | 0,11 | 13 | 0,113 |
| Total | 0,34 | 26 | 0,275 |

**Table 2.** Results of one-way ANOVA

| | | Sum of squares | df | Mean of squares | F | Sig. |
|---|---|---|---|---|---|---|
| Betw. groups | (Comb.) | 1,417 | 1 | 1,417 | 70,165 | 0,000 |
| Within groups | | 0,485 | 24 | 0,020 | | |
| Total | | 1,902 | 25 | | | |

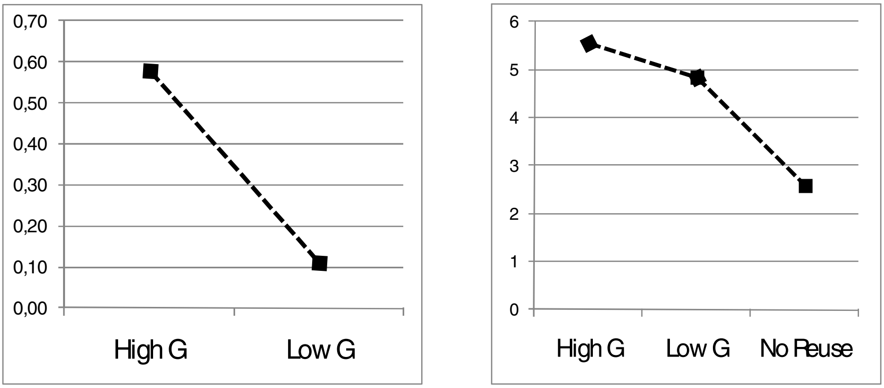


**Fig. 5.** Plot of group means: Required Adjustments Made (left) and Model Quality (right)

**Table 3.** Post-hoc comparisons by Bonferroni test

| (I) | (II) | Mean Diff. (I-J) | Std. Error | Sig. | 95% Confidence Interval | |
|---|---|---|---|---|---|---|
| | | | | | Lower Bound | Upper Bound |
| High G | Low G<br>No Reuse | 0,718<br>2,974 | 0,165<br>0,165 | 0,000<br>0,000 | 0,301<br>2,558 | 1,135<br>3,391 |
| Low G | High G<br>No Reuse | - 0,718<br>2,256 | 0,165<br>0,165 | 0,000<br>0,000 | - 1,135<br>1,839 | - 0,301<br>2,672 |
| No Reuse | High G<br>Low G | - 2,974<br>- 2,256 | 0,165<br>0,165 | 0,000<br>0,000 | - 3,391<br>- 2,672 | - 2,558<br>- 1,839 |

This means that in this specific case the granularity of the reuse model indeed has a specific impact on the extent of triggered adjustment activities: in the low granularity case, the relative adjustments, i.e. predominantly the deletion of extraneous tasks was significantly lower ($\mu$ = 11% only) compared to the relative adjustments (i.e. the addition of fine-granular tasks) that were made in the case in which the reuse model was of high granularity ($\mu$ = 57%). These results have important implications on the design of knowledge management systems that provide the search and adaptation of reusable process models in new process design tasks. The effect of anchoring to extraneous functionality is apparently so strong that certain fine levels of granularity for reuse models appear prohibitive in light of efficiency requirements. This has a direct implication on the definition of process similarity – i.e. when adaptation costs are implied – which must underlie the search functionality provided in a repository. The cost of removing extraneous functionality from the reuse model in order to arrive at the solution model must be weighed significantly higher than the adaptation operations that involve the refinement of functionality, but would still arrive at the same solution model. The known process distance measure by [31] can therefore be complemented by our newly gained propositions about the inhomogeneous costs of different process adaptation operations.

As we wanted to also have a rough estimate of the solution quality differences of the two reuse approaches compared to a non-reuse approach, we tested the ratings of solution quality with a one-way ANOVA for significance. Because the overall result was significant (rejection of H0b), we applied a post-hoc multiple comparison procedure (Bonferroni test) to see at which mean pair the differences actually occur. As can be seen in Table 3 all pair-wise mean differences are significant. We would like to emphasize the drop of quality in the control group (no reuse of a model) compared to the other two groups (see Fig. 5 right). Apparently participants had difficulties to map the complete task requirements to their solution within the given time. This shows a clear advantage of the groups that had a reuse model provided. It seems that in this kind of setting, i.e. regarding task size, time pressure, etc., the approach of reusing models has significant efficiency benefits. The theoretical relationship between numerous extraneous information elements in a reuse model and the severe decline of solution quality should be investigated in future studies in more depth. While the quality difference between High *G* and Low *G* treatments were statistically significant, the absolute difference seems small – the practical relevance of this result should be investigated in a separate study.

Compared with our previous study [8] it must be noted that the setup has changed regarding the experimental task. While previously two tasks resided at different granularity levels, we now focused on a single task, but varied the reuse model granularity. The lower Recall from the previous study in the High *G* case could not be confirmed as the model quality (which involves recalled elements) is now higher in the High *G* case. But the previous task size was much smaller, *two* tasks were involved and therefore the outcomes of the studies can hardly be compared.

## 7 Conclusion

We have drawn upon the cognitive heuristic of anchoring which has been demonstrated in domains such as data-base design, and applied it to process design tasks under reuse of available process models to evaluate possible impacts. We identified model granularity (or modularity) as having a specific relation to certain adaptation requirements to which the process modeler can anchor. In a laboratory experiment we manipulated two treatment artifacts, i.e. reuse models of *high* and *low* granularity, and showed that the anchoring effect is significantly stronger in the low granularity case (i.e. very little adjustments were made although they were required). The failed adjustments consequently contribute to lower validity of solution models. Our study contributes to a deeper insight into cognitive effects during adaptive reuse processes in problem-oriented design tasks, particularly strong human anchoring to extraneous information. Moreover, our finding refines the cost notions of adaptation operations needed for a process similarity definition that is augmented by human judgment behavior: deleting information from reuse artifacts is relatively expensive!

Nonetheless, our study has to be seen in light of its limitations. The external reliability of the study is bound to the application domain of the supplier switch process of the power utilities sector. Whether the effects can be observed repeatedly across other application domains, remains to be investigated. In order to tackle this issue we are currently conducting the same experimental study in other application domains, in particular the telecommunication services domain. By controlling the parameters explained in our contribution, but varying the application domain, we intend to test the external reliability of the here shown effects of varied granularity of reuse artifacts on modeling performance.

We have manipulated the experimental study in a specific way, i.e. the *partition* of the process context, for which the granularity values were calculated, was specifically designed by the authors. The same granularity value could be calculated for a completely different partition of the process universe, i.e. certain coarseness may have been defined for other tasks in the process than what we have chosen, or certain other tasks may have been much finer. As the theoretical number of possible partitions may be very large, the future evaluation of varied partitions must follow practical considerations. We have concentrated our efforts on the study of the BPMN notation. It cannot be completely ruled out that other process modeling notations may evoke different effects during a ceteris paribus experiment. Choosing BPMN was a pragmatic decision in the face of large communities engaged in the discourses of BPMN, intensive standardization efforts, and large IS technology vendors that have opted to include BPMN as a process modeling language in products.

The subjects involved in the study can only be regarded as proxies for novices in organizations. An interesting question therefore still is how the effects would vary if experts only would model the given process. Moreover, the number of investigated subjects is small with heterogeneity among them left despite the control efforts. Future work should increase the sample size and apply homogeneity tests.

While the above poses some limitations, it presents interesting possibilities for new research questions carrying forward our effort. One important issue to consider is that the creators of reusable process models should be available for person-to-person assistance [38]. These relationships deserve more thorough evaluation in the future.

## References

1. Leontiev, A.N.: Activity, Consciousness, and Personality. Prentice-Hall, Englewood Cliffs (1978)
2. Majchrzak, A., Cooper, L.P., Neece, O.E.: Knowledge Reuse for Innovation. Management Science 50(2), 174–188 (2004)
3. March, J.G.: Exploration and Exploitation in Organizational Learning. Organization Science 2(1), 71–87 (1991)
4. Frakes, W., Kang, K.: Software reuse research: status and future. IEEE Transactions on Software Engineering 31(7), 529–536 (2005)
5. Rothenberger, M.A., Dooley, K.J., Kulkarni, U.R., Nada, N.: Strategies for Software Reuse: A Principal Component Analysis of Reuse Practices. IEEE Transactions on Software Engineering 29(9), 825–837 (2003)
6. Kelly, M.: Enhanced Telecom Operations Map (eTOM) - The Business Process Framework, TeleManagement Forum (2007)

7. Supply-Chain Council: Supply Chain Operations Reference-model Version 8.0, Supply-Chain Council, Inc. (2006)
8. 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., et al. (eds.) BPM 2009. LNCS, vol. 5701, pp. 245–260. Springer, Heidelberg (2009)
9. Fettke, P., Loos, P.: Classification of reference models: a methodology and its application Information Systems and E-Business Management 1, 35–53 (2003)
10. Soffer, P., Reinhartz-Berger, I., Sturm, A.: Facilitating Reuse by Specialization of Reference Models for Business Process Design. In: 8th Workshop on Business Process Modeling, Development, and Support (BPMDS 2007) in Conjunction with the 19th International Conference on Advanced Information Systems Engineering, CAiSE 2007 (2007)
11. Scaife, M., Rogers, Y.: External cognition: how do graphical representations work? International Journal Human-Computer Studies 45, 185–213 (1996)
12. Larkin, J.H., Simon, H.A.: Why a diagram is (sometimes) worth ten thousand words. Cognitive Science 11, 65–100 (1987)
13. Prieto-Diaz, R.: Status report: Software reusability. lEEE Software 10(3), 61–66 (1993)
14. Reijers, H., Mendling, J.: Modularity in process models: review and effects. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 20–35. Springer, Heidelberg (2008)
15. Moody, D.L.: The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. IEEE Transactions on Software Engineering 35(6), 756–779 (2009)
16. Batra, D., Wishart, N.A.: Comparing a rule-based approach with a pattern-based approach at different levels of complexity of conceptual data modelling tasks. International Journal of Human-Computer Studies 61(4), 397–419 (2004)
17. Plous, S.: The Psychology of Judgment and Decision Making. McGraw-Hill, New York (1993)
18. Parsons, J., Saunders, C.: Cognitive Heuristics in Software Engineering: Applying and Extending Anchoring and Adjustment to Artifact Reuse. IEEE Trans. Software Eng., 873–888 (2004)
19. Yao, Y.: Probabilistic approaches to rough sets. Expert Systems 20(5), 287–297 (2003)
20. Yao, Y.: A Partition Model of Granular Computing. LNCS Transactions on Rough Sets 1, 232–253 (2004)
21. United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT): Core Components Technical Specification, Version 3.0, United Nations (September 29, 2009)
22. Gemino, A., Wand, Y.: Evaluating modeling techniques based on models of learning. Requirements Engineering 9(4), 248–260 (2004)
23. Teo, H.-H., Chan, H.C., Wei, K.K.: Performance Effects of Formal Modeling Language Differences: A Combined Abstraction Level and Construct Complexity Analysis. IEEE Transactions on Professional Communication 49(2), 160–175 (2006)
24. Mendling, J., Reijers, H.A., Recker, J.: Activity labeling in process modeling: Empirical insights and recommendations. Information Systems 35(4), 467–482 (2009)
25. Kerlinger, F.N.: Foundations of Behavioral Research, 3rd edn. Holt, Rinehart and Winston, Orlando, FL (1986)
26. Lindland, I., Sindre, G., Solvberg, A.: Understanding quality in conceptual modeling. IEEE Software 11, 42–49 (1994)
27. Krogstie, J., Sindre, G., Jørgensen, H.: Process models representing knowledge for action: a revised quality framework. European Journal of Information Systems 15, 91–102 (2006)

28. Davies, I., Green, P., Rosemann, M., Indulska, M., Gallo, S.: How do practitioners use conceptual modeling in practice? Data & Knowledge Engineering 58, 358–380 (2006)
29. Khatri, V., Vessey, I., Ramesh, V., Clay, P., Park, S.-J.: Understanding Conceptual Schemas: Exploring the Role of Application and IS Domain Knowledge. Information Systems Research 17(1), 81–99 (2006)
30. Ehrig, M., Koschmider, A., Oberweis, A.: Measuring Similarity between Semantic Business Process Models. In: Fourth Asia-Pacific Conference on Conceptual Modelling (APCCM 2007). Australian Computer Society, Inc., Ballarat (2007)
31. Dijkman, R., Dumas, M., García-Banuelos, L.: Graph Matching Algorithms for Business Process Similarity Search. In: Dayal, U., et al. (eds.) BPM 2009. LNCS, vol. 5701, pp. 48–63. Springer, Heidelberg (2009)
32. van Dongen, B.F., Dijkman, R.M., Mendling, J.: Measuring Similarity between Business Process Models. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 450–464. Springer, Heidelberg (2008)
33. Robinson, P.: Task Complexity, Task Difficulty, and Task Production: Exploring Interactions in a Componential Framework. Applied Linguistics 22(1), 27–57 (2001)
34. Campbell, D.J.: Task Complexity: A Review and Analysis. Academy of Management Review 13(1), 40–52 (1988)
35. Larkey, L.S.: Automatic Essay Grading Using Text Categorization Techniques. In: 21st annual international ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 90–95. ACM, New York (1998)
36. Prabhu, N.: Second Language Pedagogy. Oxford University Press, Oxford (1987)
37. Signavio GmbH: Signavio (2009), http://academic.signavio.com [cited 17.3.2010]
38. Boh, W.F.: Reuse of knowledge assets from repositories: a mixed methods study. Information & Management 45 (July) 365–375 (2008)

## Appendix A: Process Models Provided for Reuse

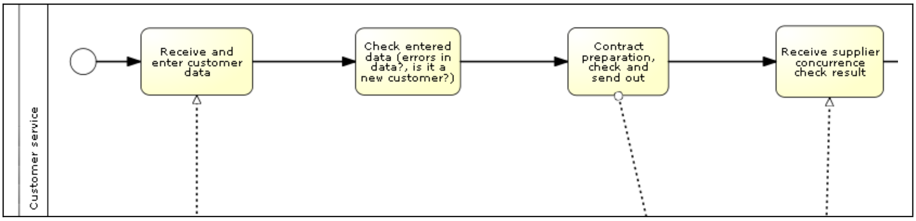

Customer service
Receive and enter customer data
Check entered data (errors in data?, is it a new customer?)
Contract preparation, check and send out
Receive supplier concurrence check result

**Fig. 6.** Excerpt of reuse process model of high granularity (for treatment group 1)

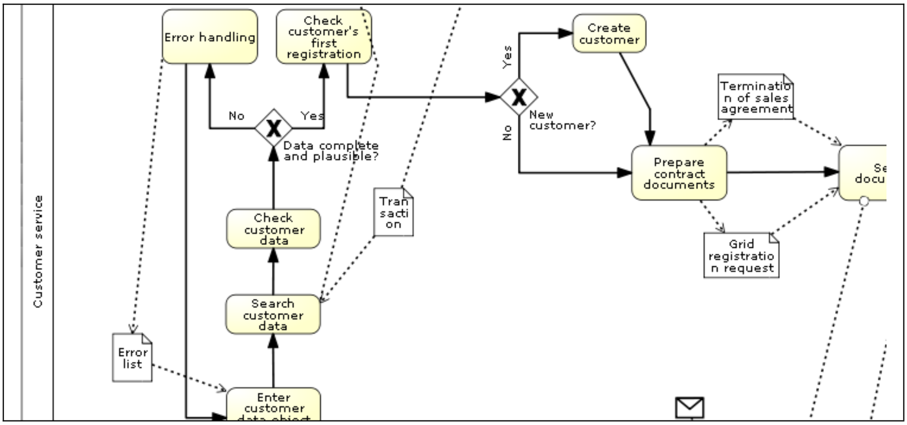

Customer service
Error handling
Check customer's first registration
Create customer
Yes
No
New customer?
Termination of sales agreement
No
Yes
Data complete and plausible?
Prepare contract documents
Se docu
Check customer data
Tran sacti on
Grid registration request
Search customer data
Error list
Enter customer data object

**Fig. 7.** Excerpt of reuse process model of low granularity (for treatment group 2)

# Machine-Assisted Design of Business Process Models Using Descriptor Space Analysis

Maya Lincoln[1], Mati Golani[2], and Avigdor Gal[1]

[1] Technion - Israel Institute of Technology
mayal@technion.ac.il, avigal@ie.technion.ac.il
[2] Ort Braude College, Israel
matig@braude.ac.il

**Abstract.** In recent years, researchers have become increasingly interested in developing methods and tools for automating the design of business process models. This work suggests a method for machine-assisted design of new process models, based on business logic that is extracted from real-life process repositories using a linguistic analysis of the relationships between constructs of process descriptors. The analysis enables the construction of a descriptor space in which it is possible to define new process sequences. The suggested method can assist process analysts in designing new business processes while making use of knowledge that is encoded in the design of existing process repositories. To demonstrate the method we developed a software tool ("New Process Design Assistant" - NPDA) that automates the suggested design method. We tested our tool on the Oracle Applications ERP process repository, showing our approach to be effective in enabling the design of new activities within new business process models.

**Keywords:** New process model design, Business process repositories, Business process integration and management, Process choreographies.

## 1 Introduction

In recent years, researchers have become increasingly interested in developing methods and tools for automating the design of business process models. Process modeling is considered a manual, labor intensive task, whose outcome depends on personal domain expertise with errors or inconsistencies that lead to bad process performance and high process costs [12]. Hence, automating the reuse of constructs, gathered from predefined process models does not only save design time but also supports non-expert designers in creating new business process models. Research in this field encapsulates topics from the areas of software design and data mining [19,15,6,4], and is focused on structured reuse of existing building blocks and pre-defined patterns that provide context and sequences [5].

While most previous work focused on supporting the design of alternative process *steps* within *existing* process models, less work has been carried out on the design of *new* process models. We only identified a few works that address the design of new models [12,14,7]. This work aims at filling this gap by suggesting a generic method for designing new business process models related to any

R. Hull, J. Mendling, and S. Tai (Eds.): BPM 2010, LNCS 6336, pp. 128–144, 2010.

functional domain. The suggested method guides business analysts that opt to design a new business model, by suggesting process steps (activities) that are relevant to the newly created process model. The business logic for such suggestions is extracted from process repositories through the analysis of existing business process model activities. Each activity is encoded automatically as a *descriptor*, using the "PDC" notation, suggested first in [11] and further elaborated in this work for supporting the field of new process model design. The collection of all descriptors formulates a descriptor space, and distances between every two space coordinates are calculated in terms of business process conduct proximity. We show through an empirical evaluation that by utilizing the descriptor space it is possible to effectively support the design of new process models.

As a motivating example consider an airport process model of check-in related processes. Now, suppose that the airport management desires to offer to its customers a new service: "check-in from home". In addition, it is also desired to outline the "check-out" process model as an extension of the current repository. Although these process models are new, the existing repository encapsulates know-how and business logic that are relevant and useful for their creation (*e.g.*, passenger check-in policies and procedures regarding security, luggage handling, passenger handling, and document validation). In the above scenario, it would have been helpful for the process designer to design the new processes using a supporting system that relies on the reuse of previous know-how instead of doing this manually from scratch. To illustrate our methodology in this work we use a real-world case study for airport process design. Based on a "check-in" process that already exists in the repository, we demonstrate how it is possible to design the two, above mentioned, new business processes.

This work proposes an innovative method for assisting designers in designing brand new business process models while making use of knowledge that is encoded in the design of existing, related process models. Our work presents the following innovations: (a) it provides generic support to the design of new business process models; (b) it equally utilizes objects and actions for business content analysis: we make use of all activity linguistic components (object, actions and their qualifiers) concurrently, without special focus on objects (as object centric methods do) or on actions (as activity-centric methods do); (c) it extends the PDC model [11] to enable the extraction of business logic from business process repositories.

The suggested method was implemented within a software tool, that was demonstrated using the aviation industry case study and the Oracle Applications ERP process repository.

The rest of the paper is organized as follows: we present related work in Section 2, positioning our work with respect to previous research. In Section 3 we present an extended model for representing process activities based on the process descriptor notion, presented first in [11], and extended in this work to support new process model design. In Section 4 we define and discuss the descriptor space and explain how to navigate in it. Then, we describe our method for designing new business process models in Section 5. Section 6 introduces the software tool and our empirical analysis. We conclude in Section 7.

## 2 Related Work

Most of the efforts invested in developing methods and tools for designing process models focus on supporting the design of alternative process steps within existing process models. Such a method is presented in [16] aiming to provide next-activity suggestions during execution based on historical executions and optimization goals. Recommendations are generated based on similar past process executions as documented in event logs. Similarly, [5] suggests an approach for helping business users in understanding the context and consequences of applying pre-defined patterns during a new process design. Other works extend this research domain by adding both generalization and formalization layers [7,1]

Few works were devoted to the design of brand new process models within specific and predefined domains. The work presented in [12] utilizes the information about a product and its structure for modeling large process structures. [14] presents a method, named "the product-based workflow design," for designing new process models based on product specification and required design criteria.

A requirement for the support of business process design involves the performance of a structured reuse of existing building blocks and pre-defined patterns that provide context and sequences [5]. The identification and choice of relevant process components are widely based on the analysis of linguistic components - actions and objects that describe business activities. Most existing languages for business process modeling and implementation are activity-centric, representing processes as a set of activities connected by control-flow elements indicating the order of activity execution [20]. In recent years, an alternative approach has been proposed, which is based on objects (or artifacts/entities/documents) as a central component for business process modeling and implementation. This relatively new approach focuses on the central objects along with their life-cycles. Services (or tasks) are used to specify the automated and/or human steps that help move objects through their life-cycle, and services are associated with artifacts using procedural, graph-based, and/or declarative formalisms [8]. Such object-centric approaches include artifact-centric modeling [13,2], data-driven modeling [12] and proclets [17]. Further analysis of the object-centric model in terms of computing the expected coupling of object lifecycle components is presented in [20].

Although most works in the above domain are either object or activity centric, only a few works combine the two approaches in order to exploit an extended knowledge scope of the business process. The work in [9] presents an algorithm that generates an information-centric process model from an activity-centric model. The work in [11] presents the concept of business process descriptor that decomposes process names into objects, actions and qualifiers.

In this work we take this model several steps forward by: (a) describing the relationships between the model components; (b) showing how the descriptor model can automatically be generated (using NLP methods); and (c) utilizing the qualifiers for identifying the relationships between descriptor components within a process repository.

# 3 The Activity Decomposition Model

This section describes a model of business process decomposition that supports process design. To illustrate the model components we make use of the aviation example from Section 1.

## 3.1 The Descriptor Model

The Workflow Management Coalition (WFMC) [3] defines business process as a **"**set of one or more linked procedures or activities which collectively realize a business objective or policy goal.**"** An example of such business process model is the "Passenger check-in" process model, presented in Fig. 1. This figure is based on YAWL [18] with two slight visual representation modifications, convenient for our needs: (a) roles were added at the top of each activity; and (b) predecessor and successor processes are presented as nested activities at the beginning and at the end of the workflow.

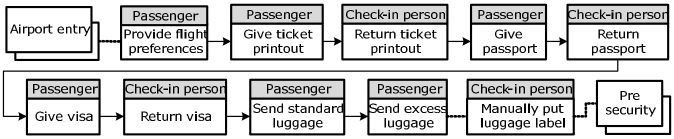


**Fig. 1.** An example: the "Passenger check-in" process model

In the Process Descriptor Catalog model ("PDC") [11] each activity is composed of one action, one object that the action acts upon, and possibly one or more action and object qualifiers, as illustrated in Fig. 2, using UML relationship symbols. Qualifiers provide an additional description to actions and objects. In particular, a qualifier of an object is roughly related to an object state. State-of the art Natural Language Processing (NLP) systems, *e.g.*, the "Stanford Parser,"[1] can be used to automatically decompose process and activity names into *process/activity descriptors.*

For example, in Fig. 1, the activity "Manually put luggage label" generates an activity descriptor containing the action "put," the action qualifier "manually," the object "label" and the object qualifier "luggage."

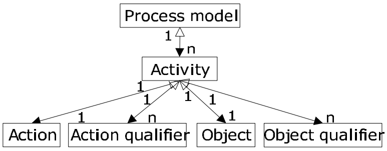


**Fig. 2.** The activity decomposition model

[1] http://nlp.stanford.edu:8080/parser/index.jsp

## 3.2 A Descriptor Model for Process Design

We now enhance the PDC model of [11] to support process design. Our model has two basic elements, namely objects and actions, and we delineate four taxonomies from them, namely an *action hierarchy model*, an *object hierarchy model*, an *action sequence model* and an *object lifecycle model.* The business action and object taxonomy models organize a set of activity descriptors according to the relationships among business actions and objects both longitudinally (hierarchically) and latitudinally (in terms of execution order), as detailed next.

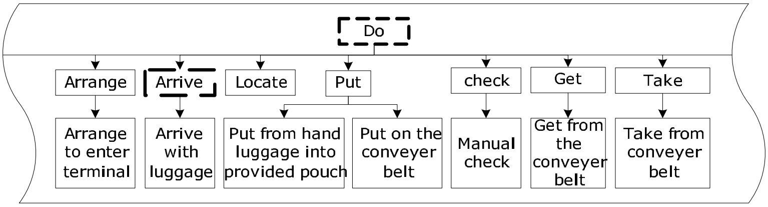


**Fig. 3.** A segment of the action hierarchy model extracted from the aviation processes

The longitudinal dimension of actions and objects is determined by their qualifiers. To illustrate the longitudinal dimension of the aviation workflows, a segment of the action hierarchy model is presented in Fig. 3 and a segment of the object hierarchy model is presented in Fig. 4. Consider the complete action (the action and its qualifier) "Manual check." It is a subclass (a more specific form) of "Check" in the action hierarchy model, since the qualifier "Manual" limits the action of "Check" to reduced action range. It is worth noting that some higher-hierarchy objects and actions are generated automatically by removing qualifiers from lower-hierarchy objects and actions. For example, the action "Arrive" was not represented without qualifiers in the aviation processes repository, and was completed from the more detailed action: "Arrive with luggage" by removing its action qualifier ("with luggage") (see Fig. 3). In Section 5 we will show how such elements assist in designing new processes by enriching the underlying process repository range. This type of objects and actions are marked with a dashed border. In addition, a root node "Do" is added to any action hierarchy model and a root node "Object" is added to any object hierarchy model, effectively generating a single object and action tree from what would have been, in graph theoretic terminology, a forest.

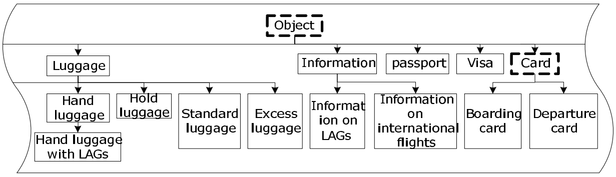


**Fig. 4.** A segment of the object hierarchy model extracted from the aviation processes

To illustrate the latitudinal dimension of the aviation process repository, a segment of the action sequence model is presented in Fig. 5 and a segment of the object lifecycle model is presented in Fig. 6. Latitudinally, each object holds: (a) a graph of ordered actions (an "action sequence") that are applied to that object. For example, the object "Luggage" is related to the following action sequence: "Arrange" followed by "Send" (see Fig. 5); (b) a graph of ordered objects that expresses the object's lifecycle, meaning - the possible ordering of the object's states. This sequence is built by locating the same object with different qualifiers along the process diagram. For example, the object "Luggage" is part of the following object lifecycle: "Luggage" –> "Standard luggage"/"Excess luggage" –> "Labeled luggage" (see Fig. 6).

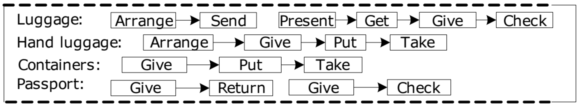


**Fig. 5.** A segment of the action sequence model extracted from the aviation processes

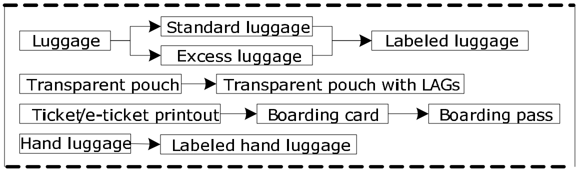


**Fig. 6.** A segment of the object lifecycle model extracted from the aviation processes

## 4 The Quad-Dimensional Descriptor Space

Based on the activity decomposition model, it is possible to visualize the operational range of a business process model as a descriptor space comprised of related objects and actions. The descriptor space is a quad-dimensional space describing a range of activities that can be carried out within a process execution flow. The coordinates represent the object dimension, the action dimension, and their qualifiers. Therefore, each space coordinate represents an activity as a quadruple $AC = \langle O, OQ, A, AQ \rangle$, where $O$ is an object, $OQ$ is a set of object qualifiers, $A$ is an action, and $AQ$ is a set of action qualifiers.

For example, the activity "Arrive at appropriate terminal with luggage" can be represented by the following coordinate: ⟨arrive, with luggage, terminal, appropriate⟩. This coordinate represents an actual activity in the business process model: "Airport entry." Once constructed, the descriptor space includes all the possible combinations of descriptor components, forming a much larger and diversified set of possible descriptors. Hence it includes several "virtual" combinations- that did not originally exist in the original process repository.

These virtual combinations, together with existing activities, form an expanded repository that is used for the design of new business processes.

For every two coordinates in the descriptor space we define a *distance function* that is tailored to our method. The proposed distance function in the descriptor space represents a linear combination of changes within each of its dimensions. Therefore, we define four specific distance measures using the structures that were gathered from existing business processes repositories (Section 3).

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

In a similar way we define *Action distance*, $AD$, calculated based on the action sequence model. For example, the action distance between "Present" and "Check" when acted on "Luggage" is 3 (see Fig. 5).

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

In a similar way we define *Action hierarchy Distance*, $AHD$, calculated based on the action hierarchy model.

$OD$, $AD$, $OHD$ and $AHD$ are combined to generate a specific distance function between any two activities $AC_i$ and $AC_j$, as follows:

$$Dist\,(AC_i, AC_j) = OD_{ij} + AD_{ij} + OHD_{ij} + AHD_{ij} \tag{1}$$

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

As an example for the use of this distance function consider the two descriptors (luggage, hand, check, null) and (luggage, null, get, from the conveyer belt). To navigate from the first descriptor to the second, we first move one step up in the object hierarchy ($OHD = 1$) from the object "Hand luggage" to the object "Luggage" (see Fig. 4). Then, we recede two steps from the action "Check" in the action sequence ($AD = 2$), resulting with the action "Get" (See Fig. 3). Finally, we drill down one step within the action hierarchy ($AHD = 1$), and retrieve the action "Get from the conveyer belt", and by that we reach the target descriptor. In total, the distance between the two above coordinates is 4.

In general, it is possible to navigate within the descriptor space (hence, move from one descriptor to another) in a meaningful way. This navigation enables us to move up to more general or drill down to more specific action and object scopes as well as to navigate to: (a) preceding and succeeding actions that act on the descriptor's object and (b) advance to a successor (more advanced) state of the object's current state or recede to a predecessor (less advanced) state. A more elaborated discussion regarding the navigation within the descriptor space is presented in [10].

## 5 The *Process Delineator* Method for Assisting the Design of Process Models

The *process delineator* method relies on an underlying process descriptor space and at any phase it either refines an existing process activity or suggests a next process activity. Since the descriptor space has a large number of elements, a general search within this space may be very expensive. Therefore, we will hereby suggest a more efficient navigation method that is tailored for our specific target.

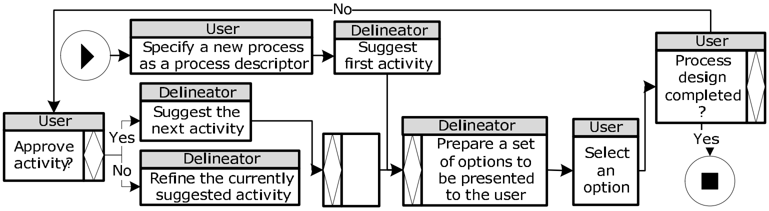

No
User
Specify a new process as a process descriptor
Delineator
Suggest first activity
User
Process design completed ?
Delineator
Suggest the next activity
User
Approve activity?
Yes
No
Delineator
Refine the currently suggested activity
Delineator
Prepare a set of options to be presented to the user
User
Select an option
Yes

**Fig. 7.** The process delineator mechanism

The process delineator is illustrated in Fig. 7. The design process starts when a process designer defines the name of the new process model. This name is decomposed into a process descriptor format. For example, a new process named: "Send luggage from home," will be transformed into the following process descriptor: object="luggage," action= "send," object qualifier="null," action qualifier="from home."

Based on the process descriptor input, the process delineator produces options for the first process activity (see Section 5.1). The process designer reviews the output option list, and either selects the most suitable first activity for the newly designed process, or suggests an alternative. At any next phase the designer either requests to refine the current activity (see Section 5.2) or advance to design the next activity (see Section 5.3). Each time the process delineator is requested to suggest activities as part of the design process it outputs a list of options, sorted and flagged according to the option's relevance to the current design phase (see Section 5.4).

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

It is worth noting that the process delineator also uses virtual activities (see Section 4). These activities can enrich and improve the design process by expanding the optional range of available building constructs.

### 5.1 Suggesting the First Process Activity

To suggest the first process activity, the process delineator searches the target object and its more specific objects within the object hierarchy model. It then creates first activity suggestions in the format of activity descriptors comprised of the retrieved objects and the first action that acts upon them in the action sequence model. Continuing the example above, the following first activity options will be suggested (see Fig. 5): “Arrange luggage” and “Give hand luggage.”

### 5.2 Refining the Currently Suggested Process Activity

A refinement can be performed by five orthogonal methods. To illustrate each of these methods we will show how the action “Get luggage” can be refined.

**Action and Object Refinement.** To refine the reference action, the process delineator navigates the descriptor space by drilling *down* the action hierarchy to more specific actions. It then combines the retrieved, more specific, actions with the reference object. The refinement of objects is done in a similar manner. By applying an action refinement to our example's reference activity, the refinement option: “Get luggage from the conveyer belt” is retrieved (see Fig. 3).

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

**Advance an Action or an Object State.** To advance the object's state within an activity, the process delineator navigates the descriptor space by moving *forward* in the object lifecycle sub-dimension. In a symmetrical manner, to advance an activity's action, the process delineator moves forward in the action sequence sub-dimension of the descriptor space. In our example the objects “Standard luggage” and “Excess luggage” represent more advanced states of the object “Luggage” (see Fig. 6) and the action “Give” follows the action “Get” in the action sequence applied on “Luggage” (See Fig. 5). Therefore, the following three refinement suggestions are constructed: “Get standard luggage”, “Get excess luggage” and “Give luggage”.

**Recede to a Less Processed State of the Object or to a Former Action.** The receding method is similar to the advancing method, only this time the process delineator navigates the descriptor space by moving *backwards* in the object lifecycle and action sequence sub-dimensions. For example, the action “Present” is acted on “Luggage” before this object is taken (before the action “Get” is applied) (see Fig. 5), hence creating the option: “Present luggage.”

**Move to a Sibling Action or Object.** In order to move to a sibling action, the process delineator moves horizontally within the action hierarchal subdimension. By fixing the reference action's level, it retrieves sibling actions for this action. Moving to a sibling object is conducted in a similar manner. Continuing our example, a navigation to sibling actions to "Get" retrieves a list of activities that includes: "Check luggage" and "Take luggage" (see Fig. 3).

### 5.3 Suggesting the Next Process Activity

This step can be achieved in two alternative ways: either by advancing to a later action that acts on the currently accepted (reference) object, or advancing to a sibling object combined with the reference activity's action. The rationale behind the last directive is that in some process flows the same action is operated on sibling objects in order to fulfill a certain process goal. For example, in the aviation processes, the "Check-in" process includes the two consecutive activities: "Send standard luggage" and "Send excess luggage."

To demonstrate this step, consider the activity following "Give passport." The process delineator finds in the action sequence model two options: "Check passport" and "Return passport" (see Fig. 5). In addition, sibling objects to "Passport" are also retrieved from the object hierarchy model, creating additional options such as "Give visa," "Give luggage," and "Give information" (see Fig. 4).

### 5.4 Preparing a Set of Output Options

The process delineator assesses the output options in each navigation phase and combines an ordered option list to assist the user in selecting the most suitable option. The process delineator sorts the options according to their relevance to the current design phase based on two considerations. First, on proximity to the design phase reference coordinate - which represents the last selected activity when suggesting a refined or next activity, or to the targeted process descriptor when suggesting the first process activity. Second, the process delineator considers to what extent was it changed comparing to actual activities that were part of the underlying process repository. Therefore, the construction of the ordered option list is conducted according to the following four stages: (a) sort by proximity to the reference activity; (b) internally sort by similarity to processes in the repository; (c) add a random option to avoid getting stuck in a local optimum; and (d) flag each option, as further detailed below.

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

**Internally Sort by Similarity to Processes in the Repository.** The process delineator also takes into account the extent to which a proposed activity was changed in comparison to actual activities in the underlying process repository. For this purpose the process delineator distinguishes between three change levels: (a) *No change*- the suggested activity is represented "as is" within the underlying business process repository. These options are not marked by any flag;

(b) *Slight modification* - there is an actual activity in the underlying business process repository containing the same object and action with different qualifiers. These options are marked with “˜”; (c) *Major change* - the object and action within the suggested activity were not coupled in any of the activities within the underlying business process repository. These options are marked with “M”.

Therefore, after sorting the options by their proximity to the reference activity, each group of options with equal distances is internally sorted in an ascending order - presenting the “no change” options at the beginning of the list, since these options posses a higher level of credibility, continuing with the “slight modification” options and terminating the list with the “major change” options. According to the example presented in Section 5.3, several options were generated as candidates for next activities to be conducted after the activity “Give passport.” Most of these options were produced by combining the action “Give” with siblings of the object “Passport,” hence having the same distance from the reference activity. Nevertheless, these options can further be differentiated. For example, “Give visa” is an actual activity in the aviation process repository, and therefore is flagged as such. Nevertheless, “Give luggage” has no representation in this repository, but since “Give hand luggage” does, this option is flagged by “˜.” Since there is no descriptor that combines the action “Give” and the object “Information” in this repository, the option “Give information” is flagged by “M.”

**Add a Random Option.** To avoid getting stuck in a local optimum, the process delineator adds at any stage a random activity from the descriptor space, that shifts the reference activity to a new random coordinate, in a similar manner as in simulated annealing (or mutation in genetic algorithms). Thus, the process delineator can provide new suggestions that are based upon a proximity sort to this new reference activity.

**Flag Each Option.** After assessing each option’s relevance to the current navigation phase and sorting the option list accordingly, the process delineator tags each option with both the numerical distance value and the change level. For example, the option “Give luggage” from the example above will be flagged “[2,˜].”

## 6 Implementation, Case Study and Experiments

### 6.1 Implementation

We have developed a system that implements the suggested method for designing new process models. We named this software system: “New Process Design Assistant” (NPDA). Given a process name, and based on an existing process repository, the NPDA guides users in creating new process models. The system implements a client-server architecture. Server side logic is implemented in PHP using a MySql database. It uses a Natural Language (NL) parser - the “Stanford Parser” - as a web service for decomposing sentences into linguistic components (see Section 3.1). The client runs within an Internet browser and is implemented in HTML and JavaScript, with AJAX calls to the server. The server side high-level architecture is further detailed in our technical report [10].

### 6.2 Case Study: An Example for Designing a New Process Model

To illustrate the proposed framework we present two short examples from the field of aviation. The full extended case-study can be found in the technical report[10]. The aviation process repository covers airport activities starting from the passenger's entry to an airport, through document handling and security checks and terminating as the passenger boards the airplane. The newly designed processes are related to the aviation field, but are not covered by the process repository. The first new process, "Passenger checkout," extends the process repository by handling passenger related activities conducted *after* an airplane arrives at its destination. The second new process, "Send luggage from home," extends the process repository by offering an additional service to passengers *before* their arrival at the airport.

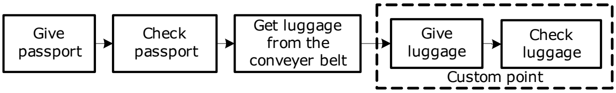
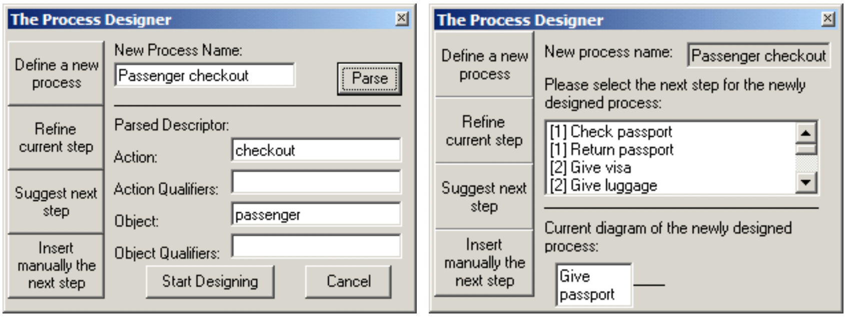


**Fig. 8.** The new designed process diagram for "Passenger checkout"

The first example supports the design of a new business process for: "Passenger checkout." The generated output (new process model) of this example is illustrated in Fig. 8 as a YAWL diagram. The design process starts when the (human) process designer inserts the following process descriptor: (action="checkout", action qualifier=null, object="passenger", object qualifier=null) to the process delineator (see Fig. 9a) and determines that the first activity is: "Give passport." Respectively, the process delineator searches the descriptor space, looking for next activity possibilities. The result set includes the following activities (see Sections 5.3 and 5.4): "[1] Check passport," "[2] Give visa" and "[2,M] Give information" (see Fig. 9b). The designer selects the option "Check passport" and decides that this activity is suitable.

The design process continues with four more design phases. The 2nd phase required a refinement for the option "Get luggage" - which was suggested as the next activity after "Check Passport." The resulted refined option list includes the option: "[1,~] Get luggage from the conveyer belt," (see Section 5.2), and this option was selected by the designer. Note that this activity was not represented "as is" in the business process repository.

The designer now wishes to design the new business process: "Send luggage from home." An interesting observation in this design process is that the designer selects more often next step activities that share the same action applied on sibling objects. For example, the the activity "Give passport" was followed by "Give flight ticket" and "Give boarding pass;" and the activity "Check flight ticket" followed the activity "Check passport" (see resulted process model diagram at [10]). The business logic behind this phenomenon is that this process expresses a more interactive business conduct in which one party (the passenger) exchanges items with the other party (the airport representative).



New Process Name: Passenger checkout

Parsed Descriptor:

Action: checkout

Action Qualifiers:

Object: passenger

Object Qualifiers:

New process name: Passenger checkout

Please select the next step for the newly designed process:

[1] Check passport
[1] Return passport
[2] Give visa
[2] Give luggage

Current diagram of the newly designed process:

Give passport

(a) The designer's request for designing the new process: "Passenger Checkout"

(b) The third phase in designing the new process: "Passenger Checkout"

**Fig. 9.** The designer's request for designing the new process: "Passenger Checkout"

### 6.3 Experiments

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

**Experiment Setup.** The "New Process Design Assistant" software (NPDA, see Section 6.1) was installed on a workstation running Windows XP, IIS6, PHP 4.8 and MySQL 5.0. This workstation served both as the server and the client, running Internet Explorer 7 as the application container and presentation layer. The "no-connection" distance (defined in Section 4) was set to 500.

**Data.** We chose a set of 14 real-life processes from the Oracle Business Model (OBM),[2] comprising: (a) nine business processes from the "Procurement" category, with 96 activities altogether; and (b) five business processes from the "Inventory" category, with 31 activities altogether. The "Procurement" data set contains related, sequential activities and therefore represents a focused operational area. The "Inventory" data set represents an extended business area, featuring loosely coupled business logic. Using the selected 14 processes we created a "process repository database" (see Section 6.1).

**Evaluation Methodology.** To evaluate the suggested method we conducted 14 experiments. At each experiment, a single process was removed from the database and then reconstructed using the NPDA software. This "machine assisted reconstruction" enables us to objectively measure the method's effectiveness.

Each experiment was conducted according to the following steps: (a) preparation: remove one of the processes from the database so that the database will not contain any of its descriptor components; (b) run the NPDA in a stepwise

[2] `http://www.oracle.com/applications/tutor/index.html`

manner. At each phase we try to identify an activity ("goal activity") that is compatible with the removed process, according to the following steps: (1) if the goal activity's linguistic components are represented in the Process Repository Database, run the "find next activity" algorithm (see Section 5.3). If the output list contains the goal activity - continue to reconstruct the next goal activity. Else, run the "activity refinement" algorithm (see Section 5.2). If the option list produced by the refinement step does not include the goal activity, choose the activity that shares the largest amount of common descriptor components with the goal activity as a basis for an additional refinement. If, after 10 successive refinements, the required activity is still not represented by one of the output options, it is inserted manually as the next process activity and the design process is continued by locating the next activity; (2) else (the goal activity's linguistic components are not represented in the Process Repository Database), the next goal activity is inputted manually by the experimenter.

**Results and Analysis.** Table 1 presents a summary of the experiment results. Each experiment of creating a new process model was based on a database with the set of all activity descriptors in all process models, excluding the set of activity descriptors of one goal process. This means that we aim at recreating the goal activities from a partial set of activity descriptors. On average, for 89% of the goal activities, all descriptor components were contained both in the goal process and in another process (see column #3). This was the case despite the relatively small experiment size (13 processes, whereas the entire OBM includes around 1,500 processes), highlighting the amount of similarity one would expect when designing new processes based on an existing repository. For the remaining 11%, at least one descriptor component was missing. In such a case, the activity was inserted manually during the design process. It is worth noting that for the 89% of activities that had the potential of reconstruction from the database, 100% were reconstructed successfully using our method (see Table 2).

In addition, Table 1 shows that on average, two iterations are required for reconstructing a goal activity (see column #4). The design of Procurement

**Table 1.** Experiment results

| Column # | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Column name | # of total processes in DB | # of total activities in DB | % of goal activities represented in the DB | Avg. # of steps per design phase | Avg. location of correct option in 'next activity' | Avg. location of correct option in 'refine activity' | Avg. location of the correct option per design phase |
| Avg.-all | 14 | 127 | 89.0% | 2.0 | 1.2 | 2.8 | 2.6 |
| Avg.-Procurement | 9 | 96 | 90.6% | 1.9 | 0.8 | 3.0 | 2.8 |
| Avg.-Inventory | 5 | 31 | 83.9% | 2.1 | 1.9 | 2.4 | 2.3 |

processes required slightly less steps than the design of Inventory processes (1.9 vs. 2.1 steps on average, respectively). It should be noted that the location of the goal activity was very high in the ranked list of suggested activities (average location: 2.6, see column #7). This location was even higher at phases that did not involve refinement (average location: 1.2, see column #5); and was a little lower in steps in which a refinement was required (2.8 on average, see column #6). This may be due to the fact that refinement steps include a much larger amount of alternatives. Again it should be noted that results within the Procurement category were better than results within the Inventory category - probably due to the larger database representing Procurement processes. Another reason may be the consecutive nature of procurement processes vs. the loosely coupled business logic of the Inventory processes.

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

| # of refinements | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| % of successful predictions | 12% | 35% | 27% | 12% | 4% | 2% | 2% | 1% | 1% | 3% |
| Cumulative | 12% | 48% | 75% | 88% | 92% | 94% | 96% | 96% | 97% | 100% |

Table 2 analyzes the number of refinements that are needed to design the correct goal activity. For each number of refinements, we record the percentage of cases where this number of refinements was needed. We also record, for each number of refinement $i$, the cumulative percentage of cases where up to $i$ refinements were needed. We observe, for example, that in 88% of the cases the system can reconstruct the goal activity after a maximum of three refinements. These results clearly demonstrate the speed and efficiency of the suggested method. Moreover, in all experiments the refinement process converged into a maximal number of nine refinements in the worst case. As hypothesized earlier- a larger database would probably yield even better results.

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

## 7 Conclusions

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

(3) defining a learning mechanism that will take into account previous designer preferences and adjusting (in real time) the process delineator mechanism.

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

## Acknowledgments

Many thanks to Samia Mazhar and the BPM Group at QUT for providing access to the aviation process data. Also thanks to Roman Kushnarenko for supporting the experiments.

## References

1. Becker, J., Delfmann, P., Herwig, S., Lis, L., Stein, A.: Towards Increased Comparability of Conceptual Models-Enforcing Naming Conventions through Domain Thesauri and Linguistic Grammars. In: ECIS (June 2009)
2. Bhattacharya, K., Gerede, C., Hull, R., Liu, R., Su, J.: Towards formal analysis of artifact-centric business process models. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, p. 288. Springer, Heidelberg (2007)
3. Coalition, W.M.: The workflow management coalition specification - terminology & glossary. Technical report, Technical Report WFMC-TC-1011, Workflow Management Coalition (1999)
4. Golani, M., Pinter, S.S.: Generating a process model from a process audit log. In: van der Aalst, W.M.P., ter Hofstede, A., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 136–151. Springer, Heidelberg (2003)
5. Gschwind, T., Koehler, J., Wong, J.: Applying patterns during business process modeling. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 4–19. Springer, Heidelberg (2008)
6. Hofmeister, C., Kruchten, P., Nord, R.L., Obbink, H., Ran, A., America, P.: A general model of software architecture design derived from five industrial approaches. The Journal of Systems & Software 80(1), 106–126 (2007)
7. Hornung, T., Koschmider, A., Lausen, G.: Recommendation based process modeling support: Method and user experience. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (eds.) ER 2008. LNCS, vol. 5231, pp. 265–278. Springer, Heidelberg (2008)
8. Hull, R.: Artifact-centric business process models: Brief survey of research results and challenges. In: Meersman, R., Tari, Z. (eds.) OTM 2008, Part II. LNCS, vol. 5332, pp. 1152–1163. Springer, Heidelberg (2008)
9. Kumaran, S., Liu, R., Wu, F.Y.: On the duality of information-centric and activity-centric models of business processes. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 32–47. Springer, Heidelberg (2008)
10. Lincoln, M., Golani, M., Gal, A.: Machine-assisted design of business process models using descriptor space analysis. Technical Report IE/IS-2010-01, Technion (March 2010), http://ie.technion.ac.il/tech_reports/1267736757_MachineAssisted_Design_of_Business_Processes.pdf

11. Lincoln, M., Karni, R., Wasser, A.: A Framework for Ontological Standardization of Business Process Content. In: International Conference on Enterprise Information Systems, pp. 257–263 (2007)
12. Muller, D., Reichert, M., Herbst, J.: Data-driven modeling and coordination of large process structures. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, p. 131. Springer, Heidelberg (2007)
13. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. IBM Systems Journal 42(3), 428–445 (2003)
14. Reijers, H.A., Limam, S., Van Der Aalst, W.M.P.: Product-based workflow design. Journal of Management Information Systems 20(1), 229–262 (2003)
15. Schimm, G.: Process miner - a tool for mining process schemes from event-based data. In: Flesca, S., Greco, S., Leone, N., Ianni, G. (eds.) JELIA 2002. LNCS (LNAI), vol. 2424, pp. 525–528. Springer, Heidelberg (2002)
16. Schonenberg, H., Weber, B., van Dongen, B.F., van der Aalst, W.M.P.: Supporting flexible processes through recommendations based on history. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 51–66. Springer, Heidelberg (2008)
17. Van der Aalst, W.M.P., Barthelmess, P., Eliis, C.A., Wainer, J.: Proclets: A framework for lightweight interacting workflow processes. International Journal of Cooperative Information Systems 10(4), 443–482 (2001)
18. van der Aalst, W.M.P., Ter Hofstede, A.H.M.: YAWL: yet another workflow language. Information Systems 30(4), 245–275 (2005)
19. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow mining: a survey of issues and approaches. Data Knowl. Eng. 47(2), 237–267 (2003)
20. Wahler, K., Kuster, J.M.: Predicting Coupling of Object-Centric Business Process Implementations. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, p. 163. Springer, Heidelberg (2008)

# From Informal Process Diagrams to Formal Process Models

Debdoot Mukherjee[1], Pankaj Dhoolia[1], Saurabh Sinha[1],
Aubrey J. Rembert[2], and Mangala Gowri Nanda[1]

[1] IBM Research – India
{debdomuk,pdhoolia,saurabhsinha,mgowri}@in.ibm.com
[2] IBM T.J. Watson Research Center
ajrember@us.ibm.com

**Abstract.** Process modeling is an important activity in business transformation projects. Free-form diagramming tools, such as PowerPoint and Visio, are the preferred tools for creating process models. However, the designs created using such tools are informal sketches, which are not amenable to automated analysis. Formal models, although desirable, are rarely created (during early design) because of the usability problems associated with formal-modeling tools. In this paper, we present an approach for automatically inferring formal process models from informal business process diagrams, so that the strengths of both types of tools can be leveraged. We discuss different sources of structural and semantic ambiguities, commonly present in informal diagrams, which pose challenges for automated inference. Our approach consists of two phases. First, it performs structural inference to identify the set of nodes and edges that constitute a process model. Then, it performs semantic interpretation, using a classifier that mimics human reasoning to associate modeling semantics with the nodes and edges. We discuss both supervised and unsupervised techniques for training such a classifier. Finally, we report results of empirical studies, conducted using flow diagrams from real projects, which illustrate the effectiveness of our approach.

## 1 Introduction

Business Process Models are key artifacts that are created during the early stages of a business-transformation project. A *business process model* depicts how various tasks are coordinated to achieve specific organizational goals. Such models are used to build a consensus among the stakeholders during the requirements-elicitation phase and then drive the subsequent transformation phases. Free-form diagramming tools, such as Powerpoint and Visio, are widely used for creating informal sketches of process models.

On the one hand, these tools are easy-to-use, ubiquitous, offer creative expression, and have a low barrier to adoption. On the other hand, the diagrams created using such tools have no formal underpinnings; therefore, they are not amenable to automated analysis—*e.g.,* for model checking, process improvements, process

R. Hull, J. Mendling, and S. Tai (Eds.): BPM 2010, LNCS 6336, pp. 145–161, 2010.

reuse, and bootstrapping process realization. Unlike the free-form diagramming tools, formal process-modeling softwares offer many such benefits, but suffer from a high barrier to adoption; this occurs for different reasons, such as complexity, costs, and the requirement of some level of formal training. Empirical studies reveal that the authoring constraints imposed by formal-modeling tools have generated mixed reactions from designers and have resulted in limited adoption of the tools [10].

To take advantage of the merits of free-form diagramming and yet leverage the benefits of formal modeling, automated techniques for converting informal sketches to formal process models are essential. A manual approach can be tedious and error-prone, especially when enterprises want to harvest formal models from a large corpus of legacy flow diagrams—a scenario that is attracting increasing interest.

Diagramming tools offer a rich collection of shapes (known as *stencils*) from which designers freely choose depictions for process flow entities, such as activities, events, gateways, etc. Many existing formal-modeling tools (*e.g.,* Websphere Business Modeler[1] (WBM), ARIS,[2] System Architect,[3] and Lombardi[4]) offer, to various degrees, capabilities to import informal diagrams, such as Visio diagrams. They perform mainly a shape-based transformation, using fixed or pluggable mappings from names of drawing shapes in stencils to process modeling entities. This is inadequate because often the same shape is used to represent different semantics. Further, these tools are not able to interpret diagrammatic cues (*e.g.,* dangling connectors) commonly used in describing the flow connections; therefore, they identify imprecise flow structures.

To address the limitations of existing tools, we present an automated approach for extracting formal process models, that conform to a given target metamodel,[5] from informal process-flow diagrams. It consists of two phases: a structure-inference phase and a semantic-interpretation phase. In the first phase, the approach precisely infers the flow-graph structure of a diagram, in terms of nodes and edges (*i.e.,* flow elements). It identifies each shape, or set of shapes, that could correspond to a flow node. Next, it uses a novel edge-inference algorithm to trace the lines between the identified nodes and infer the set of directed edges in the diagram. In this phase, the approach also infers the association of unlinked texts to appropriate flow elements. In the second phase, our approach annotates each node and edge with a process-modeling semantic, defined in the target metamodel. To perform the annotation, we use pattern classification. Specifically, we use a classifier trained on relational, geometric, and textual

[1] http://www.ibm.com/software/integration/wbimodeler/advanced/features/

[2] http://www.ids-scheer.com/en/ARIS_ARIS_Platform/3730.html

[3] http://www.ibm.com/software/awdtools/systemarchitect/

[4] http://www.lombardisoftware.com/enterprise-bpm-software.php

[5] A target metamodel lists a set of process modeling elements. For example, the Business Process Modeling Notation (BPMN) defines a metamodel comprising activities, gateways, events, swimlanes, artifacts and connecting objects. See http://www.omg.org/spec/BPMN

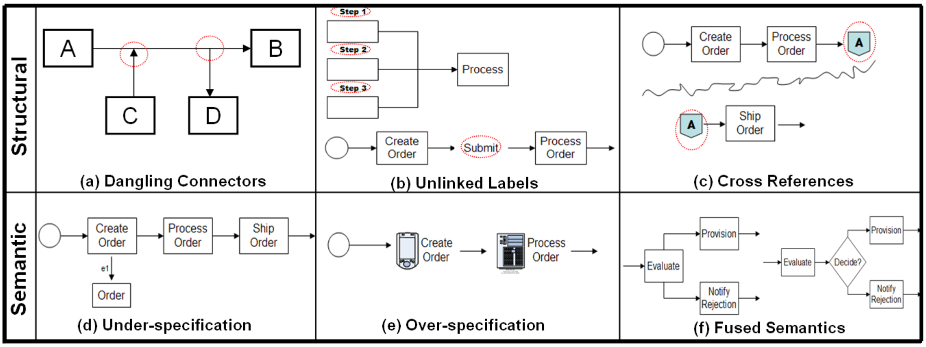


**Fig. 1.** Common structural and semantic ambiguities in process flow diagrams

features of flow elements to perform semantic disambiguation. We present both supervised and unsupervised approaches for training such a classifier.

To evaluate the effectiveness of our approach, we implemented it in a tool called *i*DISCOVER that infers formal models from Visio diagrams. We conducted empirical studies using flow diagrams taken from real business-transformation projects. Our results illustrate that, for the diagrams considered, *i*DISCOVER infers formal models with high precision and recall, and outperforms existing commercial tools. Our results also indicate that most standard classifiers are applicable for interpreting process semantics with our feature space modeling. Interestingly, an unsupervised clustering approach, which may be used in practical settings where training data is unavailable, also proves nearly as effective as supervised ones.

The main benefit of our approach is that it automates, with a high degree of accuracy, a transformation task that is tedious to perform manually. In doing so, the approach enables process engineering to leverage the strengths of both free-form diagramming tools and formal-modeling tools. More importantly, such a facility can help greater industrial adoption of formal methods developed in BPM research—currently the unavailability of formally specified process models in enterprises proves to be an impediment in applying such research.

The main contributions of this work are

- The development of a novel end-to-end approach for converting informal flow diagrams to formal process models, addressing both structural and semantic ambiguities that are commonly present in the informal diagrams.
- The implementation of the approach in a tool called *i*DISCOVER that converts Visio process flow diagrams to BPMN process models.
- An empirical evaluation, which demonstrates effectiveness of the approach.

## 2 Diagram Interpretation Challenges

Figure 1 presents some ambiguities, which present challenges in interpreting the structure and semantics of flow models. Our empirical study (Section 4) shows that such ambiguities are indeed common in real process diagrams.

### 2.1 Structural Ambiguities

Identifying the correct set of drawing shapes corresponding to a node or an edge is hard when, for example, the edges are not properly connected to nodes or multiple lines are connected to form a single flow. The top part of Figure 1 illustrates three common structural ambiguities.

*Dangling Connectors.* Existing tools can recognize a line to be an edge only if the line is properly glued[6] at both ends of two 2D shapes. However, users can join multiple lines to represent a single edge. Moreover, the endpoints of a line may be left dangling (*i.e.,* not be properly glued). In Figure 1(a), four edges exist: $(A, B)$, $(A, D)$, $(C, B)$, and $(C, D)$. But, existing tools can recognize only $(A, B)$ because it is the only properly glued edge.

*Unlinked Labels.* People often use separate drawing shapes to specify a flow element and its text label. In Figure 1(b), `Submit` is intended to be a label on the edge from `Create Order` to `Process Order`. Label association becomes a challenge when nearness alone does not suffice to tie unlinked texts with shapes identified as flow elements. Tracking patterns of text label usage may help—for example, if text labels are consistently placed on the top of shapes (*e.g.,* as illustrated by the `Step x` labels in Figure 1(b)), we can apply that pattern to resolve ambiguous cases.

*Cross-references.* Cross-reference linkages across diagrams are often required to split a large diagram across pages for convenience, as shown in Figure 1(c). Use of cross references can occur within a single page as well.

### 2.2 Semantic Ambiguities

Inferring the semantics for flow elements is straightforward if each drawing shape is used consistently to convey a single modeling semantic. However, in practice, the following scenarios are extremely common and pose challenges for semantic interpretation.

*Under-specification.* This occurs when different instances of the same shape are used to convey different semantics. For example, in Figure 1(d), a rectangle is used to denote both the output data artifact `Order` and the step `Create Order`. In general, under-specification lowers the effectiveness of a simple shape-based mapping of diagram elements to process model entities.

*Over-specification.* This occurs when the same semantic is being conveyed by different shapes. In Figure 1(e), both `Create Order` and `Process Order` are activities, but are represented using different graphics. Over-specification too tends to reduce the usefulness of shape-based mapping: the number of shapes to be enumerated by such approaches can become prohibitively large.

[6] Most diagramming formats support a notion of proper connection or glue. On clicking a connector in Visio, the endpoints appear red or green depending upon whether they are glued or dangling.

*Fused Semantics.* In Figure 1(f), the two flow fragments are semantically equivalent. The left fragment has an `Evaluate` block that represents a fusion of a task and a decision. In the fragment on the right, `Evaluate` and `Decide` are separate entities. Automatic interpretation of such fused semantics is difficult.

## 3 Automated Process Model Discovery

Our approach consists of two phases: *structural inference* and *semantic interpretation.* The structural-inference phase takes as input a *flow diagram*, and extracts a *flow graph*, which consists of nodes and edges. Additionally, the first phase computes information, such as structure, geometry, and text, for each flow element (*i.e.,* node and edge). The second phase of the algorithm constructs the *process model* from the flow graph by associating modeling semantics with each flow element using pattern classification. Specifically, this phase applies a classifier that, based on the relational, geometric, and textual features of the flow elements, performs semantic disambiguation. The resulting process model is well-defined in terms of both structure and semantics, and thus, can be formally analyzed.[7]

### 3.1 Structure Inference

The goal of the first phase is to infer the flow graph nodes and edges. It does this in three steps. First, it parses the input flow diagram to identify the basic diagram elements, which consists of shapes, lines, and text. Second, it constructs the set of nodes, selects candidate edges from diagram elements, and determines associations of text with nodes and candidate edges. Finally, this phase applies an edge-inference algorithm to compute the flow.

**Diagram Element Extraction.** An informal flow diagram is a collection of diagram elements such as: (1) *shapes*, which are candidates for nodes, (2) *lines*, which are candidate edges, and (3) *text*, which may be used to label either nodes or edges. We identify the diagram elements by parsing XML representations of diagrams or using tool specific APIs. For each diagram element, we extract information about its coordinates, dimensions, text, geometry, and group (if any) in the diagram; this information is used for node and edge discovery. All coordinates are expressed in some standard unit with respect to the origin of the page, which we consider to be the bottom, left corner.

Some properties, such as text labels and arrowheads for lines, may be readily available if they are linked to drawing shapes. If not, they need to be inferred later as part of flow element discovery. For a drawing shape that is taken from a stencil, the geometry is identified by the shape name as per the stencil. For a manually created shape, the geometry is encoded by the counts of different types of lines (*e.g.,* straight lines, elliptical arcs, bezier arcs) used to form the shape.

[7] An XML serialization of the annotated flow graph can be easily transformed to conform to specific XML process-modeling schemas, such as BPMN 2.0 and the WBM XML schema.

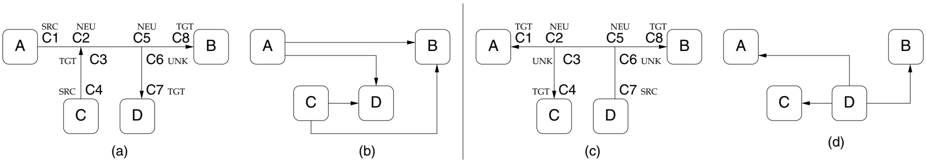


**Fig. 2.** Examples to illustrate flow-edge inference

The *group* feature in diagramming tools, which lets a user tie related diagram elements together, often conveys important structural cues. Therefore, for each diagram element, we identify elements that are grouped with it.

**Flow Element Discovery.** The second step of structure inference discovers flow elements: nodes, candidate edges, and associations of text with nodes and candidate edges. It uses the following heuristics to perform the element discovery.

1. Ignore elements, such as bounding boxes, title bars, legends, etc., that are close to page boundaries and do not have possible connections.
2. Trace undirected connected lines that form closed paths to form new nodes.
3. Recognize arrowheads that are depicted as separate shapes and associate them with nearby lines.
4. Identify shapes that are near possible connectors as nodes.
5. Lines whose endpoints lie near nodes or other lines are recursively identified to be part of candidate edges.
6. Identify as nodes, shape groups that have connectors emanating from their boundaries and those that do not have a possibility of a flow within them.
7. *Label association:* Text in unbordered shapes are taken as labels of the nodes or edges with which they are grouped, or as labels of their nearest node or edge.

**Flow-Edge Inference.** The key step of structural inference is the identification of directed connections between the nodes: that is, we need to list possible source and target nodes for each candidate edge discovered from the diagram elements.

The edge-inference algorithm uses the concept of a connection point. A *connection point* is the notional point of connection at which the endpoint of a line connects to a node or an intermediate point on another line. The intersection of the endpoint of a line $l$ with an intermediate point on another line $l_x$ generates a pair of connection points—one at the end of $l$ and the other at the intersection point on $l_x$. To illustrate, consider the flow graph shown in Figure 2(a), which has eight connection points (labeled $C1$–$C8$). The line $(A, B)$ has connection points $C1$ and $C8$ at its endpoints; it also has two connection points, labeled $C2$ and $C5$, at its intersections with the lines from $C$ and $D$. We track the set of connection points with each line and each node.

A connection point has five attributes:

1. An end point $C_{ep}$ to mark the coordinates
2. An associated line $C_{line}$ on which it is either an endpoint or an intermediate intersection point

```
algorithm InferEdges
 input   $W_n, W_e$  nodes and candidate edges
 output  $edgeList$  inferred flow edges
begin
    // create CP for line-node junctions
    foreach $l \in W_e$ and endpoint $ep$ do
        foreach node $n \in W_n$ do
            if $n$ is nearby $ep$ then
                create connection point $C$:
                        $C_{ep} = ep, C_{line} = l, C_{conn} = n$
                        $C_{isEnd} = \texttt{true}, C_{dir} = \text{SRC} \mid \text{TGT}$
                add $C$ to CP sets for $l$ and $n$
    // create CP for intersection of lines
    foreach $l \in W_e$ and endpoint $ep$ do
        foreach $l_x \in W_e$ do
            if $l$ is nearby $ep$ then
                create connection point $C_x$:
                        $C_{x,ep} = ep, C_{x,line} = l_x, C_{x,conn} = C$
                        $C_{x,isEnd} = \texttt{false}, C_{x,dir} = \text{NEU}$
                add $C_x$ to CP set for $l_x$
                create connection point $C$:
                        $C_{ep} = ep, C_{line} = l, C_{conn} = C_x$
                        $C_{isEnd} = \texttt{true}, C_{dir} = \text{TGT} \mid \text{UNK}$
                add $C$ to CP set for $l$
    // connect nodes
    foreach node $n \in W_n$ do
        foreach conn. point $C$ of $n$ do
            $nodeSet = \emptyset$; $visit(C) = \texttt{false}$
            propagateDirection($C_{dir}, C_{line}, nodeSet$)
            foreach $n' \in nodeSet$ do
                if $C_{dir} = \text{SRC}$ then
                    add $(n, n')$ to $edgeList$
                else add $(n', n)$ to $edgeList$
end

procedure propagateDirection($dir, l, ns$)
begin
    foreach conn. point $C$ of $l$ do
        if $visit(C)$ then continue
        $visit(C) = \texttt{true}$
        if $C_{isEnd} \land \texttt{match}(C_{dir}, dir)$ then
            continue
        if $\neg C_{isEnd} \land \neg\texttt{match}(C_{dir}, dir)$ then
            continue
        if $C_{conn}$ instanceof Node then
            add $C_{conn}$ to $ns$
        else if $C_{conn}$ instanceof ConnPt then
            $C_x = C_{conn}$; $l_x = C_{x,line}$
            if $\neg\texttt{match}(C_{x,dir}, dir)$ then continue
            propagateDirection($dir, l_x, ns$)
end

function match($interDir, destDir$)
begin
    if $interDir = destDir$ then
        return true
    else if $interDir = \text{NEU} \lor \text{UNK}$ then
        return true
    return false
end
```

**Fig. 3.** The edge-inference algorithm

3. A boolean value $C_{isEnd}$ indicating whether it is an endpoint or an intermediate intersection point
4. An associated node or connection point $C_{conn}$ on which it is an endpoint
5. A direction $C_{dir}$, which can be SRC (source), TGT (target), NEU (neutral), or UNK (unknown)

The direction of a connection point $C$ is found using the following rules

1. If $C$ lies at the junction of a node and a line:
   $C_{dir} = \text{TGT}$, if there is an arrowhead at $C_{ep}$
   $C_{dir} = \text{SRC}$, otherwise
2. If $C$ is at the junction of the endpoint of a line $l$ and an intermediate point on line $l_x$:
   $C_{dir} = \text{NEU}$, if $C$ lies on $l_x$ (*i.e.*, $C_{isEnd} = \texttt{false}$)
   $C_{dir} = \text{TGT}$, if $C$ lies on $l$ and there is an arrowhead at $C_{ep}$
   $C_{dir} = \text{UNK}$, if $C$ lies on $l$ and there is no arrowhead at $C_{ep}$

Figure 3 presents the edge-inference algorithm: `InferEdges`. The algorithm takes as inputs the set of nodes and candidate edges in the flow graph, and returns as output the inferred edges. Intuitively, the algorithm traverses the flow graph, starting at a connection point on a node, and identifies reachable nodes such that the directions encountered during the traversal are consistent.

Lines 1–5 of the algorithm create connection points for the junctions of lines and nodes. For each line $l$ and each endpoint $ep$ of $l$, the algorithm iterates over the nodes (lines 1–2). It uses thresholds for *nearness*[8] to determine whether a node $n$ could be connected to $ep$ (line 3). It then creates a connection point $C$, with the appropriate attributes, and adds $C$ to the set of connection points for $l$ and $n$. The direction of $C$ is set to SRC or TGT depending on whether an arrowhead occurs at $ep$. Similarly, lines 6–12 of `InferEdges` create connection points for the intersection of two lines $l$ and $l_x$. In this case, two connection points ($C_x$ and $C$) are created. $C_x$ is created for the intermediate intersecting point on $l_x$, whereas $C$ is created for the endpoint of $l$.

After creating the connection points, `InferEdges` connects the nodes by traversing the flow graph starting at each node $n$ and following each connection point on $n$ (lines 13–16). After the traversal for a connection point is complete, appropriate edges are created between $n$ and the nodes reached during the traversal (lines 17–20).

Procedure `propagateDirection` traverses the flow graph, along paths in which the directions are consistent, and identifies the reached nodes. Given a line $l$ and direction $dir$, the procedure processes each connection point $C$ of $l$ that has not been previously visited (lines 21–22). If $C$ is an endpoint and $C_{dir}$ and $dir$ match—*i.e.,* either both are SRC or both are TGT—the procedure abandons the traversal as the path does not represent a valid edge (lines 24–25). If $C$ is an intermediate point, the traversal terminates if both of the following conditions hold: (1) $C_{dir}$ does not match $dir$ and (2) $C_{dir}$ is neither NEU nor UNK (lines 26–27). If $C$ occurs at a node, the procedure has found an edge; therefore, it adds the node to the set of nodes (lines 28–29). Alternatively, if $C$ occurs at an intermediate intersection, the algorithm continues traversing if the directions are consistent (lines 30–33).

To illustrate the steps of the algorithm, consider the traversal performed by `InferEdges`, starting at connection point $C1$ in Figure 2(a). `InferEdges` processes each connection point on line $(A, B)$ (line 21 of `propagateDirection`). We illustrate the processing of $C2$ and $C5$. In both cases, the condition at line 24 evaluates `false` because $C2$ and $C5$ do not occur at endpoints: the condition at line 26 evaluates false because their $dir$ = NEU (which causes line 36 to evaluate `true`); the condition on line 28 evaluates `false` as well; but line 30 evaluates `true`. In the case of $C5$, the condition on line 32 evaluates to `false` and `propagateDirection` is invoked recursively (line 33) to continue the traversal toward node $D$. In this invocation, the algorithm reaches $C7$, whose direction is TGT (*i.e.,* does not match $C1_{dir}$); thus, a valid edge is detected. Therefore, it adds $D$ to the node set (at line 29), and later adds $(A, D)$ to the edge list (at line 19). However, the line towards $C$ is not traversed since $C3_{dir}$ = TGT, which causes the condition on line 32 to evaluate to `true` during the processing of $C2$. Note that for the traversal starting at $C7$, `InferEdges` reaches $C6$, where $isEnd$ = `true` and direction is UNK which causes line 24 to evaluate `true` and the traversal stops. This ensures

[8] Such nearness bounds are set relative to the width and height over which diagram elements span in a page and are thus scale invariant.

that the edge $(A, D)$ does not get added twice in the edge set. The final set of edges for the flow graph is shown in part(b) of Figure 2. Parts (c) and (d) of the figure illustrate a different example in which the line from $D$ is split into three edges, one each to nodes $A$, $B$ and $C$.

In this manner, lines get converted to edges that have sources and targets. For each edge, we also obtain its text label, color, and line type by aggregating these attributes for all lines that constitute it.

**Cross Reference Resolution:** We identify nodes that have the same text label and exist in different pages of a drawing to be *cross references* if either their indegree or their outdegree is zero. We reduce the flow graph by fusing such nodes and merging the incoming and outgoing edges on the fused nodes.

### 3.2 Semantic Interpretation of Flow Elements

Phase 1 of our approach infers a well-formed graph, which has none of the structural ambiguities present in the flow diagram from which it was inferred. Next, Phase 2 associates semantics with the nodes and edges in the graph, based on similarity of the nodes and edges. Semantic similarity of nodes and edges quite often follows from similarity in their geometry, relational attributes, and textual content. We formulate semantic disambiguation as a pattern-classification problem [5]. Using a representative corpus of business process diagrams, we train a classifier to learn patterns in features of flow elements that indicate the class of process semantic of an element. For semantic interpretation of new diagrams, we extract pertinent features for each flow element and feed them into the trained classifier, which detects learnt patterns to decide process semantics. We discuss both supervised and unsupervised schemes for learning that may be used depending upon whether a corpus of diagrams is available for training or not.

**Feature Extraction.** We attempt to mimic human reasoning used in recognizing process semantics from a diagram. Humans usually analyze a range of visual and textual cues to understand diagram semantics. We abstract such cues as symbolic or numeric features that can be acted upon by standard classifiers.

Table 1 lists a set of features for nodes, grouped into three categories: relational, geometric, and textual. For each group, the table lists examples of features (column 2), and discusses how the features are indicative of process semantics in nodes (column 3). (A similar list can be formulated for edges; for space constraints, we do not present it.) Relational features such as indegree and outdegree can be obtained directly from the extracted flow graphs, whereas geometric and textual features are aggregated from attributes of the diagram elements involved in the flow. For each process entity, a set of *cue words* that characterize expressions in the labels for the entity is taken to be a textual feature. For example, interrogative words (*e.g.,* “Whether,” “Is,” “Does”) in the text are typical of a *gateway*; similarly, text starting with strong verbs (*e.g.,* “Create,” “Process”) indicate an *activity*. If training data is available, we can perform text classification on labels to identify such representative words for each target entity; otherwise, these word lists have to be created with inputs from human experts.

**Table 1.** Features used for disambiguating node semantics

| Category | Features | Comments |
|---|---|---|
| Relational | No. of incoming edges (*indegree*), no. of outgoing edges (*outdegree*), no. of nodes contained within (*numContains*), whether it is contained in another node (*isContained*) | Can discriminate amongst many entities irrespective of local styles in diagrams. For example, *indegree* and *outdegree* can easily distinguish between start, end and intermediate events; non-zero *numContains* may strongly indicate presence of a *swimlane* or a *group*. |
| Geometric | Shape name in stencil, No. of vertical lines, no. of horizontal lines, no. of arcs, line style, width, height | Can provide highly accurate insight, if data sets follow templates very rigorously. Such features can work well with small sets of process diagrams with uniform styles per entity. |
| Textual | No. of cue words for every entity in label for the node and labels for incident edges | Humans comprehend text to determine semantics in highly ambiguous scenarios. For example, text in outgoing edges from *gateways* is often 'yes'/'no'/'y'/'n', text in *activities* typically starts with strong verbs, 'report' and 'e-mail' are common in *data objects*. |

**Supervised Learning.** If we have a set of diagrams for which the corresponding correct process models are known, we can train a classifier, in a supervised manner, to learn classification rules from the labeled instances. The learnt rules can be used to infer the semantics of new diagrams. A decision tree learner [14] can formulate a decision task as a sequence of logical or binary operations from a series of examples. It is a set of if-then-else like classification rules over the feature set, which can be easily interpreted (and edited if required) by data mining practitioners. A Naïve Bayes classifier [11], after training on a labeled dataset, can assign probabilities that an observation (flow element) belongs in each class (process entity). Neural networks [13] consist of layers of interconnected nodes where each layer produces a non-linear function of its input, thus enabling the modeling of very general functions. Our empirical study evaluates different classifiers for their efficacy in choosing process semantics for flow elements.

**Unsupervised Learning.** Clustering is a popular and effective technique used in data mining for discovering, without any human supervision, patterns from large amounts of data [12]. We cluster flow elements based on their geometrical, relational, and textual features, and hypothesize that elements with identical process semantics get grouped into the same cluster. Next, we consider the cluster assignments as class labels for the flow elements and train a classifier. The classifier trained in this manner can perform semantic disambiguation—eliminating the need for performing clustering on each new diagram.

We define a measure of similarity (or distance) such that flow elements in the same cluster exhibit greater similarity in semantics amongst them than with elements in any other cluster. We compute similarity for each feature category: relational ($sim_r$), geometric ($sim_g$), and textual ($sim_t$). We use the euclidean distance to compute similarity between numeric attributes, a boolean measure (1 for match, 0 for mismatch) for attributes that can be enumerated (*e.g.,* shape, name, color), and string edit distances (*e.g.,* Levenshtein, Monge Elkan, Jaro) [4] for text. The aggregate feature-based similarity of two flow elements, $f_i$ and $f_j$, is obtained using a linear combination of the three similarity components:

$$sim(f_i, f_j) = w_r \times sim_r(f_i, f_j) + w_g \times sim_g(f_i, f_j) + w_t \times sim_t(f_i, f_j)$$

The weights for the different components can be set either using domain knowledge about the importance of different aspects of the similarity measure, or, alternatively, by validation over a set of labeled training instances (if available). Given the aggregated similarity measure, there are various clustering approaches, such as agglomerative, divisive, and k-means, for iteratively improving the clustering goodness. The choice of the number of clusters may be governed by a knowledge of the number of entities in the target meta-model. After clustering is run, the user can examine a few exemplars flow elements in each cluster to decide a process semantic for the cluster. Then, the semantic classification (thus obtained via clustering) of flow elements from the training corpus is used to train a classifier, and semantic interpretation proceeds as in the supervised case. Our empirical studies show that clustering on features similar to those listed in Table 1 indeed groups together elements with common process semantics, and that an unsupervised approach is almost as effective as supervised learning for recognizing certain semantics. In practice, an unsupervised approach is often more applicable because sound training data is hard to get.

## 4 Empirical Evaluation

We implemented our approach in a tool called *i*DISCOVER, and conducted empirical studies to evaluate its accuracy and compare it with the accuracy of a commercial tool. We organized the evaluation to report results on both aspects of model discovery: structure inference and semantic interpretation.

**Experimental Setup.** As experimental data, we used a set of 185 Visio process diagrams[9] created as part of real business-transformation projects. We randomly selected the diagrams from a repository of archived projects within IBM's Service Delivery practice. Our selection was not constrained by the stencil prescription of any specific tool. For comparison with *i*DISCOVER, we selected a top commercial process-modeling tool. The Visio-import capabilities of all such tools recognize diagram shapes as process elements only if the shapes come from specific Visio stencil(s) prescribed by the tools. Moreover, they cannot resolve structural ambiguities, such as dangling connectors and unlinked labels. The representative tool that we selected offers support for the largest number of Visio stencils. For proprietary reasons, we refer to the tool as PMT(Process Modeling Tool).We employed human experts to identify the true process models for the diagrams and used them to compute the accuracies of *i*DISCOVER and PMT.

We use *precision* and *recall* to measure accuracy. In the following equations, *Actual* is the set of (manually identified) correct interpretations and *Retrieved* is the set of (automatically inferred) interpretations by a tool.

$$Precision(P) = \frac{|Actual \cap Retrieved|}{|Retrieved|}, Recall(R) = \frac{|Actual \cap Retrieved|}{|Actual|}$$

[9] `https://researcher.ibm.com/researcher/files/in-debdomuk/bpm_dataset.zip`

**Table 2.** Occurrence of structural ambiguities in the dataset

| Ambiguity | Instances per File | | % Files with |
|---|---|---|---|
| | High | Average | Ambiguity |
| Dangling Conn. | 47 (100%) | 3 (14%) | 56 |
| Unlinked Labels | 46 (39%) | 2 (3.7%) | 38 |
| Cross References | 10 | 1 | 35 |

**Table 3.** Accuracy of structural inference by *i*DISCOVER and PMT

| Element | *i*DISCOVER | | PMT | |
|---|---|---|---|---|
| | Precision | Recall | Precision | Recall |
| Node | 96.93 | 95.91 | 70.44 | 86.29 |
| Edge | 93.26 | 90.86 | 63.43 | 59.87 |

### 4.1 Study 1: Structure Inference

**Goals and Method.** The goal of the first study was to compare the accuracies of the structural inference performed by *i*DISCOVER and PMT. We also evaluated how the structural ambiguities present in the dataset impact structure inference in both cases. We ran *i*DISCOVER's structure extractor to infer XML flow graphs from the Visio diagrams and compared them with actual process models, manually identified by human experts, to measure accuracy. First, we matched the text labels for nodes in *Retrieved* with those in *Actual*. Next, we traced the recovered edges between matched nodes and checked if equivalent edges were present in *Actual* models. Finally, we computed precision and recall for both node and edge detection. Similarly, we validated the flow graphs produced by PMT against the actual process models. *i*DISCOVER also reports the number of dangling connections, unlinked labels, and cross references found in the input.

**Results and Analysis.** Table 2 lists, for each type of structural ambiguity, the highest and average number of instances found in the files in the dataset. It also reports the percentage of edges that are dangling, percentage of nodes and edges that have unlinked text labels, and percentage of files that have at least one instance of the ambiguity. Over half of the files contain dangling connectors, whereas unlinked labels and cross references occur in over a third of the files in the dataset. The fact that 100% of connectors in some files were left dangling suggests that certain users may be completely unaware of notions such as proper gluing of connectors. The data indicate that structural ambiguities can occur frequently in practice; therefore, to be useful, an automated inference technique must handle them effectively.

Table 3 shows the average precision and recall of node and edge detection. The data illustrate that *i*DISCOVER performs much better than PMT. For node inference, *i*DISCOVER had $\approx 27\%$ higher precision and $\approx 9\%$ higher recall. For edge inference, the performance of *i*DISCOVER was even better: it achieved $\approx 30\%$ improvement in both precision and recall. We observed far greater correlation of the number of dangling connectors with the edge recall of PMT (Pearson's coefficient, $\rho = -0.48$) than that with the edge recall of *i*DISCOVER ($\rho = -0.08$). This clearly suggests that while *i*DISCOVER successfully resolves dangling connectors, PMT's edge-inferencing capability is adversely affected by their presence.

**Discussion.** Overall, the study shows that *i*DISCOVER consistently performs better than PMT in terms of both precision and recall. The data also indicate

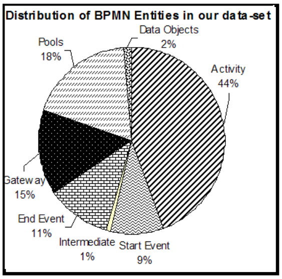


Fig. 4. Distribution of BPMN entities

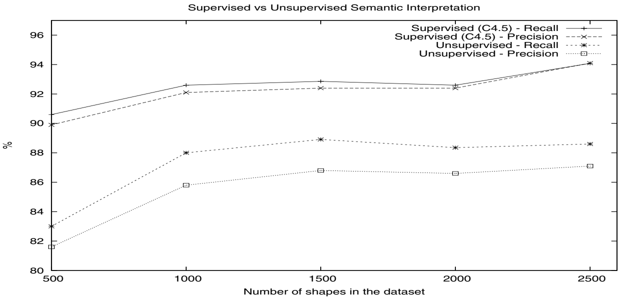


**Fig. 5.** Effects of varying the dataset size

that structural ambiguities, which complicate automated structure inference, can occur frequently in practice; therefore, an approach, such as ours, that effectively deals with such ambiguities can be valuable.

We investigated the reasons for errors in node detection in both tools, and observed the following reasons: (1) imprecise resolution of unlinked labels in ambiguous scenarios where nearness does not suffice; (2) failure to recognize some shapes when a group of diagram shapes represent a single node. An edge is taken to be accurate only if its source and target nodes are correctly inferred. Thus, although our edge-inference algorithm identifies edges precisely, the overall precision of edge inference suffers from inaccuracies in node detection.

### 4.2 Study 2: Semantic Interpretation

**Goals and Method.** The goals of the second study were to (1) evaluate the effectiveness of different pattern-classification techniques in assigning semantics to flow elements, and (2) compare the effectiveness of these techniques with that of PMT. Specifically, we evaluated three supervised classification techniques—C4.5 decision tree [14], Naïve Bayes, and Multi-layer perceptron (MLP) neural network [13]—and an unsupervised clustering technique.

We asked human experts to create BPMN models for the 185 diagrams in our dataset. To compute accuracy, we compared the semantic interpretations of the pattern classifiers and PMT against those made by the experts.[10] We chose to evaluate interpretation of node semantics only because BPMN flow-edge semantics can be resolved unambiguously by applying simple rules.[11]

To construct the training dataset for the supervised classifiers, we compared the nodes in the flow graphs extracted by *i*DISCOVER with the nodes in the

[10] Note that an expert's interpretation may differ from the actual intent of the designer in highly ambiguous scenarios. Nevertheless, we consider the expert's judgment to indicate true semantics.

[11] An edge that cuts across two *pools* is a *message flow*; an edge that exists between two nodes in the same *pool* is a *sequence flow*; an edge whose source or the target is an *artifact* is an *association*.

**Table 4.** 10-fold cross-validation results of semantic interpretation by *i*DISCOVER and PMT

| Class | Supervised | | Unsuper. | | PMT | |
|---|---|---|---|---|---|---|
| | P | R | P | R | P | R |
| Activity | 92.6 | 91.0 | 89.8 | 88.5 | 66.1 | 84.4 |
| Start Event | 82.1 | 91.0 | 77.4 | 84.9 | 18.4 | 24.2 |
| Intermediate | 14.3 | 6.7 | 0 | 0 | 0 | 0 |
| End Event | 83.6 | 84.7 | 71.7 | 87.9 | 26.5 | 35.1 |
| Gateway | 96.7 | 97.0 | 90.0 | 97.9 | 93.3 | 92.0 |
| Pool | 100 | 100 | 99.1 | 92.4 | 76.6 | 87.7 |
| Data Object | 56.5 | 57.8 | 0 | 0 | 0 | 0 |
| Overall | 91.9 | 92.1 | 87.1 | 88.4 | 60.2 | 73.7 |

**Table 5.** 10-fold cross-validation results for the three supervised classifiers

| Class | C4.5 | | Naïve Bayes | | MLP | |
|---|---|---|---|---|---|---|
| | P | R | P | R | P | R |
| Activity | 92.6 | 91.0 | 90.5 | 81.5 | 90.9 | 91.5 |
| Start Event | 82.1 | 91.0 | 81 | 78.6 | 84.4 | 83.6 |
| Intermediate | 14.3 | 6.7 | 14.3 | 53.3 | 75 | 20 |
| End Event | 83.6 | 84.7 | 75.9 | 83.8 | 78.5 | 84.3 |
| Gateway | 96.7 | 97.0 | 91.2 | 96.2 | 96.2 | 97 |
| Pool | 100 | 100 | 100 | 92.7 | 99.6 | 99.8 |
| Data Object | 56.5 | 57.8 | 28.4 | 73.3 | 60 | 40 |
| Overall | 91.9 | 92.1 | 88.9 | 85.6 | 91.2 | 91.4 |

expert-created BPMN models: we obtained 2943 matches, which formed the training set. For each node, we aggregated different features listed in Table 1. The training set contained seven classes of BPMN entities labeled by the experts; Figure 4 shows the distribution of these entities. Therefore, for the three supervised classifiers, we set up a 7-class classification problem using the *Weka* toolkit.[12] Further, to study the effects of the training-set size on the classification accuracy, we chose the best-performing classifier and experimented with different sizes of the training set.

In the unsupervised case, we performed classification via clustering. We ran $k$-means clustering using the similarity measures discussed in Section 3.2. We computed pairwise similarity between the nodes and fed the similarity matrix to a graph clusterer, which produced clusters representative of process semantics. Then, a user decided the class of BPMN semantics (out of the seven classes) for each cluster by studying a few exemplars in each cluster. Next, for each node, we compared the process semantic assigned to its cluster and that assigned by human experts to compute precision and recall. We also studied the classes of semantics that come up in new clusters as we increase $k$ in different runs of the $k$-means clusterer. Finally, as in the case for supervised classifiers, we investigated the effects of varying the dataset size on clustering results.

**Results and Analysis.** To quantify the degree of semantic ambiguities present in our dataset, we measured under-specification and over-specification. We found that 79.8% of the nodes were touched by under-specification in the sense that they were represented by shape types whose instances referred to at least two forms of semantics. We also found each class of BPMN entity to be over-specified; that is, it was represented by more than one shape in the dataset.

Table 4 reports 10-fold cross-validation results (of precision and recall) over the training dataset for the best-performing supervised classifier and the unsupervised classifier; it also presents the precision and recall results for PMT. We find that the overall precision and recall of both pattern-classification approaches are $\approx 90\%$ (within 5% of each other); and they are over $\approx 20\%$ higher than that

[12] http://www.cs.waikato.ac.nz/ml/weka/

of PMT. We observe that the classification approaches fare well in recognizing most BPMN entities present in the dataset except for intermediate events and data objects, which together constituted only 3% of the dataset. Top discriminating features were noted to be: *isContained*, *indegree*, *outdegree*, and *shape name*. PMT could detect only gateways and pools with high precision.

Table 5 reports the accuracy results for the three supervised classifiers. We find that C4.5 decision tree performs the best, MLP neural network is almost as effective, and Naïve Bayes is less effective by a few percentage points. Figure 5 evaluates both supervised and unsupervised approaches on a constant test dataset of 443 nodes (not part of the training data), when the training dataset size is varied from 500 through 2500. We find that the variation in results between the least and the greatest sizes of the training set is as low as $\approx$ 4%. Moreover, the curves show less than 2% deviation as we increase the size beyond 1000 nodes. For a training set with a balanced distribution of entities, the results for the fringe entities improved considerably but the overall results dropped by a few points.[13] In the unsupervised case, for $k = 5$, we obtained clusters that represented activity, pool, gateway, start event, and end event. However, on increasing $k$ beyond 5, we observed that the new clusters represented new process semantics, such as join/merge, fused merge, branch points, etc., but they did not isolate data objects or intermediate events, which are part of our target set.

**Discussion.** Figure 5 indicates that the classification approach could work just as well with only a third of our current dataset size. However, for such approaches to succeed, the input set should have a balanced distribution of target model entities. The results show that clustering is indeed effective in grouping together related process semantics. The relational features and shape name were found to be the most discriminative features. The set of cues words for gateways emerged as the top textual feature. We realize that textual features need to be modeled more effectively, *e.g.,* by capturing the relative position of words within a sentence, performing parts-of-speech tagging, and using WordNets.

## 5 Related Work

Although informal expressions of business process designs are extremely common, (to the best of our knowledge) there is no existing research that addresses the problem of automatically understanding process diagrams. Moreover, there have been no studies of the challenges that arise in interpreting diagrams created using stencil-based tools, such as Visio, Powerpoint, and Dia.

However, there exists a large body of work in the area of understanding line drawings and hand sketches (*e.g.,* [1,3,9,15]). The primary problem addressed by sketch recognition is that of identifying various shapes present in a diagram; semantic interpretation follows directly from a fixed mapping between the

[13] Intermediate event (P:64%, R:44%); Data Objects (P:80%, R:90%); Overall (P:88%, R:88%). Reference [17] provides the details of this experiment and illustration of *i*DISCOVER and PMT outputs.

geometry of source shapes and the semantics in the target metamodel [3,9]. In contrast, the primary challenge in inferring formal process models from informal flow diagrams is the detection of higher-level semantics—the basic shapes are readily parsable from the diagram format. Thus, unlike the research in sketch recognition, our work focuses on semantic interpretation of informal diagrams.

Gross [8] presents an approach for sketch interpretation, which consists of: low-level glyph recognition, detection of spatial relations among glyphs, and assembly of glyphs into high-level configurations. Although the approach is organized in a similar manner as ours, the high-level structural patterns (*e.g.,* tree and polyline) that they discover do not have any semantic significance. Moreover, polyline recognizers in these sketching tools can detect only pre-specified patterns of inter-linked lines and are not as general as our edge-inference algorithm. Barbu et al. [2] apply frequent graph discovery to symbol recognition from line drawings: their approach extracts a feature vector to represent nodes and applies an unsupervised hierarchical clustering algorithm. Unlike our approach, they focus on inferring graphic symbols only and not semantic classes. Also, we consider a richer set of features that includes higher-level relational attributes, such as indegree and outdegree.

Approaches in visual language theory (*e.g.,* [6,7,16]) rely on upfront codification of the production rules for interpretation. Our work can be viewed as the first step toward learning such grammars. Wittenburg [16] shows that relational grammars are required to parse process-modeling constructs. Our current work attempts semantic classification at the level of a single node. Future work can attempt learning more complex relational patterns involving a sequence of nodes (*e.g.,* loop, fork, and merge).

## 6 Conclusions and Future Work

We presented a comprehensive approach for discovering formal process models from informal process diagrams that contain structural and semantic ambiguities. We presented techniques for resolving the structural ambiguities to extract precisely a flow graph underlying a process diagram. We also showed that standard pattern-classification techniques can be successfully employed in interpreting process semantics if the feature space is carefully modeled. Our approach mimics human reasoning used in recognizing diagram semantics: it models relational, geometric, and textual attributes of flow elements as features during pattern classification, instead of simply relying on shape name as existing tools do. Our empirical results showed that unsupervised clustering can almost match supervised techniques in performance; thus, such an approach can work well in practical scenarios where sound training data may not be available. Our tool, *i*DISCOVER, has better precision and recall, in both structural inference as well as semantic interpretation, than the state-of-the-art Visio import capabilities.

Future work can strive to improve the precision and recall of semantic interpretation through more efficient modeling of textual cues because text is the only reliable feature in highly ambiguous scenarios. Label association can also be perfected by tracking spatial patterns of label assignments that emerge due to local styles followed by designers. Finally, future research can investigate the identification of higher-level relations (block structures) between model entities (*e.g.,* sub-process, loop, and fork-merge) and extend the approach to other varieties of flow diagrams.

**Acknowledgements.** We would like to thank Indrajit Bhattacharya, David Marston, Juhnyoung Lee, Rakesh Mohan, Sugata Ghosal and Kathleen Byrnes for their valuable inputs and many useful discussions on the topic. We thank Vanitha Nachimuthu and Mary Joshua for their efforts in preparing the training data.

## References

1. Apte, A., Kimura, T.: Recognizing multistroke geometric shapes: an experimental evaluation. In: Proc. of ACM UIST, pp. 121–128 (1993)
2. Barbu, E., et al.: Frequent graph discovery: Application to line drawing document images. Electronic Letters on Computer Vision and Image Analysis 5(2), 47–57 (2005)
3. Chen, Q., Grundy, J., Hosking, J.: SUMLOW: early design-stage sketching of UML diagrams on an E-whiteboard. Software Focus 38(9), 961–994 (2007)
4. Cohen, W.W., Ravikumar, P., Fienberg, S.E.: A Comparison of String Distance Metrics for Name-Matching Tasks. In: IIWeb, pp. 73–78 (2003)
5. Duda, R., Hart, P., Stork, D.: Pattern Classification. John Wiley, New York (2001)
6. Futrelle, R.P., et al.: Understanding diagrams in technical documents. IEEE Computer 25(7), 75–78 (1992)
7. Golin, E., Reiss, S.: The specification of visual language syntax. In: IEEE Workshop on Visual Languages, pp. 105–110 (1989)
8. Gross, M.: Recognizing and interpreting diagrams in design. In: Workshop on Advanced Visual Interfaces, pp. 88–94 (1994)
9. Hammond, T.: Tahuti: A geometrical sketch recognition system for UML class diagrams. In: ACM SIGGRAPH 2006 Courses, p. 25 (2006)
10. Iivari, J.: Why are CASE tools not used? Communications of ACM 39(10), 103 (1996)
11. Jain, A., Duin, R., Mao, J.: Statistical pattern recognition: A review. IEEE Transactions on Pattern Analysis and Machine Intelligence 22(1), 4–37 (2000)
12. Jain, A.K., Dubes, R.C.: Algorithms for Clustering Data. Prentice Hall, Englewood Cliffs (1988)
13. Pal, S., Mitra, S.: Multilayer perceptron, fuzzy sets, and classification. IEEE Transactions on Neural Networks 3(5), 683–697 (1992)
14. Quinlan, J.: C4. 5: Programs for Machine Learning. Morgan Kaufmann, San Francisco (2003)
15. Rubine, D.: Specifying gestures by example. In: Proc. of the Conf. on Computer Graphics and Interactive Techniques, pp. 329–337 (1991)
16. Wittenburg, K., Weitzman, L.: Relational grammars: Theory and practice in a visual language interface for process modeling. Visual Language Theory, pp. 193–217 (1998)
17. Mukherjee, D., Dhoolia, P., Sinha, S., Rembert, A.J., Nanda, M.G.: From Informal Process Diagrams To Formal Process Models. IBM Technical Report No. RI09014 (2010), `http://domino.research.ibm.com/library/cyberdig.nsf/index.html`

# Value-Oriented Coordination Process Modeling

Hassan Fatemi, Marten van Sinderen, and Roel Wieringa

Information Systems (IS) Research Group,
Electrical Engineering, Mathematics and Computer Science (EEMCS) Department,
University of Twente, Enschede, The Netherlands
h.fatemi@utwente.nl, m.j.vansinderen@ewi.utwente.nl, roelw@cs.utwente.nl

**Abstract.** Business webs are collections of enterprises designed to jointly satisfy a consumer need. Designing business webs calls for modeling the collaboration of enterprises from different perspectives, in particular the business value and coordination process perspectives, and for mutually aligning these perspectives. However, business value modeling and coordination process modeling have different goals and use different concepts. Nevertheless, the resulting models should be consistent with each other because they refer to the same system. In this paper we define consistency between value models and coordination models in multi-perspective e-business web design and give guidelines to produce consistent coordination process models from business value models in a simple and stepwise manner. We provide an initial validation of these guidelines with a real-world example of business web design.

## 1 Introduction

A business web is a collection of enterprises designed to jointly satisfy a complex consumer need [1]. In a business web each enterprise contributes with its own specific products or services to satisfy a consumer need. Each partner wants to be sure that participation in such a collaboration network is economically rational and, if so, specify the coordination process. Hence, business value modeling, where economic sustainability can be analyzed, and coordination modeling, in which coordination can be specified complement each other.

The main goal of business value modeling is to reach agreement amongst profit-and-loss responsible stakeholders regarding the question "Who is offering what of value to whom and expects what of value in return?" In contrast, an important goal of coordination process modeling is to reach a common understanding about which coordination activities should be carried out, by whom and in which order. These are two different modeling goals, asking for different modeling methods with different constructs [2]. Nevertheless, despite the differences, a business value model and its corresponding coordination process model should be consistent with each other because they both refer to the same system.

In the current line of research two approaches for maintaining consistency between the value and coordination perspectives are used: (1) informally, by giving a set of guidelines how to use e.g. the business value perspective for finding a related coordination process perspective and vice versa [3,4,5,6,7], and (2) formally,

R. Hull, J. Mendling, and S. Tai (Eds.): BPM 2010, LNCS 6336, pp. 162–177, 2010.

by stating consistency rules between perspectives, which e.g. can be checked by model checkers [8,9,10]. The proposal to maintain consistency as discussed in this paper uses both approaches. The contribution of this paper consists of the description of an improved definition of consistency between business value and coordination process models of a business web, and also a method to design a coordination process model from a value model resulting in a consistent pair of models. The contribution of the paper is unique because it shows how to move from a business value model to a coordination process model in a structured and stepwise way using coordination patterns.

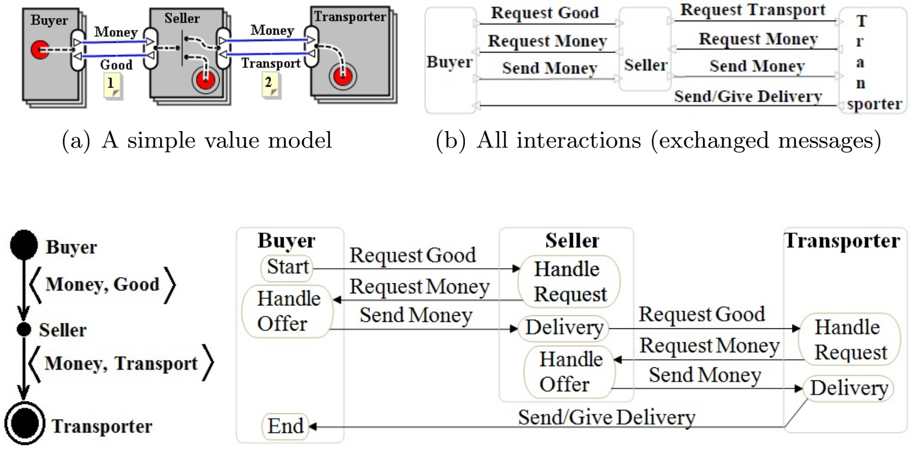
For representing the business value perspective, we use value models of $e^3$*value* [11], and for the coordination process perspective, we use BPMN diagrams (see http://www.bpmn.org/). At the end of the paper we will discuss the generality of our results beyond these two particular notations.

In section 2, we discuss business value modeling in $e^3$*value* methodology and coordination process modeling in BPMN and their properties. Next, we propose a stepwise approach to generate a coordination process model from a business value model in section 3. We apply our method on a real-life case in section 4. Finally we conclude with discussion, conclusion and future research in section 5.

## 2 (Business) Value Models and Coordination (Process) Models

In $e^3$*value* we model a business web as a graph in which the nodes represent economic actors and the edges represent economic transactions. In addition, an $e^3$*value* model shows how a consumer need is met by a set of economic transactions between actors in this network [11,12,13]. Consider the simple $e^3$*value* model (figure 1(a)) in which *Buyer* gives *Money* to *Seller* and receives *Good* in return. The *Seller*, in his turn, gives *Money* to the *Transporter* and receives *Transport*. This simple model illustrates all modeling constructs of $e^3$*value*:

- *Contract Period.* A value model describes economic transactions during a specific period of time which is called contract period. The contract period should be specified in supporting documentation to the model and the model will be used to analyze economic sustainability during this period only.
- *Actor.* An actor is an independent economic (and often also legal) entity with a specific interest in the collaboration (making profit, increasing utility, earning experience, ...). Actors in figure 1(a) are *Buyer*, *Seller* and *Transporter*. The actor for whom the business web is made to satisfy his needs is called the *consumer*. We represent the consumer need by a bullet placed inside this actor.
- *Value Object.* A value object is a service, good, money, or experience, that is of economic value to at least one actor and is exchanged between actors. In our example value objects are *Money*, *Good*, *Money* and *Transport*.
- *Value Port.* An actor uses a value port to provide or request value objects to or from other actors. A value port is a conceptual construct indicating



(a) A simple value model

(b) All interactions (exchanged messages)

(c) Transaction decomposition tree

(d) Coordination process model

**Fig. 1.** From Business Value model to Coordination Process model

that during the contract period, an actor is capable of giving or receiving a value object. Value ports are represented by small triangles on the edge of the shapes representing actors.

- *Value Interface.* Value interfaces group value ports and indicate atomicity: if one value port in the interface is triggered in the contract period, all of them are triggered in this period (however the model makes no statement about when this will happen: this will be specified in the coordination model). Value Interfaces are represented by oval shapes surrounding the value ports.
- *Value Transfer.* A value transfer connects two value ports of different actors with each other representing that the actors are willing to transfer value objects in the indicated direction.
- *Market Segment.* A market segment is a set of actors that assign economic value to objects equally. They are shown as overlapping rectangles.
- *Value Transaction.* Value transfers should come in economic reciprocal pairs, which are called value transactions.
- *Transaction Decomposition Tree (TDT).* This is a rooted directed acyclic graph with the consumer at the root and other nodes labelled by other actors linked with business transactions (see figure 1(c)). [1] The graph presents the AND/OR logic of the transactions: each complete path from the root (making a choice at every OR node) to the leaves represents one set of business transactions that jointly fulfill the consumer need. Figure (1(c)) shows the Transition Decomposition Tree for the value model in figure 1(a). It is a

[1] In $e^3$*value* this is called a dependency path but for consistency checking it is important to emphasize that this is actually a tree.

simple TDT without AND/OR splits. To illustrate this more suppose that we had two different Transporters (Normal and Special). In that case the two transporters were linked to the Seller by an OR split in the value model. So, we would have had the two transporters linked to the seller in the TDT with an OR logic between them. In that case we could enumerate two different ways of satisfying the consumer need by traversing the TDT from the root to the leaves.

The temporal meaning of a *transaction decomposition tree* is that if the need at the root occurs during the contract period, then the transactions in the tree also occur in the contract period, namely to fulfill the need.

All transactions outside the scope of the business value model (because they are not relevant to the economic sustainability estimation) are represented by a bull's eye. The bull's eyes represent the model boundaries and are the leaves in *transaction decomposition tree.*

Given an $e^3value$ model attributed with quantitative estimations (for example, the number of consumer needs per contract period and the valuation of objects exchanged) and a contract period we can estimate the revenue of each actor in the specified contract period. This is a first indication whether the model at hand can be economically rational for each actor.

### 2.1 Differences

Consider the coordination model in figure 1(d), which is consistent (in a way to be explained later) with the value model of figure 1(a). There are a number of differences between these two models. In general the conceptual gap between value models and coordination models is caused mostly by the following properties of these models:

1. **Ordering:** The key concept in value modeling is *value* while its counterpart in coordination modelling is *time.* In an $e^3value$ model there is no notion of time ordering at all [11]. Behavior and temporal order are beyond the value perspective and are part of the coordination perspective.
2. **Time-related properties:** From the value perspective, when value V is transferred from actor A to actor B, it does not make any difference whether this transfer occurs at once or in some steps, and also there is no difference between a time-continues and time-discontinues value transfer. In the coordination model all these time-related properties should be determined.
3. **Value versus coordination objects:** In a value model every object should be of value to at least one partner. But in a coordination model objects are not included necessarily because they are of economic value to a partner. They can also be included because they help coordinating the activities of the partners (for example, *messages*). We call objects in the coordination model *coordination objects.*
4. **Third parties:** A direct value transfer between two partners in a value model does not necessarily imply that there will be a direct coordination

object exchange between these partners in the corresponding coordination model. Sometimes a third party will be involved and the path for value object exchange becomes an indirect path for control object exchange. In the example at hand (figure 1) there is a direct value transfer between the buyer and the seller, while the physical transfer of the good that is the subject of the value exchange will require an indirect control object exchange between the buyer and the seller involving a transporter.

5. **Payment methods:** Money transfers are the most common transfers in value models that indicate paying a partner some money in exchange of his/her service or good. A money transfer between two partners in the value model, does not indicate the payment method. There is a wide variety of payment methods that can make the coordination model look very different from its value model.

Moving from one type of model to the other needs the conceptual gap caused by the above factors to be bridged.

### 2.2 Similarities

Despite the aforementioned conceptual gap, value modeling and coordination modeling also address some common aspects. This is the source of consistency requirements. Firstly, they have the same actors/partners. In the business world, an actor joins a business web only if (s)he earns something of value to herself/himself. Hence, every actor in a business web must perceive some value and therefore will be present in the value model independently. Secondly, a coordination model has a contract period too, with the same meaning as the contract period of the value model: The actors have agreed to behave in a certain way during the contract period. Finally, each value transaction indicates that something should happen to realize it.

In the coordination model we abstract from internal activities of actors, i.e. from activities that don't involve communication with another actor. In fact, internal business processes are an important asset of enterprises and therefore few enterprises like to disclose information about them to the outside world. This means that the properties of the overall business-to-business collaboration must not be based on the internal processes of the participating enterprises, but rather on the externally visible behavior and the associated models to represent it [14].

Without loss of generality, we make some simplifying assumptions to reduce the complexity of the problem and converge different solutions. The most important simplifying assumption is that all actors are trusted so that we don't need to consider security mechanisms to mitigate the risk of frauds. In a realistic business model this assumption needs to be dropped but before building such a realistic model, the partners need to check whether the cooperation is economically sustainable (value model) and practically possible (coordination model) under the assumption that they can trust each other. If economic sustainability and practical possibility cannot be shown under the assumption of mutual trust, it is not worth the effort to check this under the more complicated conditions of lack of trust [9]. In this paper we therefore make this simplifying assumption but in future work we will drop it. We also abstract from some interactions like confirmation messages. This does not decrease the utility of our guidelines because any set of interactions between two actors can be elaborated with more detailed protocol information without creating an inconsistency with the value model.

Under the above simplifying assumptions, for each value transfer a pair of messages (coordination objects) are enough to realize it. This pair consists of a request message and a message referring to the actual value object of the corresponding value transfer.

### 2.3 Consistency

The similarities between business value models and coordination process models motivate the definition of consistency between these models. Zlatev and Wombacher [8] were the first to define consistency between an $e^3$*value* and a coordination model based on an equivalence of a common semantic model (reduced model that contains the common concepts from two models). Wieringa and Gordijn [9] try to generate correctness formulas for value models based on the correctness formulas provided by the designer for each business transaction. Pijpers and Gordijn [10] check consistency by constructing an intermediate model that captures the physical transfers in a value model, thereby reducing the conceptual gap between value and process models. This physical transfer model can then be checked for consistency with a process model via a *reduced model* approach. Bodenstaff [15,16] introduces another definition based on checking the revenue estimation of value model with the runtime behavior of the coordination model. First we define two concepts and then introduce our definition which integrates and generalizes the original definitions by [9] and [8].

All these models make the *basic correspondence assumption* that each value transfer should correspond to some message exchange in the coordination model and that inversely each message exchange should correspond to some value transfer. Under the simplifying assumptions of the previous section, each value transfer corresponds to a request/reply pair. To elaborate this into a consistency definition, we need the concepts of transaction path and execution sequence.

A *transaction path* in the $e^3$*value* model is a complete (containing all children of each AND node) and non-redundant (containing exactly one child of each OR-node) path from the root of a transaction decomposition tree ending in bull's eyes. An *execution sequence* in a coordination model is a trace from start to end state.

Like the value model the coordination model is valid in a specific period of time (contract period). We must assume that a domain expert has given the *basic correspondence assumption* for each value transaction. The consistency definition then basically tells us when a coordination model and a value model respect each others AND/OR logic. The consistency definition is general, i.e. it does not depend on simplifying assumptions. A value model and a coordination model are consistent if:

1. The sets of actors in both models are the same.
2. The contract period of both models are the same.
3. For each *transaction path* in the value model, there is an *execution sequence* in the coordination model which realizes the value transfers of that path, and
4. For every possible *execution sequence* in the coordination model, there must exist a *transaction path*, such that the message exchanges contained in the *execution sequence* represent all the value transfers in the *transaction path*. Each message in an execution path is part of the realization of a value transfer or it is there because it defines a coordination logic. It is possible that two different coordination models realize the same value model, if they only differ in the coordination logic.

This is an informal definition, but it is precise enough to be formalized.

## 3 From a Value Model to a Coordination Model

Several authors have proposed a method to build a coordination model from a value model [3,4,5,6,7]. Pijpers and Gordijn [3] proposed a method that makes an intermediate model ($e^3$*transition* model) based on the value model by extending it with independent transfers of *ownership rights* of an object and the actual object itself.

Anderson and Bergholtz [4] proposed a method that starts with a value model and in a number of steps, each value exchange is analyzed and identified as a sub-process of the coordination model. They break value exchanges to components (resource, right, custody, and document evidence).

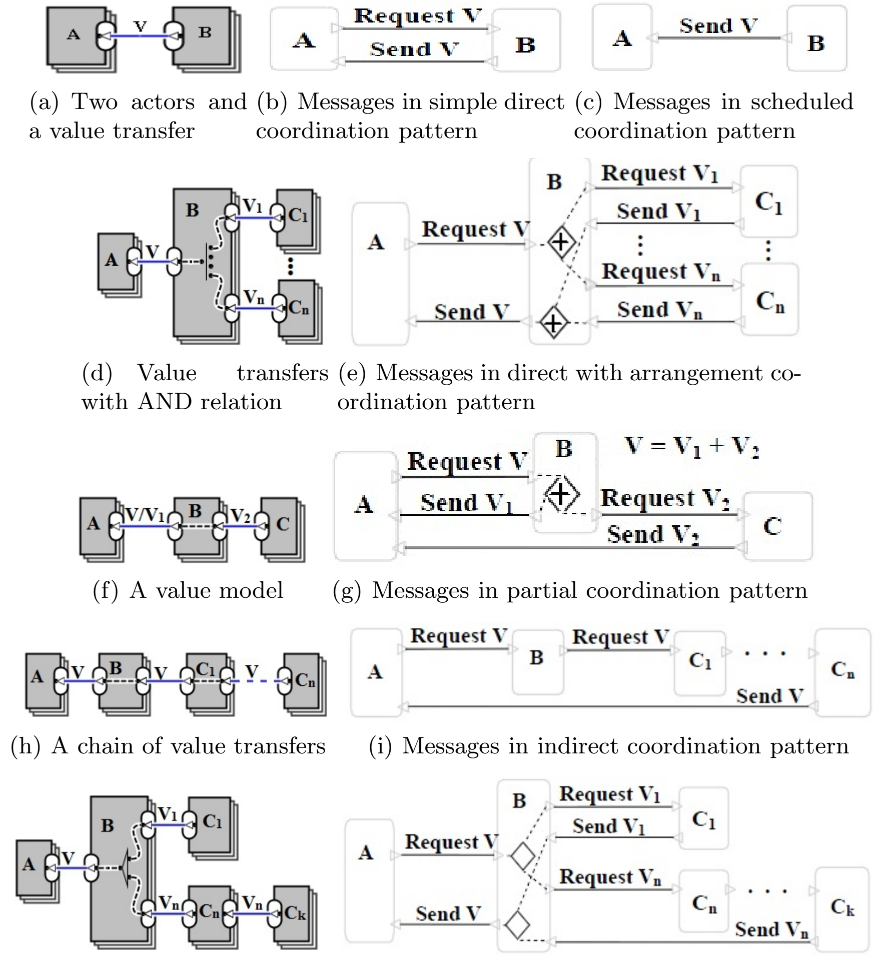
Wieringa et al [6] claim that coordination modeling is facilitated by making a *physical delivery model* first, because the value and coordination model are both views of a network of physical deliveries. They distinguish *discrete* from *cumulative* goods and *time continuous* from *time-discrete* deliveries. They also specify frequency or duration of deliveries and make a *delivery model* as an intermediate model on the way to design a coordination process model.

In our opinion, these approaches are all too complicated because they use intermediate models and/or introduce complicated concepts like ownership right, custody or physical delivery that makes it hard for others to use them in practice. Also, these methods have so far not been tested on other cases than the one they have been developed for. Our proposed method does not introduce these additional models or concepts and is therefore simpler and, as argued at the end of this paper, easier to generalize.

On the basis of the analysis in section 2, we proposed a stepwise method. The point of depature is an $e^3$*value* model. For illustration, we explain our method using $e^3$*value* model in figure 1(a).

- ***Step 1***: The first step is identifying the actors of the coordination model. The actors in both value model and coordination model must be the same. Hence, in our example they are buyer, seller, and transporter.
- ***Step 2***: In this step we aim at identifying groups of related value transfers and selecting the most suitable coordination pattern for each group. In other words, we determine the necessary interaction messages which should be included in the coordination model to realize value transactions. A domain expert must indicate for each value transfer to which sequence of coordination messages it corresponds. Logically, there are only a few possibilities, which we review here (figure 2).
  - **Simple Direct:** In this case, when an actor asks another actor to send him a value, the latter replies the former by sending him directly the requested value. This means that the receiver of the request is able to satisfy the requester's need without involving other actors (i.e. by its own). A simple value model with one value transfer and its realization according to this coordination pattern are shown in figures 2(a) and 2(b) respectively.
  - **Scheduled:** There is a special type of value transfer, we call it *scheduled transfer*, which doesn't need two messages (coordination objects) for realization in the coordination model. An example of this type of value transfer is scheduled payment in which a partner pays an already determined amount of money for a service/good on already scheduled times. In this case no party asks the other one for paying the money. Hence, in the coordination model we only have one interaction referring to the actual value object. Figure 2(c) shows the realization of the value transfer in figure 2(a) according to this coordination pattern.
  - **Direct with arrangement:** When an actor asks another actor to send him a value, the latter may, in his turn, ask some other actors to send him some values and then reply the former with the requested value. This arrangement can be both as preparation or obligation. In a preparation, the receiver of the request is not able to satisfy the requester's need only by its own, so he should involve some other actors to play role. However, in an obligation the receiver of the request is able to satisfy the requester's need by its own, but doing so obliges him to make some arrangements(value transfers). An example of preparation is ordering raw materials by a factory and as an example of obligation we can refer to clearing proprietary rights of a book or music.

    These pre and post requisites are not mutually exclusive and they both can appear in one case and from the value point of view they are the same. A value model and its realization according to this coordination pattern are shown in figures 2(d) and 2(e) respectively. In figure 2(e) all the requests are connected to the same AND-split (they execute in parallel), however they can have any ordering. The only implication is that they should be realized before the transaction which is dependent on them.



(a) Two actors and a value transfer

(b) Messages in simple direct coordination pattern

(c) Messages in scheduled coordination pattern

(d) Value transfers with AND relation

(e) Messages in direct with arrangement coordination pattern

(f) A value model

(g) Messages in partial coordination pattern

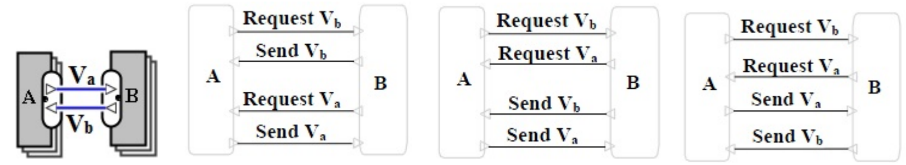
(h) A chain of value transfers

(i) Messages in indirect coordination pattern

(j) Value transfers with OR relation

(k) Messages with OR relation in combined coordination pattern

**Fig. 2.** Different ways of realization of a value transfer in Coordination Process Model



(a) A sample value model

(b) coordination messages as distinct pairs

(c) interleaved coordination messages

(d) one coordination messages pair inside the other one

**Fig. 3.** Possible ordering of the coordination messages of a value transaction

- **Partial:** If the request consists of some distinct parts and the receiver of the request is able to satisfy some parts of the request, he might send those parts directly to the requestor and ask another actor(s) to provide the other parts. An example is shown in figures 2(f) and 2(g).
- **Indirect:** In this situation, the first receiver of the request plays the role of a mediator by relaying the request to another actor. The first actor in such a relay chain that is able to satisfy the need sends back the requested value to the first requester.

  It is also conceivable that the first receiver of the request has the actual requested value but he can not provide the value to the requester or something should be done on it by a third actor before sending it to the requestor. Therefore, he asks another actor to deal with that. For example a company may transport his products to the customers via a shipping company. This situation may also be modelled in the previous way (Direct with arrangement). Sometimes there is no special distinction between these two situations and either correspondence can be defined. Figures 2(h) and 2(i) show a value model and its realization in indirect coordination pattern.
- **Combined:** This case is a combination of above situations. Basically any combination of the above cases is possible. Figures 2(j) and 2(k) show one possible value model and its realization in combined coordination pattern. Here the combination is just a matter of juxtaposition.

A value transaction between two actors aggregates two or more reciprocal value transfers. Thus, we need at least four message transfers to realize each value transaction in the coordination model. By traversing the *Transaction decomposition tree* starting from the consumer, different business scenarios and those value transactions which should occur during each scenario are identified.

Back to our example (figure 1), there is only one possible business scenario in which both transactions tagged as 1 and 2 will occur (figure 1(c)). We assume that the domain expert has given these correspondences: The two money value transfers correspond to the simple direct coordination pattern and the Good and the Transport value transfers make a chain of dependent value transfers. The coordination model after adding appropriate coordination patterns is shown in figure 1(b) which shows who is causing the transfer of some observable object to whom. Note that this is not an intermediate model and the method is not dependent on it. It is just for illustration and it does not serve any other purposes.

– ***Step 3***: To put the message transfers in a correct order in the coordination model the domain expert has to ask the following two questions regarding each value transaction of the value model:

1. *Who should first send a request to whom? (Which partner initiates?)*
2. *Which value transfer should happen first?*

Using the answers to these two questions we can put the four messages of the value transactions in a correct order. A value model containing a value transaction and three possible ordering of the coordination messages of this value transaction is shown in figure 3. Supposing that one of the actors, here actor A, starts the interaction , these three orderings are all possible ones. The second ordering(figure 3(c)) does not make sense and from the value perspective it does not differ from the first ordering(figure 3(b)). Because what really matters is the order of the value transfers and in both cases actor B sends the value first. Therefore we don't consider the ordering possibility shown in figure 3(c).

Suppose we have the following answers to the above two questions regarding value transfers tagged as 1 and 2 in the value model of figure 1(a) respectively:

1. The buyer should first send a request to the seller.
2. The payment should proceed the Good value transfer.

The first answer indicates that the buyer is the actor who initiates the process(i.e. sends the first request) so,the ordering shown in figure 3(d) applies here in which A refers to the Buyer and B to the Seller with a subtle difference that here the Seller does not send the Good to the Buyer instead he triggers the value transaction between himself and the Transporter.

- ***Step 4***: After identifying the necessary interaction messages and putting them in the coordination model in a correct order, we check time constraints of messages. For example there might be coordination messages that should occur in specific points of time or before a deadline.
- ***Step 5***: In this step we finalize the coordination model by adding necessary administrative activities to each partner (for example logging activities, confirmation messages, etc) and link the included interaction messages. Other examples are *start* and the *stop* activities (See figure 1(d)). All of these administrational activities do not correspond to value transfers but they are needed to make the coordination model work in practice. We refer to this in the discussion at the end of the paper.

Except step 2, which depends on human interpretation, the other steps can be automated through a CASE tool if we provide it with the necessary information like the answers of the questions of step 3.

The basic question here is: Is the value model generated by our method consistent with its corresponding value model? The first two requirements of the consistency definition are satisfied because the method requires that the actors in both models are the same and both models have the same contract period. Requirement 3 of the definition is satisfied because in step 2 and 3 we ensure that each transaction path corresponds to an execution sequence. And requirement 4 is satisfied because the only messages added to the coordination model are the ones needed to perform transactions (steps 2 and 3) or to control the coordination process (steps 4 and 5). In this way, we can claim that for each *transaction path* in the value model, which is a set of related transactions, there is an *execution sequence* in the coordination model and vice versa. Therefore, the models are consistent.

## 4 Case Study

To test the scalability of our method to a real-world case, we took an example that deals with the problem of clearing Intellectual Property Rights (IPR). It involves two steps: collecting fees from IPR users, i.e. radio stations, bars, discotheques and so on, who play music in public spaces with the aim of getting money from it, and repartitioning the collected fees to Right Owners, i.e. artists, song writers, producers. One of the main IPR societies in the Netherlands collecting IPR fees and repartitioning it to owners is SENA (see http://www.sena.nl/). IPR fee collection is currently done based on statistical evidence but SENA is interested in a future business model in which fees are collected on is a pay-per-play basis, in which for each music track, a track-specific network of clearing organizations is composed. This is possible once music is broadcast over the internet.

Figure 4 shows one possible value model of pay-per-play fee collecting in which BMP delivers a stream of tracks using Internet-based technology for direct playing which is not recordable at Receivers side. BMP and Receivers both should pay IPR Societies (see figure 4). In this value model actors are:

**Receivers:** A receiver is an actor who broadcasts background music to get benefits of it so, they are also IPR users.
**Background Music Providers (BMP):** A BMP is an actor who provides specialized background music in exchange of fee.
**IPR Societies:** IPR Societies collect fees for each track played in the public and repartition it to IPR owners.
**Right Owners:** Right owners of a track are those who involve in producing it, i.e., write lyric, play a musical instrument, produce and publish track etc.

Paying BUMA/Stemra is about the copyright that the composer and/or lyricist holds, whereas paying SENA is related to the rights of the performing artists and producer. After collecting money from users it should be repartitioned between appropriate right owners. SENA repartitions fees to Artists and Producers, and BUMA/Stemra does the same for Publishers, Composers and Lyricists.

We applied our method to this case. The result is shown in figure 5. According to step 1 of the proposed method, actors are the same in both models. Because of the space limitation and the high similarity that is between right owners, we only include one right owner representing all of them. Therefore actors are: *Receivers/User, BMP, BUMA, SENA, and right owner.*

Value transactions tagged as numbers 2, 3, 4, and 5 in figure 4 all match the simple direct coordination pattern. In all these transactions a right is exchanged for money. The right requestor should initiate and send money before receiving the confirmation of having right (ordering shown in figure 3(d)). Value transactions tagged as numbers 1,2 and 5 match *direct with arrangement* coordination pattern (2(d)). However because the right which the Receiver/User should clean is dependent on the tracks that BMP provides, first transaction 1 occurs and then 2 and 5.

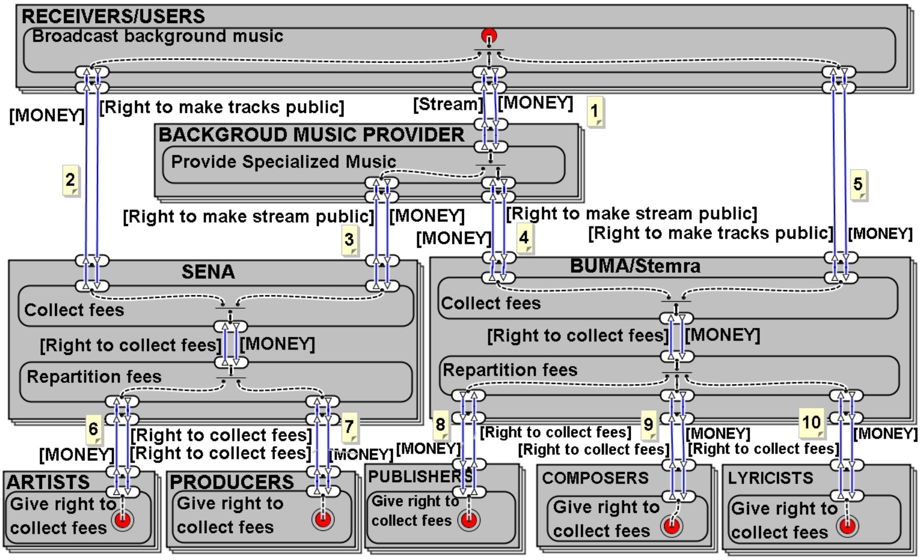


**Fig. 4.** Value model of providing music by Streaming

The confirmation sent from BMP to Receiver/User is just for the sake of efficiency. If we remove this confirmation message, the Receiver/User has to wait until the arrival of the stream before being able to clear rights.

The other transactions (numbers 6 through 10) match *scheduled coordination pattern*. Hence, we add only two messages to the coordination model to realize each value transaction (one message for each value transfer).

Here we haven't consider the time constraints and durations because the provisioning of music is a simple service for which the duration is already determined. Also the way in which the payments are being done in real life depends on the situations and the agreements between actors and there is a great variety in this that cannot be all included in the coordination model. For example, instead of paying for each stream in real time, BMP may send a *promise to pay* to SENA and at the end of the month pays all the payments in batch. In the last step we include the activities in the coordination model and using them connect the interaction messages to each other.

The two models have the same actors and we assume the same *contract period* for both. There is one *transaction path* consisting of all transactions in the model hence, there should be only one *execution sequence* in the coordination model realizing those transactions. This can be easily justified by travelling through the execution path in the coordination model starting from the Receiver/User.

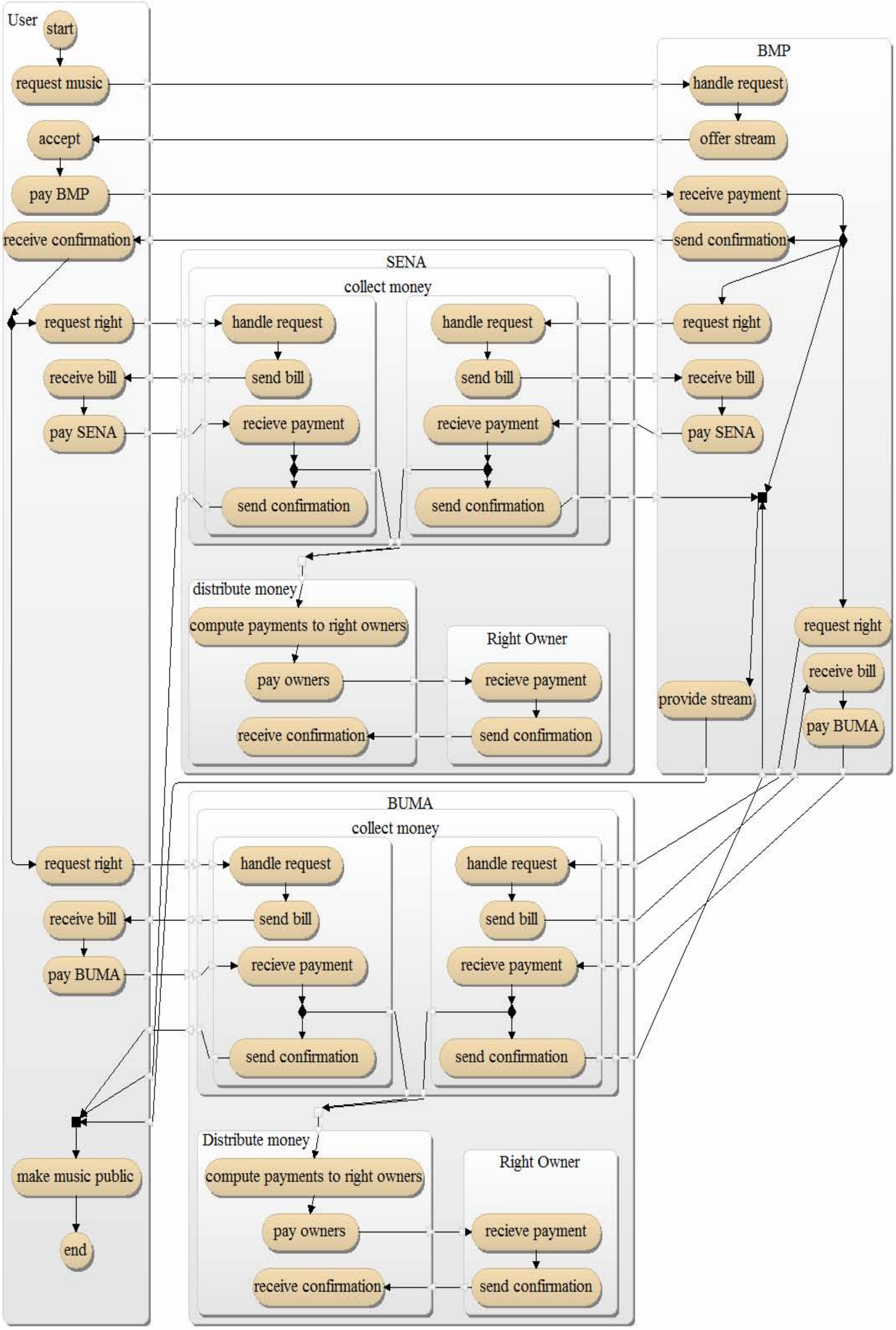


**Fig. 5.** Coordination model of providing music by Streaming

## 5 Discussion and Conclusions

In this paper we have discussed the problem of how to go from a business value model to a coordination process model in a stepwise and systematic way. Thanks to the conceptual commonalities that exist between the two models, a method could be proposed that starts with a value model where the main actors and their relationships, in the form of value exchanges, are identified. In a number of steps each value exchange is analyzed and by answering specific questions a coordination model is designed. The coordination model represents the interactions and interdependencies between the cooperating parties in terms of exchanged messages. We consider a special collection of interactions to realize the value transactions of value models. Based on the same analysis we have proposed a stepwise method for generating value model from coordination model and also check their consistency in a general and straightforward way.

The proposed guidelines make a simple method that avoids complicated concepts like property right, physical delivery, etc nor does depend on intermediate models and still is able to guide the modeler to a coordination model that is consistent with the value model. Because it does not depend on any special concept or intermediate model and nor does stipulate a special condition or attribute on the models, we claim that it is more general than existing approaches and it is easily generalizable beyond $e^3value$ and BPMN. We are currently validating this claim by applying it to more cases. We have tested our method on earlier cases done by other researchers [3,6] and observed that our application of the method produced similar results to what was obtained in those cases and we obtained coordination models that are consistent with their corresponding value models according to our definition for consistency. Because our method uses less concepts and does not depend on intermediate models, we hope that our method is both easier to use and applicable to more cases than the other methods; further validation is needed to substantiate this claim.

In addition to further validation, future work will consist of increasing the realism of the method by dropping trust assumptions and including guidelines for more complex coordination patterns. In step 5 we mentioned that we should add necessary administrational activities to the coordination model to make it work in practice. Some of these activities are related to trust issue and to have a more realistic result we should drop trust assumptions and enrich the model with necessary activities. In addition, more complex coordination patterns will include value transfers realized by multi-step coordination patterns and the inclusion of more sophisticated payment methods.

Another important addition we are working on is the addition of a reverse method. In general, coordination and value modeling are iterated, since a change in a coordination model may require a change in the value model. For example, adding a trusted third party in the coordination model requires adding this actor to the value model too. We are aiming at a reverse method that allows making such an addition in a consistent way.

## References

1. Tapscott, D., Ticoll, D., Lowy, A.: Digital Capital: Harnessing the Power of Business Webs. Harvard Business School Press, Boston (2000)
2. Gordijn, J., Akkermans, H., van Vliet, H.: Business modelling is not process modelling. In: ER Workshops, pp. 40–51 (2000)
3. Pijpers, V., Gordijn, J.: Bridging business value models and process models in aviation value webs via possession rights. In: HICSS 2007: Proceedings of the 40th Annual Hawaii International Conference on System Sciences, Washington, DC, USA, p. 175. IEEE Computer Society Press, Los Alamitos (2007)
4. Andersson, B., Bergholtz, M., Grégoire, B., Johannesson, P., Schmitt, M., Zdravkovic, J.: From business to process models - a chaining methodology. In: Proceedings of the 8th International Conference on the Advanced Information Systems and Engineering, CAiSE 2006 (2006)
5. Weigand, H., Johannesson, P., Andersson, B., Bergholtz, M., Edirisuriya, A., Ilayperuma, T.: Value object analysis and the transformation from value model to process model. In: Enterprise Interoperability, pp. 55–65. Springer, London (2007)
6. Wieringa, R., Pijpers, V., Bodenstaff, L., Gordijn, J.: Value-driven coordination process design using physical delivery models. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (eds.) ER 2008. LNCS, vol. 5231, pp. 216–231. Springer, Heidelberg (2008)
7. Fatemi, H., van Sinderen, M.J., Wieringa, R.J.: From business value model to coordination process model. In: Proceedings of the Second IFIP WG5.8 International Workshop on Enterprise Interoperability, IWEI 2009, Valencia, Spain. LNBIP, vol. 38, pp. 94–106. Springer, Heidelberg (2009)
8. Zlatev, Z., Wombacher, A.: Consistency between e3-value models and activity diagrams in a multi-perspective development method. In: Meersman, R., Tari, Z., Hacid, M.S., Mylopoulos, J., Pernici, B., Babaoglu, z., Jacobsen, H.A., Loyall, J.P. (eds.) OTM 2005. LNCS, vol. 3760, pp. 520–538. Springer, Heidelberg (2005)
9. Wieringa, R.J., Gordijn, J.: Value-oriented design of service coordination processes: Correctness and trust. In: 20th ACM Symposium on Applied Computing, pp. 1320–1327. ACM Press, New York (March 2005)
10. Pijpers, V., Gordijn, J.: Consistency checking between value models and process models: A best-of-breed approach. In: Proceedings of the Third International Workshop on Business/IT Alignment and Interoperability (BUSITAL 2008), held in conjunction with CAiSE 2008, Conference, pp. 58–72. CEUR-WS.org. (2008)
11. Gordijn, J., Akkermans, H.: Value based requirements engineering: Exploring innovative e-commerce ideas. Requirements Engineering Journal 8, 114–134 (2002)
12. Gordijn, J., Akkermans, H.: E3-value: Design and evaluation of e-business models. IEEE Intelligent Systems 16(4), 11–17 (2001)
13. Gordijn, J., Yu, E., van der Raadt, B.: e-service design using i* and e3value modeling. IEEE Software 23, 26–33 (2006)
14. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer, New York (2007)
15. Bodenstaff, L., Wombacher, A., Reichert, M.U.: Dynamic consistency between value and coordination models - research issues. Technical Report TR-CTIT-06-50, Enschede (2006)
16. Bodenstaff, L., Wombacher, A., Reichert, M.U.: On formal consistency between value and coordination models. Technical Report TR-CTIT-07-91, Enschede (October 2007)

# Coordination for Fragmented Loops and Scopes in a Distributed Business Process

Rania Khalaf[1] and Frank Leymann[2]

[1] IBM TJ Watson Research Center, 1 Rogers St, Cambridge MA 02142, USA
rhkalaf@us.ibm.com

[2] Institute of Architecture of Application Systems, University of Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany
Frank.Leymann@iaas.uni-stuttgart.de

**Abstract.** This paper addresses partitioning business processes that contain loops as well as compensation and fault handling scopes. The resulting process fragments can be distributed and wired together, recreating the execution semantics of the original process model. In earlier work, we presented BPEL fragmentation covering data and explicit control dependencies. We now extend the approach to handle fragmenting loops and scopes. Maintaining the focus on standards and maximizing extensibility of Web service runtimes and standards, the solution defines and uses new coordination protocols that plug into the WS-Coordination framework. An implementation is presented, extending the Active Endpoints BPEL engine and a WS-Coordination system.

**Keywords:** BPM, Distribution, Recovery, BPEL, WS-Coordination.

## 1 Introduction

As more work is off-shored, outsourced and globalized, business processes need to be fragmented and distributed in an agile manner. Several challenges emerge in doing so effectively: (1) handling partitions while maintaining operational semantics in the presence of fault handling, compensation handling, and loops (2) maintaining a low barrier to entry by using an open approach focused on interoperability and transparency. This leads us to a phased approach: Use the standards exclusively for a subset of use cases, identify when the standards are no longer enough, provide corresponding extensions for the identified cases, and support the extensions in a modular way avoiding force system replacement.

In [12,13], we laid the groundwork for fragmenting BPEL processes by enabling splitting data and explicit control dependencies. The approach reproduces BPEL behavior across fragments of a BPEL process by using patterns of BPEL constructs. In summary, an intra-process dependency split by the partition results in an inter-process message exchange(s) between the fragments. The algorithms ensure that the fragments are standard BPEL processes. The term 'unsplit process' refers to the process model being fragmented, 'partition' to the set-theoretic partition of an unsplit process, 'split activity' to a split loop

R. Hull, J. Mendling, and S. Tai (Eds.): BPM 2010, LNCS 6336, pp. 178–194, 2010.

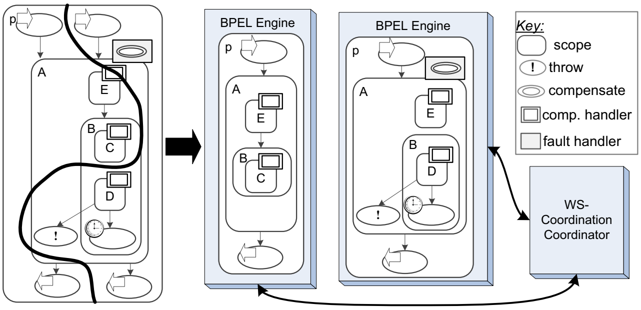


**Fig. 1.** Splitting a process with scopes across two participants

or scope, and 'split process' to the set of process fragments resulting from a partition.

In this paper, we extend the approach above to support splitting BPEL loops and scopes[1] . Scopes and loops provide shared behavior to a group of activities: fault handling, compensation handling, and variable scoping for the former and looping for the latter. The activities in a scope or loop serve different purposes and therefore may need to be executed by different organizations or servers. This results in the enclosing loop or scope being split. Splitting them using BPEL patterns in each fragment works for loops fragmented with few dependencies and participants. However, the number of exchanged messages and complexity of the patterns created when doing so for loops drastically increase with the increase in fragmentation and dependencies - and when doing so for scopes become potentially intractable. This is due to fragmenting the corresponding implicit coordination: finding a fault handler across fragments, propagating faults, synchronizing start/end of fragmented loops, scopes and handlers. A single fragment may also not have enough information, such as determining and enacting default compensation order. Thus, we find splitting scopes and loops to be a clear point where language and runtime extensions provide a better solution.

An overview of the approach is in Figure 1: A designer partitions a process (left), defined in the Web Service Business Process Execution Language (BPEL) [20] and containing multiple nested scopes with handlers, between different participants such that loops and scopes may be split(thick line). For example, this could be a new-hire onboarding process that needs to be split between the HR department and the Payroll department based on the activities each departhment should execute and control, while maintaining any common fault handling or looping behavior. The right side of Figure 1 shows the execution of the fragments by BPEL engines, recreating the operational semantics of the unsplit process by interacting with a coordinator. The extensions, compliant with the extensibility of the standards, created for this solution consist of three new

[1] Description of the protocols first shown in our technical report at http://elib.uni-stuttgart.de/opus/volltexte/2007/3050/pdf/TR_2007_01.pdf

BPEL attributes for scopes and loops and two new coordination protocols that plug into the WS-Coordination [21] framework. The overall approach is layered: Data and explicit control dependencies are split according to [12] using plain BPEL; implicit dependencies due to split loops and scopes are split using coordination as described in this paper if the process has at least one fault handler and/or contains at least one split activity. Figure 1 does not have the former, so no messages are exchanged directly between its fragments.

Section 2 presents relevant aspects of BPEL and WS-Coordination. The structure then follows the steps of the approach: Section 3 presents split scopes and loops and the corresponding BPEL language extensions. Section 4 presents using coordination for this problem, the split process information needed by the coordinator, and the new coordination protocols. Section 5 describes the implementation. We then present related work and conclude.

## 2 Background: BPEL and WS-Coordination

BPEL scopes and loops are structured activities, thus having activities nested within them. Structured activities are strictly nested and may be the source or target of explicit, conditional control links. An activity with incoming links has a 'join condition' that by default is the disjunction of the values of the incoming link conditions. If it evaluates to false, the activity throws a join failure fault that may potentially be suppressed. Suppressing it results in skipping the activity and setting its outbound links to false. A BPEL loop is similar to a while loop in programming languages. Explicit control links must not cross the loop boundary, but they may cross a scope's boundary. A BPEL scope, among other things, defines fault and compensation handlers. A fault or compensation handler may initiate compensation on one or more of its immediate child scopes: explicitly naming a target scope using the 'compensateScope' activity or all immediate children in default order using the 'compensate' activity. The default compensation handler consists of a 'compensate' activity. If a fault is thrown by an activity nested in a BPEL scope, it terminates all its nested activities, looks for a local fault handler, and runs the handler if it finds one. Once the fault handler completes, navigation continues at the scope boundary. If a handler is not found, the default fault handler runs: terminate all nested running activities, run 'compensate', then rethrow the fault to the parent scope. Termination of a scope results in triggering the default termination handler, which consists of a call to 'compensate'. All links leaving a terminated activity are fired negatively. Notice that when fault handling, faults propagate *up* the scope hierarchy, with the order of aborting siblings being irrelevant. On the other hand, default compensation occurs one level of scope nesting at a time and in an order that reverses control dependencies between peer scopes. In-depth scope behavior is described in [14].

WS-Coordination is a standard providing a pluggable framework for coordinating agreement between a set of services working together towards a joint outcome. The standard defines an Activation Service and a Registration Service. The optional Activation Service is used by the initiator of the coordination to create a unique 'coordination context' that identifies the particular instance of the distributed action being coordinated. This context is then exchanged in the header of the application messages that are part of the distributed action. The Registration Service is used by services wishing to participate in the distributed action. Usually, a service registers upon receiving a message with a coordination context. The protocol used to coordinate the distributed action is pluggable and is identified in the exchanged messages. Upon registering, each service must refer to a corresponding protocol handler that is in charge of exchanging messages according to the chosen protocol. Defining a new coordination protocol consists of defining participant and coordinator protocol services and handlers, the protocol message exchange order, and the required underlying behavior of the coordinator and the participants in relation to the exchange of these messages. One may also extend the activation and registration messages.

## 3 Split Loops and Scopes

A designer defines a split process, as in [12], by specifying a partition of the set A of all primitive activities of the process. Consider $P_n$, the set of participant names. Every participant $p$ in the set of participants $P_a$ consists of a participant name in $P_n$ and one or more activities such that $P_a \subseteq (P_n \times A)$. The following restrictions apply to the partition: A participant must have at least one activity, no two participants share the same primitive activity and every primitive activity is assigned to a participant. The splitting algorithm [12] uses this information to create one BPEL process and one WSDL for each participant, in addition to wiring information that is used to connect the fragments together at runtime.

The partition definition is used to determine if a scope or loop is split: If all activities in the body of a loop or scope or in the handlers of a scope are not assigned to the same participant, then that loop or scope is split. For example, scope E in Figure 1 is split but scope C is not. A handler is split if all the activities in the handler body are not assigned to the same participant. For loops, the designer also has to designate one participant as responsible for the loop condition. Let $L_n$ be the set of split loop names. Thus, we define a map $L_c$ that associates a participant with every split loop name: $L_c : L_n \rightarrow P_n$. The splitting algorithm then places a fragment of the scope or loop in each process fragment that has at least one activities of the scope or loop assigned to it. The algorithm denotes a loop or scope as split in the resulting fragments using the following language extension attributes on the corresponding XML elements:

- On <process>: 'belongs-to' whose value is the QName of the unsplit process.
- On <scope> and <while>: 'fragmented' whose value is 'yes' or 'no' (default 'no'), identifying the activity as split..
- On <while>: 'is-responsible' whose value is 'yes' or 'no' (default 'no'), identifying the loop fragment responsible for evaluating the loop condition.

This paper focuses on runtime enablement, thus leaving further details on fragment generation algorithms including nuances of link and activity placement to

[10]. The restrictions for this solution are: all split activities are uniquely named in the unsplit process model, compensation is a recovery mechanism and thus must not fail, the join condition of a split activity is the conjunction of the local joins of each fragment's join condition, scope fragmentation is focused on fault and compensation handling thus not covering variable scoping and event handling, and without loss of generality we do not cover splitting other types of structured activities except a 'flow' with no incoming/outgoing links. Such a 'flow' (not shown in Figure 1) is needed because BPEL syntax requires the body of a loop, scope or handler have exactly one (structured or primitive) activity. These are in addition to restrictions from [12] including that the fragments share a common correlation set serving as the split process instance identifier.

Split processes exhibit a property we call the *rubber band effect*: BPEL scopes and loops are strictly nested; it follows therefore that this nesting and resulting nesting relationships must be maintained upon splitting. Scope E in Figure 1 is split between two participants, so its ancestors A and the process itself also have fragments in those participants. A corollary is that if $x$ is a split scope or loop, all ancestor scopes and loops of $x$ (including the process), must also be split. If this property is violated, the split process could end up in an inconsistent state.

## 4 Using Coordination

Split loop and scope execution can be cast as a coordination problem: each fragment performs part of the distributed action corresponding to the behavior of the unsplit activity. For example, the split loop protocol is about agreeing whether or not to iterate again. Our protocols, by design, do not replace existing BPEL engine behavior; they complement and leverage it to enable the engines to collectively recreate the operational semantics of the unsplit process.

At first glance, it seems that the WS-BusinessActivity (WS-BA) [21] protocol could be used to coordinate scope fragments; however, this is not the case for several reasons including the fact that the fragments are peers whereas WS-BA was created to perform the coordination between a parent scope and its child scopes instead of fragments of one scope. The coordination for split loops and scopes differs from the traditional use of WS-Coordination such as WS-BA and WS-AtomicTransaction. The fragments of loops and scopes are known at design time and their location is known at deployment time. The fragments are peers and it is not known ahead of time which will start the protocol: i.e: either process fragment in Figure 1 can receive the first message to create an instance; once an instance is created, either fragment of scope $A$ can be reached first in the navigation. Our protocols are driven by lifecycle events and not by application messages. We do not use the optional Activation Service because (1)a protocol instance start is automatically detected when the first fragment of a split activity is reached in the navigation and (2)each participant has enough information to enable the coordinator to determine which protocol instance it belongs to: the fragments, their locations, the unsplit activity they are part of, and the instance of the process they are part of are known by the time the first coordination message is sent. Thus, there is no need to flow the coordination context. The required information, defined in section 4.2, consists of design-time information derived from the unsplit process, encoded in the relationship tree and Default Compensation Order (DCO) graphs, deployment time information in the wiring model, and runtime information consisting of the common correlation set value.

### 4.1 Deployment and Registration

To deploy fragments containing split loops or scopes, we extend the wiring model in [12]. The model provides connectors between the fragments themselves as well as between the fragments and external services invoked by the unsplit process. We now add a 'coordination' element containing: (1)the address of the registration service of the coordinator so that the first process fragment to start an instance can register; (2)the endpoint of a 'starter service' for each process fragment.

Protocol registration requires providing a protocol identifier; therefore a URL is used for each of the protocols. We also extend the registration message to include additional information needed to uniquely identify the instance of coordinated work and the participant registering for it.

The extensions are new sub-elements of the registration message's 'wscoord:Register' element: (1) <ProcessName>, the name of the split process the activity is in; (2) <LoopName> or <ScopeName>, the name of the split activity; (3)<ParticipantName>, The name of the participant the registration is for, used to determine which fragment is registering and match that to the information in its relationship tree; (4)<CorrelationSetValue>, an identifier of the instance of the coordinated work. Its value is the value of the common correlation set, in effect identifying the instance of the split process; (5)<counter>, a counter that further identifies which instance of a scope or loop is being coordinated, needed to distinguish multiple iterations of the a split activity in the same process instance.

### 4.2 Encoding Common Information

The a-priori information required by the coordinator about the unsplit process model is encoded in a scope and loop relationship tree and DCO graphs. They may be created at any time before the first process fragment creates an instance. They may be passed to the coordinator any time before a process fragment begins, such as at deployment time or in the registration message. Details of the formalism of these data structures and their construction are in [10], Chapter 5.

**Scope and Loop Relationship Tree.** The relationship tree, $RT = (N_{rt}, E_{rt})$, encodes information about the nesting relationships of split activities, their handlers, and their fragments. It is created from the unsplit process model and partition definition. An example is shown in Figure 2.

The set of nodes $N_{rt}$ is divided into four pair-wise disjoint sets:

- Split scope node set $S_{rt}$: Each scope node $s_{rt}$ is a tuple of a scope name, a set $M_s$ of faults the scope has a handler for, a Boolean stating whether

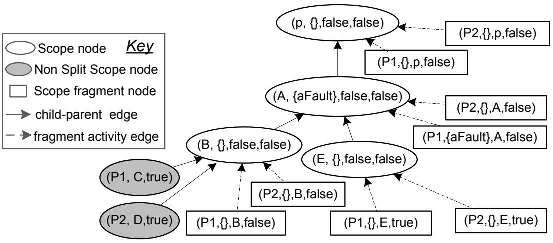


**Fig. 2.** The relationship tree for the process partition in Figure 1

the scope is in a handler of its immediately enclosing scope, and a Boolean whether it is in the compensation handler of its immediately enclosing scope. Thus, $s_{rt} = (name, M_s, in\_fault\_handler, in\_comp\_handler)$.

- Unsplit scope relevant node set $S_{ns}$: Includes scopes that are not split but need to be compensated by their ancestor split scopes. These are unsplit scopes where: their parent scope is split and they contain an explicit compensation handler either directly themselves or in any scope nested within them. Each such node $s_{ns}$ is a tuple consisting of the participant name it is in, the scope name, and a Boolean whether it has an explicit compensation handler. Thus, $s_{ns} = (p\_name, name, has\_compensation)$.
- Split loop node set $L_{rt}$: Each loop node, $l_{rt}$, is a tuple containing the name of the loop. Thus, $l_{rt} = (name)$.
- Fragment node set $F_{rt}$: A fragment node represents one fragment of a split activity and is a leaf node in the tree. They are of two kinds: Loop fragment nodes, $F_{rt_l}$, and scope fragment nodes, $F_{rt_s}$. Each loop fragment node is a tuple containing the name of the participant it is in, the name of the loop it is a fragment of, and a Boolean whether it is responsible for the loop condition. Each scope fragment node is a tuple containing the name of the participant it is in, the set $O$ of fault names this particular fragment has a handler for, the name of the scope it is a fragment of, and a Boolean whether the fragment has an explicitly defined compensation handler.

The tree has three kinds of edges forming pair-wise disjoint sets:

- The set of fragment-activity edges, $A_{rt}$, connecting a fragment node to the loop or scope node it is a fragment.
- The set of loop-scope edges, $B_{rt}$, connecting a loop to its parent scope.
- The set of child-parent edges, $C_{rt}$, connecting a scope to its immediately enclosing scope. Each scope node has exactly one such edge except for the root which has none. A scope in a fault handler of another scope is considered nested in the latter.

**Default Compensation Order Graphs.** A DCO graph, $G_D = (N_D, E_D)$, is a Directed Acyclic Graph that encodes the default compensation order of a scope. One DCO is created for each scope that has at least one explicit compensation handler nested within it and zero or more 'loopDCOS' for compensation relevant loops. Coordination only requires DCO graphs for split scopes, although the graphs are not dependent on a particular partition. The set of nodes $N_D$ of a DCO graph of a scope $s$ contains loop and scope nodes. A loop node is a member of $N_D$ if the loop is (1) an immediate child of $s$, (2) not nested in another loop in $s$ and (3) contains at least one scope having an explicit compensation handler nested at any level in $s$. A scope node is a member of $N_D$ if it is (1) an immediate child scopes of $s$, (2) not in handlers of $s$, (3) not nested in a loop that is in the DCO node set, and (4)either it or any of its nested scopes has an explicit compensation handler.

For each loop node in the DCO graph of a scope, there is an associated loopDCO graph $G_l = (N_l, E_l)$. The nodes in $N_l$ have similar characteristics to those in $N_D$ except using the corresponding loop instead of the scope $s$ to determine the relevant loop and scope.

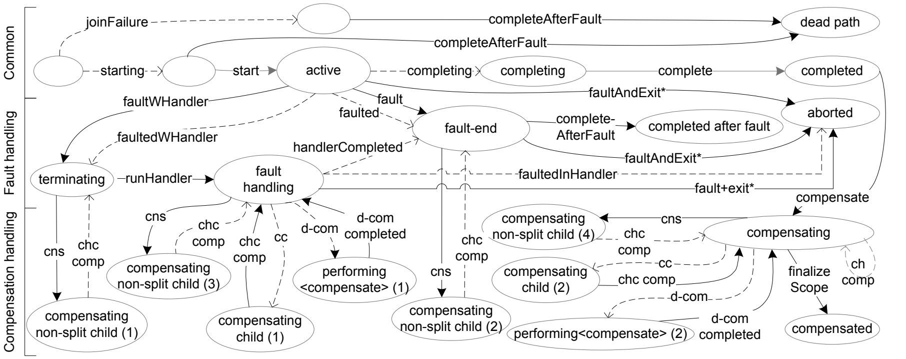
The edges in a DCO or loopDCO represent the reversal of the control path between the nodes of the DCO or loopDCO: an edge is present in $E_D$ (or $E_l$) between two nodes $x$ and $y$ in $N_D$ (or $N_l$) if there exists a control path in the process from $y$ or any of its nested activities to $x$ or any of its nested activities such that the path does not contain any other node in $N_D$ (or $N_l$).

Running compensation in default order occurs by navigating the DCO as a BPEL process with all join and transition conditions set to true. Book-keeping for instances can be encoded in 'ScopeAndLoopQ' structures in [10,14]. Upon navigating to a node, it is run and navigation continues by following its outbound edges. Running a scope node $B$ consists of a no-op if it is not completed, running its compensation handler if it has an explicit one, or running default compensation by navigating its DCO. Running a loop node $C$, consists of navigating the corresponding loopDCO, in the same way, as many times as the loop had run for that instance of the scope/loop in which the loop node is nested.

### 4.3 The Coordination Protocols for Split Loops and Scopes

A protocol is provided as a state machine encoding the messages, the order in which they are exchanged and the description of the corresponding required behavior. Each state corresponds to a point where either the coordinator(solid line) or the participant(dashed line) sends a message. The coordinator executes the protocol with each instance of a fragment of a split loop or scope. The coordinator uses the relationship tree to determine when all fragments have registered, search for handlers, and relate endpoints to participant names. The state machine for the split scope protocol is provided in Figure 3. We explain using the three subsets identified in the left margin: common(top), fault handling(middle), and compensation handling(bottom). After explaining the protocols, we will show in Figure 4 how the protocols are used to enact the split process in Figure 1.

**Common Split Activity Subset.** Consider the protocol subset common to all split activities. It synchronizes the beginning of a split activity: When a fragment



| | | | |
|---|---|---|---|
| cc: compensateChild<br>cns: compensateNonSplitChild | chc comp: childCompensationHandlerCompleted<br>ch comp: compensationHandlerCompleted | d-com: defaultCompensation<br>d-com completed: defaultCompensationComplete | *: only for process scope |

**Fig. 3.** The Split Scope Protocol

is reached in the navigation, it either starts and sends starting or it is skipped due to a 'joinFailure' fault it needs to suppress and sends joinFailure. Whether it suppresses a joinFailure depends on the activity's suppressJoinFailure attribute. The fragment then waits until the coordinator has heard from *all* fragments and sends either completeAfterFault if at least one fragment sent joinFailure or start otherwise. In the former case, the protocol ends in the 'dead path' state, otherwise it goes to 'active'. The protocol also synchronizes the end of a split activity: Once a fragment is ready to complete, it sends completing and then waits until the coordinator has received completing from all the fragments and sends it the complete message. The protocol then ends in the 'completed' state.

**Additional Messages for the Loop Protocol.** The loop protocol (not shown) adds two transitions to the common split activity subset, both for messages sent by the coordinator: continueLoop from 'completing' to 'active' and complete from 'start' to 'completed'. The fragment responsible for the loop condition sends the value of the condition in the start and complete messages to the coordinator. Once all fragments start, the coordinator sends start if the condition is true and continueLoop otherwise. Once all fragments complete, the coordinator sends continueLoop if the condition is true or complete otherwise.

**Fault Handling Subset of the Scope Protocol.** Data provided with a fault is sent to the coordinator, which then sends it to all fragments that have a matching handler. A fault handler at a fragment that has a 'faultVariable' attribute will then save this message in this variable.

Messages for fault handling are added from the 'active' state. Two special cases apply if the split scope is the process itself: (1) Upon receiving start from the first process fragment to begin, the coordinator sends a 'startInstance' message containing the value of the correlation set to the starter service of each of the other process fragments. As each fragment creates its process instance, it will send a registration message to the coordinator and the protocol proceeds as for any non-process scope. (2) A process fragment may receive a faultAndExit in the case that a fault is encountered in the split process for which no matching handler is found. The rest of the messages apply to all fragmented scopes.

If the fragment of a scope $s_m$ encounters a fault, it sends either faulted if it has no matching fault handler in its fragment or faultedWHandler otherwise. Either causes the coordinator to search the tree for the scope $s_f$ in whose protocol it will continue: $s_f$ is the scope having a matching fault handler or the process if no handler is found. In our example, the throw activity causes scope $A$'s fragment to send faultedWHandler and the coordinator finds that $s_f = A$. We now consider only the fault subset, adding messages and behavior related to potential compensation for this case in the compensation handling subset below. The coordinator first aborts the coordinator-side of scope and loop protocols nested in $s_f$ (except for scopes in the 'completed' state). If there is no handler, it sends faultAndExit to the fragments of $s_f = \mathit{process}$. If a handler is found, regardless of whether $s_f = s_m$ or one of its ancestors, the coordinator sends faultWHandler to the fragments of $s_f$ that have the handler and fault to the rest. The fault, faultWHandler, faultAndExit messages throw the fault in the scope at each process fragment, causing local engine behavior to abort nested scopes and loops. Both the participant and the coordinator sides of nested split activity protocols are therefore stopped for all fragments and no separate 'abort' message is needed. We require that messages arriving (i.e. due to a race) at the coordinator side for an aborted protocol are ignored.

The protocol synchronizes the beginning and end of split fault handlers. Therefore, a fragment having a matching handler waits for the coordinator to send runHandler before starting it. The handler in a fragment then either completes (signaled by sending handlerCompleted) or encounters a fault. In the former case, the coordinator waits for all handler fragments to complete and sends completeAfterFault. In the latter case, the participant sends faultedInHandler, placing the protocol in the 'aborted' state and causing the coordinator to search the tree for and notify the scope containing a matching fault handler.

**Compensation Handling Subset of the Scope Protocol.** Fragmented compensation handling requires coordinating the start and end of fragmented handlers as well as determining and enforcing compensation order. The latter is not possible locally because child scopes may be in multiple fragments. BPEL engines extended to support split activities are not allowed to calculate default compensation order of a split scope or any of its nested children. The coordinator will use the DCOs and relationship tree to enforce and drive compensation in the correct order. Thus, the extended BPEL engines may only start an explicit compensation handler of a split scope and can only do so upon receiving a request from the coordinator. For non-split scopes in the relationship tree($S_{ns}$), the engine can only start their explicit compensation handler if the coordinator requests it. Since there is no protocol for non-split scopes, this request arrives via the protocol of an ancestor split scope. Both the engine and the coordinator may trigger the default compensation handler of scopes in $S_{ns}$. The engine will do so if default compensation is required by the scope's termination behavior.

Consider the part of the protocol for compensating a split scope, e.g. scope $E$ in the example. It is covered by the states reachable from 'completed' because a BPEL scope may only be compensated if it has completed. If a coordinator needs to compensate $E$, it sends compensate to all fragments of $E$. If the fragment has a compensation handler, it will run the handler and send compensationHandlerCompleted upon handler completion. Once all the handler's fragments have sent this, the coordinator sends finalizeScope to all fragments. Compensating a scope in a loop requires compensating every instance of the scope that ran for one instance of the parent scope requesting the compensation. The coordinator must request the corresponding compensation the correct number of times and using the correct protocol instance for each of these scope instances.

Consider compensating children of a scope $s$ due to a 'compensateScope' or 'compensate' activity, e.g. the 'compensate' in scope $A$'s fault handler. These activities may occur only in handlers, so the corresponding states are reachable from the 'fault handling' and 'compensating' states. From either state, a fragment of $s$ can send defaultCompensation or compensateChild to request default compensation or compensation of a child scope $s'$, respectively. The coordinator then triggers the necessary compensation, which it does for the default compensation case in the order determined from navigating the DCO. Once the compensation is completed, the coordinator sends back defaultCompensationCompleted or childCompensationHandlerCompleted, respectively.

Consider the transitions and states enabling a scope such as scope $B$ to compensate a non-split scope, such as $C$, nested in $B$ and present in the relationship tree. The work is reflected in the four 'compensating non-split child' states: the protocol of $B$ for the fragment containing $C$ enters this state once the coordinator sends it compensateNonSplitChild. Once the fragment locally completes this compensation, it sends childCompensationHandlerCompleted. If $C$ has not completed, the engine must treat this request as a no-op and immediately send back the completion notification. The non-split child scopes can also be compensated from the 'terminating' and 'fault-end' states. The coordinator, in these states, is performing compensation of $C$ as part of terminating the children of $B$ and does so through $B$'s protocol because there is no protocol for non-split scopes.

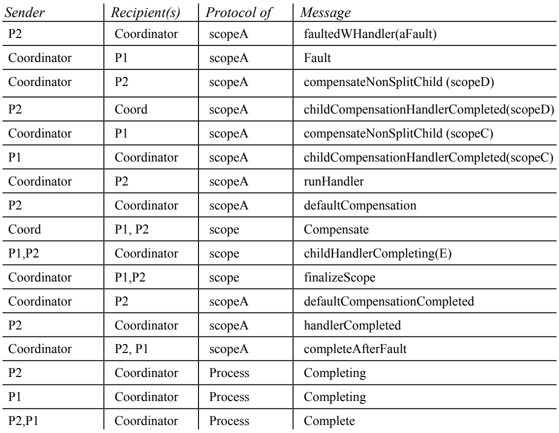
Some transitions to 'aborted' not in the figure for simplicity address faults in the fault handler while compensation is running: faultAndExit from 'compensating non-split child(2)' and faultAndExit and faultedInHandler from the three compensation states reachable from 'fault handling'.

This leaves the coordinator behavior for compensation due to termination and rethrow of a fault, e.g. scope $A$ faults while $B$ is running, which uses the protocols of multiple scopes. Recall that upon receiving a notification of a fault from a scope $s_m$, the coordinator continues from a scope $s_f$. BPEL behavior when a fault is thrown is: default termination of running activities. If the scope has no handler, then do default compensation of completed scopes followed by rethrowing the fault to the parent scope. To use less messages and simplify the runtime, we drive this behavior from $s_f$: stop all activities in $s_f$ then perform compensation in the order necessary by calculating what it would have been had we, at each scope between $s_m$ and $s_f$, actually caught and rethrown the fault.

Consider the time the coordinator has notified the fragments of $s_f$ of the fault, causing all active activities in $s_f$ to be aborted by throwing the fault locally in each fragment. Due to our restrictions on the engine's ability to trigger compensation on its own, no compensation is done during this for *split* scopes in $s_f$. Then the coordinator drives the following behavior:

1. Starting with $s = s_m$:
   (a) 'Compensation due to Termination' of activities in $s$: Perform default compensation on every split scope $s_t$ in $s$, innermost first, if $s_t$: it is not in a handler of $s$, is not completed, and has at least one immediate child scope that is completed and in the DCO of $s_t$.
   (b) If $s$ has no handler for the fault, perform default compensation on $s$ and if $s \neq process$ continue with 'Compensation due to Rethrow'. Otherwise skip steps 2 and 3.
2. perform 'Compensation due to Rethrow' of the fault: Going up the relationship tree along child-parent edges from $s_m$ to $s_f$, for each encountered scope node $s_p \notin \{s_m, s_f\}$:
   (a) If $s_p$'s *in_fault_handler* Boolean is true, skip $s_p$
   (b) Otherwise, perform step 1 for $s = s_p$, but skip any scopes already touched by previous iterations.
3. perform 'Compensation due to Termination' of $s_f$: Perform step 2 for $s = s_f$, but skip traversing down any subtrees already touched in steps 1 or 2.

Finally, continue as in the fault handling subset: if $s_f$ has a matching fault handler, send runHandler to its fragments to run the fault handler; otherwise,



| *Sender* | *Recipient(s)* | *Protocol of* | *Message* |
|---|---|---|---|
| P2 | Coordinator | scopeA | faultedWHandler(aFault) |
| Coordinator | P1 | scopeA | Fault |
| Coordinator | P2 | scopeA | compensateNonSplitChild (scopeD) |
| P2 | Coord | scopeA | childCompensationHandlerCompleted(scopeD) |
| Coordinator | P1 | scopeA | compensateNonSplitChild (scopeC) |
| P1 | Coordinator | scopeA | childCompensationHandlerCompleted(scopeC) |
| Coordinator | P2 | scopeA | runHandler |
| P2 | Coordinator | scopeA | defaultCompensation |
| Coord | P1, P2 | scope | Compensate |
| P1,P2 | Coordinator | scope | childHandlerCompleting(E) |
| Coordinator | P1,P2 | scope | finalizeScope |
| Coordinator | P2 | scopeA | defaultCompensationCompleted |
| P2 | Coordinator | scopeA | handlerCompleted |
| Coordinator | P2, P1 | scopeA | completeAfterFault |
| P2 | Coordinator | Process | Completing |
| P1 | Coordinator | Process | Completing |
| P2,P1 | Coordinator | Process | Complete |

**Fig. 4.** Protocol messages, in order, for the example in Figure 1

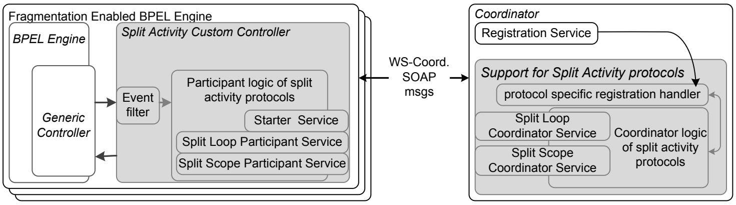

Fragmentation Enabled BPEL Engine
BPEL Engine
Generic Controller
Split Activity Custom Controller
Event filter
Participant logic of split activity protocols
Starter Service
Split Loop Participant Service
Split Scope Participant Service
WS-Coord. SOAP msgs
Coordinator
Registration Service
Support for Split Activity protocols
protocol specific registration handler
Split Loop Coordinator Service
Split Scope Coordinator Service
Coordinator logic of split activity protocols

**Fig. 5.** Architecture of the Runtime System (new artifacts in grey)

send faultAndExit to all fragments of the process. Figure 4 shows the protocol messages used to coordinate the fragments in Figure 1 once the fault is thrown by the 'throw' activity.

Race conditions exist when a state has transitions whose messages can be sent by either coordinator or participant. The resolution rules are: a fault related message wins over a compensation message; in races between fault related messages, those from the coordinator win over those from a participant; races between compensation messages result in queuing the messages.

## 5 Implementation

We focus on the runtime subsystem of the fragmentation system in [10]. The implementation of this subsystem is detailed in [22], including WSDL definitions. To support the protocols, the BPEL engine(s) used must be extended to recognize the new language extensions and support the participant side of the protocols, which affect and are affected by the engine's BPEL navigation. This includes includes a starter service, a loop participant service, and a scope participant service. Aiming to avoid new middleware where possible, we use a modular and minimally intrusive mechanism [11] for extending existing BPEL engines. The mechanism uses an engine-agnostic generic event model and an event-based approach based on a 'Generic Controller' with pluggable 'Custom Controllers' for new extensions. We created a Split Activity Custom Controller (Figure 5, left) that implements the participant services, the outbound message invocations they send, and the required behavioral logic. The controller's interacts with the Generic Controller by receiving and sending events over JMS using ActiveMQ to control and react to the behavior of the engine as needed and receiving and sending protocol messages using SOAP to and from the coordinator. [11] illustrates controlling the engine to execute a loop fragment.

Next, consider the coordinator. At deployment time, the coordinator endpoint is bound with the controllers and the coordinator is provided with the scope relationship tree, default compensation order graphs, and starter service addresses. The coordinator implementation extends the WS-Coordination system in [27] by plugging in the new protocols (Figure 5, right). This consists of providing: (1) the registration handler that provides protocol registration, including handling the extensions in the registration message and using them to create a coordinator side instance identifier, seeing whether an instance of the activity already exists, creating the reference properties needed for the coordinator address sent back in the registration response message, (2) the coordinator services for incoming protocol messages and the outgoing invocation for the outbound ones, and (3) the corresponding behavioral logic such as searching for fault handlers, grouping fragments of the same instance together based on keys generated using input from the registration handler, calculating compensation order. The coordinator stores state in stable storage using mySQL database tables, initialized using the scope and loop relationship tree. The table data is used by the handler and the coordinator logic. Upon receiving a registration message, the handler checks whether it already has a previous fragment registered for that same activity. If not, it creates a new record for that instance of the split activity. Otherwise, it adds the fragment to the table for that instance, and then creates and sends the address for the coordinator instance using the same reference properties as for other fragments of this split activity. As the messages for the protocol of that activity instance are exchanged, the tables get updated accordingly.

The runtime subsystem implementation currently supports the protocols except default compensation. It requires fragments of the same process instance to interact with the same coordinator. A set of processes was created to test that the execution semantics of the unsplit process model is preserved using the implementation. Details on this validation are in [22]. The tested processes contain split loops and/or scopes, including nesting, and different variations on split fault and compensation handling.

## 6 Related Work

WS-Coordination has been used with BPEL to coordinate nested scopes in BPEL 1.1 (of which sub-processes [9] are a special case) or provide transaction capabilities [25] to all interaction activities in a scope or related to a partnerLink. [23] compares WS-BusinessActivity to the parent-child scope relationships in a BPEL process. [15] extends WS-BA to handle scopes spanning multiple processes and their interaction with local scopes. SELF-SERV [2] uses coordination between services in the same composition to enable rapid service composition.

There is a large body of work on distributed workflows. In [3], a pub/sub event mechanism wires together processes. CrossFlow [8] focuses on inter-organizational workflow. Several approaches start with one workflow and then split it apart: using state-charts [18], deriving individual workflows for each organization using Petri Nets [26], mapping from a conversation-based language (WS-CDL [17] or a state-machine based language [7]) to BPEL. [28] and [6] split abstract process models, recreating dependencies using exchanged messages and [6] adds synchronization patterns for loops, multiple instances and a discriminator but no scopes. [1] provides a high level approach for splitting BPEL resulting in BPEL fragments. Other work has distributed the execution of one workflow for runtime optimization, such as OSIRIS [24] or the tuplespace approach in [16]. [4] uses program analysis and node reordering to maximize the throughput by breaking down a BPEL process into several BPEL processes. [19] shows a model to analyze the effects of local and remote check-pointing for fault handling and envisions embedding it in agent based workflow systems for run-time fault tolerance technique selection. Aspect Oriented Programming has been used to inject behavior into BPEL processes [5]. It may be used to add the participant side of the protocol to AOP-enabled BPEL engines. The above approaches have partial to no support for splitting compensation and fault handling. Some have different goals. The goal misalignment is due to the use of non-BPEL meta-models (UML Activity diagrams, state charts, custom abstractions), requiring of new execution runtimes (coordination spaces, SELF-SERV, Osiris), and/or lack of transparency because runtime artifacts (tuples) are in different meta-models than build-time artifacts (process model).

## 7 Conclusion

We have provided an automatic and operational semantics-preserving decomposition of recovery, rollback, and looping capabilities in fragmented business processes. The approach is modular, extending existing standards and runtimes; the implementation build on an existing BPEL engine and WS-Coordination system. The paper complements our prior work on process decomposition to result in a decomposition solution, extensively detailed in [10], for comprehensive business process functionality in a transparent and interoperable manner. Future research includes fragment evolution, autonomy and reuse, and a study of how, when and why one moves along the middleware-requirements spectrum of fragmentation solutions: from plain BPEL engines to modularly adding extensions to fully distributed process execution platforms.

**Acknowledgement.** Michael Paluszek's hard work building the runtime.

## References

1. Baresi, L.: Towards distributed BPEL orchestrations. In: Proc. of SeTra workshops, EASST (2006)
2. Benatallah, B., Dumas, M., Maamar, Z.: Definition and Execution of Composite Web Services: The SELF-SERV Project. IEEE Data Eng. Bull. 25(4) (2002)
3. Casati, F., Discenza, A.: Supporting Workflow Cooperation Within and Across Organizations. In: Proc. of SAC., ACM, New York (2000)
4. Chafle, G., Chandra, S., Mann, V., Gowri Nanda, M.: Decentralized Orchestration of composite Web Services. In: WWW Alternate Track. ACM, New York (2004)
5. Charfi, A., Mezini, M.: AO4BPEL: An aspect-oriented extension to BPEL. World Wide Web Journal 10(3) (2007)
6. Fdhila, W., Godart, C.: Toward synchronization between decentralized orchestrations of composite web services. In: Proc. of CollaborateCom (2009)
7. Fu, X., Bultan, T., Su, J.: A top-down approach to modeling global behaviors of Web services. In: Proc. of REOS Workshop. IEEE, Los Alamitos (2003)

8. Grefen, P., Hoffner, Y.: CrossFlow - cross-organizational workflow support for virtual organizations. In: Prof. of RIDE Workshop, Washington, DC. IEEE, Los Alamitos (1999)
9. IBM and SAP. WS-BPEL Extension for Sub-processes – BPEL-SPE (2005), http://www-128.ibm.com/developerworks/library/specification/ws-bpelsubproc/
10. Khalaf, R.: Supporting Business Process Fragmentation While Maintaining Operational Semantics: A BPEL Perspective. PhD thesis, University of Stuttgart, dissertation.de (2008), ISBN 978-3-86624-344-6, http://elib.uni-stuttgart.de/opus/volltexte/2008/3514/
11. Khalaf, R., Karastoyanova, D., Leymann, F.: Pluggable Framework for Enabling the Execution of Extended BPEL Behavior (Vienna, Austria). In: Di Nitto, E., Ripeanu, M. (eds.) ICSOC 2007. LNCS, vol. 4907, pp. 376–387. Springer, Heidelberg (2009)
12. Khalaf, R., Kopp, O., Leymann, F.: Maintaining data dependencies across BPEL process fragments. Int'l Journal of Cooperative Information Systems 17(3) (2008)
13. Khalaf, R., Leymann, F.: Role-based decomposition of business processes using BPEL. In: ICWS, Industry Track, Chicago, USA, pp. 770–780 (2006)
14. Khalaf, R., Roller, D., Leymann, F.: Revisiting the behavior of fault and compensation handlers in ws-bpel. In: Meersman, R., Dillon, T., Herrero, P. (eds.) OTM 2009. LNCS, vol. 5870, pp. 286–303. Springer, Heidelberg (2009)
15. Kopp, O., Leymann, F.: The influence of an external transaction on a BPEL scope (Vilamoura, Portugal). In: Meersman, R., Dillon, T., Herrero, P. (eds.) OTM 2009. LNCS, vol. 5870, pp. 381–388. Springer, Heidelberg (2009)
16. Martin, D., Wutke, D., Leymann, F.: Synchronizing control flow in a tuplespace-based, distributed workflow management system. In: Proc. of ICEC. ACM, New York (2008)
17. Mendling, J., Hafner, M.: From Inter-Organizational Workflows to Process Execution: Generating BPEL from WS-CDL. In: Proc. of OTM, Springer, Heidelberg (2005)
18. Muth, P.r., Wodtke, D., Weissenfels, J., Dittrich, A.K., Weikum, G.: From centralized workflow specification to distributed workflowexecution. Journal of Intelligent Information Systems 10(2) (1998)
19. Nichols, J., Demirkan, H., Goul, M.: Towards a model of fault tolerance technique selection in static and dynamic agent-based inter-organizational workflow management systems. In: Proc. of HICSS. IEEE, Los Alamitos (2005)
20. OASIS. Web Services Business Process Execution Language (WS-BPEL) Version 2.0 (2007), http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html
21. OASIS. WS-TX 1.2 OASIS Standards (2007), http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-tx#technical
22. Paluszek, M.: Coordinating distributed loops and fault handling, transactional scopes using WS-Coordination protocols layered on WS-BPEL services. University of Stuttgart Diploma Thesis 2586 (2007)
23. Sauter, P., Melzer, I.: A comparison of WS-Business Activity and BPEL4WS long-running transaction. In: Kommunikation in Verteilten Systemen (2005)
24. Schuler, C., Weber, R., Schuldt, H., Schek, H.J.: Peer-to-peer process execution with Osiris (Trento, Italy). In: Orlowska, M.E., Weerawarana, S., Papazoglou, M.P., Yang, J. (eds.) ICSOC 2003. LNCS, vol. 2910, pp. 483–498. Springer, Heidelberg (2003)

25. Tai, S., Khalaf, R., Mikalsen, T.: Composition of coordinated Web services. In: Jacobsen, H.-A. (ed.) Middleware 2004. LNCS, vol. 3231, pp. 294–310. Springer, Heidelberg (2004)
26. van der Aalst, W.M.P., Weske, M.: The P2P approach to interorganizational workflows. In: Dittrich, K.R., Geppert, A., Norrie, M.C. (eds.) CAiSE 2001. LNCS, vol. 2068, p. 140. Springer, Heidelberg (2001)
27. Vetter, T.: Anpassung und implementierung verschiedener transaktionsprotokolle auf WS-Coordination. University of Stuttgart Diploma Thesis 2386 (2006)
28. Yildiz, U., Godart, C.: Towards decentralized service orchestrations. In: Proc. of ACM Symposium on Applied Computing. ACM, New York (2007)

# PAPEL: A Language and Model for Provenance-Aware Policy Definition and Execution

Christoph Ringelstein and Steffen Staab

WeST - Institute for Web Science and Technologies
University of Koblenz-Landau
Universitaetsstrasse 1, 56070 Koblenz, Germany
{cringel,staab}@uni-koblenz.de
http://west.uni-koblenz.de

**Abstract.** The processing of data is often restricted by contractual and legal requirements for protecting privacy and IPRs. Policies provide means to control how and by whom data is processed. Conditions of policies may depend on the previous processing of the data. However, existing policy languages do not provide means to express such conditions. In this work we present a formal model and language allowing for specifying conditions based on the history of data processing. We base the model and language on XACML.

**Keywords:** Context-aware processes, Distributed process execution, Process tracing.

## 1 Introduction

Contracts (e.g. service level agreements) and laws (e.g. privacy laws) entitle customers to control the processing of their data. *Policies are statements of the goals for the behavior of a system* [7] and thus provides means to control the processing of data. However, conditions of policies frequently depend on environmental (or contextual) information (e.g. sharing a health record is only allowed after the patients approval). Such policies demand for controlling the process not only with respect to the actual processing step (including actors, processed data, etc.) but also with respect to the history of the processing (e.g. the number of backups that have been made, or who has encrypted the data). In this paper, we focus on policies in business processes containing conditions based on the processing history; as is common in communities dealing with such processing histories, we call the information about processing histories *provenance information*.

We need to tackle several principal problems, to enable policy conditions to relate to provenance information: First, in closed environments the environmental information can be collected with various existing logging mechanisms and be accessed by means of various, often proprietary, solutions. However, as soon

R. Hull, J. Mendling, and S. Tai (Eds.): BPM 2010, LNCS 6336, pp. 195–210, 2010.

as processes span multiple organizations and data is transfered across organizational boundaries, the provenance information must be provided in a standardized manner. The open provenance model (OPM) [10] constitutes such an approach.

Second, the policy conditions must be able to relate to provenance information. Some policy languages allow for building policies containing conditions based on provenance information (e.g. XACML [1] or Rei [8]). However, existing policy mechanisms lack a specification of how to specify or access provenance information. Based on the open provenance model, we introduce a formal language that specifies the relation between policy condition and provenance information. We specify the language by means of an abstract syntax extending the syntactic structure of eXtensible Access Control Markup Language (XACML) and we provide a corresponding description of its execution semantics. We call our approach *PAPEL*: Provenance-Aware Policy definition and Execution Language. To show its feasibility, we implement PAPEL using Datalog.

Third, the provenance information may not be queriable, because the information may not be accessible (e.g. due to log encryption [11]). To be still able to validate compliance with the given policies, the required provenance information must stay accessible. We achieve this by introducing *attributes* and *reduced facts* in PAPEL. Both mechanisms ensure that only a minimum of confidential information is disclosed to third parties.

In this paper, we tackle these problems as follows: First, we introduce a case study, point out problematic issues and derive requirements in Section 2. Then, we discuss the foundations of PAPEL in Section 3, introduce the syntax of PAPEL in Section 4, and define the semantics in Section 5. In Section 6, we discuss the Datalog implementation of our abstract syntax. Finally, we analyze related work and its capabilities to solve the previously introduced problems in Section 7.

## 2 Case Study and Requirements

In this section, we present a case study depicting issues that occur with policies in distributed environments:

***Sharing of Health Records:*** A research institute of the `University of Koblenz (ukob)` requires data about actual patients for its research. To this end, it cooperates with the hospital `Koblenz Medical Center (kmc)`. The kmc shares its patients' health records with the research institute that can access the records on a server of the hospital. The patients may choose if their data can be used for research or not. The patients' permissions are required before the hospital is allowed to share the data. If patients permit the use, they may additionally demand that their data is only used after their names have been exchanged by pseudonyms.

Jane Doe a patient of the kmc has specified a list of policies regarding her health record (record_JD) and the forwarding of the record to the research laboratory of ukob:

- **(A)** *Jane Doe allows* kmc *to share her record for research purposes with* ukob.
- **(B)** *However, she demands that her record is de-identified before it is shared.*
- **(C)** *With the de-identified record the research laboratory is allowed to do anything, beside transferring it to another organization.*

In addition to the patient's policies, the kmc has own policies for forwarding health records:

- **(D)** *The* kmc *demands that the sharing of health records is approved and the approval confirmed before the record is accessed by* ukob.

In the case study multiple issues arise. We introduce a selection of these issues in the following list, before we describe what is required to solve the issues.

- **(1)** ukob will access health records. To this end, the institute needs to know which actions are permitted or restricted.
- **(2)** The provenance information of a patient may contain personal information. Thus, the kmc wants to encrypt the provenance information before forwarding a record. At the same time the kmc has to ensure that the information needed to interpret the policies is still available without decryption.

**Requirements**

In summary we can point out the following technical and organizational requirements:

- **Requirement 'Availability':** The policies must be available to anybody accessing the associated data (see issue (1)).
- **Requirement 'Expressiveness':** The used policy language must be able to express permissions and restrictions (see issue (1); cf. XACML [1]).
- **Requirement 'Accessibility':** The information required by policy conditions must be accessible even if confidential parts of the data and of the provenance information stay hidden (see issue (2)).

In addition to access policies such as specified in languages like XACML, we need provenance information to address these issues and thus to meet the requirements. In the following sections we present a solution consisting of three parts: first, the foundations we build on; second, an abstract syntax of our novel policy language PAPEL; and third, the semantics of this language.

## 3 Foundations of PAPEL

In this section we describe the foundations PAPEL is based on. As discussed above, we require to model provenance information and policies. To this end, we make use of concepts of the *Open Provenance Model* and of the *eXtensible Access Control Markup Language*. To integrate both we introduce an abstract syntax extending the abstract syntax of XACML.

### 3.1 Open Provenance Model

The *Open Provenance Model (OPM)* [10] defines a process as an *action or series of actions performed on or caused by artifacts, and resulting in new artifacts.* We represent the information we require and which is given by a graph structure in the OPM by a set of primitives. The transformation leads to provenance information represented by our abstract syntax, which is specified in Table 1. We use the `step` primitive:

```
step (Data, Actors, InvolvedAgents, Category, Purpose, ID, PIDs)
```

to express the following constituents of OPM:

- `Data` is the *artifact* processed during the execution of the processing step. In OPM an artifact is defined as an *immutable piece of state, which may have [..] a digital representation in a computer system.*
- `Actors` are the *agents* controlling the process, In OPM an agent is defined as a *contextual entity acting as a catalyst of a process, enabling, facilitating, controlling, affecting its execution.*
- `InvolvedAgents` are *agents* involved in a process that do not trigger the processing (e.g. the receiver of a transfer or access rights).
- `ID` is the unique *identifier* of the processing step (as defined by OPM[1]).
- `PIDs` are the unique *identifiers* of the directly preceding processing steps.

The `ID` together with the `PIDs` creates a partial order of processing steps. Beside these constituents of OPM, the `step` primitive also specifies the following properties of a processing step:

- `Category` is the category of the processing step. Concepts of categories may be used from any suitable domain specific ontology.
- `Purpose` is the purpose of executing the processing step. Concepts of purposes of processing steps may also be used from any suitable domain specific ontology.

**Example 1:** In the following we depict the provenance information of a processing step of the `Koblenz Medical Center (kmc)`: *In the depicted processing step the health record of Jane Doe is shared*[2] *for research purposes with the* `University of Koblenz (ukob)`:

```
step (record_JD, {kmc}, {ukob}, share, research, 3, {2})
```

[1] In the examples we make use of simple integers, to increase readability.

[2] In this and the following examples we assume that used concepts are defined in a domain ontology provided by kmc.

### 3.2 eXtensible Access Control Markup Language

We use The *eXtensible Access Control Markup Language (XACML)* [1] as a starting point for our formal model. XACML is a standard defining a XML based policy framework. XACML supports three policy elements, i.e. permission (*permit*), restriction (*deny*) and obligation, that we need to fulfill the Requirement 'Expressiveness'. To generalize from XACML and to integrate with the provenance information, we introduce an abstract syntax for permission, restrictions and obligations based on XACML in Section 4 below.

XACML policies define rules by connecting a set of subjects (actors) with a set of targets (data) and by specifying the conditions of the rule. If the conditions are fulfilled, XACML rules result in a given effect, which is *permit* or *deny*. In addition, a rule may require that certain obligations are met before a permission is granted. However, this kind of obligations is expressed as part of conditions in PAPEL (see Example 3) exploiting the provenance information.

## 4 Syntax of PAPEL

To evaluate rules, the provenance information can be used as source of information about the previous processing. PAPEL defines an abstract syntax (see Table 1) to express provenance information and polices rules based on provenance information.

The provenance information is provided by means of logging or monitoring mechanisms (cf. [11,12]). In PAPEL, we use primitives to define provenance information (`primitive` is the start symbol of the provenance part of the PAPEL syntax). We use the `step` primitive as introduced in Section 3 (see Example 1) and the `reduced` primitive to specify the processing provenance and the `attribute` primitive to specify attributes and their value (see Table 1). The processing steps, attribute assignments and reduced facts collected during the execution of a process build the processing history, which we define as follows:

***Definition 1:*** *We define a history $H$ as a conjunction of positive facts specifying processing steps, attribute assignments and reduced facts.*

Policy rules are provided by the person concerned or IPR (Intellectual Property Rights) owner. In PAPEL, we use rules to specify policies (`rule` is the start symbol of the policy part of the PAPEL syntax). We distinguish three types of rules: `permit`, `deny` and `assignment` rules. All three types may depend on a `condition` and have the parameter ID, which refers to the actual processing step. Assignment rules specify the change of an attribute value by use of the `set_attribute` primitive.

### Permissions and Restrictions

We model permissions as rules specifying which kinds of processing steps are permitted. Likewise, restrictions are modeled as rules specifying which processing

**Table 1.** Syntax of PAPEL

PAPEL Syntax for Provenance Information:

| Syntax Element | EBNF syntax |
|---|---|
| `Primitive` | `Step \| ReducedFact \| Attribute ;` |
| `Step` | `"step ("Data", "Actors", "InvolvedAgents", "Category", "Purpose", "ID", "PIDs")." ;` |
| `ReducedFact` | `"reduced ("Data", "(Actors \| "hidden")", "(InvolvedAgents \| "hidden")", "(Category \| "hidden")", "(Purpose \| "hidden")", "ID", "PIDs")." ;` |
| `Attribute` | `"attribute ("Data", "Name", "Value", "ID")." ;` |

PAPEL Syntax for Policies:

| Syntax Element | EBNF syntax |
|---|---|
| `Rule` | `Permission \| Restriction \| Assignment ;` |
| `Permission` | `"permit (ID) IF " Condition "." ;` |
| `Restriction` | `"deny (ID) IF " Condition "." ;` |
| `Assignment` | `"assignment (ID) IF " Condition "DO" SetAttribute \| SetReducedFact "." ;` |
| `Condition` | `Primitive \| ("(NOT" Primitive \| Condition \| "permit (ID)" \| "deny (ID)" ")") \| ("(" Primitive \| Condition \| "permit (ID)" \| "deny (ID)" BooleanOperator Primitive \| Condition \| "permit (ID)" \| "deny (ID)" ")") \| (Step \| ReducedFact "AFTER" Step \| ReducedFact ")") ;` |
| `SetAttribute` | `"set_attribute ("Data", "Name", "Value", "ID")." ;` |
| `SetReducedFact` | `"set_reduced ("Data", "(Actors \| "hidden")", "(InvolvedAgents \| "hidden")", "(Category \| "hidden")", "(Purpose \| "hidden")", "ID", "PIDs")." ;` |
| `BooleanOperator` | `"AND" \| "OR" \| "XOR" ;` |

steps are prohibited. The rules consist of two parts: the name, which indicates the type of the policy (`permit` or `deny`), and the body of the rule, which defines the conditions and obligatory processing steps. Conditions are used to express dependencies between policies and provenance information and are specified after the `IF` statement (see Table 1).

**Example 2:** This example formalizes policy (A): *All entities are denied to transfer the data* `record_JD`*, beside* `kmc` *that is allowed for transferring the data for research purposes to* `ukob`*:*

```
permit (ID) IF step (record_JD, {kmc}, {ukob}, transfer,
               research, ID, _).
deny (ID)   IF step (record_JD, _, _, transfer, _, ID, _) AND
               NOT permit (ID).
```

In PAPEL restrictions override permissions (see 'Fulfilling Policies' in section 5). However, in some instances a restriction should be overridden by a permissions, as shown in the example above: The second rule of this example denies the transfer of `record_JD`. By means of the additional part of the condition `AND NOT permit (ID)` we can define exceptions to this rule. In the example the exception is defined by the first rule and allows `kmc` to transfer the record to `ukob`.

**Example 3:** The following example depicts the implementation of policy (B) of the running example (see Section 2). *The patient demands that her* `record_JD` *is de-identified before it is transfered:*

```
permit (ID) IF step (record_JD, _, _, transfer, _, ID, {PID}) AND
            step (record_JD, _, _, update, de-identify, PID, _).
```

By using variables (in the example `PID`), we can specify that the approval step has to be performed *directly* before the processing step, which should be permitted. If the variable is not used, any preceding approval step will fulfill the condition.

**Example 4:** In this example we express the policy (C) from our running example: *All processing steps that are not transfer actions will be permitted if they are performed by* `ukob`:

```
permit (ID) IF (step (_, {ukob}, _, Category, _, ID, _) AND
            NOT (Category = transfer)).
```

In the example above we make use of the logical constant `ukob`, which is defined in a domain ontology. In addition, we use the variable `Category` and unnamed variables indicated by `_`. The `_` represents another unnamed variable each time it is used. An unnamed variable matches all possible values of its type (cf. existential quantification).

### Condition Statements

In PAPEL, condition statements can be composed using logical operators (`NOT`, `AND`, `OR`, `XOR` and `AFTER`) How these operators can be combined is specified in Table 1. In addition, Table 1 depicts, that parentheses are used to specify the interpretation order of complex statements. We define the semantics of the operators in Table 2.

**Example 5:** The following example illustrates the implementation of policy (D) from the running example. The policy is specified by means of a permission in combination with a condition: *If the approval of the access has been confirmed, permit the transfer:*

```
permit (ID) IF (step (R, {ukob}, _, access, _, ID, _) AND
            (step (R, {kmc}, _, _, confirmation, _, _) AFTER
            step (R, {kmc}, _, _, access_approval, _, _))).
```

In difference to Example 3the rule in this example uses AFTER. By means of the AFTER construct we can access the partial order of the processing steps in the history. The variable R assures that the approval and confirmation is given for the health record that should be accessed.

### Attributes and Reduced Facts

As stated above the provenance information can be used as source of information about the previous processing. However, the provenance information may contain sensitive data (e.g. the provenance information that the cancer medication has been adjusted). Thus, some logging approaches provide security mechanisms like the encryption of the provenance information (e.g. [11]). To maintain the ability to check the compliance with a policy, a method is needed to enable access to relevant information or provide the information directly (Requirement 'Accessibility'). To this end, we introduce *attributes* as well as *reduced facts*.

Attributes can be used to specify provenance information that can be expressed by a value (e.g. de-identification status, modification counter, etc.). We specify attributes by the attribute primitive (see Table 1). The attribute primitive has four parameters, the Data item the attribute relates to, the Name of the attribute, its assigned Value, and the identifier ID of the processing step, when the assignment was performed. The value of the attribute is assigned by the actor performing the processing step. The assignment is done according to the assignment rule defined by the creator of the policies.

**Example 6:** This example illustrates the permit rule and the assignment rule required to implement policy (B) of the running example by means of an attribute de-identified. In difference to the implementation of policy (B) we introduced above (see Example 3), this implementation allows for de-identifying the health record at any time: *The patient demands that her record (record_JD) is de-identified before it is transfered:*

```
permit(ID) IF (step (record_JD, _, _, transfer, _, ID, _) AND
           attribute (record_JD, de-identified, true, ID)).
```

*If the executed processing step replaces all names in record_JD with pseudonyms, set the attribute named* de-identified *to* true*, and if the record is re-identified, set the attribute to* false*:*

```
assignment(ID) IF step (record_JD, _, _, _, de-identified, ID, _)
               DO set_attribute (record_JD, de-identified, true, ID).
assignment(ID) IF step (record_JD, _, _, _, re-identified, ID, _)
               DO set_attribute (record_JD, de-identified, false, ID).
```

A reduced fact is a description of a processing step that is reduced to the necessary and contains only the required information. Which information is hidden is specified by means of assignments that are defined by the creator of the policy. Reduced facts are used if attributes are not expressive enough (e.g. has the last de-identification been performed by a specific actor). A reduced fact is added to the provenance information. In difference to the full description, the information of the reduced fact must not be encrypted to allow for querying the information. The adding of the reduced fact is achieved by the actor performing the processing step following assignments defined by the policy creator.

The parameters `Data`, `Actors`, `InvolvedAgents`, `Category`, and `Purpose` are replaced by the empty placeholders `hidden` as required. The parameters `ID` and `PIDs` must not be replaced. These two parameters are required to reproduce the (partial) order of processing steps.

**Example 7:** In the following we depict the provenance information of a processing step of the department for `nuclear medicine`: *The processing step updates the health record by adding a new cancer medication:*

```
step (record_JD, {nuclear_medicine}, {},
      update, new_cancer_medication, 3, {2})
```

*If this information is not relevant to the analysis of* `ukob`*, the* `kmc` *will encrypt the original information about the processing step and will provide reduced provenance information:*

```
reduced (record_JD, hidden, hidden, update, hidden, 3, {2})
```

## 5 Execution Semantics of PAPEL

In this section we define the semantics of PAPEL. The semantics of PAPEL specifies if execution steps of a given history violate a given set of policy rules. To this end, we define a Tarskian semantics mapping syntactic elements of PAPEL onto subsets and relations over a universe $U$. Based on this semantics we define when a set of policies is fulfilled with respect to a history $H$ (see Definition 1). First we define minimal models of a history $H$:

***Definition 2:*** *We define a minimal model $M$ of the history $H$ as a model to which no strictly smaller Herbrand model of the history $H$ exist [9].*

A minimal model of a history is only a model of the history and of subparts of the history. Based on the definition of a minimal model of a history we define when a set of policies is fulfilled:

***Definition 3:*** *We define that a set of policy rules $R$ is fulfilled with respect to a history $H$ if each minimal model $M$ of the history $H$ is also a model of the set of policy rules $R$.*

### The Universe and Basic Axioms

The universe $U$ is the set: $U = \mathit{\Delta} \cup A \cup X \cup \mathit{\Psi} \cup \mathit{\Phi} \cup N \cup V$, where $\mathit{\Delta}$ is the set of artifacts (e.g. data), $A$ is the set of agents (including actors), $X$ is the set of categories, $\mathit{\Psi}$ is the set of purposes, $\mathit{\Phi}$ is the set of processing step identifiers, $N$ is the set of attribute identifiers, and $V$ the set of attribute values. These subsets of U are mutually disjoint.

For the following definitions let $I$ be a partial interpretation function and let $P(S)$ be the power set of $S$. We define $I$ to map atomic elements of PAPEL onto the (parts of) our universe $U$ as follows: $\delta^I \in \mathit{\Delta}$, $\alpha^I \subseteq A$, $\beta^I \subseteq A$, $\chi^I \in X$, $\psi^I \in \mathit{\Psi}$, $id^I \in \mathit{\Phi}$, $\rho^I \subseteq \mathit{\Phi}$, $\mu^I \in N$, and $\nu^I \in V$. The predicate representing the processing steps *step* is interpreted as: $step^I \subseteq \mathit{\Delta} \times P(A) \times P(A) \times X \times \mathit{\Psi} \times \mathit{\Phi} \times P(\mathit{\Phi})$ and the predicate representing the attribute assignments *attribute* is interpreted as: $attribute^I \subseteq \mathit{\Delta} \times N \times V \times \mathit{\Phi} \times P(\mathit{\Phi})$.

Thereby, the partial interpretation function $I$ must satisfy the following constraints: Each identifier $id^I$ must clearly identify one processing step:

$$\forall id^I \in \mathit{\Phi} : (\delta_1^I,\ \alpha_1^I,\ \beta_1^I,\ \chi_1^I,\ \psi_1^I,\ id^I,\ \rho_1^I), (\delta_2^I,\ \alpha_2^I,\ \beta_2^I,\ \chi_2^I,\ \psi_2^I,\ id^I,\ \rho_2^I) \in step^I \Rightarrow \delta_1^I = \delta_2^I \wedge \alpha_1^I = \alpha_2^I \wedge \beta_1^I = \beta_2^I \wedge \chi_1^I = \chi_2^I \wedge \psi_1^I = \psi_2^I \wedge \rho_1^I = \rho_2^I.$$

Analogously, at each processing step specified by $id^I$ each attribute has exactly one value at a time:

$$\forall id^I \in \mathit{\Phi} : (\delta^I,\ \mu^I,\ \nu_1^I,\ id^I,\ \rho_1^I), (\delta^I,\ \mu^I,\ \nu_2^I,\ id^I,\ \rho_2^I) \in attribute^I \Rightarrow \nu_1^I = \nu_2^I \wedge \rho_1^I = \rho_2^I.$$

The processing history is defined as partial order $> \subseteq \mathit{\Phi} \times \mathit{\Phi}$ of processing steps, reduced facts, and attribute assignments. A tuple $(id^I,\ pid^I)$ is element of $>$, if and only if the processing step specified by *pid* precedes the processing step specified by *id*. Before we define $>$, we define the relation $\succ \subseteq \mathit{\Phi} \times \mathit{\Phi}$ of directly preceding processing steps, reduced facts, and attribute assignments. A processing step $s_{dpid}$ immediately precedes a processing step $s_{id}$, if $s_{id}$ is the successor of $s_{dpid}$:

$$\succ\ = \{(id^I,\ dpid^I) | \exists (\delta_s^I,\ \alpha_s^I,\ \beta_s^I,\ \chi_s^I,\ \psi_s^I,\ id^I,\ \rho_s^I) \in step^I \wedge dpid^I \in \rho_s^I \vee \exists (\mu_a^I,\ \nu_a^I,\ id^I,\ \rho_a^I) \in attribute^I \wedge dpid^I \in \rho_a^I\}.$$

A processing step precedes another processing step, if a trace of consecutive processing steps exists that connects both steps:

$$> = \{(id^I,\ pid^I) | \succ (id^I,\ pid^I) \vee \exists mid^I : \succ (id^I, mid^I) \wedge > (mid^I, pid^I)\}.$$

In the following, we extend the interpretation function $I$ by the interpretation of non-atomic syntactic expressions of PAPEL.

### Processing Steps, Reduced Facts and Attributes

***Processing Step:*** Let _ be an unspecified parameter and be $s'=(\delta',\ \alpha',\ \beta',\ \chi',\ \psi',\ id',\ \rho')$, the *step* predicate is interpreted as follows:

$$(step(\delta,\ \alpha,\ \beta,\ \chi,\ \psi,\ id,\ \rho))^I = \begin{cases} \text{true} & \text{if } \exists s' \in step^I : (\delta^I = \delta'^I \vee \delta^I = \_)\wedge \\ & (\alpha^I = \alpha'^I \vee \alpha^I = \_) \wedge\ ..., \\ \text{false} & \text{else.} \end{cases}$$

***Reduced Fact:*** The *reduced* relation is interpreted as follows:

$$(reduced(\delta_r,\ \alpha_r,\ \beta_r,\ \chi_r,\ \psi_r,\ id_r,\ \rho_r))^I = \begin{cases} \text{true} & \text{if } \exists(\delta_s^I,\ \alpha_s^I,\ \beta_s^I,\ \chi_s^I,\ \psi_s^I,\ id_s^I, \\ & \rho_s^I) \in step^I :\ id_r^I = id_s^I\ \wedge \\ & (\delta_r^I = \delta_s^I \vee \delta_r^I = hidden) \wedge\ .., \\ \text{false} & \text{else.} \end{cases}$$

***Setting of Reduced Facts:*** Be $(\delta_s,\ \alpha_s,\ \beta_s,\ \chi_s,\ \psi_s,\ id_s,\ \rho_s) \in step$ the processing step described by the reduced fact, be $id_r = id_s$ and $\rho_r = \rho_s$, and be $\delta_r = \delta_s \vee \delta_r = hidden$, $\alpha_r = \alpha_s \vee \alpha_r = hidden$, $\beta_r = \beta_s \vee \beta_r = hidden$, $\chi_r = \chi_s \vee \chi_r = hidden$, and $\psi_r = \psi_s \vee \psi_r = hidden$. The *set_reduced* predicate is interpreted as follows:

$$(set\_reduced(\delta_r,\ \alpha_r,\ \beta_r,\ \chi_r,\ \psi_r,\ id_r,\ \rho_r))^I = \begin{cases} \text{true} & \text{if } (\delta_r^I,\ \alpha_r^I,\ \beta_r^I,\ \chi_r^I,\ \psi_r^I, \\ & id_r^I,\ \rho_r^I) \in reduced^I, \\ \text{false} & \text{else.} \end{cases}$$

***Attributes:*** The *attribute* relation is interpreted as follows:

$$(attribute(\delta, n, v, id, \rho))^I = \begin{cases} \text{true} & \text{if } (\delta^I, n^I, v^I, id^I, \rho^I) \in attribute^I, \\ \text{true} & \text{if } (\delta^I, n^I, v^I, id^I, \rho^I) \notin attribute^I \wedge \\ & \exists(\delta^I, n^I, v^I, id_l^I, \rho_l^I) \in attribute^I \wedge > (id^I, id_l^I) \wedge \\ & \neg\exists(\delta_m^I, n_m^I, v_m^I, id_m^I, \rho_m^I) \in \varPhi :> (id^I, id_m^I) \wedge \\ & > (id_m^I, id_l^I), \\ \text{false} & \text{else.} \end{cases}$$

An attribute has the value assigned in the actual step, or if no value is assigned in the actual step, it will have the value that has been assigned last.

***Setting of Attributes:*** The *set_attribute* predicate is interpreted as follows:

$$(set\_attribute(\delta, n, v, id, \rho))^I = \begin{cases} \text{true} & \text{if } \forall fid^I : \succ (fid^I, id^I) \rightarrow (\delta^I, n^I, v^I, fid^I, \\ & \{id^I\}) \in attribute^I, \\ \text{false} & \text{else.} \end{cases}$$

The result of the interpretation of the *set_attribute* predicate will be true if the value of the attribute is updated in all directly succeeding processing steps.

### Logical Expressions

Above we introduced the following syntactical elements `NOT`, `AND`, `OR`, and `XOR`. Their semantics are defined in the same manner as the corresponding boolean expressions (see Table 2). For example the $not \subseteq \{true,\ false\}$ relations is defined as:

$$(not(e))^I = \begin{cases} \text{true} & \text{if } e^I = false, \\ \text{false} & \text{else.} \end{cases}$$

In addition, we introduce the syntactical element `AFTER`, which is defined as the following relation: $after^I \subseteq (step^I \cup reduced^I) \times (step^I \cup reduced^I)$, where:

$$(after(s_1, s_2))^I = \begin{cases} \text{true} & \text{if } s_1 = (.., id_1, ..) \wedge s_2 = (.., id_2, ..) \wedge id_1 > id_2, \\ \text{false} & \text{else.} \end{cases}$$

**Table 2.** Condition Statements

| Syntax | Natural language semantics |
|---|---|
| `A` | *A* is true |
| `NOT A` | *A* is not true |
| `A AND B` | *A* and *B* are true |
| `A OR B` | *A* or *B* (non-exclusive or) are true |
| `A XOR B` | either *A* or *B* (exclusive or) is true |
| `B AFTER A` | first *A* and then *B* (in the given order[3]) are true |
| `IF Condition` | if the *Condition* is fulfilled |
| `DO ObligatoryActions` | the *ObligatoryActions* have to be performed |

### Permission, Restriction and Assignment

Be $L$ the set of functions implementing logical expressions. Permissions, restrictions, and assignments are functions $permit^I : \Phi \mapsto \{true,\ false\}$, $deny^I : \Phi \mapsto \{true,\ false\}$, and $assignment^I : \Phi \mapsto \{true,\ false\}$ as follows:

[3] The processing history can be represented as a directed graph (see Section 5). Thus, processing steps can be in order if they can be connected by a path in the graph.

*permit* : $\Phi \mapsto \{true,\ false\}$ is a function that will return *true* if the processing step identified by $ID$ is permitted. If it is not permitted, it will return $false$. Be $p_1,\ p_2,\ ..,\ p_n \in L$ the functions implementing the conditions of all rules defining permissions. The predicate *permit* is interpreted as follows:

$$(permit(ID))^I = \begin{cases} \text{true} & \text{if } \exists\ (\delta^I,\ \alpha^I,\ \beta^I,\ \chi^I,\ \psi^I,\ id^I,\ \rho^I) \in step^I : \\ & ID^I =\ id^I \wedge (p_1^I \vee p_2^I \vee .. \vee p_n^I) = true \\ \text{false} & \text{else} \end{cases}$$

*deny* : $\Phi \mapsto \{true,\ false\}$ is a function that will return *true* if the processing step identified by $ID$ is *not* denied. If it is denied, it will return $false$. Be $d_1,\ d_2,\ ..,\ d_m \in L$ the functions implementing the conditions of all rules defining restrictions and be $s = (\delta,\ \alpha,\ \beta,\ \chi,\ \psi,\ id,\ \rho)$, *deny* is interpreted as follows::

$$(deny(ID))^I = \begin{cases} \text{false} & \text{if } \exists\ s^I \in step^I : ID^I =\ id^I \wedge\ (d_1^I \vee d_2^I \vee .. \vee d_m^I) = true \\ \text{true} & \text{else} \end{cases}$$

*assignment* : $\Phi \mapsto \{true,\ false\}$ is a function that will return *true* if the processing step identified by $ID$ does not violate an assignment. If it violates an assignment, it will return $false$. Be $c_1,\ c_2,\ ..,\ c_k \in L$ the functions implementing the conditions of all rules defining assignments and be $a_1,\ a_2,\ ..,\ a_k$ the *set_attribute* and *set_reduced* predicates of all rules defining assignments, *assignment* is interpreted as follows:

$$(assignment(ID))^I = \begin{cases} \text{true} & \text{if } \exists\ s^I \in step^I : ID^I\ =\ id^I \wedge \\ & ((\neg c_1^I \vee o_1^I) \wedge (\neg c_2^I \vee o_2^I) \wedge .. \wedge (\neg c_k^I \vee o_k^I)) = true \\ \text{false} & \text{else} \end{cases}$$

### Fulfilling Policies

Policies will be fulfilled with respect to a history if each minimal model of a history is also a model of the policies (see Definition 3). This definition in combination with the PAPEL semantics have the following conclusion: A processing history will fulfill a set of policy rules, if all processing steps in the history are permitted with respect to the permissions ($permit^I$) and not prohibited with respect to the restrictions ($deny^I$) and if no assignment ($assignment^I$) has been violated.

## 6 Datalog Implementation of PAPEL

Implementing PAPEL, we have demonstrated its feasibility. In the implementation the provenance information is provided as database. To access the provenance information we make use of the database query language Datalog. We have chosen Datalog to provide a general implementation with a formal grounding. In our implementation, we use facts to specify the provenance information and rules to query it.

To validate a set of policies the following preparation steps are required: (1) All Datalog rules required to specify syntactical elements are added to the database; (2) A Datalog fact is generated for each processing step, which is described by the provenance information. The facts are added to the database; (3) The Datalog rules specifying the policies are added to the database; (4) A Datalog fact specifying the processing step that should be performed is added to the database; and (5) For each attribute Datalog facts for each processing step are generated and added to the database.

Then, the database is queried by means of a rule, which specifies the query if the processing step identified by the given $ID$ will be allowed or not.

## 7 Related Work

Many policy languages exist that are applicable for our purpose. Thus, we based PAPEL on these existing languages. We have decided to use the main policy elements 'restriction' (deny) and 'permission' (permit), as defined in the *eXtensible Access Control Markup Language (XACML)* [1]. Polices, which are based on complex environmental knowledge such as provenance information, can not be specified using XACML. With our policy language we extend the expressiveness of XACML conditions to cover information about the processing history of data.

As a foundation for more complex policy conditions we used the work of Kagal et al. [8]. They present a domain-centric and entity-centric policy language named Rei. Rei provides a mechanism for modeling of speech acts and delegation of policies. The conditions in Rei are specified by means of Prolog. However, Rei does not define how to model conditions based on provenance information. We extend the work of Rei by defining a formalism for conditions based on provenance information and by providing a model-theoretical semantics.

Other work which is related to our work is the *Web Services Policy 1.5 - Framework (WS-Policy)* [2] that provides a model and syntax to specify policies about entities in a Web services-based system. As WS-Policy is used to specify the policies about entities, e.g. Web services, and not of the processed data, it is not in the target of the WS-Policy framework to model conditions based on provenance information.

In Table 3, we give an overview of further related work in the field of policy languages. The Table compares the following properties of of policy languages: Are policies directly linked to a piece of data? Does the policy language allow for expressing dataflow polices and/or access control policies? Do policy conditions allow for relating to provenance information (processing traces)? Does the policy language allow for protecting confidential provenance information. Finally, we compare if the language has a syntax definition, a formal semantics and an implementation.

**Table 3.** Related Work

| Property ╲ Approach | PAPEL | XACML [1] | EPAL [4] | WS-Policy [2] | Rei [8] | XrML [13] | Casandra [5] | e-Wallet [6] | IFAudit [3] |
|---|---|---|---|---|---|---|---|---|---|
| Linked to Pieces of Data | ✓ | - | - | - | - | ✓ | - | ✓ | - |
| Dataflow Policies | ✓ | - | - | - | - | - | - | - | ✓ |
| Access Control Policies | ✓ | ✓ | ✓ | (1) | ✓ | ✓ | ✓ | ✓ | - |
| Provenance Information | ✓ | - | - | - | - | - | - | - | ✓ |
| Protection of Provenance Information | ✓ | - | - | - | - | - | - | - | - |
| Syntax Definition | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Formal Semantics | ✓ | - | ✓ | - | (✓) | - | ✓ | (2) | ✓ |
| Implementation | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

(1) WS-Policy provides data usage policies.

(2) e-Wallet provides a formal policy processing algorithm.

However, all of them provide additional aspects of policy languages that are not in our focus. Some policy approaches provide methods to enforce policy compliance in closed environments, like organizations (cf. [4]) or data silos (cf. [6]). Other languages define policies of actors (cf. [2]) not resources. Finally some languages consider credentials to gain access rights (cf. [13,5]), support roles and role delegation (cf. [5]), or provide algorithms to identify violated policies (cf. [3]).

## 8 Conclusion

Existing policy languages provide means to control how and by whom data is processed. The conditions of policies may depend on the previous processing of the data. Such policies demand for controlling the process with respect to the history of the processing. However, existing policy languages do not specify how to access the provenance information about the previous processing. In this work we have introduced PAPEL a provenance-aware policy definition and execution language motivated by XACML. PAPEL allows for policy conditions to relate to provenance information.

We have presented the abstract syntax of PAPEL, which extends the syntactic structures of XACML and OPM. In addition, we have provided a corresponding description of its execution semantics and we have sketched an implementation of PAPEL using Datalog.

In addition we have discussed how data protection can hamper the interpretation of policy conditions. We have presented means to validate compliance with given policies even if the required provenance information is confidential. At the same time the confidential methods of PAPEL ensure that only a minimum of confidential information is disclosed to third parties.

## Acknowledgment

The basic idea of this work has been motivated by discussion with Prof. Marianne Winslett and her working group. This work has been supported by the European projects Where eGovernment meets the eSociety (WeGov, FP7-248512) and Knowledge Sharing and Reuse across Media (X-Media, FP6-26978) funded by the Information Society Technologies (IST) 6th and 7th Framework Programme.

## References

1. eXtensible Access Control Markup Language (XACML) Version 2.0. Oasis standard, OASIS (February 2005)
2. Web Services Policy 1.5 - Framework. W3c recommendation, W3C (September 2007)
3. Accorsi, R., Wonnemann, C.: Auditing workflow executions against dataflow policies. In: BIS 2010: Proceedings of the 13th International Conference on Business Information Systems (2010)
4. Ashley, P., Hada, S., Karjoth, G., Powers, C., Schunter, M.: Enterprise Privacy Authorization Language (EPAL 1.2). Submission to W3c, W3C (November 2003)
5. Becker, M.Y., Sewell, P.: Becker and Peter Sewell. Cassandra: Distributed access control policies with tunable expressiveness. In: POLICY 2004: Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks, Washington, DC, USA, 2004, p. 159. IEEE Computer Society Press, Los Alamitos (2004)
6. Gandon, F.L., Sadeh, N.M.: Semantic web technologies to reconcile privacy and context awareness. J. Web Sem. 1(3), 241–260 (2004)
7. Hinton, H.M., Lee, E.S.: The compatibility of policies. In: CCS 1994: Proceedings of the 2nd ACM Conference on Computer and Communications Security, pp. 258–269. ACM, New York (1994)
8. Kagal, L., Finin, T., Joshi, A.: A policy language for a pervasive computing environment. In: IEEE International Workshop on Policies for Distributed Systems and Networks, pp. 63–75 (2003)
9. Lloyd, J.W.: Foundations of Logic Programming. Springer, New York (1993)
10. Moreau, L., Freire, J., Futrelle, J., Mcgrath, R., Myers, J., Paulson, P.: The open provenance model: An overview. In: Freire, J., Koop, D., Moreau, L. (eds.) IPAW 2008. LNCS, vol. 5272, pp. 323–326. Springer, Heidelberg (2008)
11. Ringelstein, C., Staab, S.: Logging in Distributed Workflows. In: Proceedings of the Workshop on Privacy Enforcement and Accountability with Semantics, Busan, South-Korea (2007)
12. Ringelstein, C., Staab, S.: Dialog: Distributed auditing logs. In: IEEE International Conference on Web Services, Los Angeles, CA, USA, pp. 429–436. IEEE Computer Society Press, Los Alamitos (2009)
13. Wang, X., Lao, G., DeMartini, T., Reddy, H., Nguyen, M., Valenzuela, E.: Xrml – extensible rights markup language. In: XMLSEC 2002: Proceedings of the 2002 ACM Workshop on XML Security, pp. 71–79. ACM, New York (2002)

# A Fresh Look at Precision in Process Conformance

Jorge Muñoz-Gama and Josep Carmona

Universitat Politècnica de Catalunya, Spain
jmunoz@lsi.upc.edu, jcarmona@lsi.upc.edu

**Abstract.** Process Conformance is a crucial step in the area of Process Mining: the adequacy of a model derived from applying a discovery algorithm to a log must be certified before making further decisions that affect the system under consideration. Among the different conformance dimensions, in this paper we propose a novel measure for precision, based on the simple idea of counting these situations were the model deviates from the log. Moreover, a log-based traversal of the model that avoids inspecting its whole behavior is presented. Experimental results show a significant improvement when compared to current approaches for the same task. Finally, the detection of the shortest traces in the model that lead to discrepancies is presented.

**Keywords:** Process Mining, Process Conformance.

## 1 Introduction and Related Work

Nowadays, the organizations make use of a wide variety of *Process-Aware Information Systems* (PAISs) to conduct their business processes [13]. These systems record all kind of information about the processes in *logs*, which can be used for different purposes. *Process Mining* is an area of research that aims at the discovery, analysis and extension of formal models in a PAIS, in order to support its design and maintenance.

The problem of deriving a formal model from a log is known as *Process Discovery*. For this problem, several algorithms exist which derive models that represent (maybe partially) processes detected by observing the traces in the log. In particular, the production of a Petri net [7] whose underlying behavior is related to the traces in the log has been presented extensively in the literature [17,2,14,4,18]. The Petri nets produced by many of these algorithms represent *overapproximations* of the log, i.e. the set of traces accepted in the net is a superset of the set of traces in the log. Therefore, one can not rely on the accuracy of the discovered model unless some minimality property is guaranteed ([2,4]), or a certification provided by a metric ensures its quality.

*Process Conformance* aims at evaluating the adequacy of a model in describing a log. Analyzing conformance is a complex task which involves the interplay of different and orthogonal dimensions [8,9]:

- ⋆ *Fitness*: indicates how much of the observed behavior is captured by (i.e. "fits") the process model.

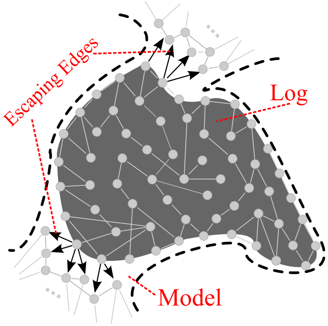
R. Hull, J. Mendling, and S. Tai (Eds.): BPM 2010, LNCS 6336, pp. 211–226, 2010.

- ⋆ *Precision*: refers to overly general models, preferring models with minimal behavior to represent as closely as possible the log.
- ⋆ *Generalization*: addresses overly precise models which overfit the given log, thus been possible to generalize.
- ⋆ *Structure*: refers to models minimal in structure which clearly reflect the described behavior.

Different algorithms for conformance checking have been presented in the literature (a complete survey can be found in [9]). In particular, some examples of approaches focused on precision are: [6] (measuring the percentage of potential traces in the model that are in the log), [5] (comparing two models and a log to see how much of the first model's behavior is covered by the second) used in [14], [19] (comparing the behavioral similarity of two models without a log), and [3] (using *minimal description length* to evaluate the quality of the model). In this work we focus on the precision between a model (a Petri net in our case) and a log. On this regard, the methods presented in [10,11] may be seen as a seminal work, were metrics for fitness, structural and precision are presented. In particular, the precision metric presented (called *advanced behavioral appropriateness*) is limited to comparing the ordering relations between events in the model with the ones from the log. This approach needs the exhaustive exploration of the model's state space, which can be impractical for large models that exhibit a high degree of concurrency, i.e. the *state-space explosion problem* arises.

Briefly, this paper presents a novel technique to measure precision, which aims at: i) complementing the precision information provided by other techniques, ii) fighting the inherent complexity that relies in checking conformance for industrial or real-life models and logs and, iii) providing useful information for later *Process Extension*, i.e. the stage where the model may be extended to better reflect the log. The initial implementation is available as the *ETConformance* plug-in in the ProM 6 framework [1].

The approach can be summarized as follows: given a model and a log, the behavior of the model restricted to the log is computed (part with gray background in the figure on the right). The border between the log's and model's behavior defines crucial points where the model deviates from the log. We call these situations *escaping edges*. By quantifying these edges and their frequency, we aim at providing an accurate measurement of the precision dimension. Moreover, the escaping edges denote inconsistencies that might be treated in the process extension phase.



In the previous figure it has been considered that the model's behavior includes all the log's behavior (perfect fitness), in order to evaluate its precision. When the fitness condition does not hold, we recommend (as in [10]), to analyse the conformance in two phases: in the first phase the fitness is evaluated, filtering the noise and analysing the discrepancies. In the second phase, the other three dimensions are evaluated. Section 7 briefly comments how to deal with non-fitting models.

### 1.1 Why a New Measure to Quantify Precision?

**A fresh look at precision.** We aim at providing a precision metric that estimates the effort needed to obtain an accurate model, focusing on the discrepancies detected. This contrasts with the existing approaches for precision that only provide discrepancies in the event relations [11].

**Efficiency.** To compute the precision metric, we present a log-based traversal of the model's behavior . The technique avoids the traversal of the complete model's behavior, i.e., only the behavior of the model reflected in the log is explored (see the figure above). Hence, the approach can handle inputs that can not be handled by other approaches which traverse the complete behavior.

**Granularity.** The closest measure to the one presented in this paper, is the *advanced behavioral appropriateness*, by Rozinat *et al* [11], denoted by $a'_B$. This approach consists in computing the precedence/follows relations between tasks in the model and in the log, and compares both relations, thus providing a behavioral metric for precision. However, these relations can only have three possible values: *Always*, *Never* and *Sometimes*. Our approach works directly with the behavior of the model, getting a deeper view of the precision problem.

**Extensionality.** Finally, we feel that, in addition to the metric, it is important to provide an appropiate mechanism for the later Process Extension. On this regard, the methods presented in this paper output also the exact points of discrepancy, i.e., the traces where the model starts to deviate from the log.

The background for the understanding of this paper is presented in Section 2. Section 3 describes informally the approach, whereas Sections 4 and 5 present the algorithm to collect escaping edges and the metric for precision analysis, respectively. Section 6 introduces the notion of disconformant traces. Section 7 discuss some extensions and experimental results are presented in Section 8.

## 2 Preliminaries

In this section we present the two main inputs needed to perform the conformance analysis: Petri nets and logs (Event Logs). Additionally, Transitions Systems will be presented to link these two inputs.

Some mathematical notation is provided for the understanding of the paper. Given a set $S$, we denote $\mathcal{P}(S)$ as the powerset over $S$, i.e. the set of possible subsets of elements of $S$. Given a set $T$, a sequence $\sigma \in T^*$ is a called *trace*.

Given a trace $\sigma = t_1 t_2 \ldots t_n$, and a natural number $0 \leq k \leq n$, $hd^k(\sigma)$ is the trace $t_1 t_2 \ldots t_k$, also called the *prefix* of length $k$ in $\sigma$. Notice that $hd^0(\sigma) = \lambda$, i.e., the empty word. Finally, given a set of traces $L$, we denote $Pref(L)$ the set of all prefixes for traces in $L$.

### 2.1 Petri Nets

**Definition 1 (Petri Net [7]).** *A Petri Net (*PN*) is a tuple* $(P, T, W, M_0)$ *where* $P$ *and* $T$ *represent finite sets of places and transitions, respectively, with* $P \cap T = \emptyset$*. And* $W : (P \times T) \cup (T \times P) \to \mathbb{N}$ *is the weighted flow relation. A marking is a mapping* $P \to \mathbb{N}$*.* $M_0$ *is the initial marking, i.e. defines the initial state of the system.*

A transition $t \in T$ is *enabled* in a marking $M$ iff $\forall p \in P : M(p) \geq W(p,t)$. An enabled transition can be *fired* resulting in a new marking $M'$ such that $\forall p : M'(p) = M(p) - W(p,t) + W(t,p)$. A marking $M'$ is *reachable* from $M$ if there is a sequence of firings $\sigma = t_1 t_2 \ldots t_n$ that transforms $M$ into $M'$, denoted by $M[\sigma\rangle M'$. A sequence of transitions $\sigma = t_1 t_2 \ldots t_n$ is a *feasible sequence* if $M_0[\sigma\rangle M$, for some $M$. The set of reachable markings from $M_0$ is denoted by $[M_0\rangle$, and form a graph called *reachability graph.* A PN is said to be *k-bounded* or simply *bounded* if $\forall p : M'(p)$ does not exceed a number $k$ for any reachable marking $M'$ from $M_0$. If no such number exists, it is said to be *unbounded.*

### 2.2 Event Logs

Event logs contain executions of a system [16]. These executions represent the ordering between different tasks, but may also contain additional information, like the task originator or its timestamp. For the purposes of this paper, all this information is abstracted:

**Definition 2 (Event Log).** *An* event log EL *is a set of traces, i.e.,* $\mathsf{EL} \in \mathcal{P}(T^*)$.

We denote by $|\mathsf{EL}|$ the number of traces of the log. As notation, these traces can be iterated through $\sigma_1 \ldots \sigma_{|\mathsf{EL}|}$.

### 2.3 Relation between a Petri Net and an Event Log

Similarly as it is done in [10], tasks in the log and transitions in the Petri net must be mapped in order to establish a relation between both objects. Besides the most simple relation between only one task and one transition, the mapping may have some more complex scenarios: (i) *Duplicate tasks*, two or more tasks in the model are associated with the same task in the log. (ii) *Invisible tasks*, a task in the model is associated with no task in the log[1]. In this paper, the invisible tasks

[1] The reasons for this lack of relation may be different: non recordable steps in the process (phone calls, meetings, ...), introduced in the model for routing purposes, relaxations of the model, or invisible tasks resulting from some discovery algorithm (e.g., [14]).

in the model will be represented as transitions filled black. (iii) *Non Modelled tasks*, a task in the log is associated with no task in the model. Non modelled tasks are not relevant for the purpose of this paper. Therefore, they can be removed from the log before starting the analysis. The techniques presented in this paper cover all the scenarios above. Note that, for the sake of clarity, we refer to task, event or transition indistinctly, whenever no mistake is possible.

### 2.4 Transitions Systems

**Definition 3 (Transition system).** *A* transition system *(*$\mathsf{TS}$*) is a tuple* $(S, T, A, s_{in})$*, where* $S$ *is a set of* states*,* $T$ *is an alphabet of* actions*,* $A \subseteq S \times T \times S$ *is a set of* (labelled) transitions*, and* $s_{in} \in S$ *is the* initial state*.*

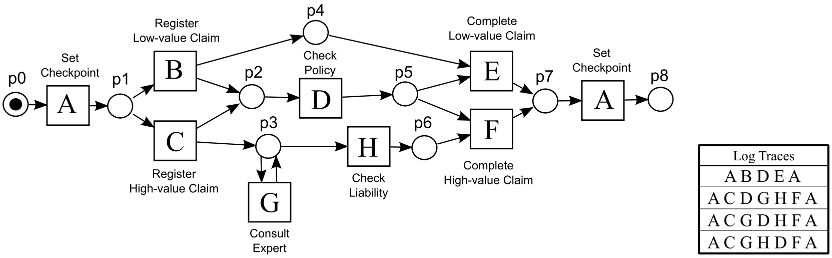
We will use $s \xrightarrow{e} s'$ as a shortcut for $(s, e, s') \in A$, and the transitive closure of this relation will be denoted by $\xrightarrow{*}$. The *language* of a transition system $\mathsf{TS}$, $L(\mathsf{TS})$, is the set of traces feasible from the initial state.

## 3 Problem Statement and Approach

As was said in the introduction, our goal is to measure the precision of a model with respect to a log. Moreover, we strive to locate inconsistencies between the model and the log, thus allowing the later extension of the model.

Let us introduce the PN and EL shown in Fig. 1(a) and (b), respectively, based on the examples used in [11]. We will use this example in the rest of the paper as a running example. The PN reflects the typical process of liability insurance claim in a bank. Note that, although the log represents a plausible situation, the control flow shown in the model is not realistic, i.e., the *Consult Expert* task should be executed exactly once. This inconsistence may seem obvious in this small scenario, but may be not for larger and realistic cases. Hence, we use it to illustrate the conformance analysis.

The precision metric introduced in this paper is computed smoothly: in contrast to other approaches that require a complete exploration of both the model



| Log Traces |
|---|
| A B D E A |
| A C D G H F A |
| A C G D H F A |
| A C G H D F A |

(a) *PN1*

(b) *EL1*

**Fig. 1.** Running example

and the log [11], in our approach the model exploration is restricted to the observed log's behavior. As a consequence, the computational requirements are bounded to the log size, thus being independent of the whole model's underlying behavior. This might be crucial for models obtained from a mining algorithm that may have an underlying behavior of intractable size.

The key concept of the approach is that of *Escaping Edges*, i.e., the situations were the model allows more behavior than the log, thus exhibiting less precision. We base our measure on the relation between the number and frequency of escaping edges with respect to model's behavior restricted to the log . Hence, the more the model deviates from the log, the less precise is its precision value. It is important to stress the fact that a model can have few escaping edges (thus exhibiting good precision) but with underlying behavior substantially bigger than the log: it is not the aim of this work to compare sizes between the model and the log, but providing an estimation of the efforts required to improve the model to precisely describe the log. Figure 2 shows the *route map* of the ETConformance approach, and indicates the section where each part is explained in detail.

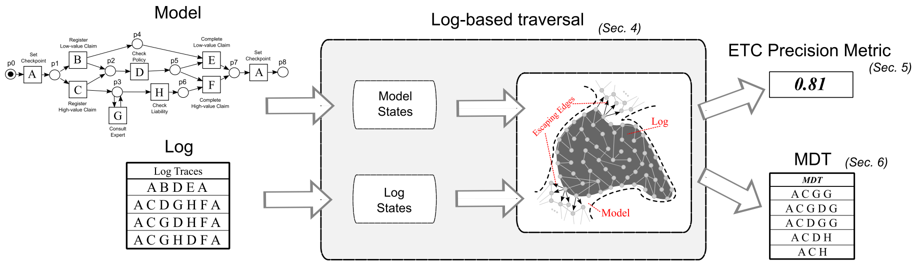


**Fig. 2.** Route map of ETConformance: a log and a Petri net are the inputs of the technique. Internally, log and Petri net states are identified and mapped. Finally, both the metric and the set of minimal discrepancies are reported.

## 4 Log-Based Traversal of the Model's Behavior

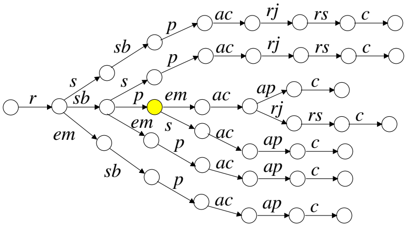
This section describes the log-based traversal of the model's behavior. We have an assumption on the input log: every trace in the log is possible in the model, i.e. it has fitness value one (see how to adapt the technique to non-fitting models in Section 7). This assumption is grounded on the correlations that exist between fitness and precision dimensions, as pointed in [10]. Hence, the fitness analysis and corresponding adjustments can be done before the precision analysis. There are some approaches to perform this task, e.g., [11].

### 4.1 Basic Idea

In order to perform a log-based traversal of the model behavior, it is necessary to find a common comparable domain between log and model, i.e. a domain where

| | |
|---|---|
| 1 | r,s,sb,p,ac,ap,c |
| 2 | r,sb,em,p,ac,ap,c |
| 3 | r,sb,p,em,ac,rj,rs,c |
| 4 | r,em,sb,p,ac,ap,c |
| 5 | r,sb,s,p,ac,rj,rs,c |
| 6 | r,sb,p,s,ac,ap,c |
| 7 | r,sb,p,em,ac,ap,c |

(a)



(b)

**Fig. 3.** (a) Event log $\mathsf{EL}$, (b) corresponding transition system $\mathsf{TS}$ such that $L(\mathsf{TS}) = \mathsf{EL}$

states of the model and log states could be mapped. For performing such task, state information must be obtained for both objects (model and log). In the log side, several algorithms are presented in [15] to incorporate state information in the log. These algorithms are parametrizable with respect to the decision of a state (*past*, *future* or *both*), the representation of the information (*sequence*, *set* and *multiset*), and its horizon (*limited* or *infinite*). In particular, using the past, sequence and infinite settings allows to derive a behavioral representation of the log (a transition system) with the same language. The states of that transition system will be the states of the log.

**Definition 4 (Prefix automaton, Log states).** *Given an event log* $\mathsf{EL}$*, let* $\mathsf{TS} = (S, T, A, s_0)$ *be the automaton derived by using the construction presented in [15] using the past, sequence and infinite settings. We call this automaton* prefix automaton*. The set* $S$ *will be denoted as the* set of states of $\mathsf{EL}$*. Given a trace* $\sigma_i = t_1 \dots t_{|\sigma_i|} \in \mathsf{EL}$*,* $s_j^i \in S$ *denotes the state in* $\mathsf{TS}$ *corresponding to the prefix* $t_1 \dots t_{j-1}$*, for* $0 \leq j \leq |\sigma_i| + 1$*.*

Fig. 3 shows an example of this transformation. The states of this transition system correspond to prefixes of the traces in the log: for instance, the state filled in Fig. 3 corresponds to the prefix $r, sb, p$, contained in traces 3, 6 and 7.

To obtain state information for a Petri net model, its reachable states can be computed. However, due to the well-known *space-explosion problem*, Petri nets can exhibit a large or even infinite behavior, making this approach impractical for these instances. Instead, the approach presented in this paper only visits those reachable markings of the net for which there is at least one state in the log (see Def. 4) mapped. Let us define formally the mapping:

**Definition 5 (Mapping between log states and Petri net markings).** *Let* $\mathsf{EL}$ *and* $\mathsf{PN} = (P, T, W, M_0)$ *be a log and a Petri net, respectively, and consider the prefix automaton* $\mathsf{TS} = (S, T, A, s_0)$ *from* $\mathsf{EL}$*. A marking* $M$ *is mapped to the state* $s \in S$*, denoted by* $M \multimap s$*, if there exists a trace* $\sigma$ *such that* $s_0 \xrightarrow{\sigma} s$ *in* $\mathsf{TS}$ *and* $M_0[\sigma\rangle M$*.*

Due to this mapping, the Petri net traversal can be controlled to reach only markings for which there is a corresponding mapped state in the log. Moreover, for each one of the markings reached on this guided traversal, the local precision on this marking can be measured by collecting the discrepancies between the behavior allowed in the model with respect to the behavior observed in the log:

**Definition 6 (Allowed Tasks and Reflected Tasks).** *Let $s$ be a state of the prefix automaton* $\mathsf{TS} = (S, T, A, s_0)$ *from* $\mathsf{EL}$*, and* $\mathsf{PN} = (P, T, W, M_0)$ *a Petri net. We define* $A_T(s) = \{t \in T \mid M \multimap s \wedge M[t\rangle M'\}$ *and* $R_T(s) = \{t \in T \mid s \xrightarrow{t} s')\}$ *as the set of allowed and reflected tasks in $s$.*

Since we are assuming fitness value one of the model with respect to the log, clearly $R_T(s) \subseteq A_T(s)$ for every state $s$ of $\mathsf{TS}$, i.e. the model overapproximates the log. Now it is possible to define the concept of escaping edge:

**Definition 7 (Escaping Edges).** *Let $s$ be a state of the prefix automaton* $\mathsf{TS} = (S, T, A, s_0)$ *from* $\mathsf{EL}$*, and* $\mathsf{PN} = (P, T, W, M_0)$ *a Petri net. The Escaping Edges ($E_E$) of $s$ is defined as* $E_E(s) = A_T(s) \setminus R_T(s)$.

An example of escaping edge for the pair (*PN1*,*EL1*) in Fig. 1 is $H$ in the log state reached after the prefix $AC$: here *PN1* accepts the $H$ task, whereas this is not reflected in *EL1*. It is important to stress that the tasks reflected of a state $s$ are the ones that appear in *any* of the traces that contain $s$. For instance, for the example of Fig. 1, the prefix $A$ has two allowed tasks in the model ($B$ and $C$) but given that both are reflected in the log (in different traces), no escaping edge arise in the log state after observing $A$. The algorithm to collect the set of escaping edges is presented as Algorithm. 1.

```
Input: EL, PN
foreach State $s$ in EL do
    RT := outEdges ($s$)
    $\sigma$ := prefix ($s$,EL)
    $mark$ := fire ($\sigma$,PN)          // Fire $\sigma$ and get the reached marking
    AT := enable ($mark$,PN)         // Get the enable transitions of $mark$
    EE := AT $\setminus$ RT                 // Allowed tasks minus Reflected Tasks
    register ($s$,EE)
```

**Algorithm 1.** ComputeEscapingEdges

### 4.2 Duplicate and Invisible Tasks

Until now, the methods have been explained without considering duplicate or invisible tasks. Although the approach presented in this paper can deal with duplicate and invisible tasks, we must remark some issues concerning the potential indeterminism that may arise.

When trying to determine the marking associated to a log state, it may happen that firing the sequence of tasks cannot be done in a deterministic way, i.e., one log task is associated to two or more enabled transitions in the model, and therefore, one of the transitions must be chosen to continue the firing. A similar situation occurs when a sequence of invisible tasks that enables a visible log task must be fired. In this cases, part of the state space of the model reachable from the current marking must be explored. To avoid producing an infinite state space (for instance, because the Petri net might be unbounded), we construct the *invisible coverability graph*: a variant of the well-known coverability graph algorithm [7] where the nodes are *$\omega$-markings* and the edges are only invisible tasks. Informally, this graph will contain all the paths of invisible tasks that lead to the enabling of a visible one. It may occur that a visible task is enabled for several and different sequences of invisible tasks, and therefore, a guess between the different sequences must be made to continue the traversal.

In real-life scenarios, dealing with indeterminism requires the use of heuristics and therefore, the inconsistencies detected (escaping edges in our framework) might be caused by a bad guess and not by a real precision problem. In [11] a heuristic that uses the shortest sequence of invisible that enables a visible task is proposed. This heuristic tries to minimize the possibility that a invisible fired task interfere the future firing of another task. In general, the availability of several heuristics can be helpful to apply ad-hoc explorations which depend on the scenario considered.

## 5 Evaluating Precision

As it has been seen before, the escaping edges are a good indicator for measuring the behavior of a model compared to the behavior reflected in the log. For that reason, we propose a metric to take into account these escaping edges and their frequency. This metric also allows us to compare between models to know which one captures better the behavior reflected of a log. Let us formalize the metric:

**Metric 1 (ETC Precision).** *Let* $\mathsf{EL} = \{\sigma_1, \ldots, \sigma_{|\mathsf{EL}|}\}$ *and* $\mathsf{PN} = (P, T, W, M_0)$ *be a log and a Petri net, respectively. For each trace* $\sigma_i$ $(1 \leq i \leq |\mathsf{EL}|)$*, state* $s^i_j$ $(1 \leq j \leq |\sigma_i| + 1)$ *denotes the* $j - th$ *state of* $\sigma_i$ *(see Def. 4). The metric is defined as follows:*

$$etc_P(\mathsf{EL}, \mathsf{PN}) = 1 - \frac{\sum_{i=1}^{|\mathsf{EL}|} \sum_{j=1}^{|\sigma_i|+1} |E_E(s^i_j)|}{\sum_{i=1}^{|\mathsf{EL}|} \sum_{j=1}^{|\sigma_i|+1} |A_T(s^i_j)|}$$

By dividing the set of escaping edges by the set of allowed tasks in the model, the metric evaluates the amount of overapproximation in each trace. Note that, for all $s^i_j$, $|E_E(s^i_j)| \leq |A_T(s^i_j)|$, and therefore $0 \leq \text{etc}_P \leq 1$. The fact that we take into account the frequency of the traces makes that the most used and appropriate traces would contribute with higher weight than the noisy traces and the wrong indeterministic choices. The metric value of the example for the pair (*PN1*, *EL1*) in Fig. 1 is

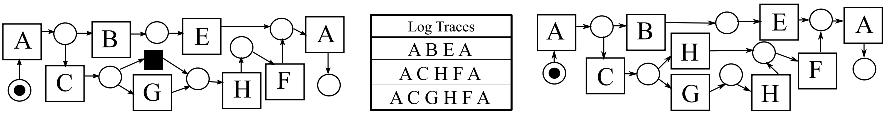
$$1 - \frac{0 + 3 + 3 + 2}{6 + 12 + 13 + 12} = 0.81$$

where every i-th summand of the numerator/denominator is processing the penalizations for the escaping edges of trace $\sigma_i$, e.g., in trace $\sigma_4 \in EL1$ there are 2 escaping edges and 12 allowed tasks.

As was done in [10], we present some of the quality requirements a good metric should satisfy and a brief justification on its fulfillment.

**Validity.** (*The metric and the property to measure must be sufficiently correlated with each other*). In the case of Metric 1, the more escaping edges, the lower value will be provided (even closer to 0 in the worst case). This inverse correlation quantifies if a model is a precise description of a log.

**Stability.** (*The metric must be as little as possible affected by the properties that are not measured*). In other words, the metric must measure only one dimension (precision in this case) independently of the others (e.g., fitness, structural, generalization). With regard to fitness or generalization, the metric is not stable since there is a correlation between precision and both dimensions. In contrast, by only focusing on the underlying behavior of the model, $\text{etc}_P$is independent to the structural dimension. To illustrate this, Fig.4 show *PN2* and *PN3*, two Petri Nets with the same behavior and different structure. The result provided by $\text{etc}_P$is 1 in both cases when compared with the log *EL2*.



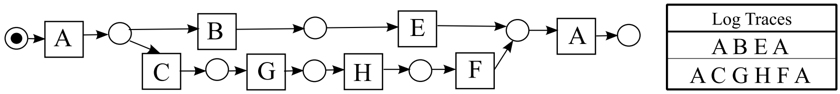
| Log Traces |
|---|
| A B E A |
| A C H F A |
| A C G H F A |

(a) *PN2* (b) *EL2* (c) *PN3*

**Fig. 4.** Stability of the metric with respect to structure

**Analyzability.** (*It relates to the properties of the measured values. In our case, the emphasis is on the requirement that the measured values should be distributed between 0 and 1, with 1 being the best and 0 being the worst value. It is important to be 1 when there is no precision problem*). The values returned by $\text{etc}_P$ are distributed between 0 and 1. In addition, the value 1 can be reached, indicating that there are no inconsistencies. Notice that, to achieve this value it is not necessary to have only one enabled at each point of the trace (like in $a_B$ [11]). This is because the metric does not depend on the idea of *more enabled tasks, more behavior*, but in the concept *behavior allowed vs reflected* itself. Moreover, the metric value can be 1, even if the whole behavior of the model is distributed in two or more different traces. This is because decision on escaping edges is done *globally* after processing the whole set of traces. This can be seen in *PN4* and *EL3* (cf. Fig. 5), that has a $\text{etc}_P$ value of 1.

**Localizability.** (*The system of measurement forming the metric should be able to locate those parts in the analyzed object that lack certain measured properties*). This is a crucial requirement in conformance analysis, due to the fact that



| Log Traces |
|---|
| A B E A |
| A C G H F A |

(a) *PN4* (b) *EL3*

**Fig. 5.** Global analysis of the traces leads to a global analysis of escaping edges

providing the discrepancy points we are making possible to identify the potential points of model's enhancement. This is done in the approach of this paper through the escaping edges, identified by their marking and the task used to *escape* from the reflected behavior in the log. However, given the importance of the localizability in conformance, we go a step further and in the next section a technique to collect the traces leading to these situations is presented.

## 6 Minimal Disconformant Traces

Given a log and a model, we are interested in identifying these minimal traces that lead to a situation where the model starts to deviate from the log. Some of these traces may represent meaningful abstractions that arise in the model and therefore no further action is required. For the rest of traces, a decision on whereas the model or the log are wrong shall be made. In the case of an erroneous model, process extension techniques must be applied.

**Definition 8 (Minimal Disconformant Traces).** *Let* $\mathsf{EL}$ *and* $\mathsf{PN} = (P, T, W, M_0)$ *be a log and a Petri net, respectively. We define the* Minimal Disconformant Traces *(*$\mathsf{MDT}$*) as the set of traces* $\sigma = \sigma' t$ *such that* $M_0[\sigma\rangle M$*,* $\sigma' \in \mathrm{Pref}(\mathsf{EL})$ *and* $\sigma \notin \mathrm{Pref}(\mathsf{EL})$*.*

The escaping edges computed in previous section can be used to obtain the Minimal Disconformant Traces. Algorithm 2 shows how to generate this set of traces using the escaping edges: for each state $s$ with a escaping edge, the sequence to reach $s$ is computed and it is concatenated with the escaping edge. Finally, Lemma 1 ensures a minimality criterion on the derived traces.

```
Input: EL
Output: $M$
foreach State $s$ in EL do
    foreach Task $t$ in $E_E(s)$ do
        $\sigma$ := prefix ($s$,EL)
        $\sigma := \sigma \cdot t$                          // Concatenate
        addTrace ($\sigma$, $M$)           // Register $\sigma$ as an MDT
return $M$
```

**Algorithm 2.** ComputeMDT

**Lemma 1.** *Algorithm 2 computes the Minimal Disconformant Traces.*

**Proof:** Let $M$ the set of traces computed by the algorithm *ComputeMDT* and $\mathsf{MDT}$ the set of traces that satisfy Definition 8. To prove $M = \mathsf{MDT}$, we will prove $M \subseteq \mathsf{MDT}$ and then $\mathsf{MDT} \subseteq M$.

Let $\sigma = \sigma' t$ be any trace of $M$. By construction $\sigma'$ is a prefix of the log. However, given the formation of $R_T$ and $E_E$ in Algorithm 1, $\sigma$ is not a prefix of the log. Furthermore, $\sigma'$ is a feasible sequence of the model because it is a prefix of the log, and all traces in the log are compliant with the model (since we assume fitness value one). In addition, by construction of $A_T$ and $E_E$, $\sigma$ is a feasible sequence by the model too. Therefore, $\sigma \in \mathsf{MDT}$.

Now, let $\sigma = \sigma' t$ be any trace of $\mathsf{MDT}$. $\sigma'$ is a prefix of the log. According to Definition 4, it must be defined a log state $s$ after the sequence $\sigma'$. The task $t$ must be in the $A_T$ of $s$, because $\sigma$ is a feasible sequence of the model. But $t$ must not appear in $R_T$ because $\sigma$ is not a prefix of the log. By construction of $R_T$, $t$ is a Escaping Edge. Therefore, $\sigma \in M$. □

Following with the running example, Fig. 6 shows the MDTs for the model *PN1* and log *EL1* (described in Fig. 1). The MDT traces shown are result of the unseen behavior produced by the loop of $G$, which even allows the possibility of skipping $G$. Note that, the set of MDTs computed by Algorithm 2 might be seen as a log (i.e sequence of traces). Consequently, all kind of Process Mining techniques can be applied to it in order to get a general view of the information. In particular, mining methods can be used to obtain a Petri Net representing this extra behavior.

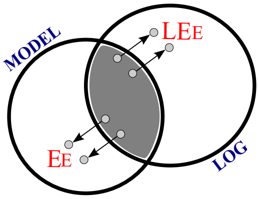
| *MDT* |
|---|
| A C G G |
| A C G D G |
| A C D G G |
| A C D H |
| A C H |

**Fig. 6.** MDTs

## 7 Extensions

**Log States as Markings.** The first possible extension is to consider log states just as Petri net markings, i.e., two traces reaching the same marking correspond to the same state. With that approach we could have a more high-level vision of the precision, e.g., the $R_T$ sets of two different states (with the same associated marking) now will be united in only one set corresponding to the new state. Although it could be useful in some occasion, we must be awared about all the information we are not considering. For instance, the etc$_P$ value of (*PN1*, *EL1*) using this new approach would be 1, losing all track about the extra behavior introduced by the $G$ loop.

**Non-fitting models.** Symmetric to the Escaping Edges ($E_E$), we can define the *Log Escaping Edges($LE_E$)*, i.e., the points where the log deviates from the model. All these points could be evaluated, providing a metric that, in this case, would mesure fitness instead of precision.



All these extensions might be incorporated to extend the applicability of the approach presented in this paper.

## 8 Experimental Results

The technique presented in this paper, implemented as the *ETConformance* plug-in within ProM 6, has been evaluated on existing public-domain benchmarks [1]. The purpose of the experiments is:

- ⋆ Justify the existence of a new metric to evaluate precision, i.e. demonstrate the novelty of the concept when compared to previous approaches.
- ⋆ Show the capacity of the technique to handle large specifications.

Table 1(a) shows a comparison of the technique presented in this paper with the technique presented in [11], implemented in ProM 5.2 as the *Conformance Checker*. The rows in the table represent benchmarks with small size (few traces). The names are shortened, e.g., GFA5 represents GroupedFollowsA5. We report the results of checking precision for both conformance checkers in columns under $a'_B$ and $\text{etc}_P$, respectively, for the small Petri nets obtained by the Parikh miner [18] which derived Petri nets with fitness value one. For the case of our checker, we additionally provide the number of minimal disconformant traces (|MDT|). We do not report CPU times since checking precision in both approaches took less than one second for each benchmark.

From Table (a) one can see that when the model describes precisely the log, both metrics provide the maximum value. Moreover, when the model is not a precise description of the log, only three benchmarks provide opposite results (GFBN2, GFl2l, GFl2lSkip). For instance, the GFl2lSkip benchmark $a'_B$ is providing a significant lower value: this is because the model contains an optional loop that is always traversed in the log. This variability is highly penalized by simply observing the tasks relations. On the other hand, metric $\text{etc}_P$ will only penalize the few situations where the escaping edges appear in the log.

Larger benchmarks for which *Conformance Checker* cannot handle are provided in Table 1(b). For these benchmarks, we report the results (precision value, number of MDT and CPU time in seconds) for the models obtained by the Parikh miner and the RBMiner [12]. These are two miner that guarantee fitness value one. For each one of the $aN$ benchmarks, $N$ represents the number of tasks in the log, while the _1 and _5 suffixes denote its size: 100 and 900 traces, respectively. The $t32$ has 200 (_1) and 1800 (_5) traces. The pair of CPU times reported denote the computation of $\text{etc}_P$ without or with the collection of MDT$s$ (in parenthesis). Also, we provide the results of the most permissive models, i.e., models with only the transitions but without arcs or places ($M_T$). These models allow any behavior and thus, they have a low $\text{etc}_P$ value, as expected.

A first conclusion on Table 1 (b) is the capability of handling large benchmarks in reasonable CPU time, even for the prototype implementation carried out. A second conclusion is the loss of precision of the metric with respect to the increase of abstraction in the mined models: as soon as the number of tasks increases, the

**Table 1.** Experimental results for small (a) and big (b) benchmarks

(a)

<table>
<tr><th>Benchmark</th><th>$a'_B$</th><th>etc$_P$</th><th>|MDT|</th><th>Benchmark</th><th>$a'_B$</th><th>etc$_P$</th><th>|MDT|</th></tr>
<tr><td>GFA6NTC</td><td>1.00</td><td>1.00</td><td>0</td><td>GFl2lOpt</td><td>1.00</td><td>0.85</td><td>7</td></tr>
<tr><td>GFA7</td><td>1.00</td><td>1.00</td><td>0</td><td>GFAL2</td><td>0.86</td><td>0.90</td><td>391</td></tr>
<tr><td>GFA8</td><td>1.00</td><td>1.00</td><td>0</td><td>GFDrivers</td><td>0.78</td><td>0.89</td><td>2</td></tr>
<tr><td>GFA12</td><td>1.00</td><td>1.00</td><td>0</td><td>GFBN3</td><td>0.71</td><td>0.88</td><td>181</td></tr>
<tr><td>GFChoice</td><td>1.00</td><td>1.00</td><td>0</td><td>GFBN2</td><td>0.59</td><td>0.96</td><td>19</td></tr>
<tr><td>GFBN1</td><td>1.00</td><td>1.00</td><td>0</td><td>GFA5</td><td>0.50</td><td>0.57</td><td>35</td></tr>
<tr><td>GFParallel5</td><td>1.00</td><td>0.99</td><td>11</td><td>GFl2l</td><td>0.47</td><td>0.75</td><td>11</td></tr>
<tr><td>GFAL1</td><td>1.00</td><td>0.88</td><td>251</td><td>GFl2lSkip</td><td>0.30</td><td>0.74</td><td>10</td></tr>
</table>

(b)

<table>
<tr><th></th><th></th><th>$M_T$</th><th colspan="5">Parikh</th><th colspan="5">RBMiner</th></tr>
<tr><th>Benchmark</th><th>|TS|</th><th>etc$_P$</th><th>$|P|$</th><th>$|T|$</th><th>etc$_P$</th><th>|MDT|</th><th>CPU</th><th>$|P|$</th><th>$|T|$</th><th>etc$_P$</th><th>|MDT|</th><th>CPU</th></tr>
<tr><td>a22f0n00_1</td><td>1309</td><td>0.06</td><td>19</td><td>22</td><td>0.63</td><td>1490</td><td>0(0)</td><td>19</td><td>22</td><td>0.63</td><td>1490</td><td>0(0)</td></tr>
<tr><td>a22f0n00_5</td><td>9867</td><td>0.07</td><td>19</td><td>22</td><td>0.73</td><td>9654</td><td>0(3)</td><td>19</td><td>22</td><td>0.73</td><td>9654</td><td>0(4)</td></tr>
<tr><td>a32f0n00_1</td><td>2011</td><td>0.04</td><td>31</td><td>32</td><td>0.52</td><td>2945</td><td>0(0)</td><td>32</td><td>32</td><td>0.52</td><td>2944</td><td>0(1)</td></tr>
<tr><td>a32f0n00_5</td><td>16921</td><td>0.05</td><td>31</td><td>32</td><td>0.59</td><td>22750</td><td>2(10)</td><td>31</td><td>32</td><td>0.59</td><td>22750</td><td>2(11)</td></tr>
<tr><td>a42f0n00_1</td><td>2865</td><td>0.03</td><td>44</td><td>42</td><td>0.35</td><td>7761</td><td>0(2)</td><td>52</td><td>42</td><td>0.37</td><td>7228</td><td>0(2)</td></tr>
<tr><td>a42f0n00_5</td><td>24366</td><td>0.04</td><td>44</td><td>42</td><td>0.42</td><td>60042</td><td>5(28)</td><td>46</td><td>42</td><td>0.42</td><td>60040</td><td>6(29)</td></tr>
<tr><td>t32f0n00_1</td><td>7717</td><td>0.03</td><td>30</td><td>33</td><td>0.37</td><td>15064</td><td>1(15)</td><td>31</td><td>33</td><td>0.37</td><td>15062</td><td>1(12)</td></tr>
<tr><td>t32f0n00_5</td><td>64829</td><td>0.04</td><td>30</td><td>33</td><td>0.39</td><td>125429</td><td>9(154)</td><td>30</td><td>33</td><td>0.39</td><td>125429</td><td>8(160)</td></tr>
</table>

miners tend to derive models less precise to account for the complex relations between different tasks. Often, these miners derive models with a high degree of concurrency, thus accepting a potentially exponential number of traces which might not correspond to the real number of traces in the log.

Finally, three charts are provided: the relation between the log size with respect to the CPU time, the etc$_P$ value and the number of MDTs are shown in Fig. 7. For these charts, we selected different log sizes for different types of benchmarks (a22f0, a22f5, a32f0,a32f5 for the two bottom charts, a42f0, t32f5 and t32f9 for the top chart). For the two bottom charts, we used the Petri nets derived by the Parikh miner to perform the conformance analysis on each log, whereas we use a single Petri net for the top chart to evaluate the CPU time (without collecting MDTs) on different logs, illustrating the linear dependance of our technique on the log size. The chart on top clearly shows the linear relation between log size and CPU time for these experiments, which is expected by the technique presented in Sect. 4. The two charts on bottom of the figure show: (left) since for the a22/a32 benchmarks the models derived are very similar independently of the log, the more traces are included the less escaping edges are found. On the other hand, the inclusion of more traces contributes to the incorporation of more MDTs, as it is shown in the right chart at the bottom.

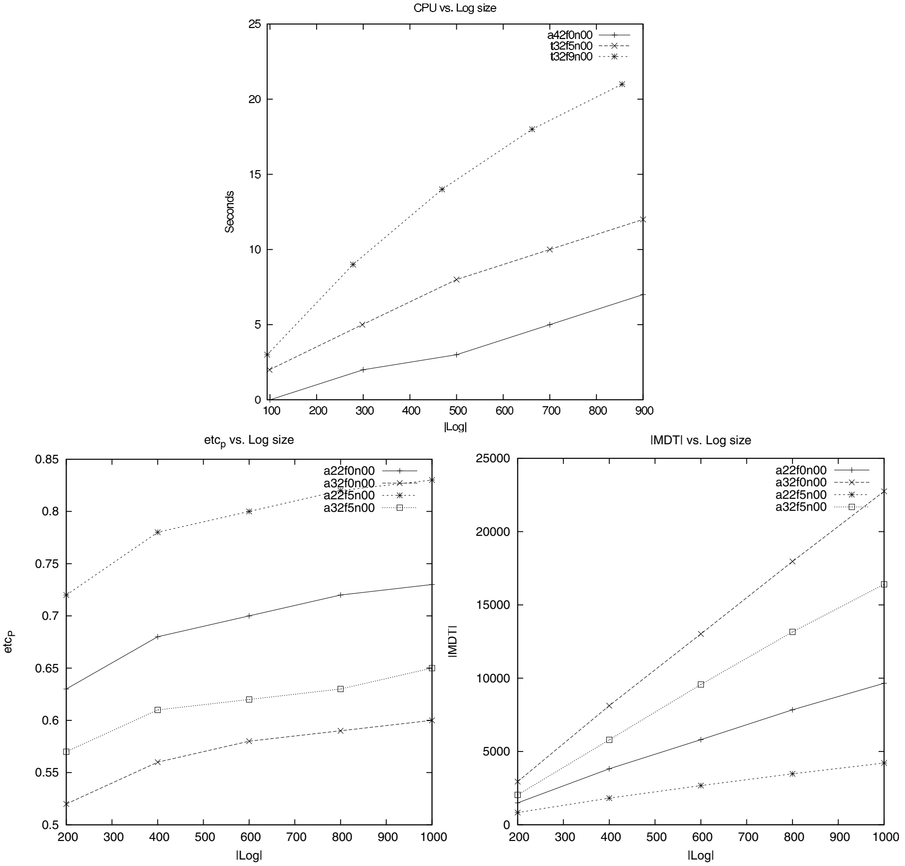


**Fig. 7.** CPU and etc$_P$ versus log size for some large benchmarks

# 9 Conclusion

This paper has presented a low-complexity technique that allows checking the precision of a general Petri net with respect to a log. By only focusing on the underlying behavior of the Petri net that is reflected in the log, the technique avoids the potential state explosion that might arise when dealing with large and highly concurrent nets. The theory has been implemented as a plugin within ProM 6 and experimental results are promising. The technique is enriched with the detection of minimal disconformant traces that may be the starting point for extension of the model to better represent the log.

**Acknowledgments.** We would like to thank M. Solé, A. Rozinat and ProM developers for their help. This work has been supported by the project FORMALISM (TIN2007-66523), and a grant by Intel Corporation.

## References

1. Process mining, http://www.processmining.org
2. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process mining based on regions of languages. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 375–383. Springer, Heidelberg (2007)
3. Calders, T., Günther, C.W., Pechenizkiy, M., Rozinat, A.: Using minimum description length for process mining. In: SAC, pp. 1451–1455. ACM, New York (2009)
4. Carmona, J., Cortadella, J., Kishinevsky, M.: A region-based algorithm for discovering Petri nets from event logs. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 358–373. Springer, Heidelberg (2008)
5. de Medeiros, A.K.A., van der Aalst, W.M.P., Weijters, A.J.M.M.: Quantifying process equivalence based on observed behavior. Data Knowl. Eng. 64(1), 55–74 (2008)
6. Greco, G., Guzzo, A., Pontieri, L., Saccà, D.: Discovering expressive process models by clustering log traces. IEEE Trans. Knowl. Data Eng. 18(8), 1010–1027 (2006)
7. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77(4), 541–580 (1989)
8. Rozinat, A., de Medeiros, A.K.A., Günther, C.W., Weijters, A.J.M.M., van der Aalst, W.M.P.: The need for a process mining evaluation framework in research and practice. In: ter Hofstede, A.H.M., Benatallah, B., Paik, H.-Y. (eds.) BPM Workshops 2007. LNCS, vol. 4928, pp. 84–89. Springer, Heidelberg (2008)
9. Rozinat, A., de Medeiros, A.K.A., Günther, C.W., Weijters, A.J.M.M., van der Aalst, W.M.P.: Towards an evaluation framework for process mining algorithms. BPM Center Report BPM-07-06, BPMcenter. org. (2007)
10. Rozinat, A., van der Aalst, W.M.P.: Conformance testing: measuring the alignment between event logs and process models. In: BETA Working Paper Series, Eindhoven University of Technology, vol. 144, pp. 203–210. Eindhoven University of Technology, WP (2005)
11. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Information Systems 33(1), 64–95 (2008)
12. Solé, M., Carmona, J.: Process mining from a basis of state regions. In: Lilius, J., Penczek, W. (eds.) PETRI NETS 2010. LNCS, vol. 6128, pp. 226–245. Springer, Heidelberg (2010)
13. van der Aalst, W.M.P.: Process-aware information systems: Lessons to be learned from process mining. T. Petri Nets and Other Models of Concurrency 2, 1–26 (2009)
14. van der Aalst, W.M.P., de Medeiros, A.K.A., Weijters, A.J.M.M.: Genetic process mining. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 48–69. Springer, Heidelberg (2005)
15. van der Aalst, W.M.P., Rubin, V., Verbeek, H.M.W.E., van Dongen, B.F., Kindler, E., Günther, C.W.: Process mining: a two-step approach to balance between underfitting and overfitting. Software and Systems Modeling (2009)
16. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow mining: A survey of issues and approaches. Data Knowl. Eng. 47(2), 237–267 (2003)
17. van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering 16(9), 1128–1142 (2004)
18. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. In: van Hee, K.M., Valk, R. (eds.) PETRI NETS 2008. LNCS, vol. 5062, pp. 368–387. Springer, Heidelberg (2008)
19. van Dongen, B.F., Mendling, J., van der Aalst, W.M.P.: Structural patterns for soundness of business process models. In: EDOC, pp. 116–128. IEEE Computer Society Press, Los Alamitos (2006)

# Trace Alignment in Process Mining: Opportunities for Process Diagnostics

R.P. Jagadeesh Chandra Bose[1,2] and Wil M.P. van der Aalst[1]

[1] Department of Mathematics and Computer Science, University of Technology, Eindhoven, The Netherlands
[2] Philips Healthcare, Veenpluis 5-6, Best, The Netherlands
j.c.b.rantham.prabhakara@tue.nl, w.m.p.v.d.aalst@tue.nl

**Abstract.** Process mining techniques attempt to extract non-trivial knowledge and interesting insights from event logs. Process mining provides a welcome extension of the repertoire of business process analysis techniques and has been adopted in various commercial BPM systems (BPM|one, Futura Reflect, ARIS PPM, Fujitsu, etc.). Unfortunately, traditional process discovery algorithms have problems dealing with less-structured processes. The resulting models are difficult to comprehend or even misleading. Therefore, we propose a new approach based on *trace alignment*. The goal is to align traces in a way that event logs can be explored easily. Trace alignment can be used in a preprocessing phase where the event log is investigated or filtered and in later phases where detailed questions need to be answered. Hence, it complements existing process mining techniques focusing on discovery and conformance checking.

## 1 Introduction

Many of today's information systems are recording an abundance of event logs. Process mining techniques attempt to extract non-trivial knowledge and interesting insights from these event logs and to exploit these for further analysis [1]. Process mining techniques aim at discovering process, control, data, organizational and social structures from event logs. The majority of research in process mining so far has focussed on process discovery (both from a control-flow and organizational perspective). One of the challenging topics in process mining is *process diagnostics*. Process diagnostics encompasses process performance analysis, anomaly detection, diagnosis, inspection of interesting patterns and the like. Research so far in diagnosing processes is limited to exploring ways and means of analyzing process models (such as conformance checking), projecting diagnostic information on these models or in dashboard like approaches over some (performance) metrics. Diagnostics of processes at the model level is cumbersome, tedious and sometimes infeasible, especially when dealing with real-life and flexible processes. We have applied process mining in more than 100 organizations and our experiences show that processes tend to be less structured than expected. Traditional process mining algorithms have problems dealing with such unstructured processes and generate spaghetti-like process models that are hard

R. Hull, J. Mendling, and S. Tai (Eds.): BPM 2010, LNCS 6336, pp. 227–242, 2010.

to comprehend. Such incomprehensible models are not amenable or are found lacking to assist in process diagnostic efforts. When diagnosing processes, a business analyst is confronted with lots of interesting questions. We list some of them below:

1. *What is the most common (likely) process behavior that is executed?* Given a bag of traces from a process, it would be interesting to know which process components are essential/critical for this process. Such essential components/functions form the backbone of the process and should be conserved. Process re-design/improvement efforts should focus on improving such critical components.
2. *Where do my process instances deviate and what do they have in common?:* In practice, there is often a significant gap between what is prescribed or supposed to happen, and what actually happens. There is a need to augment process diagnostics with techniques that can assist in finding deviations by analyzing raw traces in the event logs. There are many domains/applications where this requirement is felt. Fault diagnosis, anomaly detection, diagnosis of fraudulent insurance claims are some of the applications. Given an event log containing a mix of traces where the system process functioned normally and where it malfunctioned, analyzing these traces to find deviations in malfunctioned/anomalous traces from normal traces would give cues in understanding the cause of malfunction/anomaly.
3. *Are there any common patterns of execution in my traces?:* Analyzing logs at the granularity of an individual event might not always be result yielding as one often loses the context information during such analysis. An analyst would be interested in knowing whether there are any interesting execution (behavioral) patterns in the log. The absence or presence of such patterns may indicate the cause of an anomaly (say for e.g., fraudulent insurance claim) or a security violation or a malfunction.
4. *What are the contexts in which an activity or a set of activities are executed in my event log?:* Dependencies exist between activities in a process and activity executions are expected to happen within certain contexts. There can be short-range and long-range dependencies between activities. Long-range dependencies are difficult to discover. An analyst would be interested in understanding the contexts of execution of activities and/or activity sequences.
5. *What are the process instances that share/capture a desired behavior either exactly or approximately?:* Often in diagnostics, an analyst would be interested in finding process instances that share/comply to a particular desired behavior; The desired behavior can be represented as a manifestation of some pattern of activity sequences or some complex form (combination) of these patterns. Though temporal logic approaches can assist in addressing this problem to a certain extent by discovering process instances that capture the desired behavior exactly, one might also be interested in discovering process instances that share the desired behavior approximately.
6. *Are there particular patterns (e.g., milestones, concurrent activities etc.) in my process?:* Workflow patterns refer to recurring forms/structures addressing business requirements. For example, milestones indicate specific execution points in the process model and provide a mechanism for supporting the

conditional execution of a task or sub-process. An analyst would be interested in discovering the presence of, and in analyzing milestone patterns in the process event log. Discovery of process models with concurrency is one of the challenging problems in process mining. The presence of concurrent activities creates different permutations of activities in the event log that adds to the complexity of discovery algorithms. Detection of the presence of concurrent activities might also help in pre-processing the logs.

In this paper, taking inspiration from biological sequence alignment [2], we propose a novel approach, called *trace alignment*, of aligning traces in an event log and show the promise of such an approach in process diagnostics addressing some of the questions enumerated above. Multiple sequence alignment is a topic of extensive research of over three decades in computational biology and still remains intriguing due to the intricate challenges it poses. *There are significant challenges in adopting them to trace alignment.* We highlight some of the challenges in this paper and believe that this will *open a new area of research within process mining.* Figure 1 illustrates the traditional dotted chart analysis and the proposed trace alignment[1]. It is apparent that the proposed approach of trace alignment uncovers common execution patterns and deviations in the log yielding better insights for analysis.

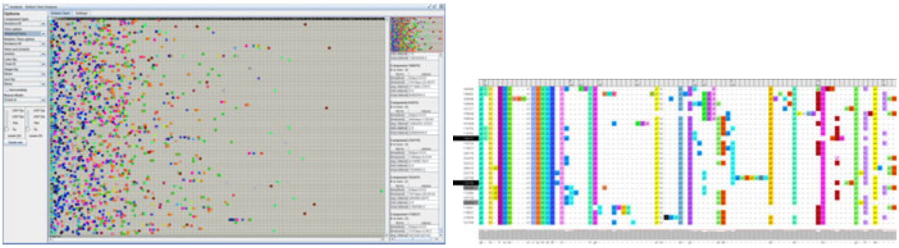


**Fig. 1.** Comparison of Dotted Chart Analysis and Trace Alignment

The remainder of this paper is organized as follows. In Section 2, we introduce the notations used in the paper. Section 3 introduces the concept of trace alignment and discusses the techniques for finding alignments. In Section 4, we propose a framework for finding alignments over a set of traces. In Section 5, we present and discuss the results of trace alignment on a synthetic log and a real-life log and show how trace alignment can assist in gaining better insights for process diagnostics. We discuss related work in Section 6. Finally, Section 7 concludes the paper.

## 2 Notations

- Let $\Sigma$ denote the set of activities. $|\Sigma|$ is the number of activities.

[1] In the dotted chart, a dot represents an activity and the x-axis represents time. In trace alignment, the x-axis represents the alignment position. y-axis represents trace indices for both the dotted chart and trace alignment.

- $\Sigma^+$ is the set of all non-empty finite sequences of activities from $\Sigma$. $T \in \Sigma^+$ is a trace over $\Sigma$. $|T|$ denotes the length of trace $T$.
- The set of all $n$-length sequences over the alphabet $\Sigma$ is denoted by $\Sigma^n$. A trace of length $n$ is denoted as $T^n$ i.e., $T^n \in \Sigma^n$, and $|T^n| = n$.
- The ordered sequence of activities in $T^n$ is denoted as $T(1)T(2)T(3)\ldots T(n)$ where $T(k)$ represents the $k^{th}$ activity in the trace.
- $T^{n-1}$ denotes the $n-1$ length prefix of $T^n$. In other words $T^n = T^{n-1}T(n)$.
- An event log, $\mathcal{L}$, corresponds to a multi-set (or bag) of traces from $\Sigma^+$.

## 3 Trace Alignment

In this section, we formally define what trace alignment is and discuss techniques for finding optimal alignments.

**Definition 1.** *Trace alignment over a set of traces* $\mathbb{T} = \{T_1, T_2, \ldots, T_n\}$ *is defined as a mapping of the set of traces in* $\mathbb{T}$ *to another set of traces* $\overline{\mathbb{T}} = \{\overline{T_1}, \overline{T_2}, \ldots, \overline{T_n}\}$ *where each* $\overline{T_i} \in (\Sigma \cup \{-\})^+$ *for* $1 \leq i \leq n$ *and*

- $|\overline{T_1}| = |\overline{T_2}| = \ldots = |\overline{T_n}| = m$,
- $\overline{T_i}$ *by removing all "$-$" gap symbols is equal to* $T_i$,
- $\nexists k, 1 \leq k \leq m$ *such that* $\forall_{1 \leq i \leq n}, \overline{T_i}(k) = -$

$m$ in the definition above is the length of the alignment. An alignment over a set of traces can be represented by a rectangular matrix $\mathcal{A} = \{a_{ij}\}(1 \leq i \leq n, 1 \leq j \leq m)$ over $\Sigma' = \Sigma \cup \{-\}$ where $-$ denotes a gap. The third condition in the definition above implies that no column in $\mathcal{A}$ contains only gaps ($-$). It is imperative to note that there can be many possible alignments for a given set of traces and that the length of the alignment, $m$, satisfies the relation $l_{max} \leq m \leq l_{sum}$ where $l_{max}$ is the maximum length of the traces in $\mathbb{T}$ and $l_{sum}$ is the sum of lengths of all traces in $\mathbb{T}$.

### 3.1 Pairwise Trace Alignment

Before we get into the details of aligning a set of traces, let us first consider a special case of trace alignment, where the number of traces to align is 2. Aligning a pair of traces is referred to as *pair-wise* trace alignment. Let us consider the example of aligning the two traces $T_1 =$ `abcac` and $T_2 =$ `acacad`. Figure 2 depicts three variants of aligning the two traces. In fact, the number of possible alignments for two traces of length $l$ is $\approx (1+\sqrt{2})^{2l+1}l^{-1/2}$ [2], e.g., for two traces of length 100, the number of possible alignments is approximately $10^{77}$. Therefore, it is infeasible to enumerate all possible alignments even for moderate values of $l$. Moreover, not all of these alignments would be interesting. In order to compute "best" alignments, we need a means of associating a score to an alignment.

Alignment between a pair of traces, $T_1$ and $T_2$ can be considered as a transformation of the trace $T_1$ to $T_2$ or viceversa through a set of editing operations

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| $\overline{T_1}$ | a b c a c - - | $\overline{T_1}$ | a b c a c - | $\overline{T_1}$ | a b c a c - - - - - - |
| $\overline{T_2}$ | a - c a c a d | $\overline{T_2}$ | a c a c a d | $\overline{T_2}$ | - - - - - a c a c a d |
| (i) |  | (ii) |  | (iii) |  |

**Fig. 2.** An Example of Pair-wise Trace Alignments

applied to one of the traces iteratively. The traces are said to be aligned after the transformation, and can be represented as a rectangular matrix as mentioned earlier. Assuming that $\overline{T_1}$ is written over $\overline{T_2}$ in the alignment (as in Figure 2), the following edit operations are defined for any column $j$ in the alignment:

- the activity pair $(a, b)$, $a, b \in \Sigma$, denotes a substitution of activity $a$ in $T_1$ with activity $b$ in $T_2$,
- the activity pair $(a, -)$ denotes the deletion of activity $a$ in $T_1$, and
- the activity pair $(-, b)$ denotes the insertion of activity $b$ in $T_1$.

It is important to note that insertion and deletion operations are complementary in that an insertion in one trace can be considered as a deletion in another trace. Henceforth, we refer to insertion and deletion operations as *indel* operation. *indels* should be sensitive to the context in which the operations are performed. For example, it is ok to have an activity `fread` after `fopen` but not after `fclose`. Hence, we consider the *indel* operation as `indelRightGivenLeft` which indicates the insertion of an activity to the right of another activity. A score function needs to be defined for the substitution and indel operations. The substitution score is a function $S : \Sigma \times \Sigma \rightarrow \Re$ where $S(a, b)$ denotes the score for substitution of activity $a$ with activity $b$ for all $a, b \in \Sigma$. The `indelRightGivenLeft` score is a function $\mathcal{I}_l : \Sigma \cup \{-\} \times \Sigma \cup \{-\} \rightarrow \Re$ where $\mathcal{I}_l(a, b)$ denotes the score for inserting activity $a$ given that the left activity is $b$ for all $a, b \in \Sigma$. $\mathcal{I}_l(a, -) = \mathcal{I}_l(-, a) = \mathcal{I}_l(-, -) = 0$ for all $a \in \Sigma$. Given $S$ and $\mathcal{I}_l$, *the score of a pair-wise alignment can be defined as the sum of the scores of the edit operations across all columns in the alignment.* In other words, if $\overline{T_1}$ and $\overline{T_2}$ are the aligned traces of $T_1$ and $T_2$, and the alignment is of length $m$, then:

$$Score(\overline{T_1}, \overline{T_2}) = \sum_{j=1}^{m} e_j$$

where

$$e_j = \begin{cases} S(a, b) & \text{if } \overline{T_1}(j) = a \text{ and } \overline{T_2}(j) = b \\ \mathcal{I}_l(a, b) & \begin{cases} \text{if } \overline{T_1}(j) = a, \overline{T_1}(j-1) = b \text{ and } \overline{T_2}(j) = - \text{ or} \\ \text{if } \overline{T_1}(j) = -, \overline{T_2}(j) = a \text{ and } \overline{T_2}(j-1) = b \end{cases} \end{cases}$$

$\overline{T_1}(0) = \overline{T_2}(0) = -$. Assuming a simple scoring function where a substitution of activity pair $(a, b)$ is associated with a score of 1 if $a = b$ and a score of $-1$ otherwise, and an indel scoring function, $\mathcal{I}_l(a, b) = -1$, for all $a, b \in \Sigma$, the alignments enumerated in Figure 2 have the scores 1, $-4$ and $-9$ respectively. A "best" alignment can be considered to be the one with the maximum score. It is imperative to note that the best scoring alignment is sensitive to the substitution and indel score functions.

**How to Compute Alignments.** Needleman and Wunsch [3] have proposed a dynamic programming algorithm for finding the optimal alignment between two amino acid sequences. The basic idea is to build up an optimal alignment using previous solutions for optimal alignments of smaller subsequences. Let $T_1$ and $T_2$ be two traces. A matrix $F$ indexed by $i$ and $j$, is constructed where the value $F(i,j)$ is the score of the best alignment between the prefix $T_1^i$ of $T_1$ and the prefix $T_2^j$ of $T_2$. $F(i,j)$ is constructed recursively by initializing $F(0,0) = 0$ and then proceeding to fill the matrix from top left to bottom right. It is possible to calculate $F(i,j)$ if $F(i-1,j-1)$, $F(i-1,j)$ and $F(i,j-1)$ are known. There are three possible ways that the best score $F(i,j)$ of an alignment up to $T_1^i$ and $T_2^j$ could be obtained: $T_1(i)$ could be aligned to $T_2(j)$, in which case $F(i,j) = F(i-1,j-1) + S(T_1(i), T_2(j))$; or $T_1(i)$ is aligned to a gap, in which case $F(i,j) = F(i-1,j) + \mathcal{I}_l(T_1(i), T_1(i-1))$; or $T_2(j)$ is aligned to a gap, in which case $F(i,j) = F(i,j-1) + \mathcal{I}_l(T_2(j), T_2(j-1))$. The best score up to $(i,j)$ will be the largest of these three options. In other words, we have

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + S(T_1(i), T_2(j)), \\ F(i-1,j) + \mathcal{I}_l(T_1(i), T_1(i-1)), \\ F(i,j-1) + \mathcal{I}_l(T_2(j), T_2(j-1)). \end{cases} \qquad (1)$$

The values along the top row (when $i = 0$) and left column (when $j = 0$) need to be handled as follows. The values $F(i,0)$ represent alignments of a prefix of $T_1$ to all gaps in $T_2$. So, we can define $F(1,0) = 0$ and for $i > 1, F(i,0) = F(i-1,0) + \mathcal{I}_l(T_1(i), T_1(i-1))$. Similarly, we can define $F(0,j)$. The value in the bottom right cell of the matrix, $F(|T_1|, |T_2|)$, is the best score for an alignment of $T_1$ and $T_2$. To find the alignment itself, we must find the path of choices from (1) that led to this best score, i.e., we move from the current cell $(i,j)$ to one of the cells $(i-1,j-1)$, $(i-1,j)$ or $(i,j-1)$ from which the value $F(i,j)$ was derived. While doing so, we add a pair of symbols onto the front of the alignment: $T_1(i)$ and $T_2(j)$ if the step was to $(i-1,j-1)$, $T_1(i)$ and the gap symbol '$-$' if the step was to $(i-1,j)$, or '$-$' and $T_2(j)$ if the step was to $(i,j-1)$. At the end we will reach the start of the matrix, $i = j = 0$. The above procedure, called traceback, will retrieve only one of the alignments that gives the best score; there can be cases where multiple options of (1) are equal. In these cases, an arbitrary choice is made. The set of all possible alignments for the best score can be enumerated by using graph traversal techniques.

### 3.2 Multiple Trace Alignment

Having discussed the alignment of two traces, let us move on to the alignment of a set of traces. One of the most popular scoring mechanisms for multiple sequence alignment of genomic sequences is the *sum-of-pairs* (SP) method. We adopt the sum-of-pairs method for trace alignment as well. Let $\overline{T_j}$ and $\overline{T_k}$ be two distinct rows extracted from a multiple trace alignment $\mathcal{A}$ (over a set of set of $n$ traces), and let $Score(\overline{T_j}, \overline{T_k})$ be the alignment score calculated in the same way as ordinary pairwise alignment of $T_j$ and $T_k$, then the SP score of a multiple trace alignment $\mathcal{A}$ is defined as

$$Score_{SP}(\mathcal{A}) = \sum_{1 \leq j \leq k \leq n} Score(\overline{T_j}, \overline{T_k})$$

It is possible to generalize the pairwise dynamic programming alignment approach to the alignment of $n$ traces. However, it is impractical for more than a few traces. Assuming that the traces are all of roughly the same length $l$, the space complexity of the multidimensional dynamic programming algorithm is $\mathcal{O}(l^n)$ and the time complexity is $\mathcal{O}(2^n l^n)$ [4]. Multiple sequence alignment that maximizes the SP score was shown to be NP-complete [5].

We adopted the progressive alignment approach for trace alignment. The basic idea of progressive alignment is to iteratively construct a succession of pairwise alignments. Alignment is allowed between a pair of traces, a trace and an alignment and between alignments. The selection of traces for alignment at each iteration is based on their similarity. Traces that are most similar to each other are aligned first. Once similar traces have been aligned, align the resulting clusters of traces against each other. A guide tree is built to assist this process. We use the agglomerative hierarchical clustering algorithm (AHC) for generating this tree. We can use either distance metrics such as Euclidean distance or similarity measures for clustering. The choice of AHC is due to the fact that it produces the tree naturally as a dendrogram while the tree has to be constructed subsequently if other clustering algorithms such as $k$-means is used.

Figure 3 illustrates an example of the progressive alignment strategy. In this example, we consider 5 traces. A guide tree is generated using AHC. Based on the guide tree, the traces $T_2$ and $T_3$ would first be aligned using pairwise trace alignment. Next traces $T_4$ and $T_5$ would be aligned using pairwise trace alignment. Subsequently, trace $T_1$ is aligned with the alignment obtained from $T_2$ and $T_3$. Finally the two alignments obtained from the set of traces $\{T_1, T_2, T_3\}$ and $\{T_4, T_5\}$ are aligned.

While aligning an alignment $\mathcal{A}$, with another alignment $\mathcal{B}$, (1) is modified as

$$F(i,j) = \max \begin{cases} F(i-1, j-1) + \overline{S}(C^i_{\mathcal{A}}, C^j_{\mathcal{B}}), \\ F(i-1, j) + \overline{\mathcal{I}_l}(C^i_{\mathcal{A}}, C^{i-1}_{\mathcal{A}}), \\ F(i, j-1) + \overline{\mathcal{I}_l}(C^j_{\mathcal{B}}, C^{j-1}_{\mathcal{B}}). \end{cases} \tag{2}$$

where $\overline{S}(C^i_{\mathcal{A}}, C^j_{\mathcal{B}})$ denotes the score of substituting column $i$ of alignment $\mathcal{A}$ with column $j$ of alignment $\mathcal{B}$ and is defined as

$$\overline{S}(C^i_{\mathcal{A}}, C^j_{\mathcal{B}}) = \sum_{\forall a,b \in \Sigma} n^i_{\mathcal{A}}(a).n^j_{\mathcal{B}}(b).S(a,b) \tag{3}$$

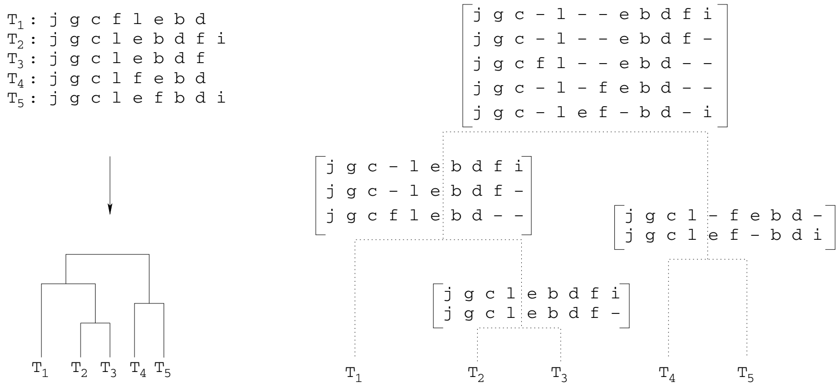


**Fig. 3.** Example of Progressive Alignment Approach for Multiple Trace Alignment

where $n^i_{\mathcal{X}}(a)$ denotes the frequency (count) of activity $a$ in column $i$ of alignment $\mathcal{X}$. $\overline{\mathcal{I}_l}(C^i_{\mathcal{A}}, C^{i-1}_{\mathcal{A}})$ denotes the score of inserting column $i$ in alignment $\mathcal{A}$ given that its left column is $i-1$ and is defined as

$$\overline{\mathcal{I}_l}(C^i_{\mathcal{A}}, C^{i-1}_{\mathcal{A}}) = \sum_{\forall a,b \in \Sigma} f^i_{\mathcal{A}}(a,b).\mathcal{I}_l(a,b) \tag{4}$$

where $f^i_{\mathcal{A}}(a,b)$ is the frequency of activity $a$ in column $i$ of alignment $\mathcal{A}$ given that its neighboring activity is $b$ in column $i-1$. The procedure for finding the "best" alignment is similar to that of pairwise alignment. Note that the guide tree enables the visualization of alignments for different subsets of the traces. The alignment at the root of the tree corresponds to the alignment of all the traces in the event log whereas an alignment at any internal node of the guide tree depicts the alignment corresponding to the traces constituting the leaves of the sub-tree at the node. It is often the case that event logs contain traces capturing different execution behavior of a process and clustering assists in grouping together a coherent set of traces.

## 4 Framework for Trace Alignment

We propose the framework depicted in Figure 4 for trace alignment. The framework identifies the following parts:

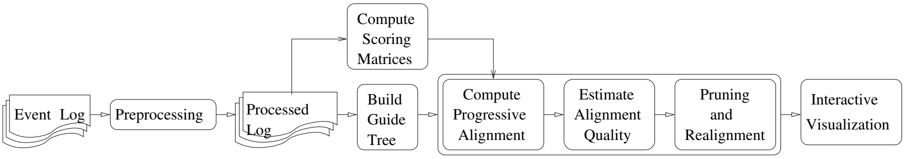


**Fig. 4.** Framework for Multiple Trace Alignment

- *Preprocessing:* Preprocessing involves steps such as removal of outliers, removal of loop-constructs, and encoding of log into character streams. The detection and removal of outliers is critical for obtaining interesting alignments.
- *Compute Scoring Matrices:* As discussed in Section 3, alignments are sensitive to the substitution and indel score functions, $S$ and $\mathcal{I}_l$ respectively. We use the approach presented in [6] for deriving the substitution and indel score functions from the event log.
- *Build Guide Tree:* A guide tree assists in progressive alignment of multiple traces as illustrated in Figure 3. We use the agglomerative hierarchical clustering (AHC) for building the guide tree. However, other approaches such as neighbor joining [7] can be used.
- *Estimate the Quality of Alignment:* Progressive alignment being a heuristics-based approach, the alignment that is obtained need not be optimal. Further any error in alignment done in early stages of progressive alignment cannot be undone. Hence it is essential to estimate the quality of an alignment. In this work, we adopt a metric based on the information score as a means for assessing the quality of an alignment. The information score of a column in an alignment is defined as $1-E/E_{max}$, where $E$ is the entropy of activities in the column[2] and $E_{max}$ is the maximum entropy which is equal to $\log_2(|\Sigma|+1)$.
- *Pruning and Realignment:* Construction of multiple trace alignment is a very complex problem, and most heuristic algorithms usually fail to generate an optimal alignment. Disturbances in an alignment can creep in from many sources thereby making the final alignment far from optimal. Disturbances here refer to the misplacement of gaps in an alignment. Efficient techniques for pruning and realigning alignments need to be supported. We will discuss more about this later in this section.
- *Interactive Visualization:* Apart from just pictorially depicting the alignment it is desirable to have additional interactive features for the analysts to explore into the patterns and the alignments uncovered. Features such as editing an alignment, sorting and/or filtering alignment columns based on activities of interest would all lead to gaining further insights into the execution of processes.

Though the definition of what constitutes an outlier is left open, in the current exploration, we have adopted one simple definition of outliers based on the length of the traces. It could be the case that in an event log there are certain process instances whose lengths deviate a lot from the average trace length in the log, e.g., one of the real life event logs that we analyzed had an average trace length of 47 activities (across 223 traces) while there were 5 traces with lengths above 250. Since an alignment is at least as long as the maximum trace length, such outlier traces in the log can lead to an alignment with too many gap symbols. Hence the removal of such traces is important. Note that the definition of outliers can change based on the perspective of analysis. If we are interested in finding common execution patterns or the backbone sequence of a process,

[2] The entropy of a column is defined as $E = \sum_{a \in \Sigma \cup \{-\}} -p_a \log_2(p_a)$ where $p_a$ is the probability of occurrence of $a$ in the column.

the above definition of outliers may work fine. However, if we are interested in finding non-conforming traces or deviations in anomalous traces from normal traces, then the above definition might not always be appropriate.

**Realigning alignments.** Variation in the lengths of the traces (mostly due to recurring patterns and loops), the choice of scoring matrices used, the method and parameter choices used in the generation of guide tree, strategies used in resolving conflicts during traceback can all lead to disturbances in the alignment. Furthermore, disturbances in earlier stages of progressive alignment strategy percolate to later stages. Detecting such disturbances and realigning them might vastly improve the quality of the final alignment. Figure 5 depicts an example of an alignment before and after realignment. For the original alignment, the information score for the columns 4 and 6 is 0.66 whereas the information score for the same columns after realignment is 0.73. More than the improvement in score, what is important it that the conserved activity sequence `ahbd` is preserved after realignment.

$$\begin{bmatrix} a & h & b & \overset{\downarrow}{-} & f & \overset{\downarrow}{d} & k & a & - & h & b & d & i \\ a & h & b & d & f & - & k & a & - & h & b & d & - \\ a & h & b & d & f & - & k & a & - & h & b & d & i \\ a & h & b & - & - & d & k & a & f & h & b & d & i \end{bmatrix} \xrightarrow{\text{Realignment}} \begin{bmatrix} a & h & b & - & f & d & k & a & - & h & b & d & i \\ a & h & b & d & f & - & k & a & - & h & b & d & - \\ a & h & b & d & f & - & k & a & - & h & b & d & i \\ a & h & b & d & - & - & k & a & f & h & b & d & i \end{bmatrix}$$

**Fig. 5.** Example of Realignment

The alignment procedure described in Section 3 is also called as *global trace alignment*. Depending on the scoring functions, global trace alignment can sometimes penalize gaps at the beginning and/or end of the traces in the alignment. In order to allow gaps to be inserted at the beginning/end of any trace in an alignment, a variant of the global trace alignment called the *semi-global trace alignment* can be considered. Here the best score of the alignment is defined to be the one that is the maximum in the last row or last column of the $F$ matrix defined in Section 3. Traceback procedure starts from that cell and proceeds until it stops at the first position it reaches in the top row or left column. Gaps can then be inserted in the appropriate trace in the positions subsequent to the maximum value cell in the last row/column and prior to the position it reached in the top row or left column. Figure 6 depicts the difference between global trace alignment and semi-global trace alignment of two traces aligned using the same scoring functions. It is easy to see that the alignment obtained using semi-global alignment is preferable to the one obtained using global-alignment. We recommend to consider semi-global trace alignment (at any iteration of progressive alignment) in scenarios where the traces to be aligned differ in their lengths vastly (for example, due to the manifestation of loop constructs).

$$\begin{array}{lccccccccccccc} \overline{T}_1 & j & g & c & - & a & h & b & - & - & - & - & f & d \\ \overline{T}_2 & j & g & c & f & a & h & b & d & k & a & h & b & d \end{array} \qquad \begin{array}{lccccccccccccc} \overline{T}_1 & j & g & c & - & a & h & b & f & d & - & - & - & - \\ \overline{T}_2 & j & g & c & f & a & h & b & - & d & k & a & h & b & d \end{array}$$

Global Trace Alignment

Semi−global Trace Alignment

**Fig. 6.** Example of global trace alignment and semi-global trace alignment

## 5 Experimental Results and Discussion

Based on the techniques and framework presented in sections 3 and 4, we built a trace alignment plug-in in ProM[3]. We present the results of applying trace alignment on two event logs in the subsequent sections.

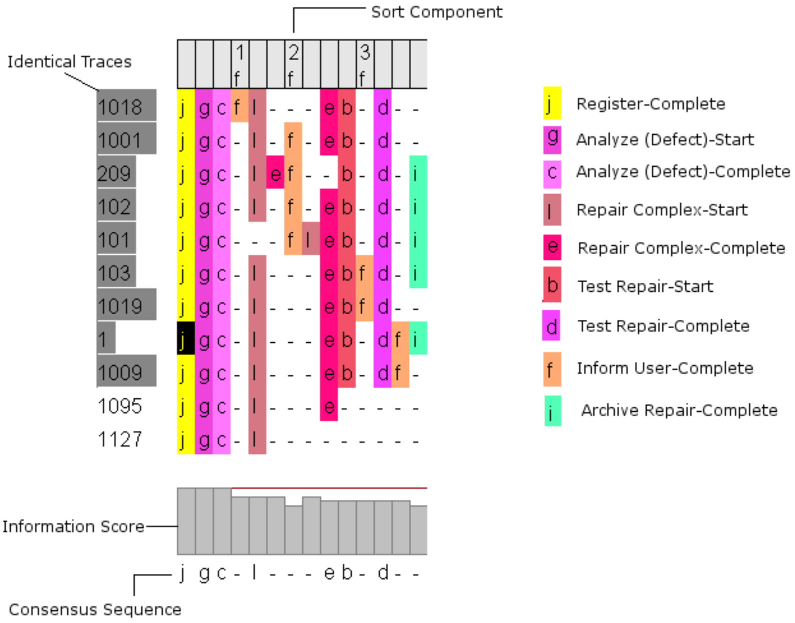
### 5.1 Telephone Repair Log

The telephone repair event log [8] is defined over 12 event classes and consists of 1104 traces, of which only 77 traces are distinct when represented as activity sequences. Since duplicate traces add to the complexity of alignment without yielding any additional benefits, we applied the trace alignment on these 77 traces (but at the same time maintain the fact that there exists identical traces in the log). The log consists of cases where the repair can be classified as a simple or complex one. For our discussion here, we further distinguish two types of cases based on the difficulty level of repair viz., cases where the telephone repair was easy and cases where it was difficult (in both simple and complex types). Difficult cases required multiple tries of the repair diagnosis for failing the quality assessment test.

As mentioned earlier, the guide tree inherently captures the notion of clustering. We have split the event log into four clusters for the example log and Figure 7 depicts the trace alignment for one of the four clusters. This cluster corresponds to traces where the repair type was easy and a complex repair procedure was done to fix the problem. The length of the alignment is 14 for this cluster. The left panel depicts the process instance identifier (as in the log) and identifiers with a grey background indicate traces that have identical duplicates. For example there are traces identical to process instance 1018 (corresponding to activity sequence `jgcflebd` in the event log) while there are no identical traces for the process instance 1127. The top panel depicts a sorting component where the traces involved in the alignment can be sorted based on the activities in a column and the number in the column indicates the priority of sorting. For example in Figure 7, the traces are sorted based on activity `f` (which indicates the `inform user` activity) with traces having `f` in column 4 having first priority and then with those having `f` in column 7 and finally with those having `f` in column 11. The bottom panel depicts the information score metric for each column as well as a consensus sequence for the alignment. *The consensus sequence captures the major activity in each column and can be considered as a back-bone sequence for the process.* Columns with an information score of 1.0 indicate well conserved patterns. For example in this alignment, the columns $1-3$ depicting the encoded activity sequence `jgc` (corresponding to activities `Register-complete`, `Analyze (Defect)-start` and `Analyze (Defect)-complete`) is well conserved and appears in all the traces as the beginning subsequence. It is obvious to see that the encoded activity `f` corresponding to `Inform User - complete`

[3] ProM is an extensible framework that provides a comprehensive set of tools/plugins for the discovery and analysis of process models from event logs. See http://www.processmining.org for more information and to download ProM.

is a concurrent activity. *Concurrent activity manifests in mutually exclusive traces across different columns in the alignment.* The encoded activities `l`, `e`, `b`, `d` and `i` correspond to `Repair Complex-start`, `Repair Complex-complete`, `Test Repair-start`, `Test Repair-complete` and `Archive Repair-complete` respectively. Annotating the traces with additional information such as performance metrics, customer feedback etc over the alignment might give further insights. For example, let us assume that the customer was not happy for the cases 1 and 1009, it is obvious to see that these traces differ from the rest in that the activity `f` appears quite late in these traces. It could be inferred that these customers were not timely informed about the status of their complaint and thus were not satisfied. The rest of the 3 clusters for this event log corresponded to the following difficulty level and repair type categories: *easy* and *simple*, *difficult* and *complex* and *difficult* and *simple/complex* where the last cluster pertained to cases where a simple repair procedure was first tried and finally a complex repair procedure was done.



| | | | | 1<br>f | | | 2<br>f | | | | 3<br>f | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1018 | j | g | c | f | l | - | - | - | e | b | - | d | - | - |
| 1001 | j | g | c | - | l | - | f | - | e | b | - | d | - | - |
| 209 | j | g | c | - | l | e | f | - | - | b | - | d | - | i |
| 102 | j | g | c | - | l | - | f | - | e | b | - | d | - | i |
| 101 | j | g | c | - | - | - | f | l | e | b | - | d | - | i |
| 103 | j | g | c | - | l | - | - | - | e | b | f | d | - | i |
| 1019 | j | g | c | - | l | - | - | - | e | b | f | d | - | - |
| 1 | j | g | c | - | l | - | - | - | e | b | - | d | f | i |
| 1009 | j | g | c | - | l | - | - | - | e | b | - | d | f | - |
| 1095 | j | g | c | - | l | - | - | - | e | - | - | - | - | - |
| 1127 | j | g | c | - | l | - | - | - | - | - | - | - | - | - |
| | j | g | c | - | l | - | - | - | e | b | - | d | - | - |

| | |
|---|---|
| j | Register-Complete |
| g | Analyze (Defect)-Start |
| c | Analyze (Defect)-Complete |
| l | Repair Complex-Start |
| e | Repair Complex-Complete |
| b | Test Repair-Start |
| d | Test Repair-Complete |
| f | Inform User-Complete |
| i | Archive Repair-Complete |

**Fig. 7.** Trace alignment of traces in telephone repair log for one of the clusters

## 5.2 Rental Agency Log

We applied trace alignment on a real life log of a rental agency where the cases corresponded to cancellation of a current rental agreement and subsequent registration of a new rental agreement. This log was provided by a large Dutch agency that rents houses and apartments (the organization has approximately

1000 employees and handles $80,000$ houses). There were 74 event classes, one event type, 210 traces and 6100 events in this log. As we can see, this log is sufficiently complex in terms of the size of the alphabet and the number of cases. The traces are first encoded into activity sequences where each activity is encoded as a two character sequence. Figure 8 depicts the alignment for one of the four clusters of this log. Since the whole alignment is not legible[4], we highlight the interesting patterns/activities (that we refer to for our further discussion) at the top and the bottom of the figure. The length of the alignment is 88. At the outset, we can see certain patterns in the form of well conserved regions (columns) in the alignment. *Deviations and exceptional behavior are captured in regions that are sparsely filled i.e., regions with lot of gap symbols* $(-)$. We will present the results of analysis of some of these deviations. It could be seen that only one of the traces (third trace in the alignment) has the activity subsequence b4a8b0 in columns $9 - 11$. Activity b0 in column 8 corresponds to the check, is first inspection done? and the activity subsequence b4a8b0 corresponds to the scenario where the result of the check was negative due to the fact that the tenant was not at home. b4 corresponds to the activity of sending a letter to the tenant and a8 corresponds to the activity of rescheduling the first inspection.

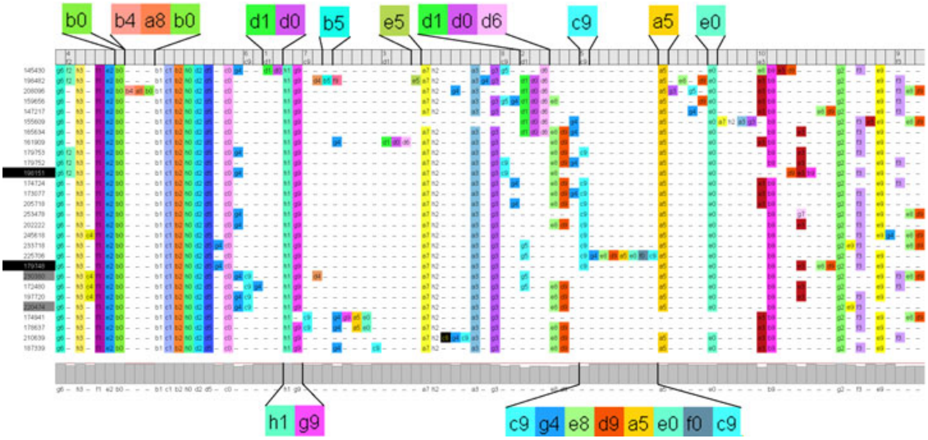


**Fig. 8.** Trace alignment for one of the clusters of rental agency log

The activity sequence h1g9 corresponding to the checks is final inspection done? (h1), and are there new/repaired defects? (g9) is well conserved across all but one of the traces . We see an exceptional activity sequence d1d0 corresponding to the offering of a flat in one of the traces (first trace) before the activity sequence h1g9. It is strange that a flat was offered before the final inspection was done as in all the other traces where the flat was offered, it happened subsequent to the final inspection. Upon further inspection, we observed

[4] The actual alignment can be inspected at www.win.tue.nl/~jcbose/AlignmentAnaysis3a.png

that though the flat was offered, the actual registration/check of the candidate corresponding to activity `d6` happened subsequent to the final inspection. Furthermore, in all the cases where the flat was offered and the candidate registered, the activity sequence `d1d0d6` was well conserved except for the first trace. Trace alignment helps us uncover such anomalies and deviations. Similarly, we notice that in only one of traces (second trace) there was a need for second inspection (activity `b5` corresponds to the planning of second inspection and `e5` corresponds to the check, `is second inspection done?`).

The activity `c9` corresponds to the determination of a candidate tenant and the activities `a5` and `e0` correspond to registration of lease and signing of contract respectively. It could also be observed from the alignment that there is an exceptional behavior in one of the cases where we see a manifestation of the activity subsequence `c9g4e8d9a5e0f0c9` (the activity `c9` appears twice). This indicates the fact that for this case, there was a need for determining the candidate tenant twice. The determination of the second candidate tenant followed the activity `f0` which corresponds to the termination of provisional lease. In this fashion trace alignment assists the analyst in getting diagnostic insights by uncovering interesting patterns and deviations.

Finding good quality alignments is intriguingly challenging. Efficient preprocessing techniques for transforming the log with abstractions might help in finding better alignments. One can try to find alignments in a multi-phase approach, with abstractions defined over conserved patterns in each iteration of the alignment. Alternatively, one can also adopt our abstraction techniques proposed in [9].

## 6 Related Work

Song and van der Aalst [10] have proposed the dotted chart analysis to analyze process performance by depicting process events in a graphical way. The dotted chart analysis (analogous to Gantt charts) primarily focuses on the time dimension of events and presents a “helicopter view” of the event log along with some metrics for performance such as the minimum, maximum and average interval between events. The business analyst need to *manually* investigate the dotted chart to identify any potential performance issues. For logs with medium to large number of activities (of the order of a few tens to hundreds), the manual inspection and comprehension of the dotted chart becomes cumbersome and often infeasible to identify interesting patterns. Trace alignment alleviates this problem, by finding those patterns automatically and depicting it to the user. In the parlance of dotted chart analysis, trace alignment considers the *logical relative* time perspective of the event log. Furthermore, it would be simple and a natural extension to project the performance metrics proposed in [10] onto the aligned traces.

Conformance checking compares an *apriori* model with the observed behavior as recorded in the log and aims at detecting inconsistencies/deviations between a process model and its corresponding execution log [11]. Conformance checking has inherent limitations in its applicability especially for diagnostic purposes. Firstly, it assumes the existence of a process model (the current realization of the conformance checker plugin in ProM requires the process to be modeled as Petrinet). However, in reality, process models are either not present or if present are incorrect or outdated (their quality typically leaves much to be desired). One can argue that process models can be discovered from the event logs and conformance checking be applied on the discovered models. However, this approach is not suitable for the analysis of highly complex and/or flexible processes, the class of models which most of the real-life logs fall into and where the discovered models are “spaghetti-like”. Even in cases where the process models are available as Petrinets, it is difficult to look inside of the processes to identify and locate problems especially with models that are large. Trace alignment analyzes the raw event traces and highlights the deviations.

Multiple sequence alignment (MSA) is an active area of research in bioinformatics. Heuristic methods such as progressive alignment [12,13] and iterative alignment [14] have been proposed for MSA. However there are challenges in adapting these techniques for trace alignment. Alignment of biological sequences typically happens over sequences with less variation in length. However, traces in an event log in process mining can be of different lengths. Variation in lengths can occur due to variation in execution paths of the instances and due to manifestation of process model constructs such as choice/loop constructs. In biological sequence alignment, there are standard scoring matrices for substitution that are derived based on physio-chemical properties of the amino acids. Insertion/deletion operations are primarily considered either with a constant gap-score (or penalty) or as an affine function. Scoring matrices for trace alignment need to be derived automatically from the event log or provided by the domain experts. Biological sequences deal with an alphabet size of either 4 (for four nucleic acids) or 20 (for amino acids). However, the number of distinct activities (event classes) in a typical process mining log can be of the order of a few hundreds. This adds to the complexity of deriving good scoring matrices and aligning traces. We took inspiration from MSA techniques [12,13,15] and adapted them for trace alignment.

## 7 Conclusions

In this paper, we proposed a novel approach of aligning traces and showed that this approach uncovers interesting patterns and assists in getting better insights on process executions. We have listed some of the interesting questions in process diagnostics and showed how trace alignment can help in diagnostic efforts. Due to the computational complexity of multiple trace alignment, automatic generation of high-quality alignments is still challenging. Traces that are outliers (noise) in the log might mislead the alignment procedure and thereby result in a low quality alignment. Better techniques to identify and discard outliers during alignment are required. Metrics and realignment strategies in the perspective of process modeling constructs and their manifestation in traces is highly desirable and is an open area of research.

**Acknowledgments.** The authors are grateful to Philips Healthcare for funding the research in process mining.

## References

1. van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering 16(9), 1128–1142 (2004)
2. Waterman, M.S.: Introduction to Computational Biology: Maps, sequences and genomes. Chapman & Hall/CRC (2000)
3. Needelman, S., Wunsch, C.: A general method applicable to the search for similarities in the amino acid sequences of two proteins. Journal of Molecular Biology 48, 443–453 (1970)
4. Durbin, R., Eddy, S., Krogh, A., Mitchison, G.: Biological Sequence Analysis: Probabilistic models of proteins and nuclei acids. Cambridge University Press, Cambridge (2002)
5. Wang, L., Jiang, T.: On the complexity of multiple sequence alignment. Journal of Computational Biology 1(4), 337–348 (1994)
6. Bose, R.P.J.C., van der Aalst, W.M.P.: Context aware trace clustering: Towards improving process mining results. In: Proceedings of the SIAM International Conference on Data Mining, pp. 401–412. SDM, Philadelphia (2009)
7. Simonsen, M., Mailund, T., Pedersen, C.N.S.: Rapid neighbor-joining. In: Algorithms in Bioinformatics, pp. 113–122 (2008)
8. de Medeiros, A.K.A., van der Aalst, W.M.P.: Process mining towards semantics. In: Advances in Web Semantics-I, pp. 35–80 (2008)
9. Bose, R.P.J.C., van der Aalst, W.M.P.: Abstractions in process mining: A taxonomy of patterns. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 159–175. Springer, Heidelberg (2009)
10. Song, M., van der Aalst, W.M.P.: Supporting process mining by showing events at a glance. In: Proceedings of the 17th Annual Workshop on Information Technologies and Systems (WITS), pp. 139–145 (2007)
11. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Information Systems 33(1), 64–95 (2008)
12. Feng, D., Doolittle, R.: Progressive sequence alignment as a prerequisite to correct phylogenetic trees. Journal of Molecular Evoluation 25, 351–360 (1987)
13. Feng, D., Doolittle, R.: Progressive alignment of amino acid sequences and construction of phylogenetic trees from them. Methods in Enzymology 266, 368–382 (1996)
14. Barton, G., Sternberg, M.: A strategy for rapid multiple alignment of protein sequences, confidence levels from tertiary structure comparisons. Journal of Molecular Biology 198(2), 327–337 (1987)
15. Daniel, C., Paul, D., Vidhya, M., Marco, O., Eun-Jong, H., Yaoyu, W., Shyamal, S., Brian, C., Shobha, P., Enoch, H.: PFAAT version 2.0: A tool for editing, annotating, and analyzing multiple sequence alignments. BMC Bioinformatics 8(1), 381 (2007)

# Content-Aware Resolution Sequence Mining for Ticket Routing

Peng Sun[1], Shu Tao[2], Xifeng Yan[3], Nikos Anerousis[2], and Yi Chen[1]

[1] Computer Science and Engineering, Arizona State University
{peng.sun,yi}@asu.edu
[2] IBM T. J. Watson Research Center
{shutao,nikos}@us.ibm.com
[3] Computer Science Department, University of California at Santa Barbara
xyan@cs.ucsb.edu

**Abstract.** Ticket routing is key to the efficiency of IT problem management. Due to the complexity of many reported problems, problem tickets typically need to be routed among various expert groups, to search for the right resolver. In this paper, we study the problem of using historical ticket data to make smarter routing recommendations for new tickets, so as to improve the efficiency of ticket routing, in terms of the Mean number of Steps To Resolve (MSTR) a ticket.

Previous studies on this problem have been focusing on mining ticket resolution sequences to generate more informed routing recommendations. In this work, we enhance the existing sequence-only approach by further mining the text content of tickets. Through extensive studies on real-world problem tickets, we find that neither resolution sequence nor ticket content alone is sufficient to deliver the most reduction in MSTR, while a hybrid approach that mines resolution sequences in a content-aware manner proves to be the most effective. We therefore propose such an approach that first analyzes the content of a new ticket and identifies a set of semantically relevant tickets, and then creates a weighted Markov model from the resolution sequences of these tickets to generate routing recommendations. Our experiments show that the proposed approach achieves significantly better results than both sequence-only and content-only solutions.

## 1 Introduction

Ticket routing is a critical issue in IT problem management. When a problem is reported to the IT service provider, a ticket is created to describe the problem symptoms and to serve as a token in the problem management process. Due to the increasing complexity of the reported IT problem, many tickets need to be routed among various expert groups, to search for the one with the right expertise to resolve it, i.e., the *resolver* group. Obviously, the goal of ticket routing is to quickly identify the resolver, so that the caused disruptions can be minimized.

Today, ticket routing is usually driven by human decisions. It is common that tickets can sometimes be mistakenly routed, which leads to unnecessary ticket routing steps. If this happens, not only resources are wasted, but also it would take longer time to close a ticket, possibly cause customer dissatisfaction. The goal of this study is to develop

R. Hull, J. Mendling, and S. Tai (Eds.): BPM 2010, LNCS 6336, pp. 243–259, 2010.

an approach to systematically reducing the number of ticket routing steps by mining historical ticket data.

Tickets typically are categorized based on the nature of the reported problems, e.g., AIX, Windows, DB2, etc. This categorization is rather coarse-grained and tells little about the problem details. Besides the problem category, two types of other information in the tickets can be utilized to improve ticket routing: (1) ticket content, which contains the text description of problem symptoms; (2) resolution sequence, which records the sequence of expert groups that have processed a ticket [23], including the final resolver group. A ticket example that contains both content and resolution sequence is shown in Table 1.

**Table 1.** A ticket example with its problem description (top) and resolution sequence (bottom)

| ID | Description |
|---|---|
| 28120 | GUI is failing with ‘‘Unable to Logon: RT11844: Security exception: [IBM][CLI Driver] SQL30081N A communication error has been detected. Communication protocol being used: ‘‘TCP/IP".Communication API being used: ‘‘SOCKETS". Location where the error was detected. |

| ID | Time | Entry |
|---|---|---|
| 28120 | 2007-05-14 | New Ticket: GUI logon failure |
| 28120 | 2007-05-14 | Transferred to Group SMRDX |
| 28120 | 2007-05-14 | Check password correctness |
| 28120 | 2007-05-14 | Transferred to Group SSDSISAP |
| 28120 | 2007-05-14 | Check authorization of user account ... |
| 28120 | 2007-05-15 | Transferred to Group ASWWCUST |
| 28120 | 2007-05-15 | Web server checking |
| ... | ... | ... |
| 28120 | 2007-05-18 | Transferred to Group SSSAPHWOA |
| 28120 | 2007-05-22 | Network checking...Resolved |

Previous works in this area have been focusing on mining only ticket resolution sequences [23,22]. In [23], a Markov-model-based method was proposed to predict the next expert group that should diagnose the problem, based the groups previously processed the ticket. While this method was shown to be effective, the semantic information embedded in ticket content was ignored. Intuitively, the higher content similarity between a historical ticket and the new ticket, the higher similarity of their routing sequences. Thus different historical tickets can have different importance in guiding the routing of the new ticket. In this paper, we seek to extend the method in [23] by utilizing this information in ticket routing.

An intuitive way of using content information is to build text classifiers that can directly label each new ticket, based on its content, with its potential resolver group. As we shall see, such a method only works for tickets that are (1) rich in content, and (2) reporting very similar problems occurred in the history. As a result, it can only resolve a portion of the studied tickets and the resulting Mean number of Steps To Resolve (MSTR) is not always reduced, compared to existing solutions using the sequence-based method. The reason is as follows.

Each expert group corresponds to specific problem diagnostic steps. When a ticket is transferred among expert groups, corresponding diagnostic steps will be taken. The problem is resolved only when the ticket is transferred to a group that performs the diagnosis relevant to the root cause. By mining the resolution sequences of historical tickets (even though they are not reporting the same problem), the sequence-based method can increase the likelihood of finding the right expert group given that certain diagnostic steps have been taken and were not able to solve the problem. While for the content-only method, it can only try the resolver groups for those not-so-similar tickets, resulting in worse performance.

The insufficiency of considering ticket sequences only or content only motivates us to integrate both information and develop content-aware resolution sequence mining techniques.

In the hybrid method, we first identify a set of existing tickets that are similar to the new ticket in content. Then, we use the resolution sequences of these similar tickets to generate a weighted Markov model. Compared with existing approach [23], in this model tickets having different similarity levels are weighted differently. To evaluate content similarity, we extend the existing text-mining techniques [27,6,20]. Specifically, we develop a *Cosine-similarity-based* weight function for model generation. Our study shows that the parameters in these weight functions can make a salient difference of the model effectiveness. Thus for the weight function, we develop an algorithm to tune its parameters to optimally fit the new ticket based on the models built for the historical ticket that is most similar to this ticket. Furthermore, we observe the situation where there are a lot of tickets that are dissimilar to the new ticket, whose combined weight may low down the effect of the highly similar tickets. Thus, we performed a model normalization to generate a training set of tickets with uniformly distributed similarities, even though the original training set can have a skewed distribution on similarities.

We conduct extensive experiments on a set of 1.4 million problems tickets. The results show that the Cosine-similarity-based weight function with normalization outperforms the other alternatives. Overall, the proposed method can reduce the MSTR of a ticket by 12.23% over the sequence-only approach proposed in [23].

To the best of our knowledge, this work is the first attempt to combine both ticket contents and resolution sequences to generate optimal ticket routing recommendations. Our contributions in this paper include:

- We explore the potential of mining ticket content to complement the resolution sequence mining method proposed in [23] to improve the accuracy in predicting ticket routing.
- We develop a hybrid approach to mine resolution sequences in a content-aware manner, and design algorithms to normalize the training data, as well as to fine-tune the parameters to achieve the optimal prediction results.
- We conduct extensive experiments, with real-world ticket data, to verify the proposed method. Our study shows it can significantly improve the efficiency of ticket routing, hence reducing MSTR. Therefore, it has great potential in serving as an on-line recommendation tool for ticket routing.

The rest of this paper is organized as follows. We first formally define our problem in Section 2. We briefly review the sequence-only method proposed in [23] in Section 3. Then we present a content-aware sequencing mining approach in Section 4. We introduce an approach for training data normalization in Section 5, and the algorithm to generate ticket routing recommendations in Section 6. In Section 7, we evaluate the effectiveness and robustness of the proposed approach. The related works are reviewed in Section 8. Finally, Section 9 concludes the paper.

## 2 Problem Formulation

In this section, we formally define the ticket routing problem. We consider a ticket processing system that involves a set of expert groups $\mathcal{G} = \{g_1, g_2, ..., g_m\}$. A ticket $t$ is a tuple $t = (\tau, s)$, where $\tau$ is the description of the problem and $s$ is the resolution sequence that consists of an ordered list of groups that processed the ticket. Take the ticket shown in Table 1 as an example, the content describes the problem as ”unable to logon”, while its resolution sequence records the routing among several groups, who may leave comments regarding the diagnostic steps taken. To simplify the problem setting, this work does not consider diagnostic comments.

In many cases, a ticket could be resolved by only one group. However, it may be transferred and diagnosed by multiple groups before the resolver group is found. Although a problem can be attributed to many different causes, there could be only one that led to the reported problem. Using the ticket in Table 1 as an example, the logon failure problem can be due to wrong password, unauthorized user account, server down, or network outage. As illustrated in Figure 1, each cause is checked by a different expert group. The problem can be resolved only if the group responsible for checking the actual cause (in this case, SSSAPHWOA) receives the ticket. Note that in this process, the ticket can be routed to groups in different orders.

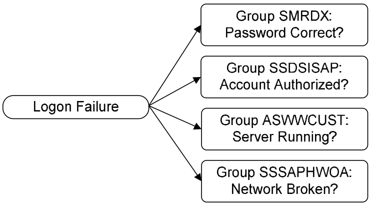

Logon Failure
Group SMRDX: Password Correct?
Group SSDSISAP: Account Authorized?
Group ASWWCUST: Server Running?
Group SSSAPHWOA: Network Broken?

**Fig. 1.** Possible causes and corresponding expert groups for the ticket in Table 1

In this study, we focus on the problem of efficient ticket routing where tickets all have a single resolver. Given this assumption, our goal is to find the resolver as quickly as possible, so that the routing delay could be minimized. For a ticket that needs multiple resolvers, the algorithm proposed in this work is going to find its last resolver group. We measure the routing efficiency with the Mean number of Steps To Resolve (MSTR) [23]. Given a set of resolved tickets, $T = \{t_1, t_2, \ldots, t_n\}$, MSTR is defined as

$$\text{MSTR}(T) = \frac{\sum_{i=1}^{n} |s_i|}{n} - 1. \tag{1}$$

Note that we assume the initial group $g_1$ is given. Therefore, MSTR represents the average number of routing steps a ticket takes, starting from $g_1$, to reach its resolver group. Obviously, the smaller the MSTR, the more efficient the ticket routing method. For a set of new problem tickets (each contains problem description in its content and has a known initial group), $T'$, the objective of this work is to *develop a routing system based on a training ticket set, $T$, which could predict the resolver of tickets in $T'$, so as to minimize the MSTR of $T'$.*

## 3 Sequence-Based and Content-Based Ticket Routing

Given a resolved ticket dataset, there are multiple ways to model ticket routing among expert groups. A traditional approach is to build text classifiers based on ticket contents and then assign tickets to different experts. In [23], we proposed a routing algorithm based on resolution sequences. Since historical ticket resolution sequences provide rich information about the relationship and dependency between experts, the sequence-based approach has demonstrated good performance. In this section, we will briefly review the sequence-based and content-based approaches, and then discuss their strengths and weaknesses, respectively.

### 3.1 Sequence-Based Routing

The sequence-based approach proposed in [23] relies on a Markov model to capture the transfer decisions made during ticket routing. In this model, each Markov state represents a group that processed the ticket. For the first-order Markov model, the transition probabilities between two groups $A$ and $B$, represent the likelihood that group $A$ transfers a ticket to $B$ when $A$ is not able to resolve it. In [23], we developed a more sophisticated approach using a variable-order Markov model.

Let $s_{(k)}$ be the set of $k$ expert groups, i.e., $s_{(k)} = \{g_{(1)}, g_{(2)}, ..., g_{(k)}\}$. Given a resolved ticket dataset $T$, the total number of tickets that have been processed by $s_{(k)}$ is denoted as $N(s_{(k)})$; and the total number of tickets that are transferred to group $g$ after all the groups in $s_{(k)}$ processed them is denoted as $N(g, s_{(k)})$. We could derive the conditional probability of transferring a ticket to $g$, given that it has been processed by $s_{(k)}$:

$$P(g|s_{(k)}) = \begin{cases} \frac{N(g,s_{(k)})}{N(s_{(k)})} & \text{if } N(s_{(k)}) > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

In [23], the conditional entropy is used to determine the optimal order of the Markov model. Using the above transition probability, we built a sequence-based routing algorithm, *Variable-order Multiple active state Search* (VMS) to predict the next group a ticket should be routed to.

VMS works as follows. Given the set of all groups that have processed ticket ($L_v$), VMS considers all its subsets $s_{(k)} \subseteq L_v$, and selects the next group from a candidate list $L_c$ ($L_c \cap L_v = \varnothing$) to maximize the transfer likelihood:

$$g^* = \text{argmax}_g \; P(g|s_{(k)}), \forall g \in L_c, s_{(k)} \subseteq L_v, \tag{3}$$

using all of transfer probabilities calculated through Eq. 2. This prediction can be conducted interactively, at any stage of ticket routing, until the final resolver group is found.

Note that if group $g^*$ identified by Eq. 3 is not the resolver, it will then be added to $L_v$ in the next iteration. More details of the VMS method can be found in [23].

The VMS method is a *sequence-based routing method*. It assumes that tickets related to similar problems are available to build the model. In [23], this is guaranteed by the manual ticket categorization by the experts. In practice, however, such categorization can be coarse-grained or inaccurate, which could undermine the effectiveness of this method.

### 3.2 Content-Based Routing

Ignored by the sequence-based approach, ticket content contains informative descriptions of reported problems, such as where and when the problem occurred, the affected system, the phenomena etc. Intuitively, the content information should be very useful to identify the right resolver to a ticket. A straightforward approach of leveraging content information is to predict resolvers from ticket description. This is a classic text classification problem, for which various known algorithms (e.g. support vector machine (SVM), k-nearest neighbor, etc.) can be applied. For example, one could create a feature vector from the problem description. Each vector will then be mapped to a resolver group – the class label.

In our study, we trained an SVM classifier with the RBF kernel using the training ticket set. For each testing ticket, we use this classifier to generate a list of candidate groups, ordered by their matching probabilities. We then assign groups in this list, starting from the top, until the true resolver group is found. We refer to this method as the *content-based routing approach*.

### 3.3 Discussion

Figure 2 compares the cumulative prediction accuracy of the sequence-based and content-based methods as a function of the number of routing steps. The ticket dataset is obtained from the IBM problem management system, and related to the AIX operating system. It is clear that the prediction accuracy of the content-based method is better than the sequence-based method at the beginning. However, it barely improves in the following

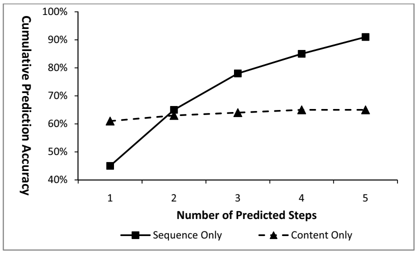


**Fig. 2.** Cumulative prediction accuracy as the number of routing steps allowed increases: sequence-only method vs. content-only method

steps. As a result, it is outperformed by the sequence-based method gradually. A careful examination of the tickets shows that the content-only method performs better for those content-rich tickets than the sequence-only method. Unfortunately, for the tickets that are either not semantically rich in their descriptions, or their reporting problems never happened before, the content-based approach will perform poorly. In contrast, the sequence-based method can be more effective for those tickets since its decision is based on ticket transfer probability, which is complementary to ticket contents.

We illustrate this effect using the example in Table 1. If the reported problem has appeared in the training data and there is only one root cause of this problem, the content-based method will perform well. If either of the conditions is not met, it could make huge classification errors and needs more steps to resolve the problem. For the sequence-based method, it predicts the next step based on the actions that have been taken: if the actions represented by *SMRDX* and *SSDSISAP* have been taken, it predicts *SSSAPHWOA* as the most likely group to solve the problem. This decision process does not rely on the fact that the same or similar problems have happened before. Instead it can be inferred from many other ticket processing patterns in the training data.

From the above discussion, we can see clearly that both the sequence- and content-only methods have their own strength and weakness. This motivates us to develop a hybrid approach that combines these two methods together so that the predication accuracy of ticket routing can be maximized.

## 4 Content-Aware Resolution Sequence Mining

In this section, we introduce a content-aware sequence mining method that customizes the VMS routing algorithm for each new ticket: an individual VMS model is derived for each new ticket $t'$ based on the similarity between $t'$ and the tickets in the training dataset. The basic idea is as follows: For a new ticket, we first evaluate the content similarity between the existing tickets and the new ticket; Then, the sequences of those similar tickets are used to learn a sequence-based routing model. In particular, the sequences of the training tickets weigh differently according to their similarity to the new ticket in content.

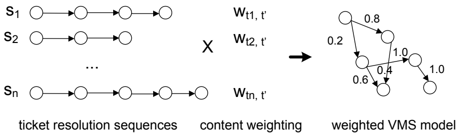


**Fig. 3.** Content-aware Weighted VMS Model

### 4.1 Overview

Given a training ticket set $T$ and a new ticket $t'$, we first evaluate the similarity between the new ticket and the existing ones, written as $w_{t_i,t'}$, $t_i \in T$. The similarity function will be discussed in Section 4.2. For each ticket $t$ in $T$, we use $s_{(k)}$ to denote the set of $k$ groups that have processed it in the past, and if a group $g$ processed the ticket after all the groups in $s_{(k)}$, we denote it as $s_{(k)} \to g$. We then define $I(g, s_{(k)}, t)$ as the indicator function of whether $s_{(k)} \to g$ occurred in the routing sequence of $t$, i.e.,

$$I(g, s_{(k)}, t) = \begin{cases} 1 \text{ if } (s_{(k)} \to g) \text{ is found in } t \\ 0 \text{ otherwise.} \end{cases}$$

Similarly, we define $I(s_{(k)}, t)$ as the indicator function of whether a set of groups $s_{(k)}$ ever processed ticket $t$, i.e.,

$$I(s_{(k)}, t) = \begin{cases} 1 \text{ if } s_{(k)} \text{ found in } t \\ 0 \text{ otherwise.} \end{cases}$$

Thus, for a new ticket $t'$, the weighted transition probabilities of the VMS model is defined as:

$$P(g|s_{(k)}) = \frac{\sum_{t_i \in T} w_{t_i,t'} I(g, s_{(k)}, t_i)}{\sum_{t_i \in T} w_{t_i,t'} I(s_{(k)}, t_i)}. \tag{4}$$

Here, the weight function $w_{t_i,t'}$ controls the contribution of ticket $t_i$ to the calculation of transition probability. Figure 3 illustrated the content-aware weighted VMS model. When $w_{t_i,t'} = 1$, the learned VMS model will be the same to all new tickets. When $w_{t_i,t'}$ reflects content similarity between training tickets $t_i$ and $t'$, it becomes a customized model for ticket $t'$.

### 4.2 Content Similarity-Based Weight Functions

To measure the content similarity, we adopt the vector space model that represents text as vectors [19,4]. Vector-based similarity models have been reported to have limitations for representing long documents [12]. However, this is not an issue for our studied ticket data set, in which we found 96% of the tickets contain less than 80 words.

Before vectorizing tickets, we preprocess tickets using stopword deletion, word stemming [2], etc. After preprocessing, only 35,690 dimensions (distinct words) were left for all the studied tickets. Then the bag-of-words approach is employed to convert tickets to vectors. Formally, let $V$ be the word set. For each ticket $t_i = (\tau_i, s_i)$ in a ticket dataset $T$, we have a $|V|$-dimension vector $\overrightarrow{\tau}_i = \langle v_{i1}, ..., v_{i|V|} \rangle$, where

$$v_{ij} = \log(c(w_j, \tau_i) + 1) \log(\frac{|T| + 1}{df_j}).$$

Here, $c(w_j, \tau_i)$ is the frequency of word $w_j$ in ticket $\tau_i$; $df_j$ is the number of tickets in $T$ that contain word $w_j$. Using this vector definition, we can compute similarity between tickets, and define weight functions.

In this paper, we examine the commonly used Cosine similarity function.

$$cos(\tau_i, \tau') = \frac{\overrightarrow{\tau}_i \cdot \overrightarrow{\tau}'}{||\overrightarrow{\tau}_i|| \cdot ||\overrightarrow{\tau}'||}, \tag{5}$$

where $\overrightarrow{\tau_i}$ and $\overrightarrow{\tau}'$ are the vectors derived from ticket contents of $t_i$ and $t'$, respectively.

Specifically, we define $w_{t_i,t'}$ as an exponential of the Cosine between two tickets.

$$w_{t_i,t'} = cos(\tau_i, \tau')^m, \tag{6}$$

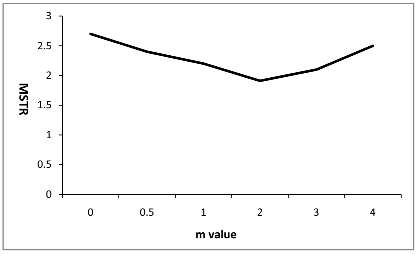


**Fig. 4.** The impact of $m$ on the routing model

where $m \geq 0$ is a parameter. When $m = 0$, the model falls back to the unweighted version. When $m \rightarrow +\infty$, the most similar ticket dominates the transition probability. Figure 4 shows the MSTR achieved for a testing data set with 1,000 tickets. The accuracy of the routing model is seemingly a convex function of $m$.

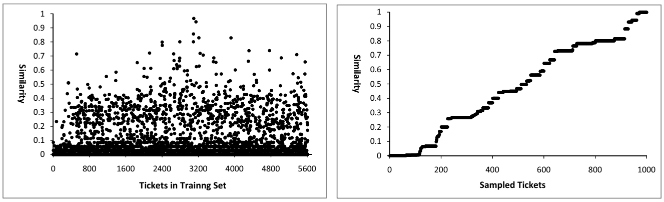
The optimal value of $m$ depends on the distribution of the similarity scores between $t_i$ and $t'$. Hence, it should be tuned for each new ticket. Since the resolver of a new ticket $t'$ is unknown beforehand, we are not able to directly tune the parameter $m$ for $t'$. Instead, we choose the training ticket (or a set of tickets to reduce noise) that is the most similar to $t'$ to tune $m$. Specifically, we leave the most similar one $t^*$ out as the new "testing" ticket, and use the rest of the tickets in the training set $T$ to construct the model in Eq.(4). We gradually increase $m$ starting from 0 with step 0.5 in each iteration, and compare the MSTR of $t^*$ as $m$ increases. The value of $m$ that minimizes the MSTR of $t^*$ will be chosen.

## 5 Training Data Normalization

The weighted VMS model proposed in Section 4 relies on a common assumption: the similarity between a new ticket and all training tickets is uniformly distributed. In practice, this assumption may not hold. For instance, Figure 5(a) shows the distribution of similarity between a randomly selected new ticket and 5,600 training tickets in the *AIX* problem category, using the Consine weight function. It shows that there are far more dissimilar tickets than similar ones.

As a consequence, even though the individual weight assigned to a dissimilar ticket is less than that assigned to a similar one, the overall transition probability can be overwhelmed by the dissimilar training tickets. For example, suppose there are 99 tickets that have the similarity value of 0.1, while only one ticket has the similarity value of 0.9. Ideally, this very similar ticket should dominate the routing recommendations generated by our model. However, if all the 99 less-similar tickets contain the same transition pattern, the transition probability for this pattern is likely higher than other patterns extracted from the most similar ticket. This might largely impact the accuracy of our method.

To overcome this problem, we use a *Bin-based Gibbs Sampling* method [7] to normalize the training tickets, so that their similarity to the new ticket is uniformly distributed. The approach works as follows.



(a) Original training set (b) Normalized training set

**Fig. 5.** Similarity distribution

First, we partition all the existing tickets into 50 buckets which are made by dividing the similarity range $[0, 1]$ into 50 equal size bins. We consider that all the existing tickets are represented by a $n$-variate joint probability distribution $p(\tau_i) = p(\tau_{i1}, \tau_{i2}, ..., \tau_{in})$, from which we wish to sample. Here, $\tau_i$ represents the content of ticket $i$, $n$ is the number of dimensions in the word vector space. Suppose we choose a random ticket as the initial sample. In each step of Gibbs Sampling, we replace the value in dimension $k$ (i.e., $\tau_{ik}$) by a new value drawn from the distribution of that variable conditioned on the values of the remaining variables. That is, the value of $\tau_{ik}$ is replaced by $\tau'_{ik}$ drawn from distribution $p(\tau_{ik}|\tau_i\backslash k)$, where $\tau_i\backslash k = (\tau_{i1}, ..., \tau_{i(k-1)}, \tau_{i(k+1)}, ..., \tau_{in})$. One iteration of the sampling consists of $n$ such steps that renew the values for all $n$ dimensions. Then after each iteration, we find a ticket whose content vector is the most similar to the newly generated vector $(\tau'_{i1}, \tau'_{i2}, ..., \tau'_{in})$. The iteration continues until an equal number of tickets are sampled from each bucket.

The above method ensure that we obtain a set of tickets whose similarity to the new ticket is uniformly distributed. For example, Figure 5(b) shows the similarity distribution of 1,000 sampled tickets obtained from the original data set in Figure 5(a), using the proposed method. Clearly, the similarity distribution is now much closer to uniform than the original distribution was. Using these sampled tickets as the training data, we can then apply the models proposed in Section 4 to generate routing recommendations. As shown later in Section 7, this normalization can significantly improve the performance of our model.

# 6 Implementation

We implemented the proposed content-aware ticket routing algorithm in C#. The algorithm runs in three phases as described in Algorithm 1.

In Phase One, our algorithm first vectorizes the new ticket $t$ as discussed in Section 4.2 and then calculates the similarity between $t$ and the resolved tickets. Note that to reduce the delay of this step, all historical tickets are pre-vectorized and the resulting vectors are stored with inverted indices [19].

In Phase Two, it retrieves the most similar tickets to the new ticket $t$ in the historical ticket dataset. Then, it finds the optimal parameter $(m)$, based on the leave-one-out methods described in Section 4.

In Phase Three, our algorithm creates a weighted Markov model with the optimal parameter determined in Phase Two. Then, routing recommendations are generated for the new ticket $t$ using the weighted VMS algorithm, given its initial group $g$.

**Algorithm 1.** Content-aware ticket routing ($t = (\tau, \{g_1\})$)

```
ticket content: $\tau$, initial group: $g_1$
Phase 1:
Build vector for ticket $t$: $\overrightarrow{\tau}$.
Evaluate similarity between $\overrightarrow{\tau}$ and the vector of each historical ticket.

Phase 2:
Select the set of tickets $T_s$ that are most similar to $t$.
Learn optimal $m$ based on $T_s$.

Phase 3:
Generate a normalized training set for $t$.
Calculate the weighted transition probabilities using Eq. 4 with the optimal $m$ (or $\sigma$) found
in Phase 2.
Set initial routing sequence $s = \{g_1\}$
while resolution group is not found do
    Use the weighted VMS routing algorithm with input $s$ to recommend the next group $g$.
    $s = s \cup \{g\}$.
end while
```

We tested the performance of our code on a machine with 3.60 GHz CPU, 2GB memory. On average, the time spent in Phases One and Two in constructing the Markov model is about 11 seconds. Once the model is obtained, the time for computing the next routing group in Phase Three is negligible. Therefore, our algorithm can be readily used as an online recommendation system.

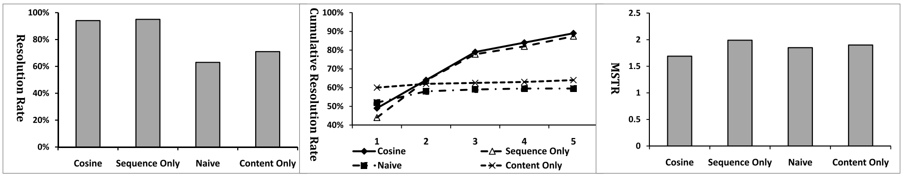
## 7 Experiments

In this section, we evaluate the proposed approaches empirically. Our evaluation is based on 1.4 million problem tickets obtained from IBM's problem management system over half a 1-year period, from Jun. 1, 2006 to Dec. 31, 2006.

These tickets were pre-classified into 553 problem categories. This is the coarse-grained ticket categorization, typically done by the helpdesk when a ticket is first opened. In each problem category, 50 to 900 expert groups were involved in the ticket routing process.

Before explaining our experiment result, we will first introduce another ticket routing approach based on the resolution sequence only: the simplest ticket routing strategy, which we call *Naive Approach*. In this approach, the training dataset is composed of the pairs of initial group and the resolver group of each routing sequence in the historical ticket database. For an incoming ticket, based on its initial group, we calculate the transition probabilities of possible resolver groups. The resolver groups are ranked in the descending order of the transition probabilities, and are attempted in the order until the resolver group is found.

Besides MSTR, we introduce another effectiveness evaluation criteria: *Resolution Rate*. It measures how many tickets in the testing set can be resolved using a routing



(a) Resolution Rate (b) Cumulative Resolution Rate (c) Performance (MSTR)

**Fig. 6.** Comparing cosine based content-aware, sequence-only, content-only, naive approaches

strategy. Specifically, for a testing set $T = \{t_1, t_2, \ldots, t_n\}$, resolution rate is defined as:

$$\mathrm{RR}(T) = \frac{\sum_{t_i \in T} R(t_i)}{|T|}. \tag{7}$$

where the routing sequence of $t_i$ has the last group denoted as $g_i$,

$$R(t_i) = \begin{cases} 1 & \text{if } g_i = g_i^* \ , \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

where $g_i^*$ is the resolver of $t_i$ determined by human decision.

Our experiments mainly aim to study the *effectiveness* of our approach: We first study the relationship between resolution rate and MSTR among four approaches: naive approach, sequence-only, content-only approach and our proposed cosine based content-aware approaches

Then we will focus on cosine based content-aware approach and sequence-only approach to show why content-aware approach is better than sequence-only approach.

### 7.1 Resolution Rate Comparison

First, we conduct the experiments to evaluate the resolution rate and MSTR of four different approaches. From 15392 AIX tickets, we randomly select 75% tickets as the training set, and use the rest of the 25% as the testing set to simulate new tickets that need to be routed.

The resolution rate of previous four approaches is shown in Figure 6(a). As we can see, while the other approaches have the resolution rate of at least 94%, the naive and content-only approaches only have a resolution rate of 63% and 71% respectively, which are not feasible to use in practice. Intuitively, since the naive approach only considers the correlation between the initial groups and the resolver groups, ignoring intermediate groups, it has a limited set of training instances and fails to transit many tickets to their resolvers. Content-only approach also only considers resolver groups, therefore it suffers a low resolution rate for the same reason as naive approach.

In addition, Figure 6(b) shows the cumulative resolution rate of tickets resolved within a given number of routing steps, when different routing approaches are used. As shown in the figure, cosine based content-aware approaches consistently outperform the sequence-only approach. This clearly demonstrates the benefit of incorporating ticket content information as well as sequence information for ticket routing: more tickets can be resolved within a given number of predicted steps.

Besides, the content-only and naive approaches outperform the other models at the first step of routing, showing that both of them are very effective for easy-to-resolve tickets. However, they become less effective as the routing continues. This indicates that for more difficult tickets, both content and sequence information should be considered to make effective ticket routing decisions.

### 7.2 MSTR Comparison

Figure 6(c) shows the performance of four approaches, where the cosine based approach has the best performance comparing to the other approaches.

**Effect of Normalization.** As mentioned in Section 5, to avoid lowing down the effect of the highly similar tickets, we generate uniformly distributed tickets as the training set. Now we evaluate the effect of the normalization.

In Figure 7, we know that normalization can reduce the MSTR about 15.28%. This shows that the normalization approach proposed in Section 5 is effectively when handling skewed data, and largely improves the MSTR.

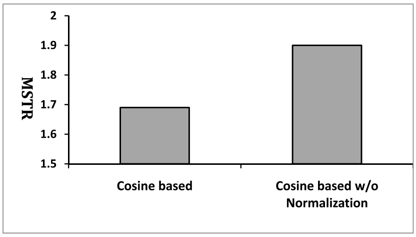


**Fig. 7.** The effect of normalization

### 7.3 Content-Aware Approach vs. Sequence-Only Approach

Using the same set of AIX tickets, we evaluate the MSTR of cosine based content-aware approach in improving the effectiveness of ticket routing, i.e., reducing MSTR, compared with the sequence-only approach.

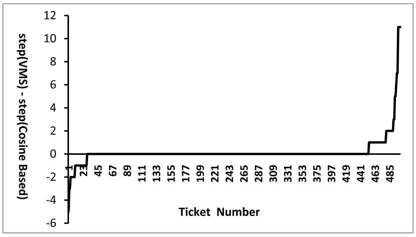
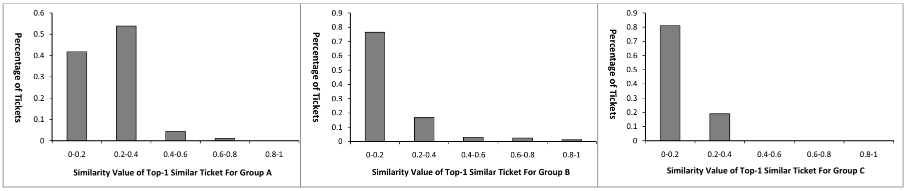


**Fig. 8.** The differences between sequence-only approach and the consine based content-aware approach in the resulting number of steps needed to resolve a ticket



(a) Group A (b) Group B (c) Group C

**Fig. 9.** Top-1 similar tickets distribution for different groups

Figure 8 shows the comparison between the sequence-only approach and the content-aware approach, in terms of the difference in number of steps needed to resolve a ticket in ascending order, for 500 randomly selected tickets. The figure shows that, for the majority of the tickets, the performance of cosine based content-aware approach is either the same as or better than that of sequence-only approach (i.e., area with value greater than 0 in the figure). Nevertheless, the content-aware approach can still be outperformed by the sequence-only approach in some cases.

To illustrate why this happens, we first partition all the tickets into three groups based on difference of MSTRs for two approaches. In group $A$, cosine based content-aware approach performs better than sequence-only approach. In group $B$, both approaches have the same MSTR. In group $C$, the cosine based content-aware approach has a larger MSTR than that of sequence-only approach.

Then, we analyze the distribution for most similar ticket of each ticket in Figure 9. Figure 9 (a) shows the distribution for group $A$. As we can see, most of the top-1 similar tickets are located in the similarity range of [0.2, 0.4). While the top-1 similar tickets of group $B$ (Figure 9 (b))and $C$ (Figure 9 (c)) are mostly located in the range of [0, 0.2). Since the tickets in group $A$ can find highly similar tickets to tune the parameters when building the routing model, they can be more effectively routed by our approach. Also, we can see that for group $C$, there is no ticket that can find a similar ticket with score larger than 0.4, resulting a larger MSTR. These observations confirm our intuition that the content-aware approach is more effective than the sequence-only approach when there are more tickets reporting on similar problems in the training data.

We have compared our content-aware approach with the sequence-only approach in all 533 problem categories, and found the former consistently outperforms the latter. Overall, the resulting MSTR of our content-aware approach is lowed by 12.23%, compared to that achieved by the sequence-only approach in [23].

## 8 Related Work

**Text Mining.** Related to this paper are the works on text mining [13,17], which covers several important research areas, including text classification [20,6], text association[16], topic modeling [27], etc. The Vector Space Model (VSM) applied in our system has also been studied before in the literature [19]. For instance, [15] first introduced SVM into the text classification applications based on the VSM model; the robustness of different text categorization methods was studied in [25]; [20] proposed methods to combine content and link information for document classification; [6] studied manifold methods; [6] introduced text classification using graph-based methods; [16] extracted word associations from text using synonyms or terms that tend to co-occur; [27] developed statistical models to find topics within a collection of documents. The weighted k-nearest neighbor classification method proposed in [14] extended the basic k-nearest neighbor algorithm by taking into account the distances to the nearest neighbors. In this paper, we extend these techniques in ticket routing applications, and address the unique challenges of parameter tuning and training set normalization in this context.

**Expert Finding.** The ticket routing problem is also related to expert finding: given a keyword query, find the most knowledgeable persons regarding the keyword query. Expert finding algorithms in [5,10] use a language model to calculate the probability of an expert candidate to generate the query terms. [21] enhances these models by allowing candidates' expertise to be propagated within networks such as email networks, while [9] explores the links in documents such as DBLP [1]. Since most of expert finding algorithms are content-based, they have the same weakness as the content-based classification methods, as illustrated in Section 3.

**Sequence Mining.** The problem of sequential pattern mining was introduced by Agrawal *et al.* [3]. Various combinatorial algorithms such as SPADE [26], PREFIX [18], were developed for efficient mining in large sequence databases. Besides combinatorial solutions, probabilistic sequence mining was also studied in the literature [8,11,24]. For instance, Cook *et al.* [8] developed neural network and Markov approaches for mining software engineering processes. In the specific problem of ticket routing, [23] was the first work that proposed Markov-model-based methods to predict ticket routing steps. This paper extends that work to incorporate both sequence and content to accelerate ticket routing.

## 9 Conclusions

In this paper, we study the problem of improving the efficiency of ticket routing by mining both ticket content and resolution sequences. We propose a novel content-aware sequence mining technique to build ticket routing models. Specifically, We build a weighted Markov model with tickets having different similarity levels weighted differently. Ticket content similarity is measured using a *Cosine-similarity-based* weight function, where the parameters are tuned to optimally fit the new ticket. Furthermore, our technique performs a normalization on training set to effectively handle the training set with diverse distribution on ticket similarity. Extensive experiments on real-world ticket data show that, because of the incorporated content information, our proposed approach is consistently more effective than both sequence-only and content-only approaches. In particular, comparing to the sequence-only approach in [23], our approach has almost the same resolution rate, with a significant reduction of 12.23% in MSTR, and thus effectively accelerates the ticket routing processes.

## Acknowledgment

This material is based on work partially supported by the U.S. National Science Foundation under grants IIS 0915438, 0917228 and 0954125.

## References

1. DBLP: http://www.informatik.uni-trier.de/~ley/db/
2. Aas, K., Eikvil, L.: Text categorisation: A survey (1999)
3. Agrawal, R., Srikant, R.: Mining sequential patterns. In: Proc. ICDE (1995)
4. Atkeson, C.G., Moore, A.W., Schaal, S.: Locally weighted learning (1996)
5. Balog, K., Azzopardi, L., de Rijke, M.: Formal models for expert finding in enterprise corpora. In: SIGIR, pp. 43–50 (2006)
6. Belkin, M., Niyogi, P., Sindhwani, V., Bartlett, P.: Manifold regularization: A geometric framework for learning from examples. Technical report, Journal of Machine Learning Research (2004)
7. Bishop, C.M.: Pattern Recognition and Machine Learning (Information Science and Statistics). Springer, Heidelberg (October 2007)
8. Cook, J., Wolf, A.: Discovering models of software processes from event-based data. ACM Trans. Software Eng. and Methodology 7(3), 215–249 (1998)
9. Deng, H., King, I., Lyu, M.R.: Formal models for expert finding on dblp bibliography data. In: ICDM 2008: Proceedings of the 2008 Eighth IEEE International Conference on Data Mining, pp. 163–172 (2008)
10. Fang, H., Zhai, C.: Probabilistic models for expert finding. In: Amati, G., Carpineto, C., Romano, G. (eds.) ECIR 2007. LNCS, vol. 4425, pp. 418–430. Springer, Heidelberg (2007)
11. Gaaloul, W., Bhiri, S., Godart, C.: Discovering workflow transactional behavior from event-based log. In: Meersman, R., et al (eds.) OTM 2004. LNCS, vol. 3290, pp. 3–18. Springer, Heidelberg (2004)
12. Garcia, E.: Description, advantages and limitations of the classic vector space model (2006)
13. Hearst, M.: What is text mining? (2003), http://people.ischool.berkeley.edu/hearst/text-mining.html
14. Hechenbichler, K., Schliep, K.: Weighted k-nearest-neighbor techniques and ordinal classification. Technical report, Ludwig-Maximilians University (2007)
15. Joachims, T.: Text categorization with support vector machines: Learning with many relevant features. In: Nédellec, C., Rouveirol, C. (eds.) ECML 1998. LNCS, vol. 1398, pp. 137–142. Springer, Heidelberg (1998)
16. Lin, D.: Extracting collocations from text corpora. In First Workshop on Computational Terminology (1998)
17. New York Times. Text mining, http://blogs.zdnet.com/emergingtech/?p=304
18. Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., Hsu, M.-C.: PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In: Int. Conf. Data Engineering (2001)
19. Salton, G., McGill, M.J.: Introduction to Modern Information Retrieval. McGraw-Hill, Inc., New York (1986)
20. Sen, P., Namata, G.M., Bilgic, M., Getoor, L., Gallagher, B., Eliassi-Rad, T.: Collective classification in network data. Technical report (2008)
21. Serdyukov, P., Rode, H., Hiemstra, D.: Modeling multi-step relevance propagation for expert finding. In: CIKM, pp. 1133–1142 (2008)

22. Shao, Q., Chen, Y., Tao, S., Yan, X., Anerousis, N.: Easyticket: A ticket routing recommendation engine for enterprise problem resolution. In: 34th Int'l Conf. VLDB, Auckland, New Zealand (2008)
23. Shao, Q., Chen, Y., Tao, S., Yan, X., Anerousis, N.: Efficient ticket routing by resolution sequence mining. In: KDD 2008, pp. 605–613 (2008)
24. Silva, R., Zhang, J., Shanahan, J.G.: Probablistic workflow mining. In: Proc. 1998 Int'l Conf. Knowledge Discovery and Data Mining, pp. 469–483 (1998)
25. Yang, Y., Liu, X.: A re-examination of text categorization methods. In: SIGIR (1999)
26. Zaki, M.: SPADE: An efficient algorithm for mining frequent sequences. Machine Learning 40, 31–60 (2001)
27. Zhai, C., Velivelli, A., Yu, B.: A cross-collection mixture model for comparative text mining. In: KDD 2004 (2004)

# Symbolic Execution of Acyclic Workflow Graphs

Cédric Favre and Hagen Völzer

IBM Research — Zurich
{ced,hvo}@zurich.ibm.com

**Abstract.** We propose a new technique to analyze the control-flow, i.e., the workflow graph of a business process model, which we call *symbolic execution*. We consider acyclic workflow graphs that may contain inclusive OR gateways and define a symbolic execution for them that runs in quadratic time. The result allows us to decide in quadratic time, for any pair of control-flow edges or tasks of the workflow graph, whether they are sometimes, never, or always reached concurrently. This has different applications in finding control- and data-flow errors. In particular, we show how to decide soundness of an acyclic workflow graph with inclusive OR gateways in quadratic time. Moreover, we show that symbolic execution provides diagnostic information that allows the user to efficiently deal with spurious errors that arise due to over-approximation of the data-based decisions in the process.

## 1 Introduction

With the increased use of business process models in simulation, code generation and direct execution, it becomes more and more important that the processes are free of control- and data-flow errors. Various studies (see [1] for a survey) have shown that such errors frequently occur in process models.

Some of these errors can be characterized in terms of relationships between control-flow edges or tasks of the process. For example, a process is free of *deadlock* if any two incoming edges of an AND-join are always marked concurrently. We can say that such a pair of edges is *always concurrent*. A process is free of *lack of synchronization* if any two incoming edges of an XOR-join are *mutually exclusive*, i.e., they never get marked concurrently. A data-flow hazard may arise if two conflicting operations on the same data object are executed concurrently. That can happen only if the tasks containing the data operations are *sometimes concurrent*, i.e., not mutually exclusive. Similar relationships have also been proposed for a behavioral comparison of processes [2].

Such control-flow relations can be computed by enumerating all reachable control-flow states of the process by explicitly executing its *workflow graph*, i.e., its control-flow representation. However, there can be exponentially many such states, resulting in a worst-case exponential time algorithm. We propose in this paper a form of symbolic execution of a workflow graph. We consider acyclic workflow graphs that may contain inclusive OR (IOR) gateways and define a symbolic execution of such graphs that runs in quadratic time. It captures enough information to allow us to decide, using a complementing graph analysis technique, the above mentioned relationships for any pair of control-flow edges in quadratic time. In particular, we obtain a control-flow

R. Hull, J. Mendling, and S. Tai (Eds.): BPM 2010, LNCS 6336, pp. 260–275, 2010.

analysis that decides *soundness*, i.e., absence of deadlock and lack of synchronization in quadratic time for any acyclic graph that may contain IOR gateways.

The symbolic execution keeps track of which decision outcomes within the process flow lead to which edge being marked. Therefore, it can provide information, in case of a detected error, about which decisions potentially lead to the error. We show how this leads to more compact diagnostic information than obtained with prior techniques. In particular, we show how this allows the user to efficiently deal with spurious errors that arise due to over-approximation of the data-based decisions in the process.

Some existing techniques can decide soundness of a workflow graph without IOR gateways, or equivalently a Free Choice Petri net, in polynomial time: A technique based on the *rank theorem* [3] in cubic time and techniques based on a complete reduction calculus [4] in more than cubic time. However, diagnostic information is not provided by the former technique and was not yet worked out for the latter.

Techniques based on *state space exploration* return an *error trace*, i.e., an execution that exhibits the control-flow error, but they have exponential worst-case time complexity. It has been shown [5] for industrial processes without IOR gateways that the latter problem can be effectively addressed in practice using various reduction techniques. Various additional structural reduction techniques exist in the literature, e.g., [6,7].

Wynn *et al.* [8] provide a study of verifying processes with IOR gateways. They apply state space exploration and use a different IOR-join semantics.

We are not aware of approaches that provide diagnostic information to deal with the over-approximation due to data abstraction in workflow graphs. Existing approaches to check other notions of soundness such as *relaxed soundness* [9] or *weak soundness* [10] have exponential complexity.

The paper is structured as follows: After setting the preliminary notions, we introduce symbolic execution in Sect. 3 and show how the relationship ‘always-concurrent’ and the absence of deadlock can be decided. Then, we discuss the ‘sometimes-concurrent’ and ‘mutually-exclusive’ relationships and lack of synchronization in Sect. 4. In Sect. 5, we show how the diagnostic information provided by symbolic execution can be used to deal with the over-approximation that results from abstracting from data-based decisions.

The proofs are omitted in this version but are available in a technical report [11].

## 2 Preliminaries

In this section, we define preliminary notions which include workflow graphs and their soundness property.

### 2.1 Basic Notions

Let $U$ be a set. A *multi-set* over $U$ is a mapping $m : U \rightarrow \mathbb{N}$. We write $m[e]$ instead of $m(e)$. For two multi-sets $m_1, m_2$, and each $x \in U$, we have : $(m_1 + m_2)[x] = m_1[x] + m_2[x]$ and $(m_1 - m_2)[x] = m_1[x] - m_2[x]$. The *scalar product* is defined by $m_1 \otimes m_2 = \sum_{x \in U}(m_1[x] \times m_2[x])$. By abuse of notation, we sometimes use a set $X \subseteq U$ in a multi-set context by setting $X[x] = 1$ if $x \in X$ and $X[x] = 0$ otherwise.

A *directed graph* $G = (N, E)$ consists of a set $N$ of *nodes* and a set $E$ of ordered pairs $(s, t)$ of nodes, written $s \to t$. 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 null values. If $c$ maps $e \in E$ to an ordered pair $(s, t) \in N$, 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 $s = \text{null}$, then we say that $e$ is a *source* of the graph. If $t = \text{null}$, 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 $n\circ$. If $n$ has only one incoming edge $e$, $^\circ n$ denotes $e$ ($\circ n$ would denote $\{e\}$). If $n$ has only one outgoing edge $e'$, $n^\circ$ denotes $e'$.

A *path* $p = \langle x_0, ..., x_n \rangle$ *from* an element $x_0$ *to* an element $x_n$ *in* a graph $G = (N, E, c)$ is an alternating sequence of elements $x_i$ in $N$ and in $E$ such that, for any element $x_i \in E$ with $c(x_i) = (s_i, t_i)$, if $i \neq 0$ then $s_i = x_{i-1}$ and if $i \neq n$ then $t_i = x_{i+1}$. If $x$ is an element of a path $p$ we say that $p$ *contains* $x$. A path is *trivial*, if it is contains only one element. A *cycle* is a path $b = \langle x_0 \dots x_n \rangle$ such that $x_0 = x_n$ and $b$ is not trivial.

### 2.2 Workflow Graphs

A *workflow graph* $W = (N, E, c, l)$ consists of a multi-graph $G = (N, E, c)$ and a mapping $l : N \to \{\text{AND}, \text{XOR}, \text{IOR}\}$ that associates a *logic* with every node $n \in N$, such that: 1. The workflow graph 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 $n$. $W$ is *cyclic* if there exists a cycle in $W$.

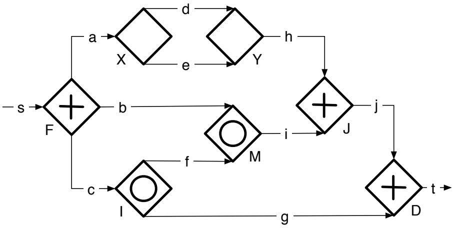


**Fig. 1.** A workflow graph

Figure 1 depicts an acyclic workflow graph. 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 single outgoing edge is called a *join*. For the sake of presentation simplicity, we use workflow graphs composed of only splits and joins.

Let, in the rest of this section, $W = (N, E, c, l)$ be an acyclic graph. Let $x_1, x_2 \in N \cup E$ be two elements of $W$ such that there is a path from $x_1$ to $x_2$. We then say that $x_1$ *precedes* $x_2$, denoted $x_1 < x_2$, and $x_2$ *follows* $x_1$. Two elements $x_1, x_2 \in N \cup E$ of $W$ are *unrelated*, denoted $x_1 \parallel x_2$, if $x_1 \neq x_2$ and neither $x_1 < x_2$ nor $x_2 < x_1$. A *prefix* of $W$ is a workflow graph $W' = (N', E', c', l')$ such that $N' \subseteq N$, for each pair of nodes $n_1, n_2 \in N$, if $n_2 \in N'$ and $n_1 < n_2$ then $n_1 \in N'$, an edge $e$ belongs to $E'$ if there exists a node $n \in N'$ such that $e$ is adjacent to $n$, for each node $n \in N'$, we have $l'(n) = l(n)$, and for each edge $e \in E'$, we have $c'(e) = (s', t')$, $c(e) = (s, t)$, $s' = s$, $t' = t$ if $t \in N'$, and $t' = \text{null}$ otherwise.

The semantics of workflow graphs is, similarly to Petri nets, defined as a token game. If $n$ has AND 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 elwhere [12], complies with the BPMN standard and BPEL's dead path elimination.

The outgoing edge or set of outgoing edges to which a token is added when executing a node with XOR or IOR logic is non-deterministic, by which we abstract from data-based or event-based decisions in the process. In the following, this semantics is defined formally.

A *marking* $m : E \to \mathbb{N}$ of a workflow graph with edges $E$ is a multi-set over $E$. When $m[e] = k$, we say that the edge *$e$ is marked with $k$ tokens* in $m$. When $m[e] > 0$, we say that the edge *$e$ is marked* in $m$. The *initial marking $m_s$* of $W$ is such that the source edge is marked with one token in $m_s$ and no other edge is marked in $m_s$.

Let $m$ and $m'$ be two markings of $W$. A tuple $(E_1, n, E_2)$ is called a *transition* if $n \in N$, $E_1 \subseteq \circ n$, and $E_2 \subseteq n\circ$. 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 any of the following propositions:

- $l(n) = \text{AND}$, $E_1 = \circ n$, and $E_2 = n\circ$.
- $l(n) = \text{XOR}$, there exists an edge $e$ such that $E_1 = \{e\}$, and there exists an edge $e'$ such that $E_2 = \{e'\}$.
- $l(n) = \text{IOR}$, $E_1, E_2$ are non-empty, $E_1 = \{e \in \circ n \mid m(e) > 0\}$, 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 $m' = m - E_1 + E_2$.

An *execution sequence* of $W$ is an alternate sequence $\sigma = \langle m_0, T_0, m_1, T_1 ... \rangle$ 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 $m_i$. An *execution* of $W$ is an execution sequence $\sigma = \langle m_0, ..., m_n \rangle$ of $W$ such that $n > 0$, $m_0 = m_s$ 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.

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 $m_s$.

### 2.3 Soundness

A deadlock occurs when a token stays 'blocked' on one edge of the workflow graph: A *deadlock* of $W$ is a reachable marking $m$ of $W$ such that 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 $W$ *contains* a deadlock if and only if there exists a reachable marking $m$ of $W$ such that $m$ is a deadlock. The workflow graph in Fig. 1 permits the execution $\sigma = \langle [s], [a, b, c], [b, c, d], [b, c, h], [b, f, h], [h, i], [j] \rangle$. The marking $[j]$ is a deadlock.

A *lack of synchronization* of $W$ is a reachable marking $m$ of $W$ such that 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 and only if there exists a reachable marking $m$ of $W$ such that $m$ is a lack of synchronization.

A workflow graph is *sound* if it contains neither a deadlock nor a lack of synchronization. Note that this notion of soundness is equivalent to the notion presented by van der Aalst [13] for workflow nets.

## 3 Symbolic Execution and Always-Concurrent Edges

In this section, we introduce *symbolic execution* and show how we use it to detect deadlocks and determine whether two edges are always-concurrent. We start by giving a characterization of deadlock, then introduce the symbols and the propagation rules of the symbols, we show how to compute a normal form of a symbol and discuss the complexity of the proposed technique.

Let, in this section, $W = (N, E, c, l)$ be an acyclic workflow graph prefix that is free of lack of synchronization. We describe in Sect. 4.3 how we determine such prefix.

### 3.1 Equivalence of Edges and a Characterization of Deadlock

A deadlock arises at an AND-join when one of its incoming edges $e$ is marked during an execution $\sigma$ but another edge $e'$ does not get marked during $\sigma$ because, as $e'$ never gets marked during $\sigma$, the AND-join cannot execute and the token marking $e$ is 'blocked'. Thus, in order to have no deadlock, the incoming edges of an AND-join need to get marked 'together' in each execution. We can precisely capture this through *edge equivalence* or the notion *always-concurrent*. In an acyclic workflow graph, only an AND-join can can '*cause*' a deadlock. An IOR-join can 'block' a token if and only if there exists a preceding node that blocks another token. Thus, whenever there is a deadlock in an acyclic workflow graph, there exists an AND-join with non-equivalent incoming edges. Nodes that are nor AND-join or IOR-join cannot block a token. To introduce edge equivalence, we define the *Parikh vector* of an execution, which records, for each edge, the number of tokens that are produced on that edge during the execution.

**Definition 1 (Parikh vector).** *The* Parikh vector *of an execution $\sigma = \langle m_0, T_0, \ldots\rangle$, written $\vec{\sigma}$, is the multi-set of edges such that $\vec{\sigma}[s] = 1$ for the source $s$ of $W$ and otherwise $\vec{\sigma}[e] = k$ such that $k = |\{i \mid T_i = (E, n, E') \wedge e \in E'\}|$.*

For example, given the execution $\sigma = \langle [s], (\{s\}, F, \{a, b, c\}), [a, b, c], (\{a\}, X, \{d\}), [b, c, d], (\{d\}, Y, \{h\}), [b, c, h], (\{c\}, I, \{f\}), [b, f, h]\ (\{b, f\}, M, \{i\}), [h, i], (\{h, i\}, J, \{j\}), [j]\rangle$ of the workflow graph of Fig. 1, we have $\vec{\sigma}[s] = \vec{\sigma}[a] = \vec{\sigma}[b] = \vec{\sigma}[c] = \vec{\sigma}[d] = \vec{\sigma}[f] = \vec{\sigma}[h] = \vec{\sigma}[i] = \vec{\sigma}[j] = 1$ and $\vec{\sigma}[e] = \vec{\sigma}[g] = 0$.

**Definition 2 (Edge equivalence, always-concurrent)**

- *Two edges are* parallel *in an execution $\sigma$ if there is a marking in $\sigma$ in which both edges are marked. Two executions $\sigma, \sigma'$ are* interleaving equivalent *if $\vec{\sigma} = \vec{\sigma}'$.*

*Two edges are* concurrent *in $\sigma$ if there is an execution $\sigma'$ such that $\sigma$ and $\sigma'$ are interleaving equivalent and the edges are parallel in $\sigma'$. Two edges are* always-concurrent *if they are concurrent in every execution of $W$.*

- *Two edges $e_1$ and $e_2$ of $W$ are* equivalent*, written $e_1 \equiv e_2$, if for any execution $\sigma$ of $W$, we have $\vec{\sigma}[e_1] = \vec{\sigma}[e_2]$.*

Two executions that are interleaving equivalent execute the same transitions; possibly in a different order. Note that these definitions are founded only for acyclic workflow graphs.

**Proposition 1.** *Two edges $e_1, e_2$ are always-concurrent iff $e_1 \equiv e_2$ and $e_1 \parallel e_2$.*

In the workflow graph depicted by Fig. 1, we have $a \equiv b \equiv h$ and $a \not\equiv d \not\equiv g$. Note that we have discussed earlier an execution of the workflow graph of Fig. 1 where $\vec{\sigma}[a] = \vec{\sigma}[d]$. However, there exist another execution such that $\vec{\sigma}[a] \neq \vec{\sigma}[d]$ and therefore $a \not\equiv d$. Moreover, $a$ is always-concurrent to $b$ but not to $h$.

**Proposition 2.** *$W$ contains a* deadlock *iff there exist two incoming edges of an AND-join of $W$ that are not equivalent, or equivalently, that are not always-concurrent.*

In the workflow graph depicted by Fig. 1, the edges $j$ and $g$ are not always-concurrent. Therefore, we get a deadlock at the AND-join $D$.

In the following, we show how we can compute edge equivalence and therefore also whether two edges are always-concurrent.

### 3.2 Symbolic Execution

The first step to compute edge equivalence is the symbolic execution of the workflow graph. During symbolic execution, each edge is labeled with a symbol, which is a set of *outcomes* of the workflow graph. An *outcome* is the source edge, an outgoing edge of an XOR-split, or an outgoing edge of an IOR-split in the graph. Figure 2 shows the labeling of the workflow graph of Fig. 1 that results from its symbolic execution.

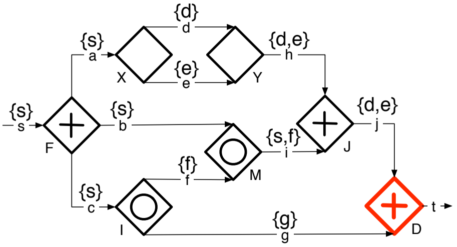


**Fig. 2.** The assignment resulting from the symbolic execution of the workflow graph of Fig. 1

The symbolic execution starts with labeling the source $s$ with $\{s\}$. All other edges are yet unlabeled. If all incoming edges of a node are labeled, we may label the outgoing edges of the node by applying one of the propagation rules depicted by Fig. 3, depending on the logic of the node.

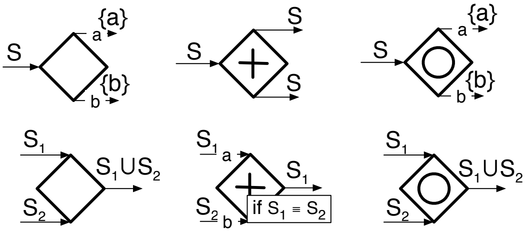


**Fig. 3.** The propagation rules

The intuition behind symbolic execution is to label an edge $e$ with a set $S$ of outcomes such that $e$ is marked during an execution $\sigma$ if and only if some of the outcomes in $S$ get marked during $\sigma$. In general, the label of the outgoing edges depends on the labels of the incoming edges. However, if the node is an XOR-split or an IOR-split, then the symbol that is assigned to one of the outgoing edges only contains that outgoing edge. The symbol associated to the incoming edge of the node is then ignored. In case of an AND-join, the propagation rule additionally requires the symbol labeling its incoming edges to be equivalent (which we will describe in Sect. 3.3) in order to be applied. The AND-join rule then chooses one of the labels of the incoming edges non-deterministically as the label for the outgoing edge. The symbol labeling an outgoing edge of a node that is an XOR-join or an IOR-join, is the union of the symbols labeling the incoming edges of the node. The symbolic execution terminates when there is no progation rule that can be applied. In the following, we define these propagation rules formally.

**Definition 3 (Symbolic execution).** *An* outcome *of $W$ is the source, an outgoing edge of some XOR-split, or an outgoing edge of some IOR-split of $W$. A* symbol *of $W$ is a set of outcomes of $W$. An* assignment *is a mapping $\varphi$ that assigns a symbol to each edge of some prefix of $W$. If $e$ is an edge of that prefix, we say that $e$ is* labeled under $\varphi$.

*For every node $n$ of a workflow graph, we describe the* propagation *by the node $n$ from an assignment $\varphi$ to an assignment $\varphi'$, written $\varphi \xrightarrow{n} \varphi'$. The propagation $\varphi \xrightarrow{n} \varphi'$ is* activated *when all the incoming edges of $n$ are labeled under $\varphi$ and no outgoing edge is labeled under $\varphi$. Additionally, if $n$ is an AND-join, the symbol associated to each incoming edges of $n$ must be equivalent (according to Def. 4) for the propagation to be activated. If $n$ is activated in $\varphi$, we have $\varphi \xrightarrow{n} \varphi'$ where $\varphi'$ is obtained as follows, for any edge $e$ of $W$:*

- *If $l(n) = AND$ and there exists an edge $e' \in \circ n$, then $\varphi'(e) = \varphi(e')$ for $e \in n\circ$ and $\varphi'(e) = \varphi(e)$ otherwise.*
- *If $n$ is an XOR-split or an IOR-split, then $\varphi'(e) = \{e\}$ for $e \in n\circ$ and $\varphi'(e) = \varphi(e)$ otherwise.*
- *If $n$ is an XOR-join or an IOR-join, for $\varphi'(e) = \bigcup_{e' \in \circ n} \varphi(e')$for $e \in n\circ$ and $\varphi'(e) = \varphi(e)$ otherwise.*

As said above, the propagation rules establish that an edge $e$ is marked during an execution $\sigma$ if and only if some of the outcomes in $\varphi(e)$ are marked during $\sigma$:

**Lemma 1.** *For any execution $\sigma$ of $W$ and any edge $e \in E$, $\vec{\sigma} \otimes \varphi(e) > 0 \Leftrightarrow \vec{\sigma}[e] > 0$.*

### 3.3 A Normal Form for Symbols

To detect a deadlock or to label the outgoing edge of an AND-join, we need to check edge equivalence. If two incoming edges of an AND-join are not equivalent, we have found a deadlock.

We will exploit that the equivalence of edges corresponds to an equivalence of the symbols they are labeled with. This symbol equivalence can be defined as follows:

**Definition 4 (Symbol equivalence).** *Two symbols $S_1, S_2$ are* equivalent *w.r.t. $W$, written $S_1 \equiv S_2$ if, for any execution $\sigma$ of $W$, $S_1 \otimes \vec{\sigma} = 0 \Leftrightarrow S_2 \otimes \vec{\sigma} = 0$.*

As $W$ is free of lack of synchronization, for any edge $e$ and for any execution $\sigma$, we have $\vec{\sigma}[e] = 1$ or $\vec{\sigma}[e] = 0$. Thus, given two edges $e_1, e_2$ labeled under $\varphi$, the edges $e_1$ and $e_2$ are equivalent if and only if the symbols $\varphi(e_1)$ and $\varphi(e_2)$ are equivalent.

We will decide the equivalence of two symbols by computing a normal form for each of them. The normal form of a symbol $S$ is the 'largest' set of outcomes that is equivalent to $A$. Two symbols are equivalent if and only if they have the same normal form. To show this, we define:

**Definition 5 (Maximal equivalent extension, Closure).** *Let $\varphi$ be an assignment of $W$ and $e$ be an edge such that $e$ is labeled under $\varphi$. Let $O$ be the set of outcomes of $W$ that are labeled under $\varphi$.*

– *A* maximal equivalent extension *of $\varphi(e)$ w.r.t. $\varphi$ is a set $\varphi^*(e) \subseteq O$ such that $\varphi^*(e) \equiv \varphi(e)$ and there exist no other set $S \subseteq O$ such that $\varphi^*(e) \subsetneq S$ and $S \equiv \varphi(e)$.*
– *The* closure *of $\varphi(e)$ w.r.t. $\varphi$ is the smallest set $\overline{\varphi}(e)$ such that $\varphi(e) \subseteq \overline{\varphi}(e)$ and for each XOR- or IOR-split $n$ such that $e'$ is labeled under $\varphi$ for each $e' \in n\circ$, we have $\varphi(^\circ n) \subseteq \overline{\varphi}(e)$ iff $n\circ \subseteq \overline{\varphi}(e)$.*

The existence of a maximal equivalent extension is clear. We can also show that it is unique.

**Lemma 2.** *Let $\varphi$ be an assignment of $W$ and $e$ an edge that is labeled under $\varphi$. Then $\varphi^*(e)$ is unique.*

It is also clear that the closure exists and is unique. Moreover, we can prove that the closure is equal to the maximal equivalent extension:

**Theorem 1.** *Let $\varphi$ be an assignment of $W$. For every edge $e$ that is labeled under $\varphi$, we have $\varphi^*(e) = \overline{\varphi}(e)$.*

That is, we obtain a unique normal form that is equivalent with a given label of an edge. We show in Sect. 3.4 that the closure can be computed in linear time. Thus, from the characterization as a closure, we can compute the normal form in linear time. Moreover, the normal form has the desired property:

**Theorem 2.** *$\overline{\varphi}(e) = \overline{\varphi}(e')$ whenever $e$ and $e'$ are equivalent.*

We are now able to compute the closure for the edges $g$, $h$, $i$, and $j$ of the example from Fig. 2. We have $\overline{\varphi}(g) = \{g\}$, $\overline{\varphi}(h) = \overline{\varphi}(i) = \overline{\varphi}(j) = \{s, d, e, f, g\}$. As $\overline{\varphi}(h) = \overline{\varphi}(i)$, $h$ and $i$ are equivalent. Thus, there is no deadlock at the AND-join $J$. On the contrary, $\overline{\varphi}(g)$ differs from $\overline{\varphi}(j)$ which implies, by Thm. 2, that $g$ and $j$ are not equivalent. Therefore, we detect a deadlock located at the AND-join $D$.

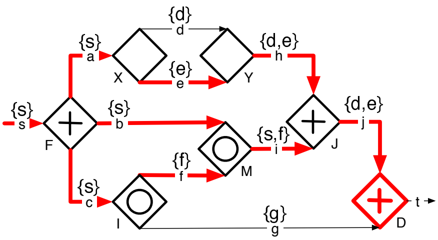


**Fig. 4.** Display of a deadlock

When we detect a deadlock because two incoming edges of an AND-join are not equivalent, we say that the AND-join is the *location* of the deadlock. To display the deadlock, we can, based on the assignment, generate in linear time an execution, called *error trace*, that exhibits the deadlock. Figure 4 depicts how we would display a deadlock: we highlight the location of the deadlock and the error trace, i.e., the edges marked during the execution leading to the deadlock. We discuss in Sect. 5 a form of diagnostic information and user interaction that goes beyond this error trace.

### 3.4 Complexity of the Computation

In this section, we first describe an approach to compute the closure in linear time and then discuss overall the complexity of symbolic execution.

Let, in this section, $\varphi$ be an assignment of $W$ and $D$ be the set of IOR-splits and XOR-splits of $W$ such that, for every node $d \in D$, every edge in $d\circ$ is labeled under $\varphi$. We define a *closure operation of* a node $d \in D$ *on* a symbol $S$ that changes $S$ *to* a symbol $S'$ such that $S' = S \cup \varphi(^\circ d) \cup d\circ$. A closure operation of a node $n$ changing $S$ to $S'$ is enabled when $\varphi(^\circ d) \in S$ or $d\circ \in S$ and $S \neq S'$. The computation of the closure comprises two phases:

1. We go through the nodes from the maximal to the minimal element in $D$ w.r.t. the precedence relation $<$, i.e., from the right most nodes in the graph to the left most nodes of the graph. For each node $n$, we execute the closure operation of $n$ if it is enabled.
2. We go through the nodes from the minimal to the maximal element in $D$ w.r.t. the precedence relation $<$. For each node $n$, we execute the closure operation of $n$ if it is enabled.

It is clear that this computation requires linear time. Note that $D$ must be sorted. This sorting is obtained once and for all before performing symbolic execution and requires linear time with respect to the size of the workflow graph. Moreover, we show that the this sequence of phases is sufficient to ensure completeness of the closure computation:

**Lemma 3.** *After performing phase 1 and phase 2 on a symbol $S$, there exists no node $n \in D$ such that a closure operation of $d$ is enabled on $S$.*

Symbolic execution needs just one traversal of the workflow graph. The closure is the most expensive operation. We have shown how to compute the closure of a symbol in linear time with respect to the size of $D$. Therefore, each transition takes at most linear time and the overall worst-case time complexity is quadratic.

## 4 Lack of Synchronization and Sometimes-Concurrent Edges

The workflow graph depicted in Fig. 5 permits the execution $\sigma = \langle [s], [a, b], [b, d], [d, e], [e, g], [g, g], \ldots\rangle$. The edge $g$ is marked by two tokens in the marking $[g, g]$. Thus, the workflow graph depicted by Fig. 5 contains a lack of synchronization. In this section, we describe an algorithm that detects lack of synchronization and sometimes-concurrent edges. The technique has quadratic time complexity.

We first give a characterization of lack of synchronization in terms of *handles* of the graph and then show how handles can be computed in quadratic time. We describe how to combine the symbolic execution and handle detection to detect control-flow errors. Finally, we show how to compute whether two edges are sometimes-concurrent, which has separate applications such as data-flow analysis.

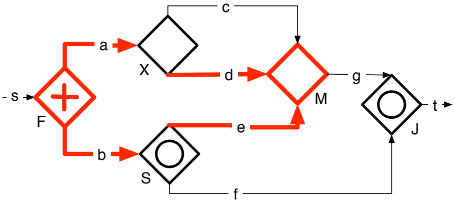


**Fig. 5.** A workflow graph that contains a lack of synchronization

### 4.1 Handles and Lack of Synchronization

To characterize lack of synchronization, we follow the intuition that paths starting with an IOR-split or an AND-split, should not be joined by an XOR-join. In the following, we formalize this characterization and show that such structure always leads to a lack of synchronization in deadlock-free acyclic workflow graphs.

**Definition 6 (Path with an AND-XOR or an IOR-XOR handle).** *Let $p_1 = \langle n_0, ..., n_i \rangle$ and $p_2 = \langle n'_0, ..., n'_j \rangle$ be two paths in a workflow graph $W = (N, E, c, l)$.*

*The paths $p_1$ and $p_2$ form a* path with a handle[1] *if $p_1$ is not trivial, $p_1 \cap p_2 = \{n_0, n_i\}$, $n_0 = n'_0$, and $n_i = n'_j$. We say that $p_1$ and $p_2$ form a path with a handle* from $n_0$ to $n_i$. *We speak of a* path with an IOR-XOR handle *if $n_0$ is an IOR-split and $n_i$ is an XOR-join. We speak of a* path with an AND-XOR handle *if $n_0$ is an AND-split, and $n_i$ is an XOR-join. In the rest of this document, we use handle instead of path with an AND-XOR handle or path with an IOR-XOR handle. The node $n_0$ is the* start node *of the handle and the node $n_i$ is the* end node *of the handle.*

**Theorem 3.** *In an acyclic workflow graph that contains no deadlock, there is a lack of synchronization iff there is a handle.*

The outline of the 'only if' direction of the proof of Thm. 3 is that, whenever there is a handle, this handle can be 'executed' in the sense that there exists an execution such that a token reaches the incoming edge of the start node of the handle and then two tokens can be propagated to reach two incoming edges of the end node of the handle to create a lack of synchronization. We believe that, due to its direct relationship with an erroneous execution, the handle is an adequate error message for the process modeler. In Fig. 5, the handle corresponding to the lack of synchronization is highlighted. We say that the end node of the handle is the *location* of the lack of synchronization. Note that it is necessary that the workflow graph is deadlock-free in order to show that the handle can be executed and thus a lack of synchronization be observed. However, even if the workflow graph contains a deadlock, a handle is a design error because, once the deadlock is fixed, the handle can be executed and a lack of synchronization can be observed.

[1] Strictly speaking, one path is the handle of the other path and vice versa.

Our notion of handles is similar to the one of Esparza and Silva [14] for Petri nets. If we restrict ourselves to workflow graphs without IOR gateways, one of the directions of our characterization follows from a result of Esparza and Silva [14]. The converse direction does not directly follow. Our notion of handles has been described by van der Aalst [13] who shows that, given a Petri net $N$, the absence of some type of handle in $N$ is a sufficient condition to the existence of an initial marking $i$ of $N$ such that $(N, i)$ is sound. He points out that path with handles can be computed using a maximum flow approach. Various algorithms exist to compute the maximum flow (see [15] for a list). The complexity of these algorithms ranges between $O(|N| \cdot |E|^2)$ and $O(|N| \cdot |E| \cdot log(|N|))$. The existence of a handle can be checked by applying a maximum flow algorithm to each pair of transition and place of the net. Therefore, the complexity of detecting handles with such an approach is at best $O(|N|^3 \cdot |E| \cdot log(|N|)$.

### 4.2 Computing Handles

Given an acyclic directed graph $G = (N, E)$ and four different nodes $s_1, s_2, t_1, t_2 \in N$, Perl and Shiloach [16] show how to detect two node-disjoint paths from $s_1$ to $t_1$ and from $s_2$ to $t_2$ in $O(|N|\cdot|E|)$. We extend their algorithm in order to detect two edge-disjoint paths between two nodes of an acyclic workflow graph. We sketch our extension here while the details can be found in a separate report [17].

Perl and Shiloach [16] describe how to detect two node-disjoint paths in a directed graph whereas we want to detect two edge-disjoint paths in a workflow graph which is a directed multi-graph. To do so, we transform the workflow graph into its *line graph*. A line graph $G'$ of a graph $G$ represents the adjacency between edges of $G$. Each edge of $G$ becomes a node of $G'$. Additionally, we carry over those nodes from $G$ to $G'$ that can be start or end nodes of a handle, i.e., $S = \{x \mid x \in N \wedge$ $x$ is an AND-split or an IOR-split$\}$ and $T = \{x \mid x \in N \wedge$ x is an XOR-join$\}$. The edges of $G'$ are such that the adjacency in $G$ is reflected in $G'$. For the workflow graph in Fig. 5, we obtain the line graph shown in Fig. 6. The line graph has two node-disjoint paths from an AND- or IOR-split to an XOR-join if and only if the workflow graph has a handle from that split to that join.

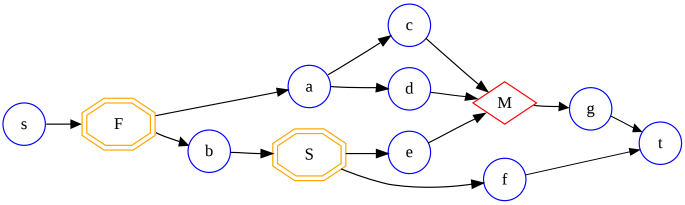


**Fig. 6.** The line graph for the workflow graph in Fig. 5

To decide whether there are such two node-disjoint paths in the line graph, we can now apply the approach by Perl and Shiloach [16], which is the construction of a graph that we call the *state graph*. To this end, we extend the partial ordering $<$ of the nodes in the line graph to a total ordering $\prec$. A node of the state graph is a pair $(n, m)$ of nodes of the line graph such that either $n = m \in S \cup T$ or $n \neq m$ and $n \prec m$. There is an edge in the state graph from $(n, m)$ to $(n', m)$ (or to $(m, n')$) if there is an edge from $n$ to $n'$ in the line graph.

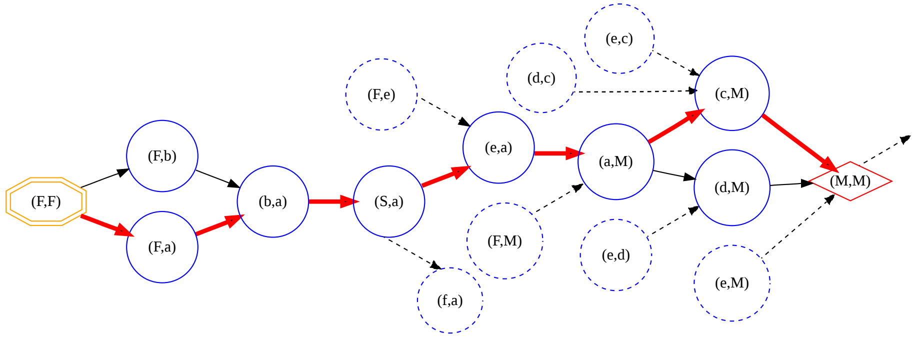


**Fig. 7.** A portion of the state graph for the line graph in Fig. 6

Figure 7 depicts a portion of the state graph for the line line graph in Fig. 6. We have two node-disjoint paths from an AND- or IOR-split $s$ to an XOR-join $j$ in the line graph if and only if there is a path from $(s, s)$ to $(j, j)$ in the state graph. In Fig. 7, one such path is highlighted which indicates two disjoint paths from the AND-split $F$ to the XOR-join $M$. The number of edges in the state graph is in $O(|N| \cdot |E|)$ and the number of nodes is in $O(|N|^2)$ in terms of the line graph [16]. The entire algorithm can be implemented to run in quadratic time in the size of the workflow graph, cf. [17].

### 4.3 Combining Symbolic Execution with Handle Detection

Symbolic execution detects deadlocks in a prefix of the workflow graph that is free of lack of synchronization. Therefore, we first check the workflow graph for handles. We use the end nodes of the handles to delimit a maximum prefix of the workflow graph that is free of handles. We perform a symbolic execution of this prefix. If a deadlock is detected, we report the deadlock. If symbolic execution labels the incoming edges of the end node of a handle, we report the corresponding lack of synchronization. If no deadlock is detected and there is no handle detected, the workflow graph is sound.

### 4.4 Sometimes-Concurrent

A data-flow hazard may arise if two conflicting operations on the same data object are executed concurrently. This can happen only if the tasks containing the data operations are sometimes-concurrent. A task of a process is represented as an edge in the corresponding workflow graph. Thus for the purpose of data-flow analysis, we are interested in detecting sometimes-concurrent edges for data-flow analysis.

**Definition 7.** *Two edges are* sometimes-concurrent *if there exists an execution in which they are parallel. They are* mutually-exclusive *or* never-concurrent *if they are not sometimes-concurrent.*

The notion of sometimes-concurrent edges is tightly related to lack of synchronization: It follows from the proof of Thm. 3 that two incoming edges $e, e'$ of an XOR-join are sometimes-concurrent if and only if there is handle to this XOR-join such that one path goes through $e$ and the other goes through $e'$. To decide whether two arbitrary edges of a sound graph are sometimes-concurrent, we show the following:

**Lemma 4.** *In a sound prefix of the workflow graph W, if two edges $e_1, e_2$ are sometimes-concurrent, then $e_1 \parallel e_2$.*

Lemma 4 can be proven by contradiction: Without loss of generality, assume that $e_1 < e_2$. As $e_1$ and $e_2$ are sometimes-concurrent, there exists a reachable marking $m$ such that $m[e_1] = m[e_2] = 1$. As there is no deadlock, we can move the token on $e_1$ on the path to $e_2$ until reaching a marking $m'$ such that $m'[e_2] = 2$. The marking $m'$ is a lack of synchronization which is ruled out by the soundness assumption.

We can now determine whether two edges are sometimes-concurrent: Let $W^*$ be the graph obtained by removing all the elements of the workflow graph that follow either $e_1$ or $e_2$ and add an XOR-join $x$ to be the target of $e_1$ and $e_2$. The edges $e_1$ and $e_2$ are sometimes-concurrent if and only if $x$ is the end node of a handle in $W^*$. We obtain:

**Theorem 4.** *It can be decided in quadratic time in the size of the workflow graph whether a given pair of edges is sometimes-concurrent.*

Kovalyov and Esparza [18], propose a technique to detect sometimes-concurrent edges for sound workflow graphs that do not contain IOR logic in cubic time.

# 5 Dealing with Over-Approximation

In this section, we show how the labeling that is computed in the symbolic execution can be leveraged to deal with errors that are detected in the workflow graph but may not arise in a real execution of the process due to the correlation of data-based decisions.

## 5.1 User Interaction to Deal with Over-Approximation

When we capture the control-flow of a process in a workflow graph, we abstract from the data-based conditions that are evaluated when executing an XOR-split or an IOR-split of the process. Such a data-based decision can be, for example, isGoldCustomer(*client*). The data-abstraction may result in errors that occur in the workflow graph but not in an *actual* execution of the process. We use in the following the term *actual execution* to refer to an execution of the real process as opposed to its workflow graph, which is an abstraction of the process.

For example, the graph in Fig. 8 contains a deadlock located at *J*. However, if the data-based decisions in all actual executions are such that outcome *d* is taken whenever *e* is taken, this deadlock would never occur in an actual execution. For example, the data-based condition on *d* could be exactly the same as on *e*. The user should therefore have the opportunity to inspect the deadlock and decide whether outcomes *d* and *e* are related as mentioned above and then dismiss the deadlock. Analysis of the graph should then continue.

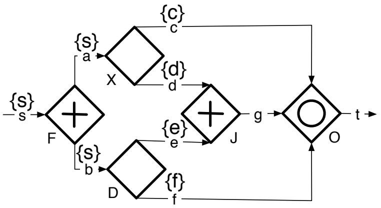


**Fig. 8.** A deadlock

To inspect a deadlock, we provide the AND-join, two incoming edges $e, e'$ of the join, and their non-equivalent labels $\varphi(e), \varphi(e')$ to the user. Then, she has to decide whether for each outcome $o \in \varphi(e)$ and each actual execution where $o$ is taken, there is an outcome $o' \in \varphi(e')$ that is also taken in that execution and vice versa. If the user affirms the latter, she can dismiss the deadlock. This basically postulates the equivalence of the two symbols in actual executions. Henceforth, we continue the symbolic execution by treating, internally to the analysis, the AND-join as an IOR-join.

To inspect a lack of synchronization, we provide the XOR-join that terminates the detected handle and the two incoming edges $e, e'$ of the XOR-join that are part of the handle to the user. Furthermore, we provide the labels $\varphi(e), \varphi(e')$. Then, the user has to determine that for each pair of outcomes $o \in \varphi(e)$ and $o' \in \varphi(e')$, we have that $o$ is taken in an actual execution implies that $o'$ is not taken in that execution. If the user affirms the latter, she can dismiss the lack of synchronization. This basically postulates that $o$ and $o'$ are mutually-exclusive in actual executions. If this is done for all incoming edges of the XOR-join, we can henceforth continue the symbolic execution by treating, internally to the analysis, the XOR-join as an IOR-join. Figure 9 shows an example with a lack of synchronization located at $J$. The user may dismiss it because for example, the conditions on $c$ and $e$ are the same, i.e., $d$ and $e$ are mutually-exclusive.

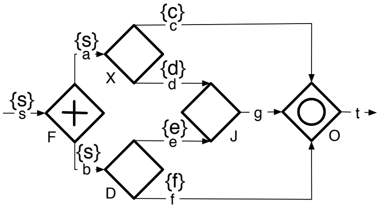


**Fig. 9.** A lack of synchronization

Figure 10 shows another example where the deadlock can be dismissed if $b$ and $c$ are deemed to be equivalent. Once the user dismissed the deadlock, we continue the symbolic execution and label the edge $d$ with the symbol $\{b, c\}$ according to the IOR-join propagation rule. To dismiss the lack of synchronization at $M$, the user then has to check the pair $a, b$ and the pair $a, c$ for mutual exclusion.

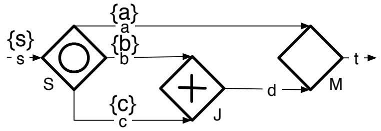


**Fig. 10.** A deadlock and a lack of synchronization

The deadlock displayed on Fig. 4, can be dismissed if $g$ is equivalent to $s$, i.e., $g$ is deemed to be marked in every execution of the process.

Note that, if we provided an execution, i.e., an error trace, rather than the symbolic information to dismiss an error, we would present exponentially many executions that contain the same error in the worst case. The analysis of the outcome sets precisely gives the conditions under which one deadlock or one lack of synchronization occurs. It does not contain information that is irrelevant for producing the error.

### 5.2 Relaxed Soundness

In some cases, the user should not be allowed to dismiss an error. Figure 11 shows a deadlock that cannot be avoided unless $d$ and $e$ are never taken which clearly indicates a modeling error. This is related to the notion of *relaxed soundness* [9]. A workflow graph is *relaxed sound* if for every edge $e$, there is a *sound execution* that marks $e$, where an execution is *sound* if it neither reaches a deadlock nor a lack of synchronization.

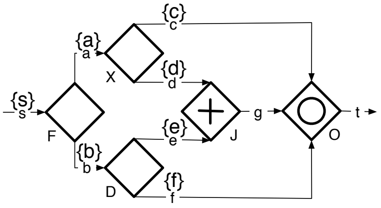


**Fig. 11.** A deadlock located at $J$ that should not be dismissed

The graph in Fig. 11 is not relaxed sound. We do not know any polynomial-time algorithm to decide relaxed soundness for acyclic workflow graphs. However, we provide here necessary conditions for relaxed soundness that can be checked in polynomial time.

One necessary condition for relaxed soundness is that for every AND-join $j$ and every pair of incoming edges $e, e'$ of $j$, $e$ and $e'$ are sometimes-concurrent. Likewise, for every XOR-join $j$ and every pair of incoming edges $e, e'$ of $j$, $e$ and $e'$ must not be always-concurrent. Moreover, we have the following stronger necessary conditions:

**Theorem 5.** *Let $W$ be an acyclic workflow graph.*

- *If for an AND-join $j$, and a pair of incoming edges $e, e'$ of $j$ and one outcome $o \in \varphi(e)$, we have that all outcomes $o' \in \varphi(e')$ are mutually-exclusive with $o$, then $W$ is not relaxed sound.*
- *If for an XOR-join $j$, and a pair of incoming edges $e, e'$ of $j$, we have $\overline{\varphi}(e) \cap \overline{\varphi}(e') \neq \emptyset$, then $W$ is not relaxed sound.*

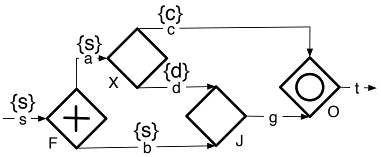


**Fig. 12.** A lack of synchronization that should not be dismissed

Based on the previous results in this paper, we can compute these necessary conditions for relaxed soundness in polynomial time. If one of them is true, the corresponding error should not be dismissible. For example, the deadlock in the workflow graph depicted by Fig. 11 cannot be dismissed because $d$ and $e$ are mutually-exclusive. The lack of synchronization located at $J$ in the workflow graph depicted by Fig. 12 cannot be dismissed because $\overline{\varphi}(d) = \{d\}$ and $\overline{\varphi}(b) = \{s, c, d\}$ and thus $\overline{\varphi}(e) \cap \overline{\varphi}(e') \neq \emptyset$.

Note that, deciding soundness and relaxed soundness complement each other. If we only decided relaxed soundness, we would not detect the deadlock that may be present in an actual execution of Fig. 8 for example.

## 6 Conclusion

We have shown how basic relationships between control-flow edges of a process can be decided in polynomial time for acyclic workflow graphs with inclusive OR gateways. This has various applications, for example, to detect control-flow errors, to perform data-flow analysis, or to compare processes at a behavioral level. Moreover, we have proposed a control-flow analysis that decides soundness in quadratic time and gives concise error information that precisely characterizes a single error. We outlined how the diagnostic information can be used to efficiently dismiss spurious errors that may not occur in actual executions of the process due to correlated data-based decisions.

Note that, to increase the applicability of this approach, we can combine it with workflow graph parsing using the Refined Process Structure Tree [19], which allows us to decompose the workflow graph into fragments and to analyze each fragment in isolation (see [5] for details). Thus, our approach can be used to analyze every acyclic fragment of a cyclic workflow graph. However, it has to be worked out how the user interaction proposed in Sect. 5 can be extended to that class. Some cyclic fragments can be analyzed using suitable heuristics [20] which can be applied in linear time. Moreover, we would like to extend symbolic execution to cyclic workflow graphs in future work.

## References

1. Mendling, J.: Empirical Studies in Process Model Verification. T. Petri Nets and Other Models of Concurrency (ToPNoC) 2, 208–224 (2009)
2. Weidlich, M., Polyvyanyy, A., Mendling, J., Weske, M.: Efficient Computation of Causal Behavioural Profiles using Structural Decomposition. Technical Report BPT 10, HPI (2010)
3. Desel, J., Esparza, J.: Free choice Petri nets. Cambridge tracts in theoretical computer science, vol. 40. Cambridge University Press, Cambridge (1995)
4. Esparza, J.: Reduction and synthesis of live and bounded free choice Petri nets. Information and Computation 114(1), 50–87 (1994)
5. 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. LNCS, vol. 5701, pp. 278–293. Springer, Heidelberg (2009)
6. Sadiq, W., Orlowska, M.E.: Analyzing process models using graph reduction techniques. Inf. Syst. 25(2), 117–134 (2000)
7. Mendling, J.: Detection and Prediction of Errors in EPC Business Process Models. PhD thesis, Vienna University of Economics and Business Administration, Vienna, Austria (2007)
8. Wynn, M., Verbeek, H., Aalst, W., Hofstede, A., Edmond, D.: Business process verification-finally a reality! Business Process Management Journal 15(1), 74–92 (2009)
9. Dehnert, J., Rittgen, P.: Relaxed soundness of business processes. In: Dittrich, K.R., Geppert, A., Norrie, M.C. (eds.) CAiSE 2001. LNCS, vol. 2068, pp. 157–170. Springer, Heidelberg (2001)
10. Martens, A.: On compatibility of web services. Petri Net Newsletter 65, 12–20 (2003)
11. Favre, C., Völzer, H.: Symbolic execution of acyclic workflow graphs. Technical Report RZ3780, IBM Research (2010)
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. 285–300. Springer, Heidelberg (2010)
13. van der Aalst, W.: Workflow verification: Finding control-flow errors using Petri-net-based techniques. In: van der Aalst, W., Desel, J., Oberweis, A. (eds.) BPM 2000. LNCS, vol. 1806, pp. 161–183. Springer, Heidelberg (2000)
14. Esparza, J., Silva, M.: Circuits, handles, bridges and nets. Advances in Petri nets 483, 210–242 (1990)
15. Goldberg, A.V., Tarjan, R.E.: A new approach to the maximum-flow problem. J. ACM 35(4), 921–940 (1988)
16. Pearl, Y., Shiloach, Y.: Finding two disjoint paths between two pairs of vertices in a graph. Journal of the ACM (JACM) 25(1), 1–9 (1978)
17. Favre, C.: An efficient approach to detect lack of synchronization in acyclic workflow graphs. In: ZEUS. CEUR Workshop Proceedings, vol. 563, pp. 57–64 (2010)
18. Kovalyov, A., Esparza, J.: A Polynomial Algorithm to Compute the Concurrency Relation of Free-Choice Signal Transition Graphs. In: Proc. of the International Workshop on Discrete Event Systems, WODES 1996, The Institution of Electrical Engineers, Edinburgh, pp. 1–6 (1996)
19. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. Data Knowl. Eng. 68(9), 793–818 (2009)
20. Vanhatalo, J.: Process Structure Trees: Decomposing a Business Process Model into a Hierarchy of Single-Entry-Single-Exit Fragments. PhD thesis, Universität Stuttgart (2009)

# Structuring Acyclic Process Models

Artem Polyvyanyy[1], Luciano García-Bañuelos[2], and Marlon Dumas[2]

[1] Hasso Plattner Institute at the University of Potsdam, Germany
Artem.Polyvyanyy@hpi.uni-potsdam.de
[2] Institute of Computer Science, University of Tartu, Estonia
{luciano.garcia,marlon.dumas}@ut.ee

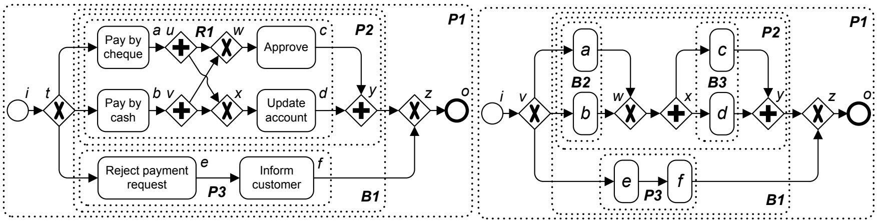
**Abstract.** This paper addresses the problem of transforming a process model with an arbitrary topology into an equivalent well-structured process model. While this problem has received significant attention, there is still no full characterization of the class of unstructured process models that can be transformed into well-structured ones, nor an automated method to structure any process model that belongs to this class. This paper fills this gap in the context of acyclic process models. The paper defines a necessary and sufficient condition for an unstructured process model to have an equivalent structured model under fully concurrent bisimulation, as well as a complete structuring method.

## 1 Introduction

In the Business Process Modeling Notation (BPMN) and in similar notations, a process model is composed of nodes (e.g., tasks, events, gateways) connected by a "flow" relation. Although BPMN allows process models to have almost any topology, it is often preferable that process models follow some structure. In this respect, a well-known property of process models is that of *(well-)structuredness* [1], meaning that for every node with multiple outgoing arcs (a *split*) there is a corresponding node with multiple incoming arcs (a *join*), such that the set of nodes between the split and the join form a single-entry-single-exit (SESE) region. For example, Fig. 1(a) shows an unstructured process model, while Fig. 1(b) shows an equivalent structured model. Note that Fig. 1(b) uses short-names for tasks (a, b, c . . . ), which appear next to each task in Fig. 1(a).

This paper studies the problem of automatically transforming process models with arbitrary topology into equivalent well-structured models. The motivation for such a transformation is manifold. Firstly, it has been empirically shown that structured process models are easier to comprehend and less error-prone than unstructured ones [2]. Thus, a transformation from unstructured to structured process model can be used as a refactoring technique to increase process model understandability. Secondly, a number of existing process model analysis techniques only work for structured models. For example, a method for calculating cycle time and capacity requirements of structured process models is outlined in [3], while a method for analyzing time constraints in structured process models is presented in [4]. By transforming unstructured process models to structured

R. Hull, J. Mendling, and S. Tai (Eds.): BPM 2010, LNCS 6336, pp. 276–293, 2010.



(a) (b)

**Fig. 1.** Unstructured process model and its equivalent structured version

ones, we can extend the applicability of these techniques to a larger class of models. Thirdly, a transformation from unstructured to structured process models can be used to implement converters from graph-oriented process modeling languages to structured process modeling languages, e.g., BPMN-to-BPEL.

In the context of flowcharts, without parallel splits and joins, it has been shown that any unstructured flowchart can be transformed into a structured one [5]. If we add parallel splits and joins, this result no longer holds: There exist unstructured process models that do not have equivalent structured ones [1]. Several authors have attempted to classify the sources of unstructuredness in process models [6,7,8] and to define automated methods for structuring process models [9,10,11]. However, these methods are incomplete: There is currently no full characterization of the class of *inherently* unstructured process models, i.e., unstructured process models that have no equivalent structured model. Also, none of the existing structuring methods is complete. In fact, this problem has not been fully solved even for acyclic process models. This paper fills this gap.

To streamline the presentation, we make several assumptions. Firstly, we consider process models composed of nodes (tasks, events, gateways) and control flow relations. In terms of BPMN, this means that we abstract away from other process model elements such as artifacts, annotations, associations, groups, pools, lanes, message flows, sub-process invocations and attributes associated to sub-process invocations, e.g., repetition. Nonetheless, the proposed method is applicable even if these types of elements are present in the input model. Simply, these ancillary elements and attributes need to be moved along with the tasks or events to which they are attached. In the same vein, we do not distinguish between events and tasks since, for the purpose of the transformation, both of these elements are treated equally. Secondly, we consider only sound process models [12]. This restriction is natural since soundness is a widely-accepted correctness criterion for process models. Thirdly, we consider process models in which every node has only one incoming or one outgoing arc. This restriction is merely syntactical because one can trivially split a node with multiple incoming and multiple outgoing arcs into two nodes: one node with a single outgoing arc and the other with a single incoming arc. Fourthly, we consider models with only one start node and one end node. Again, this is not a restriction since every sound model with multiple end nodes can be transformed into an equivalent sound model with a unique end node [12]. The reverse technique can be applied to models with multiple start nodes. Finally, we do not deal with the following BPMN constructs: OR gateways, complex gateways, error events and non-interrupting events. Lifting this latter restriction is left as future work.

The next section presents a taxonomy of (unstructured) process components in process models and reviews related work. Next, Sect.3 introduces the formalism used to represent process models. Sect.4 then introduces the behavioral equivalence used in this paper, viz. fully concurrent bisimulation (FCB), and shows that two acyclic process models are equivalent under this equivalence notion iff they have the same set of ordering relations. This result is used in Sect.5 to characterize the class of acyclic process components that can be structured and to define a structuring algorithm. Finally, Sect.6 concludes the paper.

## 2 Background and Related Work

This section discusses a complete taxonomy of process components. Next, we analyze previous work with respect to the proposed taxonomy.

### 2.1 Taxonomy of Process Components

The Refined Process Structure Tree (RPST) [13] is a technique to decompose a process model into a tree of regions. Each node in the RPST maps to a SESE region, herewith called a *process component*. A component in the RPST contains all components at the lower level, and all components at a given level are disjoint.

Each component in the RPST can be classified into one out of four classes [14]: A *trivial* ($T$) component consists of a single flow arc. A *polygon* ($P$) represents a sequence of components. A *bond* ($B$) stands for a set of components that share two common nodes. Any other component is a *rigid* ($R$). Rigid components explicitly define what makes a process model unstructured.

Fig.1 exemplifies the RPST decomposition in the form of dotted boxes. For instance in Fig.1(a), polygon $P1$ is the root of the RPST and corresponds to the whole process model. Polygon $P1$ contains bond $B1$ that, in turn, contains polygons $P2$ and $P3$. Observe that trivial components and polygons that are composed of two flow arcs are not visualized for simplicity reasons.

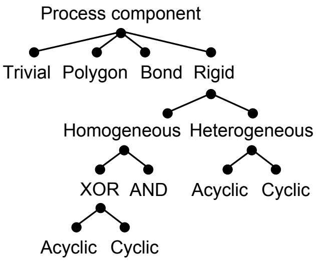


**Fig. 2.** Taxonomy

Trivials, polygons, and bonds are structured process components. If one could transform each rigid component in the RPST into an equivalent structured component, the entire model could be structured by traversing the RPST bottom-up and replacing each rigid by its equivalent structured component. Accordingly, the rest of the paper focuses on structuring rigid components.

The methods for structuring rigid components differ depending on the types of gateways present in the rigid and whether the rigid contains cycles or not. We classify rigids as follows. A homogeneous rigid contains either only *xor* or only *and* gateways. We call these rigids *(homogeneous) and rigids* and *(homogeneous) xor rigids*, respectively. A heterogeneous rigid contains a mixture of *and*/*xor* gateways. Heterogeneous and homogeneous *xor* rigids are further classified into cyclic, if they contain at least one cycle, or acyclic. Importantly, a safe process model cannot contain homogeneous *and* rigids with cycles. Upon this background, a taxonomy of process components is provided in Fig.2.

### 2.2 Related Work

The problem of structuring process models is relevant in the context of designing BPMN-to-BPEL transformations. However, BPMN-to-BPEL transformations such as [11] treat rigids as black-boxes that are translated using BPEL links or event handlers, rather than seeking to structure them. In this sense, the present contribution is complementary to this previous work.

A large body of work on flowcharts and GOTO program transformation [5], has addressed the problem of structuring *xor* rigids. In some cases, these transformations introduce additional boolean variables in order to encode part of the control flow, while in other cases they require certain nodes to be duplicated.

In [1], the authors show that not all acyclic *and* rigids can be structured. They do so by providing one counter-example, but do not give a full characterization of the class of models that can be structured nor do they define any automated transformation. Instead, they explore some causes of unstructuredness. In a similar vein, [6] presents a taxonomy of unstructuredness in process models, covering cyclic and acyclic rigids. But the taxonomy is incomplete, i.e., it does not cover all possible cases of models that can be structured. Also, the authors do not define an automated structuring algorithm.

In [7], the authors outline a classification of process components using *region trees*, a predecessor of the RPST. However, the authors do not provide a complete structuring method for acyclic heterogeneous rigids, e.g., the one in Fig.1(a). A similar remark applies to [10]. Meanwhile, [9] proposes a method for restructuring *xor* rigids based on GOTO program transformations, and extends this method to process graphs where such *xor* rigids are nested inside bonds. However, this method cannot deal with *and* rigids nor heterogeneous rigids.

## 3 Preliminaries

Below we introduce the notations used subsequently to represent process models.

### 3.1 Petri Nets

Petri nets are a well-known formalism for modeling concurrent systems. Below we present standard definitions of Petri nets and their semantics.

**Definition 1 (Petri net).** A *Petri net*, or a *net*, is a tuple $N = (P, T, F)$, with $P$ and $T$ as finite disjoint sets of *places* and *transitions*, and $F \subseteq (P \times T) \cup (T \times P)$ as the *flow* relation.

We identify $F$ with its characteristic function on the set $(P{\times}T)\cup(T{\times}P)$. We write $X = (P \cup T)$ for all nodes of a net. For a node $x \in X$, $\bullet x = \{y \in X \mid F(y,x) = 1\}$ and $x\bullet = \{y \in X \mid F(x,y) = 1\}$. A node $x \in X$ is an *input* (*output*) node of a node $y \in X$, iff $x \in \bullet y$ ($x \in y\bullet$). For $Y \subseteq X$, $\bullet Y = \bigcup_{y \in Y} \bullet y$ and $Y\bullet = \bigcup_{y \in Y} y\bullet$. We denote by $F^+$ and $F^*$ irreflexive and, respectively, reflexive transitive closures of $F$.

**Definition 2 (Net semantics).** Let $N = (P,T,F)$ be a net.

- $M : P \to \mathbb{N}_0$ is a *marking* of $N$, where $M(p)$, $p \in P$, returns the number of *tokens* in place $p$. $[p]$ denotes the marking when place $p$ contains just one token and all other places contain no tokens.
- For any transition $t \in T$ and for any marking $M$ of $N$, $t$ is *enabled* in $M$, denoted by $(N,M)[t\rangle$, iff $\forall p \in \bullet t : M(p) \geq 1$.
- If $t \in T$ is enabled in $M$, then it can *fire*, which leads to a new marking $M'$, denoted by $(N,M)[t\rangle(N,M')$. The new marking $M'$ is defined by $M'(p) = M(p) - F(p,t) + F(t,p)$, for each place $p \in P$.
- A sequence of transitions $\sigma = t_1 \dots t_n$, $n \in \mathbb{N}$, is a *firing sequence*, iff there exist markings $M_0 \dots M_n$, such that for all $1 \leq i \leq n$ holds $(N,M_{i-1})[t_i\rangle(N,M_i)$.
- For any two markings $M$ and $M'$ of $N$, $M'$ is *reachable* from $M$ in $N$, denoted by $M' \in [N,M\rangle$, iff there exists a firing sequence $\sigma$ leading from $M$ to $M'$.
- A *net system*, or a *system*, is a pair $(N,M_0)$, where $N$ is a net and $M_0$ is a marking of $N$. $M_0$ is called the *initial marking* of $N$.

Workflow (WF-)nets [15] are a subclass of Petri nets specifically designed to represent business process models. A WF-net is a net with two special places: one to mark the start and the other the end of a workflow execution.

**Definition 3 (WF-net, Short-circuit net, WF-system).** A Petri net $N = (P,T,F)$ is a *workflow net*, or a *WF-net*, iff $N$ has a dedicated *source* place $i \in P$, with $\bullet i = \emptyset$, $N$ has a dedicated *sink* place $o \in P$, with $o\bullet = \emptyset$, and the *short-circuit* net $N' = (P, T \cup \{t^\star\}, F \cup \{(o,t^\star),(t^\star,i)\})$, $t^\star \notin T$, of $N$ is strongly connected. A *WF-system* is a pair $(N,M_i)$, where $M_i = [i]$.

Soundness and safeness are basic properties of WF-systems [15]. Soundness states that every execution of a WF-system ends with a token in the sink place, and once a token reaches the sink place, no other tokens remain in the net. Safeness refers to the fact that there is never more than one token in the same place.

In the rest of the paper, we also use three structural subclasses of Petri nets (free-choice net, occurrence net, and causal net), as well as labeled Petri nets to distinguish observable and silent transitions. These are defined below.

**Definition 4 (Free-choice net, Occurrence net, Causal net).** A Petri net $N = (P,T,F)$ is a *free-choice* net, iff $\forall p \in P, |p\bullet| > 1 : \bullet(p\bullet) = \{p\}$. Let $N = (P,T,F)$ be a net such that $\forall x,y \in P \cup T : (x,y) \in F^+ \Rightarrow (y,x) \notin F^+$.

- Net $N$ is an *occurrence net*, iff $\forall p \in P : |\bullet p| \leq 1$.
- Net $N$ is a *causal net*, iff $\forall p \in P : |\bullet p| \leq 1 \wedge |p\bullet| \leq 1$.

**Definition 5 (Labeled net).** A *labeled* net is a tuple $N = (P,T,F,\mathcal{T},\lambda)$, where $(P,T,F)$ is a net, $\mathcal{T}$ is a set of *labels*, such that $\tau \in \mathcal{T}$, and $\lambda : T \to \mathcal{T}$

is a function that assigns labels to transitions. If $\lambda(t) \neq \tau$, then $t$ is *observable*; otherwise, $t$ is *silent*. $\lambda$ is *distinctive* if it is injective on a subset of observable transitions.

### 3.2 Process Model

As discussed in Sect. 1, we consider process models consisting of activities and gateways, as captured in the following definition.

**Definition 6 (Process model)**
A *process model* is a tuple $W = (A, G^{+}, G^{\times}, C, \mathcal{A}, \mu)$, where $A$ is a non-empty set of *activities* (or *tasks*), $G^{+}$ is a set of *and gateways*, $G^{\times}$ is a set of *xor gateways* (these sets are disjoint). We write $G = (G^{+} \cup G^{\times})$ for all gateways and $Z = (A \cup G)$ for all nodes of a model. $C \subseteq Z \times Z$ defines the *control flow*. $\mathcal{A}$ is a non-empty set of *names* and $\mu : A \to \mathcal{A}$ is a function that assigns names to tasks.

A task $a \in A$ is a *source*, iff $\bullet a = \emptyset$ and it is a *sink*, iff $a\bullet = \emptyset$, where $\bullet x$, $x \in Z$, stands for a set of immediate predecessors and $x\bullet$ stands for a set of immediate successors of node $x$. As discussed in Sect. 1, we assume that $(Z, C)$ is a graph with a single source, a single sink and is such that every node is on a path from the source to the sink. Each task $a \in A$ has at most one incoming and at most one outgoing arc, i.e., $|\bullet a| \leq 1 \wedge |a\bullet| \leq 1$, while each gateway is either a split or a join: A gateway $g \in G$ is a *split*, iff $|\bullet g| = 1 \wedge |g\bullet| > 1$. A gateway $g \in G$ is a *join*, iff $|\bullet g| > 1 \wedge |g\bullet| = 1$. The execution semantics of process models is defined by a mapping to labeled free-choice Petri nets (cf. Definition 7). For example, Fig. 3 shows the WF-net of the process model in Fig. 1(a). The figure highlights the subnet that corresponds to rigid component $R1$ in Fig. 1(a) (cf. dotted box).

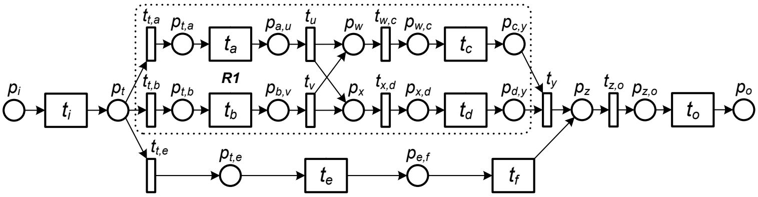


**Fig. 3.** A WF-net that corresponds to the process model in Fig. 1(a)

**Definition 7 (WF-net of a process model).** Let $W = (A, G^{+}, G^{\times}, C, \mathcal{A}, \mu)$ be a process model. Let $I$ and $O$ be sources and sinks of $W$, respectively. The labeled net $N = (P, T, F, \mathcal{T}, \lambda)$ *corresponding* to $W$ is defined by:

- $P = \{p_x \mid x \in G^{\times}\} \cup \{p_{x,y} \mid (x,y) \in C \wedge y \in A \cup G^{+}\} \cup \{p_x \mid x \in I \cup O\}$.
- $T = \{t_x \mid x \in A \cup G^{+}\} \cup \{t_{x,y} \mid (x,y) \in C \wedge x \in G^{\times}\}$.
- $F = \{(t_x, p_y) \mid (x,y) \in C \wedge x \in A \cup G^{+} \wedge y \in G^{\times}\} \cup \{(t_x, p_{x,y}) \mid (x,y) \in C \wedge x, y \in A \cup G^{+}\} \cup \{(t_{x,y}, p_y) \mid (x,y) \in C \wedge x, y \in G^{\times}\} \cup \{(t_{x,y}, p_{x,y}) \mid (x,y) \in C \wedge x \in G^{\times} \wedge y \in A \cup G^{+}\} \cup \{(p_x, t_{x,y}) \mid (x,y) \in C \wedge x \in G^{\times}\} \cup \{(p_{x,y}, t_y) \mid (x,y) \in C \wedge y \in A \cup G^{+}\} \cup \{(p_x, t_x) \mid x \in I\} \cup \{(t_x, p_x) \mid x \in O\}$.
- $\mathcal{T} = \mathcal{A} \cup \{\tau\}$. $\lambda(t_x) = \mu(x), t_x \in T, x \in A$, otherwise $\lambda(t) = \tau, t \in T$.

Definition 7 states that a task is mapped to a Petri net transition with a single input and a single output arc. An *and* gateway maps to a transition with multiple outgoing arcs (*and*-split) or multiple incoming arcs (*and*-join). An *xor* gateway maps to a place with multiple outgoing arcs (*xor*-split) or multiple incoming arcs (*xor*-join). The places corresponding to *xor*-splits are immediately followed by empty ($\tau$) transitions representing the branching conditions (cf. transitions $t_{x,y}$ introduced in the mapping). Sources and sinks of the process model are mapped to places.

A process model is sound if its corresponding WF-net is sound. In this paper we only consider sound process models. We note that a sound free-choice WF-system is guaranteed to be safe [16] and Definition 7 always produces free-choice WF-nets. Thus the rest of the paper deals with sound and safe process models.

## 4 Behavioral Equivalence of Process Models

This section motivates fully concurrent bisimulation as the equivalence notion for process models, cf., Sect.4.1, and discusses the procedure of checking equivalence for the class of behavior captured by occurrence nets, cf., Sect.4.2.

### 4.1 Fully Concurrent Bisimulation

An unstructured model and the corresponding structured model are structurally different, but behaviorally equivalent. There exist many notions of behavioral equivalence for concurrent systems [17]. A common notion of behavioral equivalence for concurrent systems is that of *bisimulation*. Related notions are those of *weak bisimulation* and *branching bisimulation*, which abstract away from silent transitions. These notions have been advocated as being suitable for comparing process models [12]. However, we argue that they are not suitable for our purposes. These three notions adopt an interleaving semantics – i.e., no two tasks are executed exactly at the same time. Thus, a concurrent system and its sequential simulation are considered equivalent. For example, Fig.4 shows the sequential simulation of the net in Fig.3. This net is structured and weakly bisimilar to the net in Fig.3, but it contains no parallel branch. We could take any process model, compute its sequential simulation, structure this sequential net using GOTO program transformations, and transform back the resulting sequential net into a structured process model. This structuring method is complete, but if we start with a process model containing *and* gateways, we obtain a (much larger) structured process model without any parallel branches.

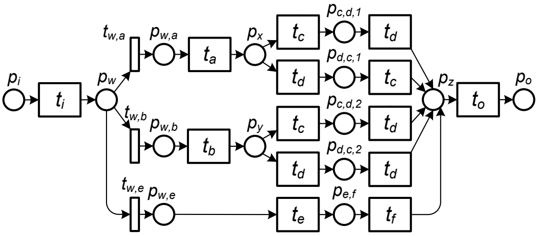


**Fig. 4.** Sequential simulation of the net in Fig.3

Accordingly, we adopt a notion of equivalence that preserves the level of concurrency of observable transitions, viz. *fully concurrent bisimulation* (FCB) [18]. FCB is defined in terms of concurrent runs of a system, a.k.a. *processes* in the literature (but not to be confused with "business processes" or workflows).

Let $N = (P, T, F)$ be a causal net. A *P-cut* $c \subseteq P$ of $N$ is a maximal set of places unordered w.r.t. $F^{+}$. Let $Min(N)$ define the set $\{x \in X \mid \bullet x = \emptyset\}$ and let $Max(N)$ define the set $\{x \in X \mid x \bullet = \emptyset\}$.

**Definition 8 (Process).** A *process* $\pi = (N_\pi, \rho)$ of a system $S = (N, M_0)$, $N = (P, T, F)$, consists of a causal net $N_\pi = (P_\pi, T_\pi, F_\pi)$ and a function $\rho : X_\pi \to X$:

- $\rho(P_\pi) \subseteq P,\ \rho(T_\pi) \subseteq T$,
- $Min(N_\pi)$ is a P-cut, which corresponds to the initial marking $M_0$, that is $\forall\ p \in P : M_0(p) = |\rho^{-1}(p) \cap Min(N_\pi)|$, and
- $\forall\ t \in T_\pi\ \forall\ p \in P : (F(p, \rho(t)) = |\rho^{-1}(p) \cap \bullet t|) \wedge (F(\rho(t), p) = |\rho^{-1}(p) \cap t \bullet|)$.

A process $\pi$ of $S$ is *initial*, iff $T_\pi = \emptyset$.

A process $\pi'$ is an extension of a process $\pi$ if it is possible to observe $\pi$ before one observes $\pi'$. Consequently, process $\pi$ is a prefix of $\pi'$.

**Definition 9 (Prefix, Process extension)**
Let $\pi = (N_\pi, \rho)$, $N_\pi = (P_\pi, T_\pi, F_\pi)$, be a process of $S = (N, M_0)$, $N = (P, T, F)$. Let $c$ be a P-cut of $N_\pi$ and let $c^{\downarrow}$ be the set $\{x \in X_\pi \mid \exists\ y \in c : (x, y) \in F^{*}\}$. A process $\pi_c^{\downarrow}$ is a *prefix* of $\pi$, iff $\pi_c^{\downarrow} = ((P_\pi \cap c^{\downarrow}, T_\pi \cap c^{\downarrow}, F \cap (c^{\downarrow} \times c^{\downarrow})), \rho|_{c^{\downarrow}})$. A process $\pi'$ is an *extension* of process $\pi$ if $\pi$ is a prefix of $\pi'$.

In order to define FCB, we need two auxiliary definitions: $\lambda$-abstraction of a process, which is a process footprint that ignores silent transitions, and the order-isomorphism of $\lambda$-abstractions.

**Definition 10 (Abstraction of a process of a labeled system)**
Let $S = (N, M_0)$, $N = (P, T, F, \mathcal{T}, \lambda)$, be a labeled system and let $\pi = (N_\pi, \rho)$, $N_\pi = (P_\pi, T_\pi, F_\pi)$, be a process of $S$. The $\lambda$-abstraction of $\pi$, denoted by $\alpha_\lambda(\pi) = (T'_\pi, <, \lambda')$, is defined by $T'_\pi = \{t \in T_\pi \mid \lambda(\rho(t)) \neq \tau\}$, $< = \{(t_1, t_2) \in T'_\pi \times T'_\pi \mid (t_1, t_2) \in F^{+}\}$, and $\lambda' : T'_\pi \to \mathcal{T}$, such that $\lambda'(t) = \lambda(\rho(t))$, $t \in T'_\pi$.

Two $\lambda$-abstractions are order-isomorphic if there exists a one-to-one correspondence between transitions of both abstractions that also preserves the ordering of the corresponding transitions in the respective abstractions.

**Definition 11 (Order-isomorphism of abstractions)**
Let $\alpha_{\lambda_1} = (T_1, <_1, \lambda_1)$ and $\alpha_{\lambda_2} = (T_2, <_2, \lambda_2)$ be two $\lambda$-abstractions, both with labels in $\mathcal{T}$. Then $\alpha_{\lambda_1}$ and $\alpha_{\lambda_2}$ are *order-isomorphic*, denoted by $\alpha_{\lambda_1} \cong \alpha_{\lambda_2}$, iff there is a bijection $\beta : T_1 \to T_2$ such that $\forall\ t \in T_1 : \lambda_1(t) = \lambda_2(\beta(t))$ and $\forall\ t_1, t_2 \in T_1 : t_1 <_1 t_2 \Leftrightarrow \beta(t_1) <_2 \beta(t_2)$.

Given the above, fully concurrent bisimulation is defined as follows.

**Definition 12 (Fully concurrent bisimulation)**
Let $S_1 = (N_1, M_0^1)$ and $S_2 = (N_2, M_0^2)$ be labeled systems, $N_1 = (P_1, T_1, F_1, \mathcal{T}_1, \lambda_1)$ and $N_2 = (P_2, T_2, F_2, \mathcal{T}_2, \lambda_2)$. $S_1$ and $S_2$ are *fully concurrent bisimilar*, denoted by $S_1 \approx S_2$, iff there is a set $\mathcal{B} \subseteq \{(\pi_1, \pi_2, \beta)\}$, such that:

(i) $\pi_1$ is a process of $S_1$, $\pi_2$ is a process of $S_2$, and $\beta$ is a relation between the non-$\tau$ transitions of $\pi_1$ and $\pi_2$.
(ii) If $\pi_0^1$ and $\pi_0^2$ are the initial processes of $S_1$ and $S_2$, respectively, then $(\pi_0^1, \pi_0^2, \varnothing) \in \mathcal{B}$.
(iii) If $(\pi_1, \pi_2, \beta) \in \mathcal{B}$, then $\beta$ is an order-isomorphism between the $\lambda_1$-abstraction of $\pi_1$ and the $\lambda_2$-abstraction of $\pi_2$.
(iv) $\forall (\pi_1, \pi_2, \beta) \in \mathcal{B}$ :
  (a) If $\pi_1'$ is an extension of $\pi_1$, then $\exists (\pi_1', \pi_2', \beta') \in \mathcal{B}$ where $\pi_2'$ is an extension of $\pi_2$ and $\beta \subseteq \beta'$.
  (b) Vice versa.

FCB defines an equivalence relation on labeled systems that is stricter than weak bisimulation and related notions. The nets in Fig.4 and Fig.3 are weakly bisimilar but not FCB-equivalent. Meanwhile, the two models in Fig.1 are FCB-equivalent (with the understanding that two process models are FCB-equivalent if the corresponding Petri nets are FCB-equivalent).

### 4.2 Behavioral Equivalence and Ordering Relations

The above definition of FCB-equivalence is abstract and hardly of any use when synthesizing structured nets from unstructured ones. Accordingly, we employ a more convenient way of reasoning about FCB-equivalence based on the *ordering relations* of occurrence nets. The idea is that any pair of nodes in an occurrence net can be in a precedence, conflict, or concurrent ordering relation as defined below, and these ordering relations can be used to reason about FCB-equivalence.

**Definition 13 (Ordering relations)**
Let $N = (P, T, F)$ be an occurrence net and let $x, y \in X$ be two nodes of $N$.

- $x$ *precedes* $y$, denoted by $x \rightsquigarrow_N y$, iff $(x, y) \in F^+$.
- $x$ and $y$ are in *conflict*, denoted by $x \#_N y$, iff $\exists\ t_1, t_2 \in T, t_1 \neq t_2 : (\bullet t_1 \cap \bullet t_2 \neq \varnothing) \land t_1 \rightsquigarrow_N x \land t_2 \rightsquigarrow_N y$.
- $x$ and $y$ are *concurrent*, denoted by $x \,\|_N\, y$, iff they are neither in precedence, nor in conflict.

The set $\mathcal{R} = \{\rightsquigarrow_N, \#_N, \|_N\}$ forms the *ordering relations* of $N$.

Let $N = (P, T, F, \mathcal{T}, \lambda)$ be a labeled occurrence net and let $T' = \{t \in T \mid \lambda(t) \neq \tau\}$. The $\lambda$-*ordering relations* of $N$ are formed by the set $\mathcal{R}_\lambda = \{\rightsquigarrow_N \cap\ T' \times T', \#_N \cap\ T' \times T', \|_N \cap\ T' \times T'\}$. We say that two ordering relations are isomorphic if for each pair of observable transitions the ordering relation coincides.

**Definition 14 (Isomorphism of ordering relations)**
Let $N_1 = (P_1, T_1, F_1, \mathcal{T}_1, \lambda_1)$ and $N_2 = (P_2, T_2, F_2, \mathcal{T}_2, \lambda_2)$ be two labeled occurrence nets with distinctive labelings. Let $T_1'$ and $T_2'$ denote non-$\tau$ transitions of $N_1$ and $N_2$, respectively. Two $\lambda$-ordering relations $\mathcal{R}_{\lambda_1}$ of $N_1$ and $\mathcal{R}_{\lambda_2}$ of $N_2$ are *isomorphic*, denoted by $\mathcal{R}_{\lambda_1} \cong \mathcal{R}_{\lambda_2}$, iff there is a bijection $\gamma : T_1' \to T_2'$, such that:

- $\forall\ t \in T_1' : \lambda_1(t) = \lambda_2(\gamma(t))$, and
- $\forall\ t_1, t_2 \in T_1' : (t_1 \rightsquigarrow_{N_1} t_2 \land \gamma(t_1) \rightsquigarrow_{N_2} \gamma(t_2)) \lor (t_2 \rightsquigarrow_{N_1} t_1 \land \gamma(t_2) \rightsquigarrow_{N_2} \gamma(t_1)) \lor (t_1 \#_{N_1} t_2 \land \gamma(t_1) \#_{N_2} \gamma(t_2)) \lor (t_1 \,\|_{N_1}\, t_2 \land \gamma(t_1) \,\|_{N_2}\, \gamma(t_2))$.

Finally, we show that two occurrence nets with isomorphic ordering relations are FCB-equivalent, and vice-versa. This result is exploited in the next section.

**Theorem 1.** *Let $S_1 = (N_1, M_i^1)$, $N_1 = (P_1, T_1, F_1, \mathcal{T}_1, \lambda_1)$, and $S_2 = (N_2, M_i^2)$, $N_2 = (P_2, T_2, F_2, \mathcal{T}_2, \lambda_2)$, be two labeled occurrence systems with distinctive labelings and $T_1' \subseteq T_1$, $T_2' \subseteq T_2$ observable transitions, such that there exists bijection $\psi : T_1' \to T_2'$ for which holds $\lambda_1(t) = \lambda_2(\psi(t))$, for all $t \in T_1'$. Let $\mathcal{R}_{\lambda_1}$ and $\mathcal{R}_{\lambda_2}$ be the $\lambda$-ordering relations of $N_1$ and $N_2$. Then, it holds:*

$$S_1 \approx S_2 \Leftrightarrow \mathcal{R}_{\lambda_1} \cong \mathcal{R}_{\lambda_2}.$$

*Proof.* We prove each direction of the equality separately.

($\Rightarrow$) Let $S_1$ and $S_2$ be FCB-equivalent. We want to show that $\mathcal{R}_{\lambda_1} \cong \mathcal{R}_{\lambda_2}$.
Let us assume that $S_1 \approx S_2$ holds, but $\mathcal{R}_{\lambda_1} \cong \mathcal{R}_{\lambda_2}$ does not hold. Furthermore, let us consider transitions $t_i^1, t_j^1 \in T_1'$ that are in one-to-one correspondence with transitions $t_i^2, t_j^2 \in T_2'$, i.e., $\lambda_1(t_i^1) = \lambda_2(\psi(t_i^2))$ and $\lambda_1(t_j^1) = \lambda_2(\psi(t_j^2))$. All scenarios can be reduced to the following two cases:

*Case 1:* ($t_i^1 \parallel_{N_1} t_j^1$ or $t_i^1 \rightsquigarrow_{N_1} t_j^1$, and $t_i^2 \#_{N_2} t_j^2$). If $t_i^1 \parallel_{N_1} t_j^1$ or $t_i^1 \rightsquigarrow_{N_1} t_j^1$, then there exists process $\pi_1$ in $S_1$ that contains $t_i^1$ and $t_j^1$. If $t_i^2 \#_{N_2} t_j^2$, then there exists no process $\pi_2$ in $S_2$ that contains $t_i^2$ and $t_j^2$.

*Case 2:* ($t_i^1 \rightsquigarrow_{N_1} t_j^1$, and $t_j^2 \rightsquigarrow_{N_2} t_i^2$ or $t_i^2 \parallel_{N_2} t_j^2$). Let $\pi_1$ be a process in $S_1$ that contains $t_i^1$ and $t_j^1$, and let $\pi_2$ be a process in $S_2$ that contains $t_i^2$ and $t_j^2$. Then, there exists no $\phi \subseteq \psi$, such that $\phi$ is an order-isomorphism between $\lambda$-abstractions of $\pi_1$ and $\pi_2$.

In both cases we reach the contradiction, i.e., systems $S1$ and $S2$ cannot be FCB-equivalent if the $\lambda$-ordering relations are not isomorphic.

($\Leftarrow$) Let $\mathcal{R}_{\lambda_1} \cong \mathcal{R}_{\lambda_2}$. We want to show that $S_1$ and $S_2$ are FCB-equivalent.
Let us assume that $\mathcal{R}_{\lambda_1} \cong \mathcal{R}_{\lambda_2}$ holds, but $S_1 \approx S_2$ does not hold. Then, for instance, in $S_1$ there exists process $\pi_1'$ that has no corresponding order-isomorphic process in $S_2$. Suppose that $\pi_1'$ has the minimal size among all such processes, i.e., any prefix of $\pi_1'$ has a corresponding order-isomorphic process in $S_2$. Let $\pi_1'$ be an extension of $\pi_1$ by exactly one observable transition $t_j^1 \in T_1'$. Let $\pi_2$ be a process in $S_2$ that is order-isomorphic with $\pi_1$. Let $t_j^2 \in T_2'$ be in one-to-one correspondence with $t_j^1$, i.e., $\lambda_1(t_j^1) = \lambda_2(\psi(t_j^2))$. All scenarios can be reduced to the following three cases:

*Case 1:* There exists process $\pi_2'$ that contains $t_j^2$ and is an extension of $\pi_2$ by one observable transition. Moreover, there exists $t_i^1 \in T_1'$ in $\pi_1$, such that $t_i^1 \rightsquigarrow_{N_1} t_j^1$. However, it holds $t_i^2 \parallel_{N_2} t_j^2$, for $t_i^2 \in T_2'$, such that $\lambda_1(t_i^1) = \lambda_2(\psi(t_i^2))$; otherwise there exists an order-isomorphism $\phi \subseteq \psi$ between $\pi_1'$ and $\pi_2'$.

*Case 2:* There exists no process $\pi_2'$ that contains $t_j^2$ and is an extension of $\pi_2$. Moreover, there exists $t_i^1 \in T_1'$ in $\pi_1$, such that $t_i^1 \rightsquigarrow_{N_1} t_j^1$. However, it holds $t_i^2 \#_{N_2} t_j^2$, for $t_i^2 \in T_2'$, such that $\lambda_1(t_i^1) = \lambda_2(\psi(t_i^2))$.

*Case 3:* There exists process $\pi_2'$ that contains $t_j^2$ and is an extension of $\pi_2$, but not by only one observable transition. Then, there exists $t_k^2 \in T_2'$ and process $\pi_2''$ in $S_2$, such that $t_k^2 \rightsquigarrow_{N_2} t_j^2$, $\pi_2''$ is prefix of $\pi_2'$, and $\pi_2$ is prefix of $\pi_2''$. However, $t_k^1 \in T_1'$, $\lambda_1(t_k^1) = \lambda_2(\psi(t_k^2))$, is not in $\pi_1'$ and, hence, $t_k^1 \not\rightsquigarrow_{N_1} t_j^1$.

In all three cases we reach the contradiction, i.e., the $\lambda$-ordering relations cannot be isomorphic if systems $S_1$ and $S_2$ are not FCB-equivalent. □

# 5 Synthesis of Structured Process Models

The key idea of the proposed structuring method is to compute the ordering relations of every rigid component, and to synthesize a structured process component from these ordering relations (if such a structured process component exists). A structured process component is one whose RPST contains only trivials, bonds and polygons. Accordingly, what we need is to find such structures in the graph induced by the ordering relations of the component. To this end, we rely on the concept of modular decomposition [19]. Below we discuss how to compute the ordering relations of a process component and then we use the output of this step to synthesize a structured component based on the modular decomposition.

## 5.1 Computing Ordering Relations

In order to compute the ordering relations of tasks in a process component, we first need to build a corresponding occurrence net, using a procedure known as unfolding [20]. For example, Fig.5 presents the occurrence net for the rigid component $R1$ in Fig.3. The occurrence net may include multiple transitions referring to the same task, e.g., transitions $t_{c,1}$ and $t_{c,2}$ refer to task $c$. If we used the ordering relations computed from the occurrence net to synthesize a structured process component, the component would contain many duplicate tasks. Fortunately, for any safe net there exists a prefix of its occurrence net, called the *complete prefix unfolding* [20], that is more compact than the occurrence net but contains all the information about markings contained in the occurrence net. Moreover, this prefix is finite (even for safe nets with cycles). The complete prefix unfolding is obtained by truncating the occurrence net in points where the information about reachable markings starts to be redundant.

**Definition 15 (Complete Prefix Unfolding, Cutoff transition)**
Let $N = (P, T, F)$ be an occurrence net.

- A *local configuration* $\lceil t \rceil$ of a transition $t$ in an occurrence net is the set of transitions that precede $t$, i.e., $\lceil t \rceil = \{t' \in T \mid (t', t) \in F^*\}$.
- The *final marking* of a local configuration $\mathit{Mark}(\lceil t \rceil)$ is the set of places that are marked after all the transitions in $\lceil t \rceil$ fire.
- An *adequate order* $\lhd$ is a strict well-founded partial order on local configurations, so that $\lceil t \rceil \subset \lceil t' \rceil$ implies $\lceil t \rceil \lhd \lceil t' \rceil$[1].
- A transition $t$ of an occurrence net is a *cutoff transition* if there exists a *corresponding transition* $t'$, such that $\mathit{Mark}(\lceil t \rceil) = \mathit{Mark}(\lceil t' \rceil)$ and $\lceil t' \rceil \lhd \lceil t \rceil$.
- A *complete prefix unfolding* is the greatest backward closed subnet of an occurrence net containing no transitions after cutoff transitions.

[1] Several definitions of *adequate order* exist; we use the one defined in [20], because it has been shown to generate compact unfoldings.

The dotted lines in Fig.[5] indicate which parts of the occurrence net are truncated in the complete prefix unfolding. In the unfolding, transition $t_v$ is a cutoff transition.

Alg.[1] (adapted from [21]) computes the ordering relations based on a complete prefix unfolding. This algorithm has a low polynomial time to the size of the net. However, the overall complexity of computing ordering relations is dominated by the exponential worst-case complexity of computing the prefix unfolding, which is an NP-complete problem. Observe that in the case of *and* rigids this step is not required as the corresponding WF-net is always an occurrence net. Besides, we do not compute the prefix unfolding over the whole net, but only on individual rigid components of the net. Tests we have conducted with sample process models show that the prefix unfolding computation takes sub-second times[2]. This finding is in line with other work that have empirically shown that prefix unfolding computation is efficient in practice [20].

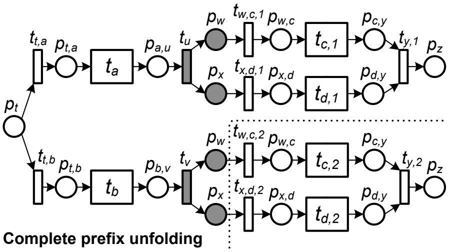


**Fig. 5.** Occurrence net and complete prefix unfolding of the running example

**Algorithm 1**: Compute Ordering Relations of a Complete Prefix Unfolding

```
Input: A WF-net $W = (P_w, T_w, F_w)$, its Complete Prefix Unfolding
       $U = (P, T, F)$, and the mapping function $l_w : T \rightarrow T_w$
Output: Matrix $ORel$ containing the ordering relations of transitions in $U$
foreach $t_i, t_j \in T$ do  Assert $(t_i \parallel_U t_j)$ in $ORel$
foreach $t_i \in T$ following a preorder traversal of $U$ do
    foreach $t_j \in T$ such that $t_j \in \bullet(\bullet t_i)$ do
        Assert $(t_j \rightsquigarrow_U t_i)$ in $ORel$
        foreach $t_k \in T$ such that $(t_k \rightsquigarrow_U t_j) \in ORel$ do
            Assert $(t_k \rightsquigarrow_U t_i)$ in $ORel$
        foreach $t_k \in T$ such that $(t_k \#_U t_j) \in ORel$ do
            Assert $(t_k \#_U t_i)$, $(t_i \#_U t_k)$ in $ORel$
    foreach $t_j \in T$ such that $t_i \neq t_j \wedge \bullet t_i \cap \bullet t_j \neq \emptyset$ do
        Assert $(t_k \#_U t_i)$ in $ORel$
        foreach $t_k \in T$ such that $(t_j \rightsquigarrow_U t_k) \in ORel$ do
            Assert $(t_k \#_U t_i)$, $(t_i \#_U t_k)$ in $ORel$
// Iterate over cutoff transitions in reverse topological order
foreach $t_i, t_j, t_k \in T$ such that $t_i$ is a cutoff in $U$, $t_j$ is not a cutoff in $U$,
$Mark(\lceil t_i \rceil) = Mark(\lceil t_j \rceil)$, $l_w(t_i)\bullet = l_w(t_j)\bullet$, and $l_w(t_i) \bullet \cap \bullet\, l_w(t_k) \neq \emptyset$ do
    foreach $t_m, t_n \in T$ such that $(t_k \rightsquigarrow_U t_m) \in ORel \vee t_k = t_m$ and $t_n \in \lceil t_i \rceil$ do
        Assert $(t_n \rightsquigarrow_U t_m)$ in $ORel$
return $ORel$
```

[2] Using the Mole tool for prefix unfolding `http://www.fmi.uni-stuttgart.de/szs/tools/mole/` which implements the algorithm in [20].

Alg. 1 comprises two phases. First, it computes the ordering relations of transitions on the unfolding according to Definition 13. Then, it updates the relations of transitions in the local configuration of every cutoff transition to overcome the effects of truncation. The update must be performed in reverse topological order. For instance, the algorithm will assert $(t_b \rightsquigarrow t_{c,1})$ in addition to $(t_a \rightsquigarrow t_{c,1})$, i.e., task $c$ may be preceded by either $a$ or $b$. Note that we impose an additional requirement i.e., postsets of a cutoff transition and its corresponding transition must map to the same set of places in the WF-net, for all cutoff transitions. If a complete prefix unfolding does not meet this requirement, it must be expanded.

### 5.2 From Ordering Relations to Process Models

This section presents the algorithm for synthesizing a well-structured process model that is fully concurrent bisimilar with a given (unstructured) model. Also, we identify the cases when an equivalent well-structured model does not exist.

According to Theorem 1, two process models are fully concurrent bisimilar, iff they demonstrate same ordering relations. Given an (unstructured) process model, the algorithm proceeds by computing its ordering relations, as discussed in Sect. 5.1. Afterwards, the algorithm attempts to synthesize a well-structured model with the same ordering relations.

Let $N = (P, T, F, \mathcal{T}, \lambda)$ be a labeled occurrence net. The *ordering relations graph* of $N$ is a triple $\mathcal{G}_\lambda = (V, E, \mathcal{L})$, where $V$ is the set of non-$\tau$ transitions of $N$, $\mathcal{L} = \{\epsilon, \rightsquigarrow, \#, \|\}$ is a set of labels, and $E : V \times V \rightarrow \mathcal{L}$ is an edge labeling function, such that $E(x, y) = \oplus$, $x, y \in V$ and $\oplus \in \mathcal{L} \setminus \epsilon$, if $x \oplus_N y$, otherwise $E(x, y) = \epsilon$. Self-relations are ignored, i.e., $E(x, x) = \epsilon$, $x \in V$. Observe that $\mathcal{G}_\lambda$ is an alternative representation of $\lambda$-ordering relations $\mathcal{R}_\lambda$ of $N$.

Fig. 6(a) shows the ordering relations graph of a complete prefix unfolding that is given in Fig. 5. As the conflict and concurrency relations are symmetric, the corresponding edges are visualized as two-sided arrows; solid and dotted for the conflict ($a \# b$) and concurrency ($c \| d$) relation, respectively. Regular arrows reflect the precedence relation, which is transitive and asymmetric. Edges that have $\epsilon$ labels are not visualized. Because of the precedence relation, most of the ordering relations graphs are asymmetric.

The RPST of a well-structured model is composed of trivial, polygon, and bond (either *and* or *xor*) components. Contrary to a rigid component that can have an arbitrary topology, the structure of each component of a well-structured model is well-defined and has a precise structural characterization in terms of the corresponding ordering relations graph. The ordering relations graph of a bond is a complete graph, or a clique. All edges in the graph have the same label: # for *xor* bonds and $\|$ for *and* bonds. This topology is consistent with the intuition behind: all nodes in a *xor* bond are in conflict, i.e., only one is executed; all nodes in an *and* bond are concurrently executed. Fig. 6(b) shows an *and* bond with three parallel branches, whereas Fig. 6(c) shows the corresponding clique of concurrent relations. In the cases of a trivial and polygon component, the ordering relations graph is a direct acyclic graph representing the transitive closure, or the total order, of the precedence relation. All edges of the graph are labeled $\rightsquigarrow$. Fig. 6(d)

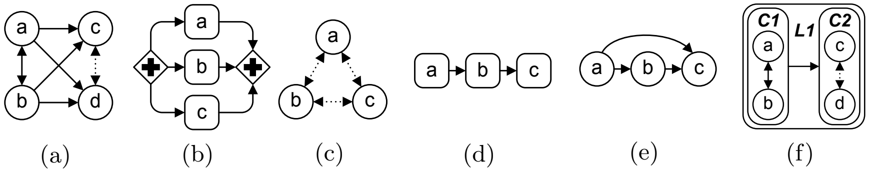


**Fig. 6.** (a) An ordering relations graph, (b) an *and* bond component, (c) an *and* complete module, (d) a polygon component, (e) a linear module, (f) the MDT of (a)

shows a polygon composed of three activities, whereas Fig. 6(e) presents the corresponding transitive closure over the precedence relation.

Let $\mathcal{G} = (V, E, \mathcal{L})$ be an ordering relations graph. A *module* $M \subseteq V$ of $\mathcal{G}$ is a non-empty subset of transitions that have a uniform relations with transitions $V \setminus M$, i.e., $\forall\ x, y \in M\ \forall\ z \in V \setminus M : E(x,z) = E(y,z) \wedge E(z,x) = E(z,y)$. Note that singleton sets of $V$ are referred to as *trivial* modules.

**Definition 16 (Complete, Linear, Primitive)**
Let $M$ be a non-singleton module of $\mathcal{G}$.

- $M$ is *complete* $(C)$, iff $\exists\ l \in \{\#, \|\}\ \forall\ x, y \in M, x \neq y : E(x,y) = l$, i.e., the subgraph induced by $M$ is a complete graph, or a clique. If $l = \#$, then $M$ is *xor* complete, otherwise $M$ is *and* complete.
- $M$ is *linear* $(L)$, iff there exists a linear order $(x_1, ..., x_{|T'|})$ of elements of $T'$, such that $E(x_i, x_j) = \rightsquigarrow$, if $i < j$, and $E(x_i, x_j) = \epsilon$ otherwise.
- If $M$ is neither complete, nor linear, then $M$ is *primitive* $(P)$.

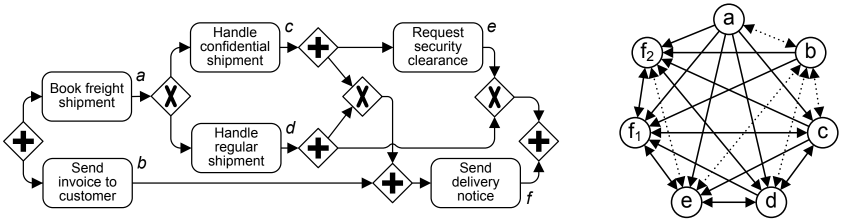
The following proposition summarizes relations between components of a process model and modules of an ordering relations graph.

**Proposition 1.** Let $C_1$ be a process component and let $M_1$ be the corresponding ordering relations graph. Let $M_2$ be an ordering relations graph and let $C_2$ be the corresponding process component.

1. If $C_1$ is trivial or polygon, then $M_1$ is linear.
2. If $M_2$ is linear, then there exists $C_2$ that is trivial or polygon.
3. If $C_1$ is *and* (*xor*) bond, then $M_1$ is *and* (*xor*) complete.
4. If $M_2$ is *and* (*xor*) complete, then there exists $C_2$ that is *and* (*xor*) bond.

Two modules $M_1$ and $M_2$ of $\mathcal{G}$ *overlap*, iff they intersect and neither is a subset of the other, i.e., $M_1 \setminus M_2$, $M_1 \cap M_2$ , and $M_2 \setminus M_1$ are all non-empty. $M_1$ is *strong*, iff there exists no module $M_2$ of $\mathcal{G}$, such that $M_1$ and $M_2$ overlap. The *modular decomposition* substitutes each strong module of a graph by a new vertex and proceeds recursively. The result is a rooted, unique tree called the *Modular Decomposition Tree*, which can be computed in linear time [19].

**Definition 17 (Modular Decomposition Tree).** Let $\mathcal{G} = (V, E, \mathcal{L})$ be an ordering relations graph. The *Modular Decomposition Tree* (MDT) of $\mathcal{G}$, denoted by $MDT(\mathcal{G})$, is a containment hierarchy of all strong modules of $\mathcal{G}$.



(a) (b)

**Fig. 7.** (a) A rigid process component and (b) its ordering relations graph

Fig. 6(f) shows the MDT of the ordering relations graph that is proposed in Fig. 6(a). Each module is enclosed in a box with rounded corners. Note that module names hint at their class. For instance, module $C1$ is a complete module, and is composed of two nodes $a$ and $b$ that are in conflict relation, $a \# b$. Therefore, $C1$ is a *xor* complete module. Similarly, $C2$ is an *and* complete module. By treating both modules as singletons, the modular decomposition identifies that they are in total order and, hence, form a linear module $L1$.

We are now ready to present the main result of this section.

**Theorem 2.** *Let $\mathcal{G}$ be an ordering relations graph. The Modular Decomposition Tree of $\mathcal{G}$ has no primitive module, iff there exists a well-structured process model $W$ such that $\mathcal{G}$ is the ordering relations graph of $W$.*

*Proof.* Let $\mathcal{G} = (V, E, \mathcal{L})$ be an ordering relations graph.

($\Rightarrow$) Assume that the MDT of $\mathcal{G}$ has no primitive module. We show now by structural induction on the MDT of $\mathcal{G}$ that there exists a well-structured process model $W$ with ordering relations $\mathcal{G}$. The MDT of $\mathcal{G}$ contains singleton, linear, *and* complete, and *xor* complete modules.

*Base:* If the MDT of $\mathcal{G}$ consists of a single module $M$, then $M$ is singleton and $W$ is a process model composed of a single task $m \in M$.

*Step:* Let $M$ be a module of the MDT of $\mathcal{G}$ such that each child module of $M$ has a corresponding well-structured process model. If $M$ is linear, then $W$ can be a trivial or polygon component composed from children of $M$, cf., 2 in Prop. 1. If $M$ is complete, then $W$ can be a bond component, either *and* or *xor*, composed from children of $M$, cf., 4 in Prop. 1. In both cases, $M$ has a corresponding well-structured process model.

Therefore, there exists a well-structured process model $W$ composed from children of module $V$ of $\mathcal{G}$ that has ordering relations graph $\mathcal{G}$.

($\Leftarrow$) Let $W$ be a well-structured process model with ordering relations graph $\mathcal{G}$. We want to show that the MDT of $\mathcal{G}$ has no primitive module. Because $W$ is well-structured, the RPST of $W$ has no rigid component. The corresponding ordering relations graph of a non-rigid component, i.e., trivial, polygon, or bond component, is either complete or linear, cf., 1 and 3 in Prop. 1. If $W$ is composed of a single task, then $\mathcal{G}$ consists of one singleton trivial module. □

Finally, we detail the approach for structuring acyclic rigid components in Alg. 2.

**Algorithm 2**: Restructure an Acyclic Rigid Process Component

```
Input: An acyclic rigid process component
Output: The RPST of a well-structured process component
Compute the complete prefix unfolding of the input process component
Compute the ordering relations of the unfolding (Alg. 1)
Restrict the ordering relations to the set of non-$\tau$ transitions
Compute the MDT of the graph formed with the restricted ordering relations
Construct the RPST by traversing each module $M$ of the MDT (in postorder)
  ∘ If $M$ is trivial singleton, then generate a task
  ∘ If $M$ is and complete, then generate an and bond component
  ∘ If $M$ is xor complete, then generate a xor bond component
  ∘ If $M$ is linear, then generate a trivial or polygon component
  ∘ If $M$ is primitive, then FAIL // component cannot be restructured
return the RPST
```

Based on all previous results, Alg.2 synthesizes, whenever possible, the RPST of an FCB-equivalent well-structured component for a given acyclic rigid component. No variables are introduced. Task duplication depends on the “quality” of the prefix unfolding. The complexity of the algorithm is determined by the exponential complexity of the unfolding (see earlier discussion). All other steps are polynomial. In the case of an *and* rigid, the unfolding is not needed because the WF-net of an *and* rigid is already an occurrence net. The algorithm fails if the input process component is inherently unstructured, such as the process component in Fig.7(a). In this particular case, the ordering relations graph forms a single primitive module, cf., Fig.7(b). Note that the unfolding step duplicates task $f$.

## 6 Conclusion

We conclude that a sound and safe acyclic process model is inherently unstructured if its RPST has a rigid component for which the modular decomposition of its ordering relations contains a primitive. In all other cases, Algorithm 2 applied to each rigid in the RPST constructs an equivalent structured model. We have thus provided a characterization of the class of structured acyclic process models under FCB equivalence, and a complete structuring method. This method is implemented in a tool, namely `bpstruct`, that structures BPMN models exported from Oryx[3]. The tool is available at `https://code.google.com/p/bpstruct/`.

This method can also be used to structure models with SESE cycles, even if these cycles contain unstructured components. In this case, the unstructured components and the cycles are in different nodes of the RPST. However, the proposed method cannot deal with models with arbitrary cycles. Also, the results do not apply to models with OR-joins, complex gateways, exception handlers and non-interrupting events. Future work will aim at lifting these restrictions.

[3] `http://oryx-project.org/`

**Acknowledgments.** This research is partly funded by the ERDF via the Estonian Center of Excellence in Computer Science.

## References

1. Kiepuszewski, B., ter Hofstede, A.H.M., Bussler, C.: On Structured Workflow Modelling. In: Wangler, B., Bergman, L.D. (eds.) CAiSE 2000. LNCS, vol. 1789, pp. 431–445. Springer, Heidelberg (2000)
2. Laue, R., Mendling, J.: The Impact of Structuredness on Error Probability of Process Models. In: UNISCON. LNBIP, vol. 5, pp. 585–590. Springer, Heidelberg (2008)
3. Laguna, M., Marklund, J.: Business Process Modeling, Simulation, and Design. Prentice Hall, Englewood Cliffs (2005)
4. Combi, C., Posenato, R.: Controllability in Temporal Conceptual Workflow Schemata. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 64–79. Springer, Heidelberg (2009)
5. Oulsnam, G.: Unravelling unstructured programs. Comput. J. 25(3), 379–387 (1982)
6. Liu, R., Kumar, A.: An Analysis and Taxonomy of Unstructured Workflows. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 268–284. Springer, Heidelberg (2005)
7. Hauser, R., Friess, M., Küster, J.M., Vanhatalo, J.: An Incremental Approach to the Analysis and Transformation of Workflows Using Region Trees. IEEE Transactions on Systems, Man, and Cybernetics, Part C 38(3), 347–359 (2008)
8. Polyvyanyy, A., García-Bañuelos, L., Weske, M.: Unveiling Hidden Unstructured Regions in Process Models. In: Meersman, R., Dillon, T., Herrero, P. (eds.) OTM 2009. LNCS, vol. 5870, pp. 340–356. Springer, Heidelberg (2009)
9. Hauser, R., Koehler, J.: Compiling Process Graphs into Executable Code. In: Karsai, G., Visser, E. (eds.) GPCE 2004. LNCS, vol. 3286, pp. 317–336. Springer, Heidelberg (2004)
10. Koehler, J., Hauser, R.: Untangling Unstructured Cyclic Flows - A Solution Based on Continuations. In: Meersman, R., Tari, Z. (eds.) OTM 2004. LNCS, vol. 3290, pp. 121–138. Springer, Heidelberg (2004)
11. Ouyang, C., Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M., Mendling, J.: From business process models to process-oriented software systems. ACM Trans. Softw. Eng. Methodol. 19(1) (2009)
12. Kiepuszewski, B., ter Hofstede, A.H.M., van der Aalst, W.M.P.: Fundamentals of Control Flow in Workflows. Acta Inf. 39(3), 143–209 (2003)
13. Vanhatalo, J., Völzer, H., Koehler, J.: The Refined Process Structure Tree. Data & Knowledge Engineering 68(9), 793–818 (2009)
14. Polyvyanyy, A., Vanhatalo, J., Völzer, H.: Simplified computation and generalization of the refined process structure tree. Technical Report RZ 3745, IBM (2009)
15. van der Aalst, W.M.P.: Verification of Workflow Nets. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 407–426. Springer, Heidelberg (1997)
16. 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.) BPM 2000. LNCS, vol. 1806, pp. 161–183. Springer, Heidelberg (2000)
17. 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)

18. Best, E., Devillers, R.R., Kiehn, A., Pomello, L.: Concurrent bisimulations in petri nets. Acta Inf. 28(3), 231–264 (1991)
19. McConnell, R.M., de Montgolfier, F.: Linear-time modular decomposition of directed graphs. Discrete Applied Mathematics 145(2), 198–209 (2005)
20. Esparza, J., Römer, S., Vogler, W.: An Improvement of McMillan's Unfolding Algorithm. FMSD 20(3), 285–310 (2002)
21. Kondratyev, A., Kishinevsky, M., Taubin, A., Ten, S.: Analysis of Petri Nets by Ordering Relations in Reduced Unfoldings. FMSD 12(1), 5–38 (1998)

# A New Semantics for the Inclusive Converging Gateway in Safe Processes

Hagen Völzer

IBM Research — Zurich, Switzerland
hvo@zurich.ibm.com

**Abstract.** We propose a new semantics for the inclusive converging gateway (also known as *Or-join*). The new semantics coincides with the intuitive, widely agreed semantics for Or-joins on sound acyclic workflow graphs which is implied, for example, by dead path elimination on BPEL flows. The new semantics also coincides with the block-based semantics as used in BPEL on cyclic graphs that can be composed from sound acyclic graphs, repeat- and while-loops. Furthermore, we display several examples for unstructured workflow graphs for which Or-joins get the desired intuitive semantics. A key insight is that not all situations where two or more Or-joins seem to be mutually dependent (known as 'vicious circles') are necessarily symmetric. Many such situations are asymmetric and can be resolved naturally in favor of one of the Or-joins. Still symmetric or almost symmetric situations exist, for which it is not clear what semantics is desirable and which result in a deadlock in our semantics. We show that enabledness of an Or-join in our semantics can be decided in linear time in the size of the workflow graph.

## 1 Introduction

The semantics of the inclusive converging gateway, also known as *Or-join*, is recognized as one of the main problems of defining an execution semantics for a business process modeling language that permits unrestricted directed graphs such as BPMN and EPCs. With an increased interest in directly executing BPMN models or generating code from them, the problem has become more important and consequently, has received a lot of attention recently [1–4, 7–9].

Or-joins were introduced into business process modeling languages to be able to synchronize a variable set of threads. The simplest example is shown in Fig. 1, which is drawn using BPMN [5]. The Or-split $s1$ produces a token for either or both of its outgoing edges, i.e., either task $A$ will be enabled, or task $B$ or both of them. The corresponding Or-join $j1$ is meant to synchronize the created threads, i.e., it should wait for all tokens that were created by the Or-split $s1$ before it consumes them and produces a token on its outgoing edge. This simple example already shows that the semantics of the Or-join is necessarily *non-local*, i.e., in contrast to all other usual gateways, its enabledness does not only depend on the tokens on its incoming edges. To see that, consider the states shown in Fig. 1(a) and (b), where a token on an edge is shown as a black dot. The Or-join behaves differently in both states: In (a), $j1$ is enabled whereas in (b), $j1$ is not enabled as it has to wait for the other token that has not yet passed task

R. Hull, J. Mendling, and S. Tai (Eds.): BPM 2010, LNCS 6336, pp. 294–309, 2010.

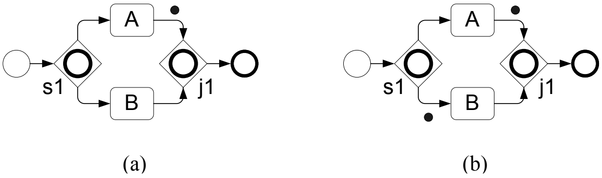


**Fig. 1.** A simple example of an Or-join

*B*. However both states are indistinguishable if we only look at the incoming edges of $j1$. Therefore, the semantics of the Or-join is non-local.

Figure 2 shows more examples for using an Or-join to synchronize a variable set of threads. In other words, the Or-join merges a set of paths that are neither necessarily pairwise alternative, in which case an Xor-join can be used, nor are the paths pairwise parallel, in which case an And-join can be used. Figure 2 shows that no Or-split is needed to create such a variable set of paths.

We henceforth do not show any tasks in our examples because they are not relevant for our considerations. The reader may imagine a task on each edge of the graph.

The Or-join is often used without a formalized semantics. The informal statement that is usually given to explain its intended behavior is that the Or-join has to wait for all tokens that *'may still arrive'* on its incoming edges, cf. [7]. This statement raises at least two fundamental problems:

1. Whether a token may still arrive is traditionally interpreted on the state space of the graph [3, 7–9]. Van der Aalst, Desel, and Kindler [7] have shown that a straightforward formalization of the informal statement then fails because 'may still arrive' implicitly refers to the very semantics that it is used to define. The same authors [7] and later, Kindler [3], showed that this self-reference can be resolved using fixed-point theory. While it remains unclear whether the resulting semantics is useful for all graphs, there is still another problem with the state-space interpretation: In order to determine whether an Or-join is enabled in a given state $s$ of the graph, one has to explore in the worst case all possible future states of $s$, of which there are exponentially many. This would potentially pose a substantial problem for an execution engine.

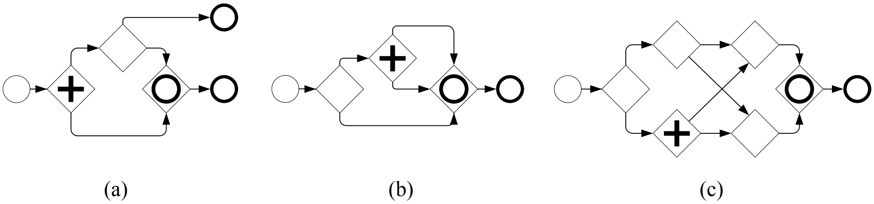


**Fig. 2.** Or-joins for synchronizing a variable set of threads

2. The second fundamental problem is that two or more Or-joins may mutually depend on each other: One join would be enabled only if the other would not and vice versa. Such situations, which can only occur in cyclic graphs, have been called ‘vicious circles’ [7]. A typical example is shown later in Fig. 5. It is not clear whether models containing such situations actually occur in practice and if so, what they are intended to model and hence how the semantics should be defined for them.

We address the first problem by using a graph-based interpretation of ‘may arrive’, which is inspired by informal statements in version 1.0 of the BPMN specification [5]. A similar approach has also been proposed by Dumas *et al.*[2]. This resolves the formalization problem and results in a low computational complexity of enactment. Dumas *et al.*[2] obtain an algorithm that runs in quadratic time which can be reduced to linear time after constructing a data structure of quadratic size. We will display a simple linear time algorithm for our new semantics.

Regarding the second problem, we argue that ‘vicious circles’ can occur in practice by displaying simple reasonable graphs that contain vicious circles. Even simple *well-structured* graphs, i.e., graphs that are composed from matching pairs of splits and joins, may contain a vicious circle. Existing proposals for Or-join semantics, including Dumas *et al.*[2], then introduce either a deadlock or an artificial non-deterministic choice to such well-structured graphs, which we do not find satisfactory. We argue that a well-structured graph has a natural semantics as it can be seen as a representation of a block-structured process, e.g. modeled in BPEL, and it can be executed accordingly as implied, e.g., by the semantics of BPEL. This was also stated as a design goal for an Or-join semantics by Mendling and van der Aalst [4].

We show how the natural semantics for well-structured graphs can be defined without referring to blocks, which gives rise to a new semantics for general workflow graphs. The new semantics agrees with the widely accepted Or-join semantics for acyclic graphs, which is implied, for example, by the semantics for BPEL flows, i.e., *dead path elimination*. The new semantics also agrees with the above mentioned natural semantics for well-structured graphs. Moreover, we display examples of unstructured cyclic graphs that also get a desired intuitive semantics. Still some models with vicious circles remain, for which it is not clear what a reasonable semantics should be. These cases create a deadlock in our semantics and hence could be sorted out by static analysis.

Our semantics was developed as part of the BPMN 2.0 standardization effort and it is included in the current BPMN 2.0 specification draft [6].

The paper is structured as follows. After introducing preliminary notions in Sect. 2, we discuss Or-join semantics for acyclic graphs in Sect. 3. Acyclic graphs form an important class because essentially, it is intuitively clear how Or-joins should behave in acyclic graphs, viz. as defined by dead path elimination (see Sect. 3.3). As said above, ‘vicious circles’ do not occur in acyclic graphs. Then, in Sect. 4, we stepwise introduce the new semantics and discuss its various aspects. Some proofs are omitted in this version but can be found in an extended version of this paper.

## 2 Preliminaries

This section defines the basic preliminary notions of this paper, which include workflow graphs, their semantics and the *soundness* property for workflow graphs. Unsound workflow graphs are usually considered as invalid models that contain modeling errors.

A *workflow graph* $G = (V, E, \ell)$ consists of set $V$ of *nodes*, a set $E \subseteq V \times V$ of *edges*[1], and a partial mapping $\ell : V \rightarrow \{\text{And}, \text{Xor}, \text{Or}\}$ such that

1. $\ell(x)$ is defined if and only if $x$ has more than one incoming edge or more than one outgoing edge,
2. there is exactly one source and at least one sink,
3. the source has exactly one outgoing edge and each sink has exactly one incoming edge, and
4. every node is on a path from the source to some sink.

The source is also called the *start node*, a sink is called an *end node*, $\ell(x)$ is called the *logic* of $x$. If the logic is And, Or or Xor, we call $x$ a *gateway*; if $x$ has no logic and $x$ is no start or end node, we call $x$ a *task*. We use BPMN to depict workflow graphs, i.e., gateways are drawn as diamonds, where the symbol "+" inside stands for And, a circle stands for Or, whereas no decoration stands for Xor. Tasks are drawn as rectangles, start and end nodes as circles. A gateway that has more than one incoming edge and only one outgoing edge is also called a *join*, a gateway with more than one outgoing but only one incoming edge is also called a *split*. We may assume for simplicity of the presentation that every gateway is either a split or a join. We say that an edge $e$ is *incident* to a node $n$ if $e$ is incoming to $n$ or outgoing from $n$. Let $x, y$ be two graph elements, i.e., nodes or edges. If there is a path from $x$ to $y$, we also say sometimes that $x$ is *upstream* of $y$ and $y$ is *downstream* of $x$. If the graph is acyclic, then 'upstream' is a partial order and we write $x \leq y$.

The semantics of a workflow graph is, similarly to Petri nets, defined as a token game. A *state* of a workflow graph is represented by tokens on the edges of the graph. Let $G = (V, E, \ell)$ be a workflow graph. A *state* of $G$ is a mapping $s : E \rightarrow \mathbb{N}$, which assigns a natural number to each edge. When $s(e) = k$, we say that edge $e$ carries $k$ *tokens* in state $s$. The semantics of the various nodes is defined as usual. An And-gateway removes one token from each of its ingoing edges and adds one token to each of its outgoing edges. An Xor-gateway nondeterministically chooses one of its incoming edges on which there is at least one token, removes one token from that edge, then nondeterministically chooses one of its outgoing edges, and adds one token to that outgoing edge. As usual, we abstract from the data that controls the flow in Xor-gateways, hence the nondeterministic choice. Enabledness of an Or-gateway will be defined later in this paper. When an Or-gateway executes, it consumes a token from each incoming edge that carries a token and produces a token for each edge of a nonempty subset of its outgoing edges. That subset is chosen nondeterministically, again abstracting from data-based decisions.

To be more precise, let $s$ and $s'$ be two states and $n$ a node that is neither a start nor an end node. At this point, we assume that a definition of when an Or-join is *enabled*

[1] We show some example workflow graphs that have multiple edges from one node to another. The reader may imagine a task on such edges to comply with this formalization.

in a state $s$ of the workflow graph is given, i.e., the following is parametrized by such a definition. Because we will introduce several such definitions later in this paper, we say also an Or-join is $X$-*enabled* in a state $s$ to make the parameter more explicit. The parameter $X$ will occur in the notions that depend on $X$-enabledness of Or-joins.

We write $s \overset{n}{\to} s'$ when $s$ changes to $s'$ by executing node $n$. We have $s \overset{n}{\to} s'$ if

1. $\ell(n) =$ And or the logic of $n$ is undefined, and
$$s'(e) = \begin{cases} s(e) - 1 & e \text{ is an incoming edge of } n, \\ s(e) + 1 & e \text{ is an outgoing edge of } n, \\ s(e) & \text{otherwise.} \end{cases}$$
Note that our assumption that every gateway is either a join or a split implies that an edge cannot be both, incoming and outgoing for the same node.
2. $\ell(n) =$ Xor and there exists an incoming edge $e'$ and an outgoing edge $e''$ of $n$ such that
$$s'(e) = \begin{cases} s(e) - 1 & e = e', \\ s(e) + 1 & e = e'', \\ s(e) & \text{otherwise.} \end{cases}$$
3. $\ell(n) =$ Or, $x$ is $X$-enabled in $s$, and there exists a nonempty set $F$ of outgoing edges of $n$ such that
$$s'(e) = \begin{cases} s(e) - 1 & e \text{ is an incoming edge of } n \text{ such that } s(e) \geq 1, \\ s(e) + 1 & e \in F, \\ s(e) & \text{otherwise.} \end{cases}$$

The *initial state* of $G$ is the state where there is exactly one token on the unique outgoing edge of the start node and no token anywhere else. A node $n$ that has Xor- or And-logic or is an Or-split is *enabled* in a state $s$ if there exists a state $s'$ such that $s \overset{n}{\to} s'$. We also say $X$-*enabled* in a context where $n$ can be an Or-join or another gateway. A state $s'$ is $X$-*reachable from* a state $s$ if there exists a finite sequence $s_0 \overset{n_1}{\to} s_1 \ldots s_{k-1} \overset{n_k}{\to} s_k, k \geq 0$ such that $s_0 = s$ and $s_k = s'$. Such a sequence is called an $X$-*execution*. A state is a $X$-*reachable* state of $G$ if it is reachable from the initial state of $G$.

An $X$-reachable state $s$ is a (*local*) $X$-*deadlock* if there exists a token on an incoming edge of a gateway such that every state that is $X$-reachable from $s$ also contains a token on that edge. A state is *unsafe* or has a *lack of synchronization* if there is an edge which carries more than one token in $s$, otherwise it is *safe*. A workflow graph $G$ is $X$-*live* if it has no $X$-deadlock and $X$-*safe* if no $X$-reachable state has a lack of synchronization. $G$ is $X$-*sound* if it is $X$-live and $X$-safe.

Note that all notions are independent of the parameter $X$ in case $G$ does not contain an Or-join.

## 3 Semantics for Acyclic Workflow Graphs

To clarify the Or-join semantics for acyclic graphs, we define a state-space based and a graph-based semantics of the Or-join in this section. We show that both coincide on sound workflow graphs. Furthermore, we show that the graph-based semantics is the same that is induced by the execution semantics for BPEL flows, i.e., *dead path elimination*.

It is agreed that at least one incoming edge of the Or-join needs to have a token for its enabledness. It is also usually assumed that, if there is a token on each incoming edge, the Or-join is enabled. We are now going to interpret what it means that the Or-join has to wait for any tokens that 'may still arrive' on the empty incoming edges.

### 3.1 The State-Space View

The problem of formalizing the state-space based interpretation of 'a token may arrive' arises already for acyclic workflow graphs. Given a state $s$ of the graph, a token *may arrive* on an edge $e$ if $s$ can evolve into a state $s'$ such that $e$ has a token in $s'$. However, before it can be defined whether a state can evolve into another state, we need to define whether an Or-join is enabled. Conversely, before we can define whether an Or-join is enabled, we need to define when a state can evolve into another state. This cyclic dependency prevents a straight-forward formalization. To avoid fixed point theory as a resolution here, we can exploit the fact that, in an acyclic workflow graph, the dependencies between multiple Or-joins are given by the partial order that is induced by the edges of the graph. In particular, minimal Or-joins with respect to this order do not depend on any other Or-join and their semantics can be defined without referring to the Or-join semantics:

**Definition 1.** *Let G be an acyclic workflow graph.*

1. *Let $j$ be an Or-join of G. The* depth *of $j$ is the largest number of Or-joins that are contained on any path from the start node to $j$ (not counting $j$ itself).*
2. *We say that a state $s'$ is* $k$-S-reachable *('S' stands for 'state-based') from a state $s$ for $k \geq 0$ if there is an S-execution that starts in $s$, ends in $s'$, and that does not contain any Or-join of depth $\geq k$.*
3. *An Or-join of depth $k$ is* S-enabled *in a state $s$ for $k \geq 0$ if $s(e) \geq 1$ for some incoming edge $e$ of $x$ and if for each incoming edge $e'$ of $x$ with $s(e') = 0$, there is no state $s'$ such that $s'$ is $k$-S-reachable from $s$ and $s'(e') \geq 1$.*

To evaluate S-enabledness of an Or-join according to this definition, one can first evaluate the S-enabledness of Or-joins of depth 0, then depth 1 and so forth. To see whether an Or-join of depth 0 is S-enabled, one has to check whether a token can be produced on an empty incoming edge of the Or-join by executing only Or-splits, And- and Xor-gateways but not Or-joins. To see whether an Or-join of depth 1 is S-enabled, one has to check whether a token can be produced on an empty incoming edge of the Or-join by executing only Or-splits, And- and Xor-gateways and Or-joins of depth 0. Consider for example Fig. 3. In part (a), $j1$ has depth 0 and is S-enabled, whereas $j2$ has depth 1 and is not S-enabled. In part (b), both $j1$ (depth 0) and $j2$ (depth 1) are S-enabled.

### 3.2 The Graph-Based View

An alternative interpretation of 'may arrive' is graph-based and was indicated in the BPMN 1.0 specification [5][2]: A token may arrive on an edge $e$ if there is an edge $e'$

[2] Dumas *et al.*[2] have proposed a different graph-based interpretation.

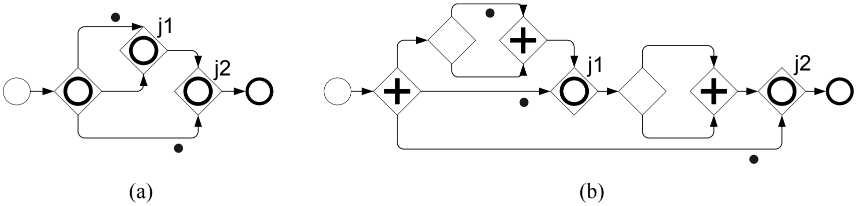


**Fig. 3.** Examples for the state-space and graph-based interpretation

that has a token in $s$ and there is a directed path from $e'$ to $e$ in the workflow graph. This interpretation has the advantage of a straight-forward formalization and simple and efficient enactment:

**Definition 2.** *Let G be an acyclic workflow graph. An Or-join j is* P-enabled *('P' stands for 'path') in a state s if* $s(e) \geq 1$ *for some incoming edge e of j and if for each incoming edge* $e'$ *of j with* $s(e') = 0$ *and for each edge* $e''$ *of the graph with* $s(e'') \geq 1$*, there is no directed path from* $e''$ *to* $e'$*.*

In Fig. 3(a), $j1$ is P-enabled, whereas $j2$ is not. In Fig. 3(b), neither $j1$ nor $j2$ is P-enabled. Hence, S-enabledness and P-enabledness coincide in Fig. 3(a) but they differ in Fig. 3(b). The reason is that the graph in Fig. 3(b) contains deadlocks. The deadlocks prevent tokens to move to the empty incoming edges of the Or-joins. While the state-space interpretation is aware of the deadlocks, the graph-based interpretation is not. Since such deadlocks are usually considered modeling errors, the difference of the two semantics on such models is not substantial. In fact, we can show that the state-space and graph-based interpretation coincide on sound acyclic graphs:

**Definition 3.** *Given a workflow graph G, we say that two semantics X and Y* coincide under soundness *if the following statements hold:*

1. *G is X-sound if and only if it is Y-sound.*
2. *If G is X-sound (or equivalently Y-sound), then a state is X-reachable whenever it is Y-reachable.*
3. *If G is X-sound (or equivalently Y-sound), then an Or-join is X-enabled in an X-reachable (equivalently: Y-reachable) state whenever it is Y-enabled.*

**Theorem 1.** *P-semantics and S-semantics coincide under soundness on acyclic workflow graphs.*

*Proof.* The proof is included in an extended version of this paper. We sketch here the main idea: If an Or-join $j$ is not P-enabled because there is a token upstream of an empty incoming edge of $j$, then it can be shown that that token can be moved to that empty incoming edge of $j$, provided the graph is deadlock-free. Then, $j$ is also not S-enabled.

Conversely, if $j$ is not S-enabled because a token can be brought to an empty incoming edge of $j$, then that token must be located upstream of the empty incoming edge of $j$ and hence $j$ is not P-enabled.

Note that P-semantics can be enacted in linear time in the number of edges of the graph, i.e., it can be determined in linear time whether an Or-join is enabled in a given state.

### 3.3 Dead Path Elimination

In this section, we show that P-semantics essentially coincides with semantics that is implied by *Dead Path Elimination* (DPE) for BPEL flows. Dead path elimination determines the enabledness of an Or-join in an acyclic graph by inspection of its incoming edges only. This works as follows. Let $G$ be an acyclic workflow graph. A *DPE-state* is again a distribution of tokens over the edges of $G$, however each token has now a value, which is either *true* or *false*. Initially there is a *true* token on the initial edge and no token elsewhere. Each gateway waits for a token, *true* or *false*, on each incoming edge. As soon as all incoming edges have a token, these tokens are consumed and a token is produced on each outgoing edge. If all inputs are false, all outputs are false as well. Upon receipt of a true token, an Or-split produces *true* tokens on a nonempty subset of outgoing edges and *false* tokens on all other edges. The Xor-split and the And-split can be thought of as special cases of the Or-split: the former always produces exactly one *true* token, whereas the latter always produces only *true* tokens. The value of the token that is produced by a join is determined by applying the logical function of the join to the inputs, i.e., an Or-join applies the Or-function, the And-join the And-function and the Xor-join the Xor-function.

Figure 4 shows an example of dead path elimination. The labeling represents an entire DPE execution. An intermediate reachable DPE state is represented by the tokens shown in italics. The following lemma characterizes reachable DPE states. It can be easily shown by induction on the reachable DPE states.

**Lemma 1.** *Let $G$ be acyclic. Then for each reachable DPE-state $x$ and each edge $e$ of $G$, exactly one of the following three statements is true: (1) $e$ has a token in $x$, (2) some edge $e' < e$ has a token or (3) some edge $e' > e$ has a token.*

We say that a DPE step is a *true* step, denoted $x \xrightarrow{n} x'$, if at least one true token is consumed, otherwise we say it is an *elimination* step, denoted $x \overset{n}{\dashrightarrow} x'$. If $x'$ can be reached from $x$ through elimination steps only (zero or more), we write $x \overset{*}{\dashrightarrow} x'$. Furthermore, we write $x \overset{\max}{\dashrightarrow} x^*$ if $x \overset{*}{\dashrightarrow} x^*$ and $x^*$ does not enable any further elimination step. It is not difficult to see that $x^*$ always exists and is unique. We say that a step $x \overset{n}{\dashrightarrow} x'$ is *unsound* if either $n$ is an And-join that consumes at least one false token in this step or $n$ is an Xor-join that consumes more than one true token in this step. We say that the graph $G$ is *DPE-sound* if no reachable DPE-state enables an unsound step. Note that if the graph is DPE-sound, all Xor- and And-joins can be replaced by Or-joins without changing the behavior.

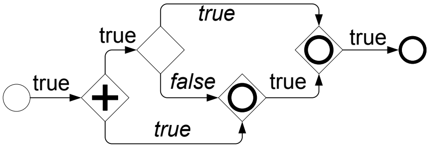


**Fig. 4.** An example for dead path elimination

We show now that for sound acyclic graphs, P-execution and DPE execution mutually simulate each other in a strong sense. To this end, we say that a safe P-reachable state $s$ and a reachable DPE-state $x$ *correspond*, denoted $s \sim x$, if for all edges $e$, $s$ has a token on $e$ if and only if $x$ has a true token on $e$. Thus a state may correspond to more than one DPE-state. DPE semantics and P-semantics coincide if elimination steps are hidden. This can be formalized as follows:

**Theorem 2.** *Let G be an acyclic workflow graph, s be P-reachable state, x a reachable DPE-state and n a node of G.*

1. *The initial state and the initial DPE-state of G correspond.*
2. *Let $x \xrightarrow{n} x'$ be a sound DPE step and $s \sim x$. Then, there exists a state $s'$ such that $s \xrightarrow{n} s'$ and $s' \sim x'$.*
3. *$s \sim x$ and $x \overset{*}{\dashrightarrow} x'$ implies $s \sim x'$,*
4. *Let G be P-safe. Then, $s \sim x$, $s \xrightarrow{n} s'$ and $x \overset{max}{\dashrightarrow} x^*$ implies that there exists a DPE state $x'$ such that $x^* \xrightarrow{n} x'$ and $s' \sim x'$.*
5. *G is P-sound if and only if G is DPE-sound.*

*Proof.* The proof is included in an extended version of this paper. We sketch the main ideas here. For part 2, where $s \sim x$, one can show by help of Lemma 1 that a false token on an edge $e$ in $x$ implies that there is not only no token on $e$ in $s$ but also no token in $s$ on any edge upstream from $e$. It immediately follows that an OR-join that is enabled in $x$ is also enabled in $s$.

For part 4, consider an Or-join $n$ that is P-enabled in $s$. If $n$ has an empty incoming edge $e$ in $s$, it can be shown, again by help of Lemma 1, that there must be a false token upstream of $e$ in $x$. Because false tokens move as far as possible when going from $x$ to $x^*$, it can be shown that the Or-join is then enabled in $x^*$.

## 4 Semantics for Cyclic Workflow Graphs

In this section, we extend the graph-based semantics to deal with cyclic workflow graphs. While the graph-based semantics addresses the first fundamental problem mentioned in Sect. 1, we encounter the second problem mentioned in Sect. 1 when we want to extend the semantics to cyclic graphs: Figure 5 shows an example of a graph with two mutually dependent Or-joins, which is known as a 'vicious circle'. Applying a state-space interpretation, $j1$ is enabled if and only if $j2$ is not enabled and $j2$ is enabled if and only if $j1$ is not enabled. If we apply the graph-based interpretation as defined in Def. 2, then the state in Fig. 5 is a deadlock. This may be satisfactory because it is not clear what the intended behavior of the graph in Fig. 5 is and hence what semantics it should have. As it defines a deadlock, it can be sorted out by static analysis as a modeling error. However, we show in the next subsection that this argument does not apply to all vicious circles.

### 4.1 Block-Based Semantics for Separable Graphs

Although, it seems that the example in Fig. 5 does not present a serious problem, there are more natural workflow graphs that exhibit the same problem. Figure 6 shows a *well-structured* workflow graph, i.e., it is composed of matching pairs of splits and joins. In

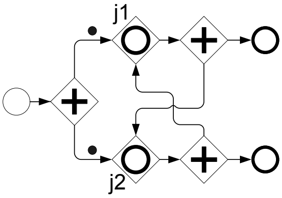


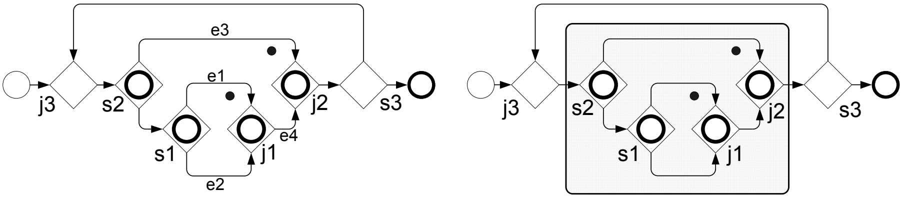
**Fig. 5.** A symmetric vicious circle

contrast to the previous example in Fig. 5, its not difficult to imagine a real business process behind it—provided that suitable tasks are inserted at the edges of the graph. However, the informal semantics produces the same problems as for the example in Fig. 5: The join $j2$ should wait for the token at $e1$ to arrive at $e4$ and the join $j1$ should wait for the token at $e3$ to arrive at $e2$, which can go there via $s3$ and $j3$. This is also reflected by the graph-based semantics as in Def. 2, now applied to a cyclic graph, which interprets the state shown as a deadlock. Previous semantic proposals [2, 3, 8] either define this as a deadlock (neither $j1$ nor $j2$ is enabled) or a nondeterministic choice between the two joins (both, $j1$ and $j2$ are enabled but execution of one disables the other). Moreover, note that a execution of $j2$ implies that the graph is then unsound.

We believe that neither of these two options is reasonable for this graph. It is a well-structured graph, in fact it can be thought of a BPMN representation of a BPEL process where a *flow*, i.e., a sound acyclic subgraph, is nested in a repeat-until-loop, cf. Fig. 6(b). BPEL semantics implies a block-based execution: Before a new iteration of the loop is started, the flow has to complete first. This means that $j2$ waits for $j1$ but not the other way around, i.e., the structure of the process implies an asymmetric dependency between $j1$ and $j2$. We believe that a block-based execution as in BPEL provides the natural semantics for such well-structured graphs. This was also advocated by Mendling and van der Aalst [4].

To capture this semantics, we define a class of workflow graphs, called *separable graphs*, which include well-structured graphs. A *separable graph* is a workflow graph that can be composed from sound acyclic and from sequential *blocks*:

**Definition 4.** *Let G be a workflow graph with nodes N and edges E such that G has a unique end node $n_\omega$ and let $n_\alpha$ be its unique start node.*



(a) (b)

**Fig. 6.** A vicious circle in a well-structured graph

1. *Let $B \subseteq N$ be a nonempty set of nodes and $E_B = E \cap (B \times B)$ the induced set of edges. $B$ is a* block *if the induced subgraph $(B, E_B)$ has a single entry edge and a single exit edge, i.e., there exist edges $e, e' \in E$ with $E \cap ((N \setminus B) \times B) = \{e\}$ and $E \cap (B \times (N \setminus B)) = \{e'\}$; $e$ and $e'$ are called the* entry *and the* exit *edge of $B$, respectively.*
2. *Let $B_0 = N \setminus \{n_\alpha, n_\omega\}$. $B_0$ is a block, called the* root block. *A* decomposition *of $G$ is a set $\mathscr{D}$ of blocks of $G$ such that $B_0 \in \mathscr{D}$ and for each pair of blocks $B, B' \in \mathscr{D}$, we have $B \subseteq B'$, $B' \subseteq B$, or $B \cap B' = \emptyset$. If $B' \subseteq B$, we say $B'$ is a* subblock *of $B$. $B'$ is an* immediate subblock *of $B$ if there is no block $B'' \in \mathscr{D}$ such that $B' \subset B'' \subset B$.*
3. *Let $B \in \mathscr{D}$ be a block. The* abstraction *of $B$ with respect to $\mathscr{D}$ is the workflow graph that results from taking the graph $(B, E_B)$, replacing each immediate subblock $B' \in \mathscr{D}$ of $B$ by a fresh task and adding a fresh start node and a fresh end node that are connected to the entry and the exit of the block respectively. $B$ is* sequential *if each gateway in its abstraction has Xor logic; $B$ is a* flow *if its abstraction is acyclic and sound. $G$ is* separable *if there is a decomposition $\mathscr{D}$ such that each block in $\mathscr{D}$ is either a flow or sequential; $\mathscr{D}$ is then called a* separating *decomposition of $G$.*

**Definition 5.** *Let $G$ be a separable workflow graph and $\mathscr{D}$ a separating decomposition of $G$. An Or-join $j$ is $\mathscr{D}$-enabled in a state $s$ if it is P-enabled in $s$ with respect to the smallest block in $\mathscr{D}$ that contains $j$.*

Note that in a separable workflow graph, the smallest block that contains an Or-join is not sequential and hence must be acyclic.

**Lemma 2.** *Let $G$ be a separable workflow graph and $\mathscr{D}$ a separating decomposition. Then $G$ is $\mathscr{D}$-sound.*

*Proof.* We say that a block is *active* in a state $s$ of $G$ if some edge $e \in E_B$ of the block has a token in $s$. It can be shown by structural induction on $\mathscr{D}$ that a block has no deadlock and lack of synchronization provided that its subblocks are sound and provided that its entry edge never gets a token while the block is active. Since the top-level block, i.e., the workflow graph $G$ itself, never gets more than one token, it follows that $G$ is $\mathscr{D}$-sound.

$\mathscr{D}$-semantics does not seem satisfactory since a decomposition is used to define it. However, the semantics does not depend on the particular decomposition. For example, the separating decomposition $\mathscr{D}_1 = \{B_0, B_1\}$ shown in Fig. 6(b), where $B_0 = \{s1, j1, s2, j2, s3, j3\}$ and $B_1 = \{s1, j1, s2, j2\}$ induces the same semantics as $\mathscr{D}_2 = \{B_0, B_1, B_2\}$ where $B_2 = \{s1, j1\}$.

**Proposition 1.** *Let $\mathscr{D}$ and $\mathscr{D}'$ be two separating decompositions of $G$. Then, $\mathscr{D}$-semantics and $\mathscr{D}'$-semantics coincide on $G$.*

*Proof.* The proof is included in an extended version of this paper. The main idea is the following. The essential proof obligation is to show that an Or-join $j$ is enabled with respect to a containing block $X$ whenever it is enabled with respect to another containing block $Y$. It can be shown that one can restrict to the case $Y \subseteq X$. One direction is trivial. For the other direction, suppose that $j$ is enabled in $Y$ but not in $X$. Then, there is a token upstream of $j$ in $X \setminus Y$. Because there is no deadlock (Lemma 2), it can be shown that we can then move the token to the entry of $Y$ and then further to the non-empty incoming edge of $j$, which would cause a lack of synchronization, which contradicts Lemma 2.

Although this block-based semantics gives a reasonable meaning to separable graphs that is aligned with block-based execution as in BPEL, its description is not yet fully satisfactory because one needs to find a decomposition in order to figure out the enabledness of an Or-join. We will see later that it is not necessary to be aware of the blocks. The same semantics can be defined in a different way that does not refer to blocks.

### 4.2 A New Semantics

We consider again the well-structured graph in Fig. 6. The block-based semantics suggests that the dependency between the joins $j1$ and $j2$ is not as symmetric as in Fig. 5. The structure suggests that $j2$ should wait for $j1$ but not vice versa. How can we characterize this asymmetry? Let us compare the path from $e1$ to $e4$ with the path from $e3$ to $e2$. The path from $e1$ to $e4$ shows how the token on $e1$ catches up to its 'sibling token' on $e3$. However, if we move the token from $e3$ to $e2$, then this starts a new iteration of the loop, causing the synchronization of two tokens from different rounds. Starting a new round is witnessed for example by the fact that the path from $e3$ to $e2$ visits the gateway $s1$ because $s1$ has also a path to $e1$ where there is already a token waiting. Executing $s1$ therefore could cause a second token on $e1$ which would manifest the synchronization error. Clearly, such a new iteration would also be started if the path visits the waiting join itself. For example, the join $j1$ should not wait for the token on $e1$ to arrive at $e2$. Any corresponding path from $e1$ to $e2$ goes through $j1$ and should therefore not be considered. We therefore define the semantics as follows:

**Definition 6.** *An Or-join $j$ is* Q-enabled *in state $s$ if*

1. *there is an incoming edge $e$ of $j$ such that $s(e) \geq 1$ and*
2. *for each edge $e'$ of the graph with $s(e') \geq 1$, we have: If there is a path from $e'$ to some incoming edge $e$ of $j$ with $s(e) = 0$ that does not visit $j$, then there is a path from $e'$ to some incoming edge $e$ of $j$ with $s(e) > 0$ that does not visit $j$.*

This definition implicitly defines *inhibiting paths* to $j$, i.e., a path from a token an empty incoming edge of $j$ such that the path does not visit $j$. Furthermore, an *anti-inhibiting path* to $j$ is a path from a token to a non-empty incoming edge of $j$ such that the path does not visit $j$. An Or-join $j$ has to wait only for those token that have an inhibiting path but no anti-inhibiting path to $j$. Note that there is no anti-inhibiting path in Fig. 6(a) to $j2$ from the token on $e1$ because the path from $e1$ to $e3$ visits $j2$.

We show now that Q-semantics and P-semantics coincide on sound acyclic graphs.

**Proposition 2.** *P-semantics and Q-semantics coincide on P-sound acyclic workflow graphs.*

**Theorem 3.** *Let $G$ be a separable workflow graph and $\mathscr{D}$ a separating decomposition. Then $\mathscr{D}$-semantics and Q-semantics coincide on $G$.*

Note that the compliance with block-based semantics is related to the fact that the Q-enabledness of an Or-join $j$ does not depend on the tokens outside the smallest block that contains $j$. In other words, the Or-join behavior is non-local only within that block. This remains true for non-separable graphs under a mild syntactic restriction[3]. This locality of Q-semantics with respect to such a block makes the behavior of a larger graph easier to understand provided it contains smaller blocks. Moreover, this means that such blocks can be understood, executed and analyzed in isolation, i.e., they constitute logically atomic parts of the workflow graph in Q-semantics. Furthermore, refining a task with such a block or replacing a block with another block does not change the behavior of the surrounding graph. This eases constructing, changing and refactoring a workflow graph. In particular, the behavior of a business process does not change when a process model is changed by encapsulating such a block into a subprocess.

## 4.3 Non-separable Graphs

We have argued in the previous section that Q-semantics suits separable graphs well. In this section, we study examples of how non-separable graphs behave under Q-semantics.

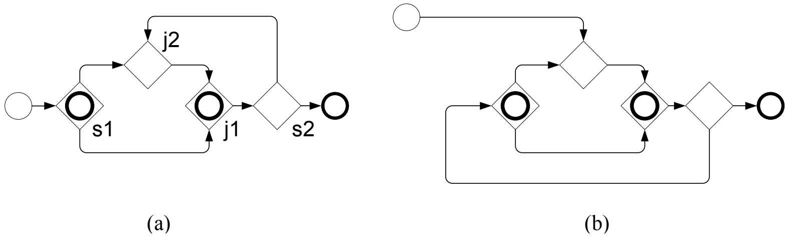


**Fig. 7.** Two non-separable cyclic graphs

We first look at some examples, for which we think that the intended meaning is clear and also achieved by Q-semantics. Figure 7 shows two cyclic graphs where an Or-region, i.e., a pair of an Or-split and an Or-join, has an additional entry on one of the paths between the split and join. Figure 8(a) can be seen as an Or-region with two parallel additional entries. Figure 7(a) and (b) can be seen as special cases of Fig. 8(a) where one additional entry was omitted. The correspondence between Fig. 7(a) and Fig. 8(a) is given by the labeling of the gateways. Figure 8(b) shows how an Or-join can be used to glue a flow with a well-structured loop. All these examples are Q-sound. The reader may verify that Q-semantics indeed produces the desired behavior for these examples.

The symmetric vicious circle in Fig. 5 still creates a deadlock in Q-semantics. As we argued before, we do not consider this as a problem. The intended behavior is not clear and therefore, the graph can be rejected by static analysis. Figure 9(a) shows a graph where the situation seems to be different at first sight. The dependency between the two Or-joins $j1$ and $j2$ does not seem to be symmetric. In fact, the graph is similar to the graph in Fig. 7(a), they only differ in the logic of $j2$. However, Q-semantics defines the

[3] One has to require that for every incoming edge $e$ of an Or-join $j$ contained in the block, there is a path from the start node to $e$ that does not visit $j$.

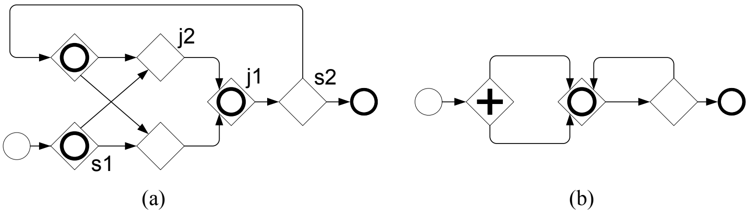


**Fig. 8.** Two more sound cyclic graphs

state shown as a deadlock. This may contrast our intuition because one might think that this graph should have the same behavior as the graph in Fig. 7(a), so that *j*1 waits for *j*2 but not the other way around. However, this intuition may be caused by the layout of the graph: Fig. 9(b) shows how the same graph can be re-drawn. Now, the layout suggests that *j*2 should wait for *j*1 and not the other way around. Thus, the dependency is more symmetric than these two layouts suggest and in fact, the graph can also be re-drawn as shown in Fig. 9(c) to resemble more the graph in Fig. 5. Again, we think that the intended behavior is not clear and that the deadlock can be used to reject the graph in static analysis.

How to treat such 'vicious circles' in practice can be seen as a meta-issue, which is fairly independent from the discussion in this paper. The deadlocks produced by these examples can be detected at analysis time or at runtime (cf. [2]) and they can be treated in a different way by an engine or tool as defined through a 'meta-semantics'. In particular, an engine could apply a more *optimistic approach* [2, 3] and resolve the deadlocks through a non-deterministic choice between the Or-joins if there are reasons to do so. This approach clearly remains possible on top of Q-semantics.

### 4.4 Enactment

In this section, we show that Q-semantics of an Or-join can be enacted in linear time.

If an Or-join *j* has at least one token on an incoming edge, we have to determine whether there are any inhibiting and anti-inhibiting paths to *j*. The algorithm consists of two parts. First we mark all edges of the graph that contribute to anti-inhibiting paths. Those can be determined by backward reachability search starting from the non-empty incoming edges of *j*. We mark all those edges in red. We stop exploration when

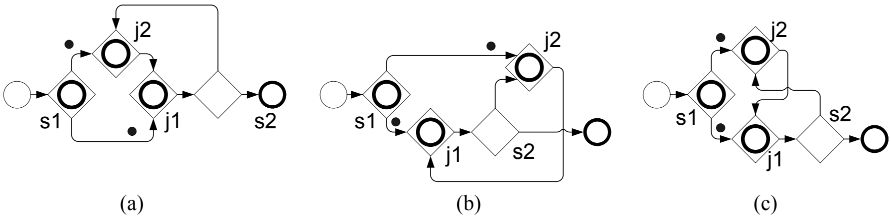


**Fig. 9.** Three different layouts of the same cyclic graph

**Procedure 1.** Returns *true* iff Or-join $j$ is Q-enabled in state $s$.

**IsEnabled(Workflow graph** $G$**, State** $s$**, Or-join** $j$**)**

```
Red := $\{e \mid e$ is an incoming edge of $j$ such that $s(e) > 0\}$
while there exist an edge $e = (n_1, n_2) \in$ Red and $e' = (n_3, n_1) \notin$ Red such that $n_1 \neq j$ do
    Red := Red $\cup \{e'\}$
Green := $\{e \mid e$ is an incoming edge of $j$ such that $s(e) = 0\}$
while there exist an edge $e = (n_1, n_2) \in$ Green and $e' = (n_3, n_1) \notin$ (Green $\cup$ Red) such that $n_1 \neq j$ do
    Green := Green $\cup \{e'\}$
return (Green $\cap \{e \mid s(e) > 0\} = \emptyset$).
```

we reach $j$ itself in order not to mark the empty incoming edges of $j$. The second part of the algorithm marks all edges in green that are not red already and that are backward-reachable from any empty incoming edge of $j$. Again, we stop exploration when we reach $j$ itself in order not to mark any non-empty incoming edge of $j$. This part computes the inhibiting paths. The Or-join $j$ is then enabled if and only if there is no token on some green edge. The Pseudo-code in Procedure 1 makes this algorithm more precise. It is not difficult to see that this algorithm can be implemented to run in linear time in the size of the workflow graph. We conclude:

**Theorem 4.** *Let G be a workflow graph. It can be computed in linear time (in the size of G) whether an Or-join is Q-enabled in a given state of G.*

## 5 Conclusion

We have presented a new semantics for the Or-join in workflow graphs. We have argued that a graph-based semantics gives rise to a straight-forward formalization as well as an efficient enactment. Furthermore, we have pointed out that reasonable process graphs exist where 'vicious circles' arise and argued that those should neither result in a deadlock nor be resolved by non-deterministic choice. We have shown that a natural semantics can be defined for those cases by extending the graph-based semantics for acyclic graphs to general workflow graphs. Our semantics is aligned with block-based execution of well-structured graphs. We have shown various examples of cyclic graphs that are not well-structured but still receive an intuitive semantics in our proposal. Nevertheless, there are cases where mutually dependent Or-joins create a deadlock in our semantics. We have argued that in those cases, the intended meaning is not clear and hence they should be sorted out by static analysis.

As usual, unsoundness is considered as a modeling error in this paper. In fact, lack of synchronization can cause undesired race conditions for Or-join enabledness in Q-semantics. Extending our semantics to models which treat lack of synchronization as a feature rather than an error is beyond the scope of this paper.

Our semantics is included in the draft for the BPMN 2.0 standard [6].

*Acknowledgement.* We would like to thank Alistair Barros and the other members of the BPMN 2.0 RFP taskforce, in particular Matthias Kloppmann and Stephen A. White, moreover we thank Ekkart Kindler, Jussi Vanhatalo and Thomas Hettel for helpful discussions. We also would like to thank the anonymous reviewers for their constructive comments to improve the presentation.

## References

1. Börger, E., Sörensen, O., Thalheim, B.: On defining the behavior of OR-joins in business process models. J. UCS 15(1), 3–32 (2009)
2. Dumas, M., Großkopf, A., Hettel, T., Wynn, M.T.: Semantics of standard process models with OR-joins. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 41–58. Springer, Heidelberg (2007)
3. Kindler, E.: On the semantics of EPCs: Resolving the vicious circle. Data Knowl. Eng. 56(1), 23–40 (2006)
4. Mendling, J., van der Aalst, W.M.P.: Formalization and verification of EPCs with OR-joins based on state and context. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 439–453. Springer, Heidelberg (2007)
5. OMG. Business process modeling notation (BPMN) version 1.0, OMG document number dtc/06-02-01. Technical report (2006)
6. OMG. Business process model and notation (BPMN) version 2.0, OMG document number dtc/2010-05-03. Technical report (2010)
7. van der Aalst, W.M.P., Desel, J., Kindler, E.: On the semantics of EPCs: A vicious circle. In: EPK, GI-Arbeitskreis Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten, pp. 71–79 (2002)
8. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: yet another workflow language. Inf. Syst. 30(4), 245–275 (2005)
9. Wynn, M.T., Edmond, D., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Achieving a general, formal and decidable approach to the OR-join in workflow using reset nets. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 423–443. Springer, Heidelberg (2005)

# From People to Services to UI: Distributed Orchestration of User Interfaces

Florian Daniel[1], Stefano Soi[1], Stefano Tranquillini[1], Fabio Casati[1],
Chang Heng[2], and Li Yan[2]

[1] University of Trento, Povo (TN), Italy
{daniel,soi,tranquillini,casati}@disi.unitn.it
[2] Huawei Technologies, Shenzhen, P.R. China
{changheng,liyanmr}@huawei.com

**Abstract.** Traditionally, workflow management systems aim at alleviating people's burden of coordinating repetitive business procedures, i.e., they coordinate *people*. Web service orchestration approaches, instead, coordinate pieces of software (the *web services*), hiding the human aspects that are intrinsically present in any business process behind the services. The recent emergence of technologies like BPEL4People and WS-HumanTask, which introduce human actors into service compositions, manifest that taking into account the people involved in business processes is however important. Yet, none of these approaches allow one to also develop the *user interfaces* (UIs) the users need to concretely participate in a business process.

With this paper, we want to go one step beyond state-of-the-art workflow management and service composition and propose an original model, language and running system for the composition of distributed UIs, an approach that allows us to bring together UIs, web services and people in a single orchestration logic and tool. To demonstrate the effectiveness of the idea, we apply the approach to a real-world home assistance scenario.

## 1 Introduction

*Workflow management systems* support office automation processes, including the automatic generation of form-based user interfaces (UIs) for executing the human tasks in a process. *Service orchestrations* and related languages focus instead on integration at the application level. As such, this technology excels in the reuse of components and services but does not facilitate the development of UI front-ends for supporting human tasks and complex user interaction needs, which is one of the most time consuming tasks in software development [1].

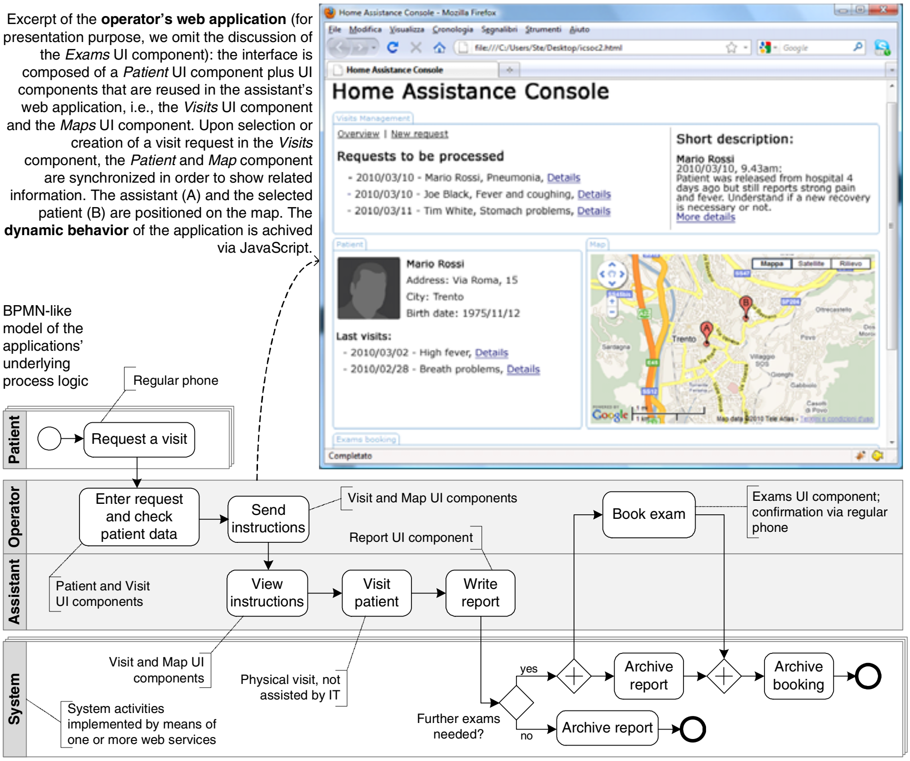
Only recently, *web mashups* [2] have turned lessons learned from data and application integration into lightweight, simple composition approaches featuring a significant innovation: integration at the UI level. Besides web services or data feeds, mashups reuse pieces of UI (e.g., content extracted from web pages or JavaScript UI widgets) and integrate them into a new web page. Mashups, therefore, manifest the need for reuse in UI development and suitable UI component technologies. Interestingly, however, unlike

R. Hull, J. Mendling, and S. Tai (Eds.): BPM 2010, LNCS 6336, pp. 310–326, 2010.

what happened for services, this need has not yet resulted in accepted component-based development models and practices.

This paper tackles the development of applications that require service composition/process automation logic but that also include human tasks, where humans interact with the system via a possibly complex and sophisticated UI that is tailored to help them in performing the specific job they need to carry out. In other words, this work targets the development of mashup-like applications that require process support, including applications that require distributed mashups coordinated in real time, and provides design and tool support for professional developers, yielding an original composition paradigm based on web-based UI components and web services.

This is a common need that today is typically fulfilled by developing UIs in ad hoc ways and using a process engine in the back-end for process automation. As an example, consider the following scenario.


Excerpt of the **operator's web application** (for presentation purpose, we omit the discussion of the *Exams* UI component): the interface is composed of a *Patient* UI component plus UI components that are reused in the assistant's web application, i.e., the *Visits* UI component and the *Maps* UI component. Upon selection or creation of a visit request in the *Visits* component, the *Patient* and *Map* component are synchronized in order to show related information. The assistant (A) and the selected patient (B) are positioned on the map. The **dynamic behavior** of the application is achived via JavaScript.

Home Assistance Console - Mozilla Firefox

File Modifica Visualizza Cronologia Segnalibri Strumenti Aiuto

file:///C:/Users/Ste/Desktop/icsoc2.html

Home Assistance Console

Home Assistance Console

Visits Management

Overview | New request

Requests to be processed

- 2010/03/10 - Mario Rossi, Pneumonia, Details
- 2010/03/10 - Joe Black, Fever and coughing, Details
- 2010/03/11 - Tim White, Stomach problems, Details

Short description:

Mario Rossi
2010/03/10, 9.43am:
Patient was released from hospital 4 days ago but still reports strong pain and fever. Understand if a new recovery is necessary or not.
More details

Patient

Mario Rossi
Address: Via Roma, 15
City: Trento
Birth date: 1975/11/12

Last visits:

- 2010/03/02 - High fever, Details
- 2010/02/28 - Breath problems, Details

Map

Exams booking

Completato

Google
Mappa
Satellite
Rilievo
Trento
A
B
Map data ©2010 Tele Atlas
BPMN-like model of the applications' underlying process logic
Regular phone
Patient
Request a visit
Operator
Enter request and check patient data
Send instructions
Visit and Map UI components
Report UI component
Book exam
Exams UI component; confirmation via regular phone
Assistant
Patient and Visit UI components
View instructions
Visit patient
Write report
System
Visit and Map UI components
Physical visit, not assisted by IT
System activities implemented by means of one or more web services
yes
no
Further exams needed?
Archive report
Archive booking
Archive report

**Fig. 1.** Simplified home assistance process: gray shaded swim lanes are instantiated only once (in form of suitable UIs) and handle multiple instances of white shaded swim lanes

**Scenario.** Figure 1 shows the high-level model of a home assistance process in the Province of Trento we want to aid in one of our projects. A *patient* can ask for the visit of a home assistant (e.g., a paramedic) by calling (via phone) an operator of the assistance service. Upon request, the *operator* inputs the respective details and inspects the patient's data and personal health history in order to provide the assistant with the necessary instructions. There is always one assistant on duty. The *home assistant* views the description, visits the patient, and files a report about the provided service. The report is processed by the *back-end system* and archived if no further exams are needed. If exams are instead needed, the operator books the exam in the local hospital asking confirmation to the patient (again via phone); in parallel, the system archives the report. Upon confirmation of the exam booking, the system also archives the booking, which terminates the responsibility of the home assistance service.

Our goal is to develop an application that supports this process. This application includes, besides the process logic, two mashup-like, web-based control consoles for the operator and the assistant that are themselves part of the orchestration and need to interact with (and are affected by) the evolution of the process. Furthermore, the UI can be itself component-based and created by reusing and combining existing UI components. The two applications, once instantiated, should be able to manage multiple requests for assistance, while the system activities will be instantiated independently for each report to be processed.

**Challenges and contributions.** The scenario requires the coordination of the individual actors in the process and the development of the necessary *distributed* user interface *and* service orchestration logic. Doing so requires (i) understanding how to *componentize UIs and compose them into web applications*, (ii) defining a logic that is able to *orchestrate both UIs and web services*, (iii) providing a language and tool for *specifying distributed UI compositions*, and (iv) developing a runtime environment that is able to *execute distributed UI and service compositions*.

**Structure of the paper**. Implementing the process of the scenario is a non-trivial composition problem. After describing the UI orchestration approach (Section 3), in this paper we show how defining a new type of binding allows us to leverage the standard WSDL [4] language to describe HTML/JavaScript UI components (Section 4). We then build on existing composition languages (in particular WS-BPEL [5]) to introduce the notions of UI components, pages, and actors to support the specification of distributed UI compositions (Section 5). The extended BPEL is compiled to generate the UI composition logic (that runs entirely on the browser, for performance reasons) and the server-side logic that performs service orchestration and distributed UI synchronization. Finally, we extend the Eclipse BPEL editor to support this extension, and we describe a system that is able to execute distributed UI compositions, starting from the extended BPEL specification. These models and tools are integrated in a hosted development and execution platform, called MarcoFlow (Section 6), jointly developed by Huawei Technologies and the University of Trento.

## 2 State of the Art in Orchestrating Services, People and UIs

In most **service orchestration** approaches, such as BPEL [5], there is no support for UI design. Many variations of BPEL have been developed, e.g., aiming at the invocation of REST services [6] or at exposing BPEL processes as REST services [7]. IBM's Sharable Code platform [8] follows a slightly different strategy in the composition of REST and SOAP services and also allows the integration of user interfaces for the Web; UIs are however not provided as components but as ad-hoc Ruby on Rails HTML templates.

**BPEL4People** [9] is an extension of BPEL that introduces the concept of people task as first-class citizen into the orchestration of web services. The extension is tightly coupled with the **WS-HumanTask** [10] specification, which focuses on the definition of human tasks, including their properties, behavior and operations used to manipulate them. BPEL4People supports people activities in form of inline tasks (defined in BPEL4People) or standalone human tasks accessible as web services. In order to control the life cycle of service-enabled human tasks in an interoperable manner, WS-HumanTask also comes with a suitable coordination protocol for human tasks, which is supported by BPEL4People. The two specifications focus on the coordination logic only and do not support the design of the UIs for task execution.

The systematic development of web interfaces and applications has typically been addressed by the web engineering community by means of **model-driven web design approaches**. Among the most notable and advanced model-driven web engineering tools we find, for instance, WebRatio [11] and VisualWade [12]. The former is based on a web-specific visual modeling language (WebML), the latter on an object-oriented modeling notation (OO-H). Similar, but less advanced, modeling tools are also available for web modeling languages/methods like Hera, OOHDM, and UWE. These tools provide expert web programmers with modeling abstractions and automated code generation capabilities for complex web applications based on a hyperlink-based navigation paradigm. WebML has also been extended toward web services [13] and process-based web applications [14]; reuse is however limited to web services and UIs are generated out of HTML templates for individual components.

A first approach to component-based UI development is represented by **portals and portlets** [15], which explicitly distinguish between UI components (the portlets) and composite applications (the portals). Portlets are full-fledged, pluggable Web application components that generate document markup fragments (e.g., (X)HTML) that can however only be reached through the URL of the portal page. A portal server typically allows users to customize composite pages (e.g., to rearrange or show/hide portlets) and provides single sign-on and role-based personalization, but there is no possibility to specify process flows or web service interactions (the new WSRP [16] specification only provides support for accessing remote portlets as web services). Also **JavaServer Faces** [17] feature a component model for reusable UI components and support the definition of navigation flows; the technology is however hardly reusable in non-Java based web applications, navigation flows do not support flow controls, and there is no support for service orchestration and UI distribution.

Finally, the web mashup [2] phenomenon produced a set of so-called **mashup tools**, which aim at assisting mashup development by means of easy-to-use graphical user interfaces targeted also at non-professional programmers. For instance, Yahoo! Pipes (http://pipes.yahoo.com) focuses on data integration via RSS or Atom feeds via a data-flow composition language; UI integration is not supported. Microsoft Popfly (http://www.popfly.ms; discontinued since August 2009) provided a graphical user interface for the composition of both data access applications and UI components; service orchestration was not supported. JackBe Presto (http://www.jackbe.com) adopts a Pipes-like approach for data mashups and allows a portal-like aggregation of UI widgets (so-called mashlets) visualizing the output of such mashups; there is no synchronization of UI widgets or process logic. IBM QEDWiki (http://services.alphaworks.ibm.com/qedwiki) provides a wiki-based (collaborative) mechanism to glue together JavaScript or PHP-based widgets; service composition is not supported. Intel Mash Maker (http://mashmaker.intel.com) features a browser plug-in which interprets annotations inside web pages allowing the personalization of web pages with UI widgets; service composition is outside the scope of Mash Maker.

In the mashArt [3] project, we worked on a so-called universal integration approach for UI components and data and application logic services. MashArt comes with a simple editor and a lightweight runtime environment running in the client browser and targets skilled web users. MashArt aims at simplicity: orchestration of distributed (i.e., multi-browser) applications, multiple actors, and complex features like transactions or exception handling are outside its scope. The CRUISe project [17] has similarities with mashArt, especially regarding the componentization of UIs. Yet, is does not support the seamless integration of UI components with service orchestration, i.e., there is no support for complex process logic. CRUISe rather focuses on adaptivity and context-awareness. Finally, the ServFace project [19] aims at supporting even unskilled web users in composing web services that come with an annotated WSDL description. Annotations are used to automatically generate form-like interfaces for the services, which can be placed onto one or more web pages and used to graphically specify data flows among the form fields. The result is a simple, user-driven web service orchestration. None of these projects, however, supports the coordination of multiple different actors inside a same process, and none of the approaches discussed in this section supports the distribution of UIs over multiple browsers.

## 3 Distributed User Interface Orchestration: Approach

If we analyze the home assistance scenario, we see that the envisioned application (as a whole) is *highly distributed* over the Web: The UIs for the actors participating in the application are composed of UI components, which can be components developed in-house (like the *Visit* component) or sourced from the Web (like the *Map* component); service orchestrations are based on web services. The UI exposes the state of the application and allows users to interact with it and to enact service calls. The two applications for the operator and the assistant are instantiated in different web browsers, contributing to the distribution of the overall UI and raising the need for synchronization.

The *key idea* to approach the coordination of (i) UI components inside web pages, (ii) web services providing data or application logic, and (iii) individual pages (as well as the people interacting with them) is to split the coordination problem into two layers: *intra-page UI synchronization* and *distributed UI synchronization and web service orchestration*.

We have seen that many of the research challenges raised by the home assistance application are not yet covered adequately by existing works. Especially the aim of providing a single development approach that is able to cover all development aspects in an integrated fashion poses requirements to the *whole life cycle* of UI orchestrations, especially in terms of design, deployment and execution support.

Indeed, supporting the *design* of distributed UI orchestrations such as the ones needed in the example scenario requires:

- Defining a new type of component, the **UI component**, which is able to modularize pieces of UI and to abstract their external interfaces in a way that conforms to the standard WSDL [4] format for service descriptions (to keep compatibility with the BPEL editors and language). We deal with the novel technological aspects introduced by UI components by defining a new type of WSDL binding, which allows us to specify how to translate the abstract WSDL operation descriptions into JavaScript function calls.
- Bringing together the needs of **UI synchronization and service orchestration** in one single language. UIs are typically event-based (e.g., user clicks or key strokes), while service invocations are coordinated via control flows. In this paper, we show how to extend the standard BPEL language in order to support UIs (BPEL comes with graphical editors and ready, off-the-shelf runtime engines that we want to reuse, not re-implement). We call this extended language *BPEL4UI*.
- Implementing a suitable, graphical **design environment** that allows developers to visually compose services and UI components and to define the grouping of UI components into pages. We achieve this by extending the Eclipse BPEL editor with UI-specific modeling constructs that are able to generate BPEL4UI in output.

Supporting the *deployment* of UI orchestrations requires:

- **Splitting the BPEL4UI specification** into the two orchestration layers for intra-page UI synchronization and distributed UI synchronization and web service orchestration. For the former we use a lightweight UI composition language (UICL), which allows specifying how UI components are coordinated in the client browser. For the latter we rely on standard BPEL.
- Providing a set of **auxiliary web services** that are able to mediate communications between the client-side UI composition logic and the BPEL logic. We achieve this layer by automatically generating and deploying a set of web services that manage the UI-to-BPEL and BPEL-to-UI interactions.

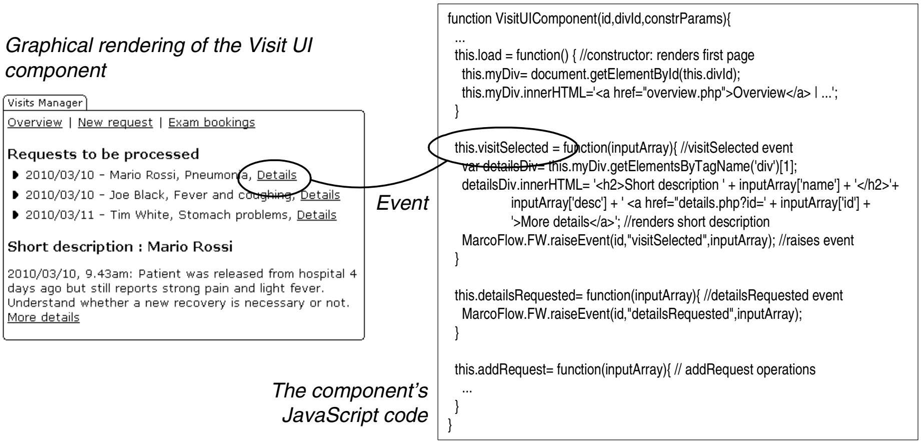
Supporting the *execution* of UI orchestrations requires:

- Providing a **client-side runtime framework** for UI synchronization that is able to instantiate UI components inside web pages and to propagate events from one component to other components, starting from a UICL specification. Events of a UI component may be propagated to components running in the same web page or in other pages of the application and to web services.
- Providing a **communication middleware layer** that is able to run the generated auxiliary web services for UI-to-BPEL and BPEL-to-UI communications. We implement this layer by reusing standard web server technology able to instantiate SOAP and RESTful web services.
- Setting up a **BPEL engine** that is able to run standard BPEL process specifications. The engine is in charge of orchestrating web services and distributed UI-UI communications. We rely on standard technology and reuse an existing BPEL engine.

These requirements and the respective hints to our solution show that the main methodological goals in achieving our UI orchestration approach are (i) relying as much as possible on existing *standards*, (ii) providing the developer with only *few and simple new concepts*, and (iii) implementing a runtime architecture that associates each concern to the *right level of abstraction and software tool* (e.g., UI synchronization is handled in the browser, while service orchestration is delegated to the BPEL engine).

## 4 The Building Blocks: Web Services and UI Components

Orchestrating remote application logic and pieces of UI requires, first of all, understanding the exact nature of the components to be integrated. For the integration of application logic, we rely on standard web service technologies, such as WSDL-SOAP services, i.e., remote web services whose external interface is described in WSDL, which supports interoperability via four message-based types of operations: *request-response*, *notification*, *one-way*, and *solicit-response*. Most of today's web services of this kind are *stateless*, meaning that the order of invocation of their operations does not influence the success of the interaction, while there are also *stateful* services whose interaction requires following a so-called business protocol that describes the interaction patterns supported by the service.


Graphical rendering of the Visit UI component
Event
The component's JavaScript code

Visits Manager

Overview | New request | Exam bookings

**Requests to be processed**

- 2010/03/10 - Mario Rossi, Pneumonia, Details
- 2010/03/10 - Joe Black, Fever and coughing, Details
- 2010/03/11 - Tim White, Stomach problems, Details

**Short description : Mario Rossi**

2010/03/10, 9.43am: Patient was released from hospital 4 days ago but still reports strong pain and light fever. Understand whether a new recovery is necessary or not. More details

```
function VisitUIComponent(id,divId,constrParams){
 ...
 this.load = function() { //constructor: renders first page
  this.myDiv= document.getElementById(this.divId);
  this.myDiv.innerHTML='<a href="overview.php">Overview</a> | ...';
 }

 this.visitSelected = function(inputArray){ //visitSelected event
  var detailsDiv= this.myDiv.getElementsByTagName('div')[1];
  detailsDiv.innerHTML= '<h2>Short description ' + inputArray['name'] + '</h2>'+
              inputArray['desc'] + ' <a href="details.php?id=' + inputArray['id'] +
              '>More details</a>'; //renders short description
  MarcoFlow.FW.raiseEvent(id,"visitSelected",inputArray); //raises event
 }

 this.detailsRequested= function(inputArray){ //detailsRequested event
  MarcoFlow.FW.raiseEvent(id,"detailsRequested",inputArray);
 }

 this.addRequest= function(inputArray){ // addRequest operations
  ...
 }
}
```

**Fig. 2.** Graphical rendering and internal logic of a JavaScript/HTML UI component

For the integration of UI, we rely instead on JavaScript/HTML **UI components**, which are simple, stand-alone web applications that can be instantiated and run inside any common web browser. Figure 2 shows an example of UI component (the *Visit* UI component of our reference scenario), along with an excerpt of its JavaScript code. Unlike web services, UI components are characterized by:

– A **user interface.** UI components can be instantiated inside a web browser and can be accessed and navigated by a user via standard HTML. The UI allows the user to interactively inspect and alter the content of the component, e.g., the short description in Figure 2. UI components are therefore *stateful*, and the component's navigation features replace the business protocol needed for services.

- **Events.** Interacting with the UI generates system events (e.g., mouse clicks) in the browser used to manage the update of contents. Some events may be exposed as component events in order to *communicate state changes*. For instance, a click on the *Details* link in Figure 2 launches a *visitSelected* event.
- **Operations.** Operations *enact state changes* from the outside. Typically, we can map the event of one component to the operation of another component in order to synchronize the components' state (so that they show related information).
- **Properties.** The graphical setup of a component may require the setting of *constructor parameters*, e.g., to align background colors or to specify the start page of a component.

In order to make UI components available in BPEL, each component is equipped with a standard WSDL descriptor that describes the events and operations (the constructor is expressed as operation) in terms of one-way and notification WSDL operations, respectively. To support the instantiation and execution of components, we have defined a new *JavaScript binding* for WSDL, which binds the abstract operations to the JavaScript functions of the component. The WSDL-UI descriptor can be used as is by the client-side runtime framework and adapted for its use by the BPEL engine.

## 5 Modeling UI Orchestrations

Specifying a UI orchestration requires modeling two fundamental aspects: (i) the *interaction logic* that rules the passing of data among UI components and web services and (ii) the *graphical layout* of the final application. Supporting these tasks in BPEL requires extending the expressive power of the language with UI-specific constructs.

### 5.1 BPEL4UI: Concepts and Syntax

Figure 3 shows the simplified meta-model of BPEL4UI. Specifically, the figure details all the new modeling constructs necessary to specify UI orchestrations (gray-shaded) and omits details of the standard BPEL language, which are reused as is by BPEL4UI (a detailed meta-model for BPEL can be found in [20]).

In terms of standard BPEL [5], a UI orchestration is a *process* that is composed of a set of associated *activities* (e.g., sequence, flow, if, assign, validate, or similar), *variables* (to store intermediate processing results), *message exchanges*, *correlation sets* (to correlate messages in conversations), and *fault handlers*. The services or UI components integrated by a process are declared by means of so-called *partner links*, while *partner link types* define the roles played by each of the services or UI components in the conversation and the *port types* specifying the operations and messages supported by each service or component. There can be multiple partner links for each partner link type.

Modeling UI-specific aspects requires instead introducing a set of new constructs that are not yet supported by BPEL. The constructs, illustrated in Figure 3, are:

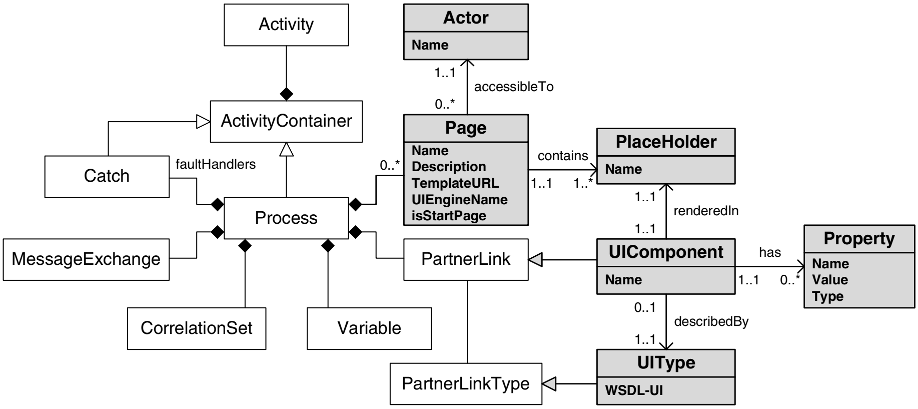


**Fig. 3.** Simplified BPEL4UI meta-model in UML. White classes correspond to standard BPEL constructs [20]; gray classes correspond to constructs for UI and user management.

- **UI type:** The use of UI components in service compositions asks for a new kind of partner link type. Although syntactically there is no difference between web services and UI components (the JavaScript binding introduced into WSDL-UI comes into play only at runtime), it is important to distinguish between services and UI components as their semantics and, hence, their usage in the model will be different. Also, it is necessary to mark UI component types as such, in order to support the generation of standard BPEL, as described in Section 6.

  As exemplified in Figure 4, we specify the new partner link type like a standard web service type (lines 10-13). In order to reflect the events and operations of the UI component, we distinguish the two roles. Lines 1-8 define the necessary name spaces and import the WSDL-UI descriptor of the UI component.

- **Page:** The distributed UI of the overall application consists of one or more web pages, which can host instances of UI components. Pages have a *name*, a *description*, a reference to the pages' *layout template*, the name of the *UI engine* (see Section 6) they will run on, and an indication of whether they are a *start page* of the application or not (similar to the start activity in process models).

  The code lines 16-21 in Figure 4 show the definition of a page called "Operator", along with its layout template and the name of the UI engine on which the page will be deployed; the page is a start page for the process.

- **Place holder:** Each page comes with a set of place holders, which are empty areas inside the layout template that can be used for the graphical rendering of UI components. Place holders are identified by a unique *name*, which can be used to associate UI components.

  Place holders are associated with page definitions and specified as sub-elements, as shown in lines 19-20 in Figure 4.

- **UI component:** UI types can be instantiated as UI components. For instance, there might be one UI type but two different instances of the type running in two different web pages. Declaring a UI component in a BPEL4UI model leads to the creation of an instance of the UI component in one of the pages of the application. Each component is part of one process and has a unique *name*.

We specify UI component partner links by extending the standard partner link definition of BPEL with three new attributes, i.e., *isUiComponent*, *pageName* and *placeHolderName*. Lines 25-31 in Figure 4 show how to declare the *Visit UI component* of our example scenario.

- **Property:** As we have seen in the previous section, UI components may have a constructor that allows one to set configuration properties. Therefore, each UI component may have a set of associated properties than can be parsed at instantiation time of the component. We use simple *name-value* pairs to store constructor parameters.

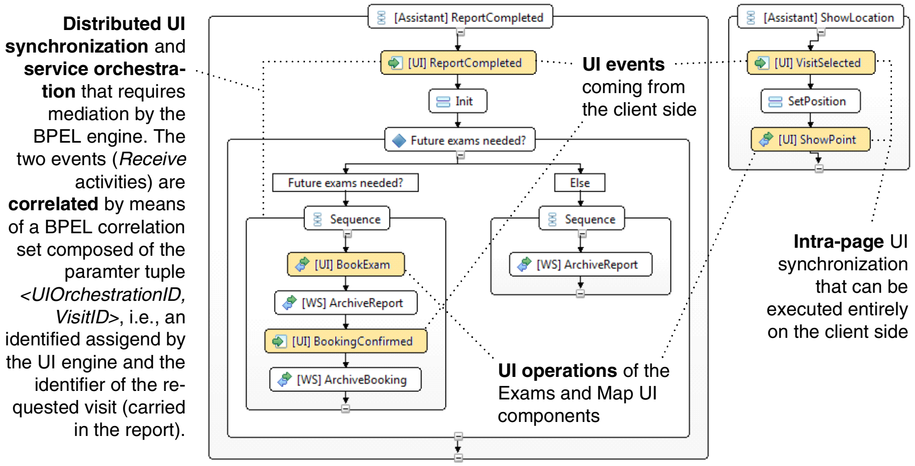
  Properties extend the definition of UI component link types by adding *property* sub-elements to the partner link definition, one for each constructor parameter, as shown in lines 29-30 in Figure 4.

- **Actor:** In order to coordinate the people in a process, pages of the application can be associated with individual actors, i.e., humans, which are then allowed to access the page and to interact with the UI orchestration via the UI components rendered in the page. As for now, we simply associate static actors to pages (using their *names*); yet, actors can easily be assigned also dynamically at deployment time or runtime by associating roles instead of actors and using a suitable user management system.

Actors are added to page definitions by means of the *actorName* attribute, as highlighted in line 18 in Figure 4.

```
 1  <bpel:process name="HomeAssistance"
 2    targetNamespace=www.unitn.it/bpel4ui/HomeAssistance
 3    xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/
 4    executable" xmlns:visit="http://www.unitnt.it/UI/VisitUIComponent" ...>
 5
 6  <bpel:import namespace="http://www.unitn.it/UI/VisitUIComponent"
 7    location="VisitUI.wsdl" importType="http://schemas.xmlsoap.org/wsdl/">
 8  </bpel:import>
 9  ...
10  <bpel:partnerLinkType name="VisitUIComponent">
11    <bpel:role name="Receive" portType="visit:VisitUI_RECEIVE"/>
12    <bpel:role name="Invoke" portType="visit:VisitUI_INVOKE"/>
13  </bpel:partnerLinkType>
14  ...
15  <pages>
16    <page name="Operator" description="Operator's home page"
17      templateURL="http://www.unitn.it/BPEL4UI/operatorLayout.html"
18      uiEngineName="UNITN" isStartPage="yes" actorName="Paul">
19        <placeholder name="marcoflow-left"/>
20        <placeholder name="marcoflow-right"/>
21    </page>
22    ...
23  </pages>
24  <bpel:partnerLinks>
25    <bpel:partnerLink name="VisitUI_Operator"
26      partnerLinkType="VisitUIComponent" myRole="Receive"
27      partnerRole="Invoke" isUiComponent="yes" pageName="Patient"
28      placeHolderName="marcoflow-left">
29        <property name="StartPage" type="xsd:string">New Visit</property>
30        <property name="BackgroundColor" type="xsd:string">white</property>
31    </bpel:partnerLink>
32  </bpel:partnerLinks>
33  ...
34  </bpel:process>
```

**Fig. 4.** Excerpt of the BPEL4UI home assistance process (new constructs in bold)



**Distributed UI synchronization** and **service orchestration** that requires mediation by the BPEL engine. The two events (*Receive* activities) are **correlated** by means of a BPEL correlation set composed of the paramter tuple <*UIOrchestrationID, VisitID*>, i.e., an identified assigend by the UI engine and the identifier of the requested visit (carried in the report).

**UI events** coming from the client side

**UI operations** of the Exams and Map UI components

**Intra-page** UI synchronization that can be executed entirely on the client side

**Fig. 5.** Part of the BPEL4UI model of the home assistance process modeled in the extended Eclipse BPEL editor (these and other *Sequence* constructs run inside a *Flow*)

## 5.2 Modeling the Orchestration Logic

The code example in Figure 4 shows that the UI-specific modeling constructs have a very limited impact on the syntax of BPEL and are concerned with the abstract specification of the layout and the declaration of UI partner links. The actual composition logic relies exclusively on standard BPEL constructs, yet – since UI components are different from web services (e.g., it is important to know in which page they are running) – it is important to understand the effect individual modeling patterns have on the execution of the final application, i.e., the *semantics* of the patterns. As hinted at in Section 2 and illustrated in Figure 5, we distinguish three main design patterns:

- **Intra-page UI synchronization:** The sequence construct in the right part of Figure 5 shows the internals of the *View instructions* task in Figure 1. When the assistant clicks on a visit request, the patient's address is shown on the Google map. In BPEL terms, we receive a message from the *Visit* UI component (the event) and forward it to the operation of the *Map* component, implementing an intra-page UI synchronization. Both UI components involved in the sequence are associated with the page of the assistant. Hence, this kind of UI synchronization can be performed on the *client side* without involving the BPEL engine.
- **Distributed UI synchronization:** The sequence construct in the left part of the figure, instead, contains a distributed synchronization that cannot be executed on the client only, as the two UI components involved in the communication (*Report* and *Exam*) run in different web pages. The event generated upon submission of a new report is processed by the *BPEL engine*, which then decides whether an additional exam needs to be booked by the operator or not.
- **Service orchestration:** The distributed UI synchronization also involves the orchestration of the *Report archiving* and *Booking archiving* web services, as well as some BPEL flow control constructs. For instance, the modeled logic

> checks whether the report expresses the need for further exams or not. In either case, the further processing of the report involves the invocation of either one or both the web services, in order to correctly terminate the handling of a visit request.

The BPEL4UI excerpt in Figure 5 shows that, when modeling a UI orchestration, it is important to keep in mind who communicates with whom and where UI components will be rendered. Depending on these two considerations, the modeled composition logic will either be executed on the client side, in the BPEL engine, or in both layers. For instance, it suffices to associate the *Map* component with a different page so as to turn the intra-page UI synchronization in the right hand side of Figure 5 into a distributed communication and, hence, to require support from the BPEL engine.

**Data transformations.** When composing services or UI components, it is not enough to model the communication flow only. An important and time-consuming aspect is that of transforming the data passed from one component to another. With BPEL4UI we support all data transformation features provided by BPEL by means of its *Assign* activity. This allows us to leverage on technologies, such as XPath, XQuery, XSLT or Java, for the implementation of also very complex data transformations. Yet, the type of data transformation may affect the logic of the UI orchestration. For instance, if the *SetPosition* activity in Figure 5 does not transform data at all or only performs simple parameter mappings (with the BPEL *Copy* construct) the intra-page UI synchronization can be executed in the client browser. If instead a more complex transformation is needed, we rely on the BPEL engine to perform it.

The reason for this choice is that UI synchronization typically involves exchanging only simple data (e.g., parameter-value pairs) and does not require complex transformations like when interacting with web services. This choice allows us to keep the client-side framework as lightweight as possible, while not giving up any data transformation capabilities. The decision of where to transform data is taken based on the nature of the involved partner links and the type of transformation.

**Correlation.** The intra-page UI synchronization in Figure 5 does not involve any *asynchronous* communication pattern or multiple entry points into the process logic. It is therefore not necessary to implement any correlation logic in BPEL4UI in order to propagate the *VisitSelected* event to the *ShowPoint* operation. The correlation of the event and the operation in the two web pages is achieved outside the BPEL engine (in the *UI engine server* in Figure 6) by sharing a common key (the *UIOrchestrationID*) that is carried by each event and used to dispatch events. This kind of correlation is automated in our runtime environment and does not require specific modeling.

The distributed UI synchronization, instead, involves two UI events from two different actors: *ReportCompleted* and *BookingConfirmed.* In this case, it is necessary to configure a so-called *correlation set* (in BPEL terminology) that allows the BPEL engine to understand whether they belong to the same process instance or not. In Figure 5, we use *UIOrchestrationID* and *VisitID* (part of the report) as correlation set.

**Graphical layout.** Defining web pages and associating UI partner links with place holders therein requires implementing suitable HTML templates that are able to host UI components. As we focus on the middleware layer for UI orchestrations, for the layout templates we rely on standard web design instruments and technologies. The

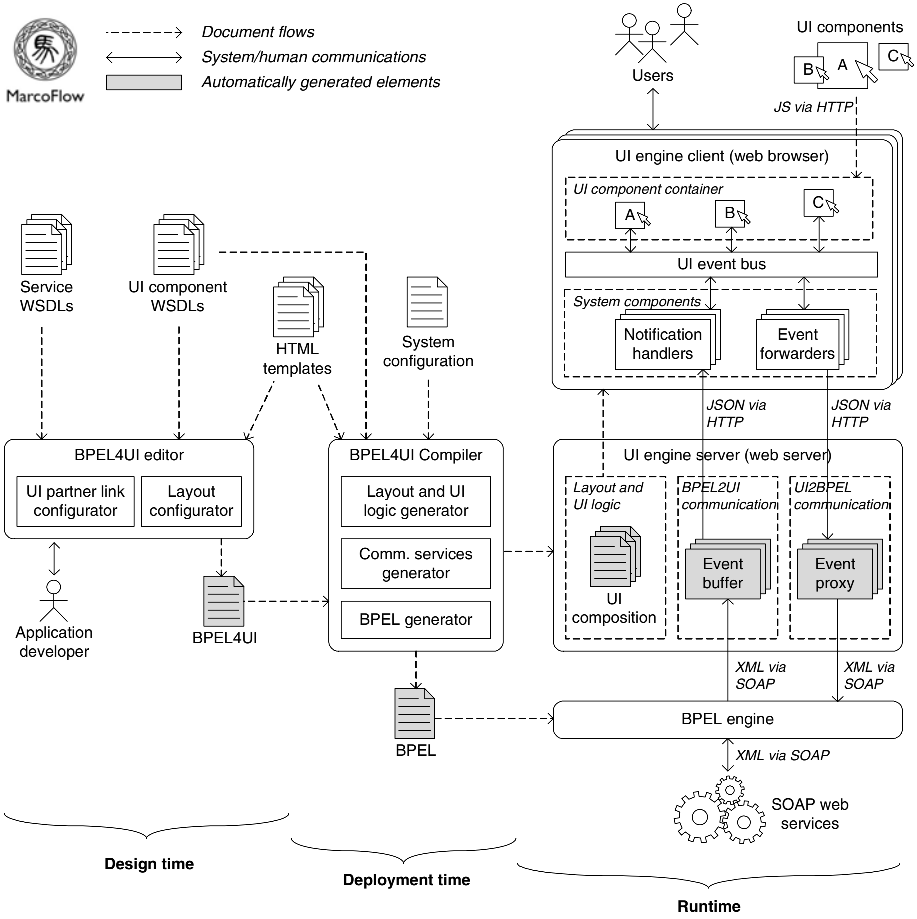


**Fig. 6.** From design time to runtime: overall system architecture of MarcoFlow

only requirement the templates must satisfy is that they provide place holders in form of HTML DIV elements that can be indexed via standard HTML identifiers following a predefined naming convention: *<div id="marcoflow-left">... </div>*.

## 6 Deploying and Running UI Orchestrations

The BPEL4UI language is only a piece of the integrated system for UI orchestration, called *MarcoFlow*. The overall architecture of the system is shown in Figure 6 (for presentation purposes, we discuss a slightly simplified version), which partitions its software components into design time, deployment time, and runtime components.

The **design** part comprises the *BPEL4UI editor* with its *UI partner link configurator* and *layout configurator*. Starting from a set of *web service WSDLs*, *UI component WSDLs*, and *HTML templates* the application developer graphically models the UI orchestration, and the editor generates a corresponding *BPEL4UI* specification in output. The composition logic in Figure 5 has been modeled in our BPEL4UI editor, an extended Eclipse BPEL editor with (i) a panel for the specification of the pages in which UI components can be rendered and (ii) a property panel that allows the developer to configure the web pages, to set the properties of UI partner links, and to associate them to place holders in the layout.

The **deployment** of a UI orchestration requires translating the BPEL4UI specification, which is not immediately executable neither by a standard BPEL engine nor by the UI rendering engine (the so-called *UI engine*, which we discuss in the following), into executable formats. This task is achieved by the *BPEL4UI compiler*, which, starting from the *BPEL4UI* specification, the set of used *HTML templates* and *UI component WSDLs*, and the *system configuration* of the runtime part of the architecture, generates three kinds of outputs:

1. A set of *communication channels* (to be deployed in the so-called *UI engine server*), which mediate between the *UI engine client* (the client browser) and the *BPEL engine*. These channels are crucial in that they resolve the technology conflict inherently present in BPEL4UI specifications: a BPEL engine is not able to talk to JavaScript UI components running inside a client browser, and UI components are not able to interact with the SOAP interface of a BPEL engine. For each UI component in a page, the compiler therefore generates (i) an *event proxy* that is able to forward events from the client browser to the BPEL engine and (ii) an *event buffer* that is able to accept events from the BPEL engine and stores them on behalf of the *UI engine client*.
2. A *standard BPEL* specification containing the distributed UI synchronization and web service orchestration logic. Unlike the BPEL4UI specification, the generated BPEL specification does no longer contain any of the UI-specific constructs introduced in Section 4.1 and can therefore be executed by any standards-compliant BPEL engine. This means that all references to UI component partner links in input to the compilation are rewritten into references to the respective communication channels of the UI components in the *UI engine server*, also setting the correct, new SOAP endpoints.
3. A set of *UI compositions* (one for each page of the application) consisting of the layout of the page, the list of UI components of the page, the assignment of UI components to place holders, the specification of the intra-page UI synchronization logic, and a reference to the client-side runtime framework. Interactions with web services or UI components running in other pages are translated into interactions with local system components (the *notification handlers* and *event forwarders*), which manage the necessary interaction with the *communication channels* via suitable RESTful web service calls.

Finally, the *BPEL4UI compiler* also manages the deployment of the generated artifacts in the respective runtime environments. Specifically, the generated *communication channels* and the UI compositions are deployed in the *UI engine server* and the standard *BPEL specification* is deployed in the *BPEL engine*.

The **execution** of a UI orchestration requires the setting up and coordination of three independent runtime environments: First, the interaction with the users is managed in the client browser by an event-based JavaScript runtime framework that is able to parse the UI composition stored in the UI engine server, to instantiate UI components in their respective place holders, to configure the *notification handlers* and *event forwarders*, and to set up the necessary publish-subscribe logic ruling the event-to-operation mapping of the components running inside the client browser. While *event forwarders* are called each time an event is to be sent from the client to the BPEL engine, the *notification handlers* are active components that periodically poll the event buffers of their UI components on the *UI engine server* in order to fetch possible events coming from the *BPEL engine* (we are currently studying suitable push mechanisms for events).

Second, the *UI engine server* must run the web services implementing the communication channels. In practice we generate standard Java servlets and SOAP web services, which can easily be deployed in a common web server, such as Apache Tomcat. The use of a web server is mandatory in that we need to be able to accept notifications from the BPEL engine and the UI engine client, which requires the ability of constant listening. The event buffer is implemented via a simple relational database (in PostgreSQL) that manages multiple UI components and distinguishes between instances of UI orchestrations by means of a session key that is shared among all UI components participating in a same UI orchestration instance.

Third, running the BPEL process requires a *BPEL engine*. Our choice to rely on standard BPEL allows us to reuse a common engine without the need for any UI-specific extensions. In our case, we use Apache ODE, which is characterized by a simple deployment procedure for BPEL processes.

The MarcoFlow system shown in Figure 6 is fully implemented and running. A demo of the tool is available at *http://mashart.org/marcoflow/demo.htm.*

## 7 Conclusion

The spectrum of applications whose design intrinsically depends on a structured flow of activities, tasks or capabilities is large, but current workflow or business process management software is not able to cater for all of them. Especially lightweight, component-based applications or Web 2.0 based, mashup-like applications typically do not justify the investment in complex process support systems, either because their user basis is too small or because there is need only for few, simple applications. Yet, these applications too demand for abstractions and tools that are able to speed up their development, especially in the context of the Web with its fast development cycles.

We introduced the idea of *distributed UI orchestration*, a component-based development technique that introduces a new first-class concept into the workflow management and service composition world, i.e., UIs, and that fits the needs of many of today's web applications. We proposed a model for UI components and showed how their use requires extending the expressive power of standard service composition languages. The language comes with a suitable modeling environment and a code generator able to produce code and instructions that can be executed straightaway by our runtime environment, which separates the problem of intra-page UI synchronization from that of distributed UI synchronization and service orchestration. The result is an approach to distributed UI orchestration that is comprehensive and free.

Unlike in our research on universal composition [3] and unlike mashup tools, in this paper we do not aim at enabling less skilled web users to develop simple applications. MarcoFlow targets skilled web developers that are familiar with BPEL and applications that are complex and possibly involve multiple actors that are distributed over the Web, but that need orchestration. While the idea of event-based UI components has been around for some time now, distributed UI orchestration and multi-browser/multi-actor applications as proposed in this paper are new.

Next, we plan to support the *dynamic selection of actors* (during deployment or at runtime), advanced *access policies*, and *data flow* mechanisms that go beyond the current event-based communication (e.g., through a suitable persistence layer).

## References

1. Myers, B.A., Rosson, M.B.: Survey on user interface programming. In: SIGCHI 1992, pp. 195–202 (1992)
2. Yu, J., Benatallah, B., Casati, F., Daniel, F.: Understanding Mashup Development and its Differences with Traditional Integration. IEEE Internet Computing 12(5), 44–52 (2008)
3. Daniel, F., Casati, F., Benatallah, B., Shan, M.-C.: Hosted Universal Composition: Models, Languages and Infrastructure in mashArt. In: ER 2009, pp. 428–443 (2009)
4. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.1. W3C Note (March 2001), `http://www.w3.org/TR/wsdl`
5. OASIS. Web Services Business Process Execution Language Version 2.0 (April 2007), `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`
6. Pautasso, C.: BPEL for REST. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 278–293. Springer, Heidelberg (2008)
7. van Lessen, T., Leymann, F., Mietzner, R., Nitzsche, J., Schleicher, D.: A Management Framework for WS-BPEL. In: ECoWS 2008, Dublin, pp. 187–196 (2008)
8. Maximilien, E.M., Ranabahu, A., Gomadam, K.: An Online Platform for Web APIs and Service Mashups. Internet Computing 12(5), 32–43 (2008)
9. Active Endpoints, Adobe, BEA, IBM, Oracle, SAP. WS-BPEL Extension for People (BPEL4People), Version 1.0 (June 2007)
10. Active Endpoints, Adobe, BEA, IBM, Oracle, SAP. Web Services Human Task (WS-HumanTask), Version 1.0 (June 2007)
11. Acerbis, R., Bongio, A., Brambilla, M., Butti, S., Ceri, S., Fraternali, P.: Web Applications Design and Development with WebML and WebRatio 5.0. In: TOOLS 2008, pp. 392–411 (2008)
12. Gómez, J., Bia, A., Parraga, A.: Tool Support for Model-Driven Development of Web Applications. In: Ngu, A.H.H., Kitsuregawa, M., Neuhold, E.J., Chung, J.-Y., Sheng, Q.Z. (eds.) WISE 2005. LNCS, vol. 3806, pp. 721–730. Springer, Heidelberg (2005)
13. Manolescu, I., Brambilla, M., Ceri, S., Comai, S., Fraternali, P.: Model-Driven Design and Deployment of Service-Enabled Web Applications. ACM Trans. Internet Technol. 5(3), 439–479 (2005)
14. Brambilla, M., Ceri, S., Fraternali, P., Manolescu, I.: Process Modeling in Web Applications. ACM Trans. Softw. Eng. Methodol. 15(4), 360–409 (2006)

15. Sun Microsystems. JSR-000168 Portlet Specification (October 2003), http://jcp.org/aboutJava/communityprocess/final/jsr168/
16. OASIS. Web Services for Remote Portlets, (August 2003) http://www.oasis-open.org/committees/wsrp
17. Oracle. JavaServer Faces Technology, http://java.sun.com/javaee/javaserverfaces/
18. Pietschmann, S., Voigt, M., Rümpel, A., Meissner, K.: CRUISe: Composition of Rich User Interface Services. In: ICWE 2009, pp. 473–476 (2009)
19. Feldmann, M., Nestler, T., Jugel, U., Muthmann, K., Hübsch, G., Schill, A.: Overview of an end user enabled model-driven development approach for interactive applications based on annotated services. In: WEWST 2009, pp. 19–28 (2009)
20. WSPER.org. WS-BPEL 2.0 Metamodel, http://www.ebpml.org/wsper/wsper/ws-bpel20.html

# Self-adjusting Recommendations for People-Driven Ad-Hoc Processes

Christoph Dorn[1], Thomas Burkhart[2], Dirk Werth[2], and Schahram Dustdar[1]

[1] Distributed Systems Group
Vienna University of Technology
1040 Vienna, Austria
lastname@infosys.tuwien.ac.at
[2] Institute for Information Systems (IWi)
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany
lastname@iwi.dfki.de

**Abstract.** A company's ability to flexibly adapt to changing business requirements is one key factor to remain competitive. The required flexibility in people-driven processes is usually achieved through ad-hoc workflows. Effective guidance in ad-hoc workflows requires simultaneous consideration of multiple goals: support of individual work habits, exploration of crowd process knowledge, and automatic adaptation to changes. This paper presents a self-adjusting approach for providing context-sensitive process recommendations based on the analysis of user behavior, crowd processes, and continuous application of process detection. Specifically, we classify users as *eagles* (i.e., specialists) or *flock*. The approach is evaluated in the context of the European research project Commius.

## 1 Introduction

Today, enterprise competitiveness is primarily determined by an organization's ability to adapt to dynamically changing environments. Keeping the pace with innovations to maintain a competitive advantage requires the rapid assembly of value chains where multiple specialized companies cooperate in the production of increasingly complex products. As a direct consequence, established work practises—especially in people driven process environments—need to become flexible and adaptable.

Traditional work-flow engines lack the required flexibility for reacting to ad-hoc changes. Their rigid underlying process model would need to foresee all possible variation, which becomes unfeasible even for simple processes. Support systems for flexible processes (e.g., Caramba [1]) recommend a user to follow a predefined process path, but allow them to deviate from that process on demand (For a exhaustive survey on flexible business support systems see [2]). This paper focuses on two major challenges that remained mostly unaddressed: (a) users in people-driven processes require a combination of personalized recommendations, while exploiting the best practises emerging from the overall user community; (b) flexible processes need to evolve across time to reflect the changes in working style, business constraints, and impact of cross-organizational cooperation.

R. Hull, J. Mendling, and S. Tai (Eds.): BPM 2010, LNCS 6336, pp. 327–342, 2010.

In this paper, we introduce a hybrid approach that combines user-centric process recommendation with crowd-based process knowledge. Specifically we provide recommendations learned from previous processes executed by that user and couple them with process decisions taken from all users involved in that particular process type. Our main contribution is a self-adjusting user classification model that determines whether a user engages in individualized process adaptations (*eagle*) or whether the user follows a generally agreed upon process step sequence (*flock*). Monitoring and recommendation evaluation continuously adjusts this classification. The underlying ad-hoc process engine allows any deviations from the modeled flow. These deviations feed back into the process model, ultimately enabling process evolution through self-learning of process patterns.

A motivating scenario sets the scene for our self-adjusting recommendation approach (Section 2). In Section 3, we continue with a brief discussion of the term *flexibility* as applied in the domain of adaptive business processes, followed by related work focusing on flexible process support systems. Section 4 describes the process recommendation algorithm and feedback mechanism. Section 5 discusses our advanced recommendation aggregation and user classification technique. We evaluate our approach based on the scenario and our prototype implementation in Section 6. Finally, Section 7 gives a short conclusion and an outlook on future work.

## 2 Supporting Flexibility in People-Driven Processes

Based on the continuum given in [3] business processes can be classified as follows:

**Structured processes** represent the traditional work flows with full automation capacity. Structured process-models determine a-priori the complete process flow, agents, alternative paths etc. and remain unchanged for all process instances.
**Semi-structured processes**—or case-oriented processes [4]—reside between ad-hoc processes and structured processes. They follow certain rules but cannot be entirely standardized.
**Ad-hoc processes** represent the most flexible type of processes because their actual execution path is completely defined at run-time with no given structure forcing a certain course of action.

We focus on ad-hoc processes where users are free to decide which process steps to execute. In order to provide recommendations, however, we allow users to model processes which are then refined through user monitoring. We apply the taxonomy by Regev et al. [5] to characterize our supported process flexibility:

**Abstraction Level:** Runtime changes affect only the process instance. However, these changes eventually cause an evolution of the process model through process learning.
**Subject of Change:** Process advice supports only behavioral changes, as we recommend only the selection and order of process steps. (Functional, operational, information, and organizational perspectives remain outside the scope of this paper).
**Properties of Change:** Our main focus lies on process learning which results in small, incremental changes to the process. Revolutionary changes through business process reengineering, however, are also supported. Changes to the process model affect only new

instances (i.e., *deferred* as opposed to *immediate*) and emerge from past user actions in an ad-hoc fashion.

### 2.1 Motivating Scenario

Within this paper, a common show-case will be used to point out different aspects of the introduced approach. The scenario (Figure 1) focuses on a business process describes the handling of incoming orders and the subsequent dispatching of the ordered items. An incoming order triggers the process. Subsequently, an order confirmation is returned to the customer. Further, credit and inventory checks confirm the credit-worthiness of the customer and the availability of the ordered items. In case the ordered goods are not on stock, the replenishment of the items is triggered. As soon as all required items are gathered, the shipment as well as the corresponding invoice are prepared. The process ends with the dispositioning of the ordered items.

The scenario describes a primarily people-driven work flow. At first sight, the process does not seem very ad-hoc. The individual workers, however, are free to select the desired selection and sequence of process steps. The main purpose of the process description is obtaining a first, generic process that provides a rough guide for most cases. As business requirements change due to internal forces (e.g., new products, different customer focus) or external forces (e.g., important customers demand special treatment) the individual workers adapt the order of steps as they see fit. A worker, for example, can decide to ship the goods before completing billing and invoicing. Our mechanism monitors such decisions and continuously adapts to recommend always the most suitable next steps.

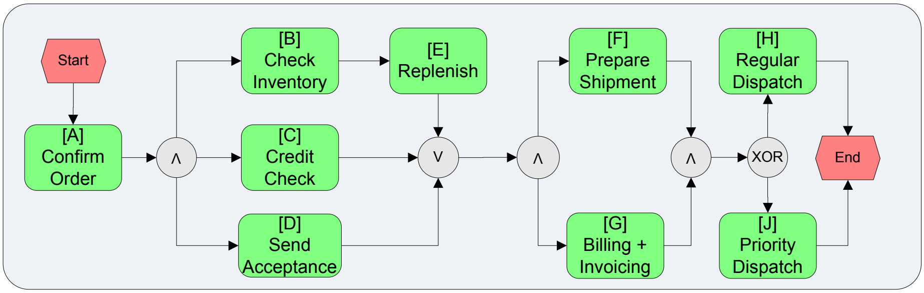


**Fig. 1.** Scenario: generic people-driven order process model (PM)

## 3 Related Work

**Defining Flexibility in Business Processes.** The term *flexibility* in the context of business processes comes with a multitude of interpretations. An overview over the most established interpretations of flexibility allows us to better compare our contribution to existing approaches.

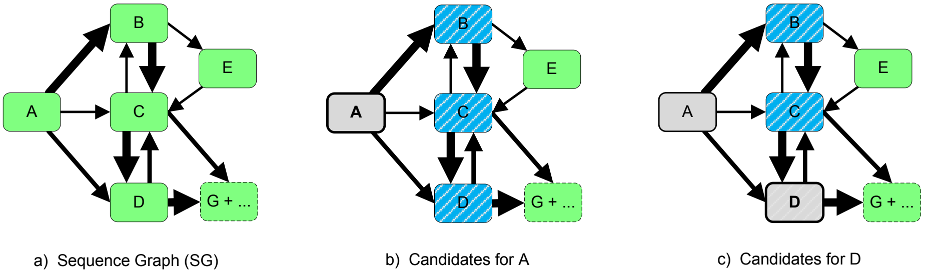
In the scope of this paper, we define process flexibility as the ability to adapt the process flow on demand through adding, skipping, or sequence reordering of process steps. This definition is closely related to the interpretation by Adams et al. [6] in which processes simply provide a guideline while the appropriate way of handling single tasks is chosen on an as-needed basis. In Reijers et al. [7], process models define the normal way of achieving a goal, but still offer the possibility to deviate based on available case data. Sadiq et al. [8], on the other hand, describe flexibility as the ability to deal with processes that are only partially defined at build-time. Soffer [9] distinguishes between short-term flexibility (i.e., deviations from a given model) and long-term flexibility (i.e. evolution of processes). Greiner and Rahm [10] limit the definition to exception handling capabilities in case of unforeseen events or policy changes. In contrast to the application specific perspectives, Adamides et al.[11], define strategic flexibility which describes a company's diversity of strategies and its capability to switch between them.

**Flexible Process Support Systems.** Research on providing recommendations in flexible workflow systems focuses on multiple aspects. The major means of providing recommendations is done by guidelines. A predefined process model assists a user in choosing how to proceed a workflow. In more detail, such a guideline can exists merely of process parts (like presented by Sadiq et al. [8]) and which thus does not require a complete process model. Alternatively, guidelines can define what has to be done in each specific process state but still not provide a complete process path [12]. Moreover, Adams et al. [13] define each process step within such a guideline as a simple placeholder task which is dynamically replaced by a context sensitive choice from an extensible catalog of suitable workflow definitions during run-time. The actual selection process is ultimately defined by so-called Ripple Down Rules [6]. In addition, recommendations can be derived based on a rough task structure [14]. In contrast to these guideline approaches that are mainly based on predefined process models and might not be instantiated at all, recommendations are based on best-practices shared by users within a company [15]. Pesic et al. and van der Aalst [16,17] provide recommendations based on past experiences and additionally on a specific process goal. This is achieved by comparing the current process instance with past executions (logs), while preferring those executions that satisfy the specified goal. A similar approach can be found in [18] where recommendations are generated based on similar past process executions by considering the specific optimization goals. Another approach is followed by Almeida and Casanova [19] whose recommendations are based on an ontology and semantic rules that generate possible process alternatives or suitable process steps if the execution of a workflow instance fails to proceed. Vanderfeesten et al. [20] follow an approach in which, based on the information available for a case, the next step to be performed is determined using a strategy of e.g. lowest cost or shortest processing time. While the previous approaches focus on concrete recommendations, the TIBCO Software Inc. provides detailed process information and context to the user. Thus, a user can identify which steps are required to achieve the process goal [21].

These flexible process support systems seem to be on the right track when comparing their capabilities to the stated definitions and statements concerning flexibility. According to several surveys, however, actual implications of the ad-hoc approach lack of a sufficient degree of process guidance during run-time due to their overly extensive degree of freedom (cf. [2]).

## 4 Process Recommendation

The recommendation mechanism applies two related data sets, the process model (PM, Figure 1) and the sequence graph (SG). Figure 2a) displays the sequence graph for the first steps of the scenario process. The sequence graph $SG(P, E)$ comprises nodes representing the individual process steps P. An directed edge $e \in E$ in SG between two nodes $A$ and $B$ describes a temporal sequence that process step $B$ follows immediately after $A$. Whenever a user conducts process step $B$ after process step $A$ we increase the edge value. The SG accumulates all individual process step sequences for a particular process type. It thus yields the likelihood (i.e., preference) of following a particular path through the process. In Figure 2a, the arc thickness indicates this preference. The flow control model describes the dependency between process steps. Joins, splits and sequential steps are extracted from the sequence graph by existing process techniques [22].



a) Sequence Graph (SG) b) Candidates for A c) Candidates for D

**Fig. 2.** Sequence Graph (SG) excerpt and process step candidates: completed process steps in dark gray, candidates in shaded blue, and inactive process steps in green. (Colors online)

When a process is started, we derive an instance of the process model. As we track the progress of the process, we utilize the process instance to select the relevant process steps that are sensible to enact in the current process state. This is the task of the *Process Instance Manager* (Fig. 5). For each point in time, it keeps track of process steps that have been completed, which are active (i.e., all process step preconditions are fulfilled, but the step has not been carried out), and which steps need to be (de)activated. For our recommendation purpose the *Process Instance Manager* provides a list of process step candidates that are ready to be carried out.

The sequence graph then provides the information to establish which of the possible process steps to carry out first. The algorithm in Listing 1 describes the recommendation procedure in detail. After completion of the order confirmation process step $A$, the *Process Instance Manager* identifies *Check Inventory B*, *Credit Check C*, and *Send Acceptance D* from the PM as valid next steps (Figure 2b). A recommendation $r$ consists of a process step $S$, and the recommendation confidence $w$ defined in the interval

$[0, 100]$, where 100 indicates absolute certainty. Within a set of recommendations $R$, the sum of confidence values will always add up to exactly 100 ($\sum w_i = 100 \; \forall \; r_i \in R$).

Subsequently, the SG weights these candidates according to the edge values (e.g., here $w(AB) = 70, w(AC) = 10$, and $w(AD) = 30$). The first phase ranks all process step candidates that are active and exhibit incoming edges from the recently finished process step (here $A$) as detailed in lines 1 to 12 in Algorithm 1. Let us assume the user does not follow the top-rated recommendation $B$ and instead selects step $D$ (Figure 2c). Now, simple recommendation on the SG would suggest primarily to continue with the process ($G$+ ...) and as second choice continuing to $C$ (because arc DG yields a higher edge weight than edge DC). A pre-selection of valid edges based on the PM, however, identifies $B$ and $C$ as the only sensible next process steps. In short, the sequence graph by itself cannot give recommendations that respect control flow constraints. The flow control model by itself, on the other hand, cannot provide suggestions on the order of which process step to carry out first.

We cannot focus on recommending subsequent process steps only, as the user is free to select any process step. Lines 13 to 26 in Algorithm 1 analyze the process for skipped and out-of-order process steps. Thus, after finishing $D$, the SG will analyze both $C$ and $B$. First, we check any preceding steps of $D$ that are still active (i.e., only $C$) and score them according to outgoing weights (i.e., $w(CD)$) — lines 13 to 18.

Finally, if the SG does not exhibit any edge between the candidates and the last completed step (i.e., $B$), the candidates are ranked based on their aggregated weight on their respective incoming edges (i.e., $w(AB)$ ). We limit the incoming edges to those that originate at already completed process steps. We, thus, prefer candidates that follow after already completed steps and that are frequently traversed (lines 19 to 26).

The algorithm recommends only the next, active process steps that need completion for sake of simplicity (as opposed to process step sequences). The process model and sequence graph, however, contain the required information to provide also multi-step recommendations. The recommendation model does not support dependencies between process steps explicitly. When the users are aware of such limitations, the process logs and subsequent mined process models will eventually reenforce such dependencies implicitly.

### 4.1 User-Based Recommendation

The generic scenario process gives rise to distinctive process adaptations as required by different environment needs. We observe the behavior of following three example users. User 1 is responsible for regular customers that order standard products which get automatically restocked once a certain threshold is undercut. Standard customers receive their goods via regular shipping. Consequently, User 1's personal process model deviates from the standard order process. Figure 3 display PM (a) and SG (b) for User 1. The graphs does not display process steps $B$, $E$, and $J$ as they are never invoked. Note that User 1 embraced the habit of always preparing the billing before triggering the shipment, having steps $G$ and $F$ in sequence (Figure 3a). From the sequence graph (Figure 3b) we learn, that User 1 has hardly any preference on whether to execute $C$ or $D$ first.

**Algorithm 1.** Crowd-based Recommendation Algorithm $\mathcal{A}(SG, P, S)$

```
for all $ProcessStep\ PS \in P$ do                ▷ Get list of process step candidates.
    if $state(PS) == active$ then
        $R \leftarrow PS$
    end if
end for                                           ▷ Initialize process step ranking scores.
$W \leftarrow \emptyset$
for all $ProcessStep\ C \in R$ do                 ▷ For all consecutive process steps of $S$.
    $W[C] += SG.getEdge(S, C).weight$
end for
if $hasActivePredecessors(S)$ then                ▷ If a preceding process step has been temporarily skipped.
    for all $ProcessStep\ C \in R$ do             ▷ Extract incoming edge weight from SG.
        $W[C] += SG.getEdge(C, S).weight$
    end for
end if                                            ▷ For any other active process step that is not directly connected to $S$.
for all $ProcessStep\ C \in R \wedge W[C] == 0$ do
    for all $Arc\ a \in SG.getInEdge(C)$ do
        if $state(sourceNode(a)) == completed$ then    ▷ Count the edge weight only if the predecessor step has been completed.
            $wsum += a.weight$
        end if
    end for
    $W[C] += w$
end for                                           ▷ Rank candidates by weights $W$
$sort(R, W)$
return $R$
```

User 2 serves to premium customers that have a high order volume, pay regularly and thus need not go through a credit check. Premium customers receive priority shipment ($J$) to deliver their order goods as fast as possible. Similar to User 1, User 2 doesn't check the availability of stock before confirming an order either. Consequently, steps $A$ and $D$ become a sequence as steps $B, C, E$, and $H$ are missing (Figure 3c+d). This user also exhibits a strong tendency to first trigger shipment preparations ($F$) and dispatching the goods ($J$) before preparing invoicing ($G$). User 3 handles special cases. Being a new employee, he tends to forget certain process steps. Specifically, he never returns order confirmations (D), and occasionally misses the preparation of billing information (G). Consequently, the process extracted from the sequence graph joins steps $F$ and $G$ via an $OR$ instead of an $AND$ (Figure 3e+f).

Each individual user exhibits a very personalized process that deviates considerably from the standard order process.[1] While personalized recommendation would yield highly relevant process step rankings, these recommendations cannot exploit alternative activities when exceptions such as delayed shipping, or partial order content is

[1] Note that for collaborative processes (i.e., multiple interacting users) the personalized process and respective recommendations cover only the part of the process in which the user is involved in.

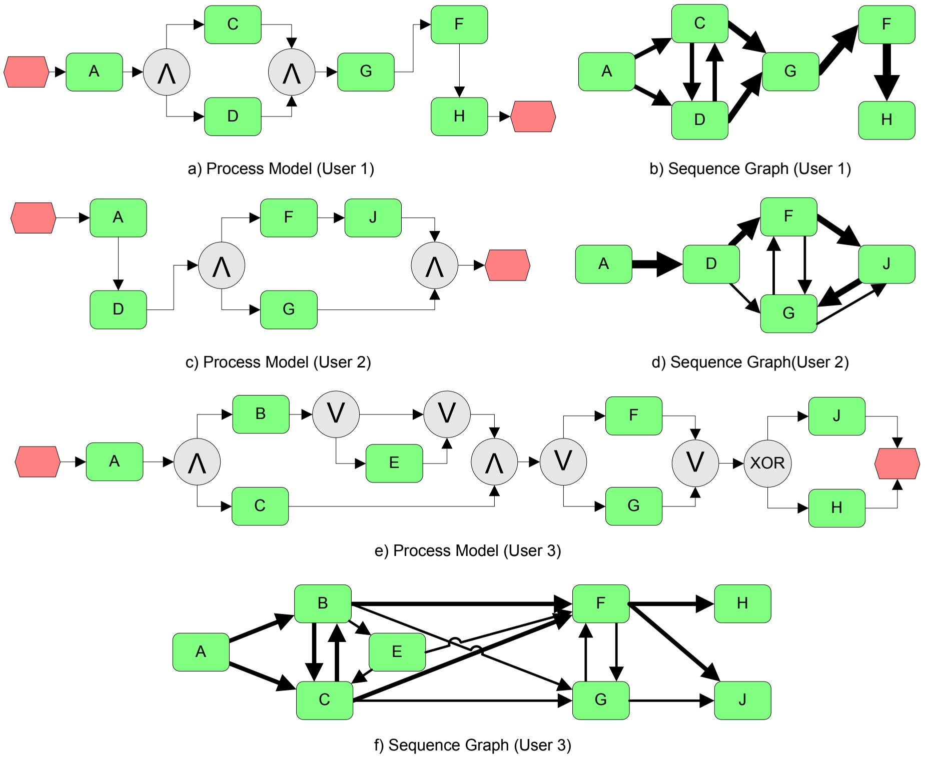

A
C
D
G
F
H
a) Process Model (User 1)
b) Sequence Graph (User 1)
J
c) Process Model (User 2)
d) Sequence Graph(User 2)
B
E
XOR
e) Process Model (User 3)
f) Sequence Graph (User 3)

**Fig. 3.** Process Model and Sequence graph for User 1 (handling standard orders), User 2 (serving premium customers), and User 3 (handling special cases)

out of stock. Moreover, pure personalized recommendations will reinforce inefficient or even incorrect sequences such as inadvertently skipping an important process step. Crowd-based recommendations mitigate this shortcoming.

## 4.2 Crowd-Based Recommendation

Crowd-based recommendations enrich the set of relevant possible process paths through aggregation of the process experiences from multiple users. Personalized processes capture the habits of an individual user. They are, however, limited to process step sequences that particular user has executed so far. Alternative sequences that potentially reduce overall processing time remain unavailable. Also, a personalized process cannot be applied for giving advice in exceptional situations that have not been encountered by the user before.

Figure 4 displays the aggregated flow control model and sequence graph for users 1, 2, and 3. The SG is a simple aggregation of all process step sequence from process instances completed by the three users. The process mining technique reference above then generates the corresponding PM. Note, due to User 3, the PM joins steps *F* and *G* via an *OR*.

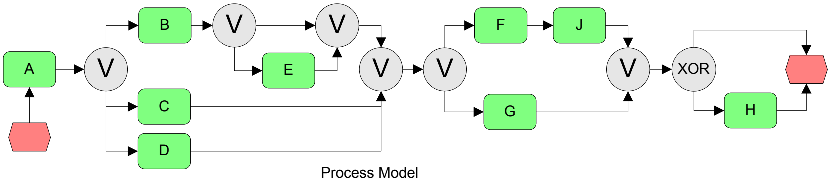


**Fig. 4.** Aggregated Process Model for User 1, 2, and 3

The complete recommendation cycle is depicted in Figure 5. An incoming request for recommendation triggers the recommendation mechanism (1). The recommender collects information from the process instance manager (2a) and the sequence graph (2b) to aggregate sensible upcoming process steps. The process recommender retrieves this information from both personal and crowd-based SG, respectively PM. The exact aggregation of personal and crowd-based recommendations is outlined in Section 5. The recommender subsequently provides the user the recommended process steps (3).The user selects one process step and enacts it by clicking, for example, on a link in his/her user interface (4). Note that the user doesn't explicitly agree or disagree with a recommendation. Instead, the system monitor observes the user's actions (5a), and other system events (5b) to determine the true process progress. The system monitor updates the process instance manager whenever a process step has been completed (6). The process instance manager in turn updates the personal and crowd-based sequence graph for each completed step (7). In regular intervals, the Process Miner takes a sequence graph (8) and generates an updated process model (9). This procedure is performed for each process type to generate the crowd-based PM and for each individual user and process type to derive the personal PM. We apply an aging mechanism to reduce the effect of old, potentially outdated, process sequences. For every new incoming process sequence we remove the oldest sequence.

## 5 Self-adjusting Recommendation Model

The overall recommendation combines user-centric and crowd-based recommendations according to the classifier $\alpha$. It describes the user on a scale between 0 and 1, where 1 denotes a user always adhering to his individual work style — the *eagle*. At the other extreme end of the classifier ($\alpha = 0$), a user follows generally applied work practices — *flock*. We determine $\alpha$ for each user and process type as a user's work style potentially deviates for each process type. The overall recommendation merges user-centric and crowd-based recommendations according to the following formula:

$$R_{overall} = \alpha * R_{user} + (1 - \alpha) * R_{crowd} \tag{1}$$

Specifically, we multiply a recommendation's weight $w_{REC}$ within $R_{user}$ with $\alpha$ and repeat the same for $R_{crowd}$ with $(1 - \alpha)$. Sorting the merged list provides the overall recommendation.

Suppose following simple example consisting of user- and crowd-based recommendations: $R_{user}$ recommending $S_1, S_3, S_4$ and $R_{crowd}$ recommending $S_2, S_3, S_5, S_6$. We

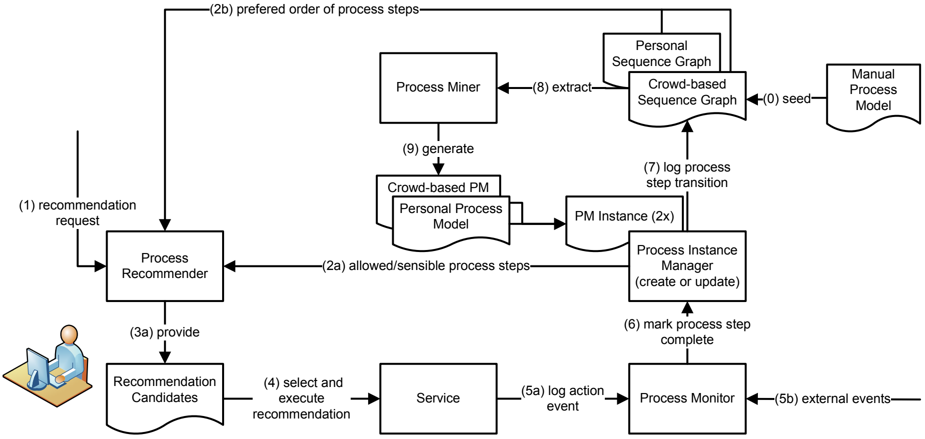

(2b) prefered order of process steps
Personal Sequence Graph
Process Miner
(8) extract
Crowd-based Sequence Graph
(0) seed
Manual Process Model
(9) generate
(7) log process step transition
Crowd-based PM
Personal Process Model
PM Instance (2x)
(1) recommendation request
Process Recommender
(2a) allowed/sensible process steps
Process Instance Manager (create or update)
(3a) provide
(6) mark process step complete
Recommendation Candidates
(4) select and execute recommendation
Service
(5a) log action event
Process Monitor
(5b) external events

**Fig. 5.** Feedback cycle for personal and crowd-centric recommendations

obtain following overall recommendation for $\alpha = 0.5$ (Note that the weights for $S_3$ are aggregated.):

$$0.5 * \begin{bmatrix} S_1 & 70 \\ S_3 & 15 \\ S_4 & 15 \end{bmatrix} + 0.5 * \begin{bmatrix} S_2 & 80 \\ S_3 & 10 \\ S_5 & 5 \\ S_6 & 5 \end{bmatrix} \longrightarrow \begin{bmatrix} S_2 & 40 \\ S_1 & 35 \\ S_3 & 12.5 \\ S_4 & 7.5 \\ S_5 & 2.5 \\ S_6 & 2.5 \end{bmatrix} \qquad (2)$$

In our example, we set $\alpha = 0.5$ to denote a user that has not been classified as *eagle* or *flock*, yet. We reject a fixed configuration of the parameter $\alpha$. Instead, dynamic classification adjustment reflects a user's adaptation to changing process requirements and learning effects. To this end, we observe the user's selection of recommended process steps. We increase the value of $\alpha$ when the user carries out a process steps that originated from $R_{user}$. Similarly, we reduce the value of $\alpha$ when the user follows crowd-based recommendations.

Following factors determine the amount to which $\alpha$ is moved:

- Similarity of user-centric and crowd-based recommendations: We cannot clearly distinguish between distinctive user behavior when both recommendation set contain similar top-rated process steps.
- Process success: When $\alpha$ remains close to 1 but process success declines, we have to assume that the personalized recommendations fail as they most likely reinforce bad decisions. In this case, we need to push $\alpha$ towards the neutral value to introduce again crowd-based recommendations.
- Current value of $\alpha$: Remaining in the middle between *eagle* and *flock* is not desirable, as we neither provide user-centric know how, nor exploit the wisdom of the crowd. Consequently, moving away from the middle is reinforced. Reaching the absolute extremes, however, is hard to allow for a quick transition between the two behavior types.

**Recommendation Similarity.** We calculate the similarity of user-centric and crowd-based recommendations implicitly by comparing the actual user actions with both recommendations. We assume that users deviate slightly from recommended process steps on a regular basis. Subsequently, we first combine the user's actions in an anonymous process step type and then compare that process step with the given recommendations.

We determine the similarity of two process steps by observing the overlap of common and individual actions. Specifically, we apply the weighted Jaccard similarity measurement.

$$sim_{wJaccard}(s_1, s_2) = \frac{\sum_{a \in s_1 \cap s_2} w_{IDF}(a)}{\sum_{a \in s_1 \cup s_2} w_{IDF}(a)} \tag{3}$$

where $w_{IDF}(a)$ is the weight function describing the frequency of action $a$ occurring in a process step $s$. Here the weight function is the inverse document frequency (IDF) of $a$, having the document base comprise all process steps defined in the process model. The weight for a particular action $a_i$ is defined as:

$$w_{IDF}(a_i) = log \frac{|S|}{s : a_i \in s} \tag{4}$$

where $|S|$ is the number of all process steps and $s : a_i \in s$ counts all process steps that contain action $a_i$. Actions that occur in most process steps will thus yield low weight when comparing two process steps, while rare actions will yield a high weight.

The similarity of user actions and recommendation derive the recommendation's success. Each recommended process step is additionally weighted by the recommendation's weight. For an anonymous process step $A$ we calculate:

$$succ(R, A) = \sum_{i}^{R} sim_{wJaccard}(A, s_i) * w_{REC}(s_i) \tag{5}$$

The overall effect on $\alpha$ moving towards *eagle* or *flock* is then simply derived through comparison of personalized and crowd-based recommendation success:

$$\delta(A) = succ(R_{user}, A) - succ(R_{crowd}, A) \tag{6}$$

**Avoiding Classification Lock-in.** An *eagle* remains locked-in his classification when he repeatedly fails to successfully complete a process but continues to receive exclusive personal recommendations. In this case, we have to abandon the underlying classification. A user is considered locked-in, when his average process success rate falls below the average process success rate of the top 50% *flock* users. Specifically, we sort all users according to their current classification value $\alpha$ in ascending order and select the process success rate $psucc$ of all users having $\alpha$ equal or below the second quartile. We set $\alpha = 0.5$ for user $u$ if he fails to meet following threshold condition:

$$ps\bar{u}cc(u) > \frac{2 * \sum_i ps\bar{u}cc(u_i)}{|U|} \qquad \forall u_i : \alpha_i \leq Q_2 \tag{7}$$

This is expected to raise the number of successful processes as the user is presented with process step alternatives he did not consider before. The user classification might again deviate towards *eagle* again, but this time resulting in more sensible process steps.

**Accelerated Classification Divergence.** When recommendations combine profile-based and crowd-based recommendation to approximately equal extent, the top recommended process steps are potentially similar or, on the other hand, completely contradicting. We apply a sigmoid function to avoid remaining too long in the middle between *eagle* and *flock* ($\alpha \sim 0.5$). The sigmoid function (see Fig. 7) ensures that we can quickly move from the middle in both directions. However, we will only move if $\delta(A)\ != 0$ (i.e., when there is a trend towards *eagle* or *flock*) otherwise we remain with the previous $\alpha$ value. Based on $\delta(A)$ and current classification value $\alpha_t$, we determine the new $\alpha_{t+1}$:

$$\alpha_{t+1} = \begin{cases} Min[Max[(1 + e^{-10*(\alpha_t+\delta(A))+5})^{-1}, 0], 1] & \text{if } \delta(A) \neq 0, \\ \alpha_t & \text{if } \delta(A) = 0. \end{cases} \tag{8}$$

where the *Min* and *Max* operators limit $\alpha$ to the interval [0, 1].

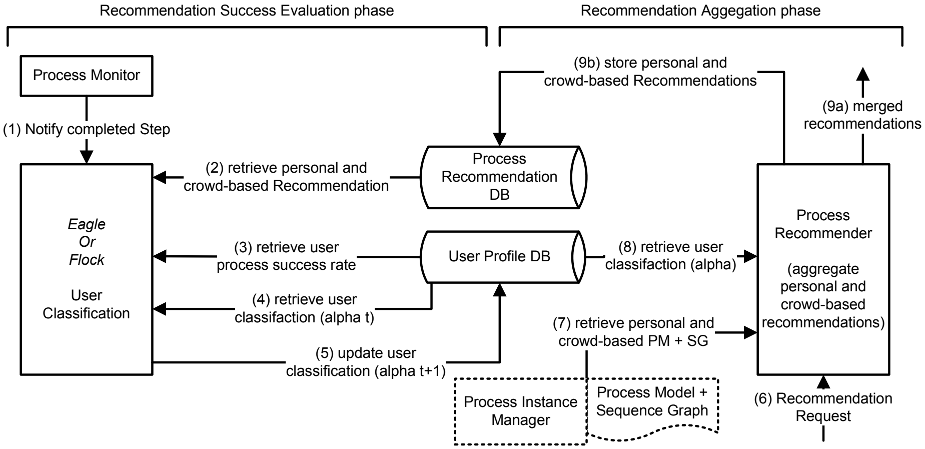

Recommendation Success Evaluation phase
Recommendation Aggegation phase
Process Monitor
(1) Notify completed Step
Eagle Or Flock
User Classification
(2) retrieve personal and crowd-based Recommendation
Process Recommendation DB
(9b) store personal and crowd-based Recommendations
(9a) merged recommendations
(3) retrieve user process success rate
User Profile DB
(8) retrieve user classifaction (alpha)
Process Recommender
(aggregate personal and crowd-based recommendations)
(4) retrieve user classifaction (alpha t)
(7) retrieve personal and crowd-based PM + SG
(5) update user classification (alpha t+1)
Process Instance Manager
Process Model + Sequence Graph
(6) Recommendation Request

**Fig. 6.** Self-tuning of classification parameter $\alpha$ based on recommendation success

**Classification Self-Tuning Cycle.** The complete classification self-tuning cycle consists of the *Recommendation Success Evaluation phase* and the *Recommendation Aggregation phase* (Figure 6). For each completed process step (1) the *User classification* component retrieves the corresponding personal and crowd-based recommendations (2) from the *Process Recommendation DB*. Next, we apply the recommendation similarity comparison. Subsequently, we evaluate the user process success rate (3) to check for classification lock-in . The *User Profile DB* manages classification values for the various process types and the corresponding process success information. We calculate the new classification value based on the previous value (4). The previous value is neglected if the lock-in check triggers a classification reset. Finally, the new classification value is stored (5).

The recommendation aggregation phase provides more details on how the *Process Recommender* — first introduced in Figure 5 — merges personal and crowd-based recommendations. Upon an incoming recommendation request (1), the recommender retrieves personal and crowd-based PM and SG (7). For each set, the ranking algorithm in Listing 1 determines the top process step candidates. The two sets are then aggregated applying the classification parameter (8). While the user receives the merged recommendations (9a), the process recommender stores the two output rankings of the recommendation algorithm separately (9b).

## 6 Experiments

**Scenario Evaluation.** We demonstrate the effect of user classification based on the motivating scenario, in particular based on the behavior of the three user types. Figure 8 provides the process recommendation evaluation results for 10 instances of the order process for each user (dashed lines) and the corresponding effect on the user classification $\alpha$ (full lines). We applied rapid aging in the experiment to visualize the convergence towards *eagle* or *flock* more clearly (i.e., new process sequences have an early and strong impact).

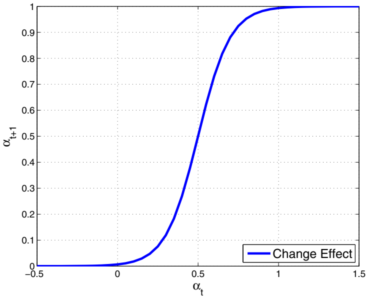


**Fig. 7.** Sigmoid function: the new user classification ($\alpha_{t+1}$) depends on the previous classification value ($\alpha_t$) and the shift towards *eagle* or *flock*

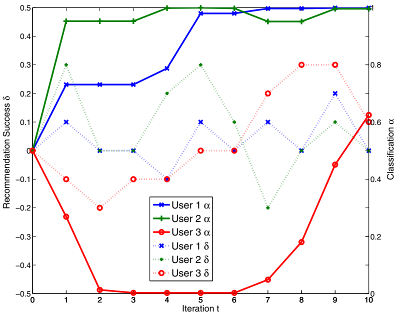


**Fig. 8.** Classification change $\alpha$ (full lines) and user recommendation success $\delta$ (dashed lines) for User 1, 2, and 3 across 10 time intervals

User 1 takes up some crowd-based recommendations but remains slightly with the personalized recommendations (i.e., $\delta$ on average between 0 and 0.1). We have a delayed convergence towards *eagle*, however, deviations towards *flock* have no effect. User 2 displays also *eagle* behavior, albeit diverges more quickly. User 3 exhibits a typical learning behavior. As he realizes to execute process step $D$, he strongly deviates from the personal recommendation, thus he becomes a *flock* member ($t_1$ to $t_4$) . As his corrected behavior becomes more present in the personal flow model, the differences between personal and crowd-based recommendations decrease ($t_5$ to $t_6$) and his personal sequence preferences start to show effect ($t_7$ to $t_{10}$). Multiple, sequential recommendation evaluations towards *eagle* ($\delta > 0$) cause his reclassification. As users learn and adapt their behavior, new flow control structures emerge from the crowd

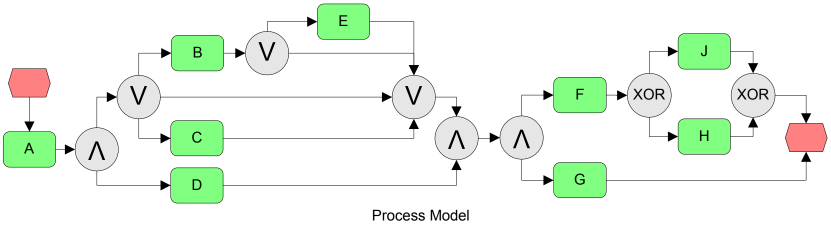


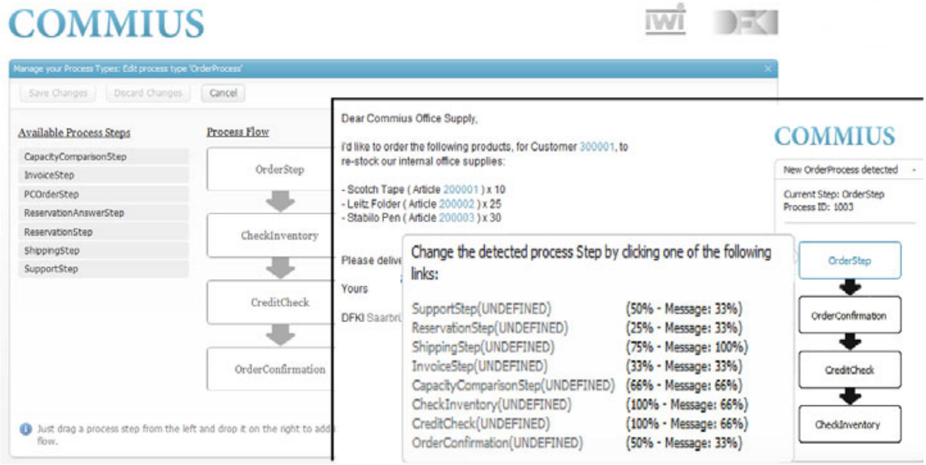
**Fig. 9.** Evolved Process Model and Sequence Graph for User 1, 2, and 3

sequence graph. Once User 3 apprehends to always send an order confirmation, the evolved crowd-based PM (Figure 9) identifies process step *D* as mandatory (and no longer optional).

**Prototype Evaluation.** The recommendation approach introduced in this paper has been implemented as a proof-of-concept within a software prototype of the European research project Commius. The prototype connects to a standard email environment—intercepting and analyzing email traffic—in order to detect process steps from the communication behavior of a user.

The users apply the process configuration tool (Figure 10) to define a coarse-grained structure of the desired process. Within Commius, we allow only a simple sequential structure to enable process modeling also for non-experts. The refined process model is later derived from user actions. When the system recognizes this predefined process steps in the email traffic, it will automatically enrich the corresponding emails with context sensitive information as well as process recommendation concerning further steps [23]. Figure 10 (inset) displays an example email enriched with process recommendations. The process step sequence with highest probability is provided on the right side (here four subsequent process steps taken from the scenario). The aggregation of personal and crowd-based recommendations exhibits a different process step sequence than the originally modeled flow. User 3 has been classified as *flock*, thus the recommendation advises him to prepare an order confirmation. The enhanced email also demonstrates the flexibility supported by our prototype. In case the user prefers not to follow any of the given recommended steps, s/he is free to select any other step from the underlying process. The popup contains the probabilities how well the alternative process steps match the current process context.

**Results.** The evaluation results are twofold. First, we achieved the successful application of our approach in email-based process environments. Recommendation support is directly integrated in the email client. Second, we demonstrated the user classification mechanism based on three user types. Classification diverges quickly (User 1, User 2), and displays the benefit of crowd-based process model to overcome erroneous process decisions (User 3) followed by subsequent reclassification.



Dear Commius Office Supply,

I'd like to order the following products, for Customer 300001, to
re-stock our internal office supplies:

- Scotch Tape ( Article 200001 ) x 10
- Leitz Folder ( Article 200002 ) x 25
- Stabilo Pen ( Article 200003 ) x 30

Please deliv

Yours

DFKI Saarbr

Change the detected process Step by clicking one of the following links:

| | |
|---|---|
| SupportStep(UNDEFINED) | (50% - Message: 33%) |
| ReservationStep(UNDEFINED) | (25% - Message: 33%) |
| ShippingStep(UNDEFINED) | (75% - Message: 100%) |
| InvoiceStep(UNDEFINED) | (33% - Message: 33%) |
| CapacityComparisonStep(UNDEFINED) | (66% - Message: 66%) |
| CheckInventory(UNDEFINED) | (100% - Message: 66%) |
| CreditCheck(UNDEFINED) | (100% - Message: 66%) |
| OrderConfirmation(UNDEFINED) | (50% - Message: 33%) |

**Fig. 10.** Commius process modeling tool and email-based process step recommendation (inset)

## 7 Conclusion and Outlook

Recommendations for people-driven ad-hoc processes exhibit maximum effectiveness when personal and crowd-based behavior is combined. Adding continuous process detection and user classification ensure valid recommendations even in case of process evolution. We introduced the concepts of *eagle* and *flock* to describe the recommendation needs of distinct user types.

Future work will focus on evaluating the recommendations in real-world environments within the scope of the Commius project. At the same time, we plan to integrate context constraints to distinguish between process sequences that depend to a large degree on data input and/or specific environmental conditions. This will allow to give even more targeted recommendations. In addition, we intend to investigate clustering techniques for discovering conflicting recommendations in the crowd-centric process model.

## Acknowledgment

This work has been partially supported by the EU STREP project Commius (FP7-213876).

## References

1. Dustdar, S.: Caramba Process-Aware Collaboration System Supporting Ad hoc and Collaborative Processes in Virtual Team. Distributed Parallel Databases 15(1), 45–66 (2004)
2. Burkhart, T., Loos, P.: Flexible business processes - evaluation of current approaches. In: Proceedings of Multikonferenz Wirtschaftsinformatik - MKWI 2010 (2010)

3. Huth, C., Erdmann, I., Nastansky, L.: Groupprocess: Using process knowledge from the participative design and practical operation of ad hoc processes for the design of structured workflows. In: HICSS (2001)
4. Dellen, B., Maurer, F., Pews, G.: Knowledge based techniques to increase the flexibility of workflow management. In: Data and Knowledge Engineering.North-Holland, Amsterdam (1997)
5. Regev, G., Soffer, P., Schmidt, R.: Taxonomy of flexibility in business processes. In: BPMDS (2006)
6. Adams, M., Hofstede, A., Edmond, D., van der Aalst, W.: Facilitating flexibility and dynamic exception handling in workflows through worklets. In: CAiSE 2005, pp. 45–50 (2005)
7. Reijers, H., Rigter, J., Aalst, W.V.D.: The case handling case. International Journal of Cooperative Information Systems 12, 365–391 (2003)
8. Sadiq, S.W., 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)
9. Soffer, P.: On the notion of flexibility in business processes. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS. Springer, Heidelberg (2005)
10. Müller, R., Greiner, U., Rahm, E.: Agent work: a workflow system supporting rule-based workflow adaptation. Data Knowl. Eng. 51(2), 223–256 (2004)
11. Adamides, E.D., Stamboulis, Y., Pomonis, N.: Modularity and strategic flexibility: a cognitive and dynamic perspective. In: Systems Dynamics Society Conference 2005 (2005)
12. Polyvyanyy, A., Weske, M.: Flexible process graph: A prologue. In: OTM Conferences, vol. (1), pp. 427–435 (2008)
13. Adams, M., Edmond, D., ter Hofstede, A.H.M.: The application of activity theory to dynamic workflow adaptation issues. In: 7th Pacific Asia Conference on Information Systems, pp. 1836–1852 (2003)
14. Eichholz, C., Dittmar, A., Forbrig, P.: Using task modelling concepts for achieving adaptive workflows. In: EHCI/DS-VIS, pp. 96–111 (2004)
15. Stoitsev, T., Scheidl, S., Spahn, M.: A framework for light-weight composition and management of ad-hoc business processes. In: Winckler, M., Johnson, H., Palanque, P. (eds.) TAMODIA 2007. LNCS, vol. 4849, pp. 213–226. Springer, Heidelberg (2007)
16. Pesic, M., van der Aalst, W.M.P.: A declarative approach for flexible business processes management. In: Business Process Management Workshops, pp. 169–180 (2006)
17. Pesic, M., Schonenberg, M.H., Sidorova, N., van der Aalst, W.M.P.: Constraint-based workflow models: Change made easy. In: OTM Conferences, vol. (1), pp. 77–94 (2007)
18. Schonenberg, H., Weber, B., Dongen, B., Aalst, W.: 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)
19. Almeida, T., Vieira, S.C., Casanova, M.A.: Flexible workflow execution through an ontology-based approach. In: Workshop on Ontologies as Software Engineering Artifacts, OOPSLA (2004)
20. Vanderfeesten, I.T.P., Reijers, H.A., van der Aalst, W.M.P.: Product based workflow support: Dynamic workflow execution. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 571–574. Springer, Heidelberg (2008)
21. TIBCO, Software, Inc.: Tibco iprocess conductor (2007)
22. Gaaloul, W., Baïna, K., Godart, C.: Log-based mining techniques applied to web service composition reengineering. Service Oriented Computing and Applications 2(2-3), 93–110 (2008)
23. Burkhart, T., Werth, D., Loos, P.: ”Commius An Email Based Interoperability Solution Tailored For SMEs. Journal Of Digital Information Management 6 (2008)

# A Collaborative Approach to Maturing Process-Related Knowledge

Hans Friedrich Witschel[1], Bo Hu[1], Uwe V. Riss[1],
Barbara Thönssen[2], Roman Brun[2], Andreas Martin[2], and Knut Hinkelmann[2]

[1] SAP AG, Dietmar-Hopp-Allee 16
69190 Walldorf, Germany
{Hans-Friedrich.Witschel,Bo01.Hu,Uwe.Riss}@sap.com

[2] University of Applied Sciences Northwestern Switzerland (FHNW),
Institut for Business Information Systems,
Riggenbachstr. 16, 4600 Olten, Switzerland
{Barbara.Thoenssen,Roman.Brun,Andreas.Martin,Knut.Hinkelmann}@fhnw.ch

**Abstract.** We introduce a new approach supporting knowledge workers in sharing process-related knowledge. It is based on the insight that - while offering valuable context information - traditional business process modelling approaches are too rigid and inflexible to capture the actual way processes are executed. Therefore, business process models are made agile and open for changes during execution. To achieve this, the strict distinction between build time modelling and run time execution are softened and process activities are represented to the users in a way that allows for individual adaptations. That can be done by attaching resources, commenting on an issue or adding problems and solutions to an activity or process. In addition activities can be delegated or new (sub-)activities can be added. Thus, the model can adapt to the reality of actual process executions and valuable resources and experiences are proactively presented to users in the right context. A double-staged approach is chosen to apply the model in the real application scenario of a university.

## 1 Introduction

Agility has emerged as an important common characteristic of successful businesses of any size, who benefit from quick response to volatile markets and rapid changing user requirements. In this work, we inspect business agility through the apparatus of knowledge sharing. More specifically, we leverage the process-related knowledge, in terms of knowledge about processes (process knowledge) and knowledge needed in processes (functional knowledge), to increase the agility of organisations. As such knowledge is used and generated during work, its sharing and maturing has to be aligned with business processes that faithfully reflect an organisation's core and mission-critical activities. Businesses need to make sure that employees, participating in mission-critical activities, share the acquired process-related knowledge to keep established experience within the organisation and optimise its performance in the face of employee fluctuation.

R. Hull, J. Mendling, and S. Tai (Eds.): BPM 2010, LNCS 6336, pp. 343–358, 2010.

In practice, we deal with process-related knowledge with the help of Business Process Management Systems (BPMS) and/or Workflow Management Systems (WfMS). BPMS mainly manipulate process knowledge on the business level by making process structures and resources explicit and by supporting process improvement. But they lack process automation. This is the function of WfMS, which automate process execution (see [28], pp 8f). WfMS's modelling functionality, however, is specialised for technical aspects and is not rich enough regarding knowledge aspects. When aligned with everyday work activities, however, existing business process modelling and execution approaches may find themselves overstretched in answering the call of agility due to the lack of flexibility and the amount of overhead required for predefined process models. Usually, process models are created by experts who attempt to bring together all relevant knowledge about a certain process and model it in a BPMS. After negotiating and compromising, a resultant process model could truly represent how a process appears under certain generalised circumstances. The model, however, often differs from the reality of process execution [16]. Variations in execution, which are seldom readily documented, become inevitable when applying process models to new situations. The problem cannot be simply remedied with business process reengineering, which is carried out in a structured and systematic way and cannot keep pace with rapidly changing businesses or markets.

Coming from a totally different perspective, knowledge sharing (for all kinds of knowledge) sometimes takes place informally, e.g., via email and telephone or by imitation (apprenticeship). Although this is flexible and can be very efficient at times, it usually restricts the benefits to the persons that are directly taking part in the exchange (i.e., the information is completely lost for all others). Even if employees have documented their experience and made it publicly available (e.g. in a company Wiki, a lessons-learned database, or on a file share), it does not mean that others are aware of the existence of such information. Needless to say there is much less chance that they will be able to find it or that they will even look for it in a given work situation where it is needed.

The intrinsic inadequacy of formal and informal process knowledge sharing inspired us to take an eclectic approach so as to bridge exactly this gap: to learn process models by doing and to enable adding and sharing individual knowledge and experience. This approach enhances agile process modelling [11] with functional knowledge used in a specific process instance and the possibility of its informal exchange through so-called task patterns, during the execution of processes.

Our vision emphasises the participation of users in a succession of phases known as seeding, evolutionary growth and re-seeding in the SER model [5] (originally applied in the area of managing complex design environments). The SER model describes an approach "between the two extremes of 'put-all-the-knowledge-in-at-the-beginning' and 'just-provide-an-empty-framework' ". It combines the strengths and avoids the weaknesses of both top-down and bottom-up approaches, respectively. The SER model assumes that once a seed is taken up by a community, there is a phase in which the knowledge artefacts evolve in a rather uncontrolled way.

According to Fischer, it is necessary not to force users to invest much effort into formalising their contributions since this would interrupt their normal work process (something most people are not prepared to accept). Contribution should be kept simple and will eventually lead to structures that are too redundant and unwieldy to be understood and managed. They are thus pruned and restructured in the reseeding phase, which is done by a knowledge engineer, removes inconsistencies and creates generalisations (i.e. removing pieces of information that are too context-specific) and formalisations of the knowledge. This is exactly what we want to achieve with the task pattern approach [22] that we introduce in this paper as mediator between process modelling and individual task execution. Projected onto the SER model, reseeding in our work is understood as a chance to understand and align the most frequent (and hence possibly most important) contributions to task patterns in order to learn about potential improvements of the original seeds. This realises a continuous improvement of process models and the task patterns based on actual work activities.

The paper is organised as follows: in Section 2 task patterns as the central building block for learning and maturing process-related knowledge is described. This is followed by a description of our approach to monitor performed tasks in order to semi-automatically support process model adaptations. In Section 3 we give an example of application of our new approach. Next, we describe some technical details of the system (Section 4), then give a brief overview on related work (Section 5) before Section 6 concludes.

## 2 Combining Knowledge Intensive Processes with Task Patterns

A business process is a collection of structured activities with a precise goal to be achieved over a period of time. In general, the activities of a process are in a pre-defined order, resources are mapped (e.g. software systems or personnel, via roles) and the process flow is depending on fixed decision rules. The KISS approach [4] aims to bridge the gap of a strict distinction between design time and run time. A knowledge intensive process (KIP) can be regarded as a collection of activities building the 'skeleton' of a business process, some activities of which can be knowledge intensive (called 'KIA'). Whereas ordinary activities are always executed (i.e., in every process instanc), KIAs are optionally executed depending on information specific for the certain process instance. That can be application data, process data or functional data.

KIAs are modelled during build time but their execution is triggered – or suggested – during run time based on rules. If, for example, an application has to be checked, several KIAs could be executed such as 'Refer to an expert', 'Ask for additional material' or 'Clarify with applicant'. Which one is selected within a specific process instance depends on rules operating on run time information: information already provided for the application, decisions taken in previous process instances or data that is available from related information sources, e.g. out of a legacy system maintaining data of former applications.

We will call the concrete instance of an activity (assigned to particular members of an organisation) task, regardless whether it is a KIA or not. A task is a definition of a particular item of work that specifies the requirements and the goal of this work (cf. [1]). We introduce task patterns as abstractions of tasks that provide information and experience that is generally relevant for the task execution. By abstraction we mean common features of a family of similar tasks, which aim at the same goals under similar conditions (for details refer to [3,27]).

In this section, we describe how agile business processes can work together with task patterns to yield a new form of knowledge sharing. Meanwhile, in order to fully understand the way informal process knowledge to be attached to agile business processes, we explain the notion of task patterns more closely.

### 2.1 Task Patterns

In our approach of maturing process knowledge described below, we will introduce a one-to-one relationship between an activity and a task pattern. That is, for each activity of an agile business process there is exactly one task pattern that serves as the basis for collecting information and experience around the tasks. Tasks concretise task patterns and thus instantiate the corresponding activity.

How does this facilitate the transfer of information and experience work? In general, task patterns provide two means for knowledge sharing (cf. [21,27]):

1. Abstraction services: these provide contextual information about resources that can be used in the task – including information objects such as files, but also persons who are to be contacted – or sub-tasks that should be started.
2. Problem/solution objects: These enable users to share experience regarding typical problems that may arise during the execution of a task, together with descriptions of possible solutions.

Users can interact with task patterns in two ways:

**Consuming** information from task patterns: when working on a task T, a suitable task pattern P can be displayed alongside the task. The user can access P's abstraction services to consume the resources that they offer and attach them to the current task T. The same applies to solutions offered for a problem that happens to occur in T.

**Contributing** to task patterns: users can also attach resources to their concrete task T while working on it. This enables them to associate such information with abstraction services or problem/solution objects of the task pattern P and publishing that information to a shared repository.

The approach maintains a clear separation between personal knowledge, contained in the individual task, and public experience, contained in the task pattern. This separation prevents an intermixture of private and public data. Therefore, it makes transparent to the user which task information is exposed and shared with others and which remains under individual control [20].

## 2.2 Sharing Work Experience

Regarding knowledge sharing, each activity of the process model is the basic unit to which information and experience gets attached. This happens via a one-to-one relationship between task patterns and activities (which are instantiated by tasks): for each activity, the corresponding task pattern collects the information and experience that users have attached to it while they were working on a task.

In the following, we will describe how agile business processes and task patterns play together. We differentiate such interplay into one that happens at design time and one that occurs during run time. At design time, for each activity, a draft of a corresponding task pattern is created with the aid of a group of experts who define an initial set of abstraction services and known problems together with their solutions. The initial pattern should be thought of as a seed that triggers the process of attaching experience to a work context. Subject to further refinement, the initial task patterns do not have to be (and will never be) complete in the beginning. These initial patterns are meant to grow larger while their users learn more about the activities and mature over time adapting to the actual way in which they are executed in practice.

All process information is stored in an ontology-based data store. Specifically, this store contains all information on task instances, encapsulated in so-called task description objects (TDOs). TDOs are filled dynamically by the back-end system and loaded into the front-end once a task gets accepted by a user (see below) - they thus serve as a means of communication between the two.

At run time, when the process is actually executed, the existing task pattern for an activity, which we denote by P, will be retrieved. A task pattern management system will first consult the task ontology to retrieve the abstraction services of P. It then helps users to instantiate P by recommending candidate fillers or each abstraction service. More specifically, one proceeds as follows (this flow of action is also depicted in Fig 1):

1. The workflow engine identifies which task T with corresponding task pattern P should be performed next.
2. A task T is instantiated and a set of organisational members is selected as potential executors of T. The selected persons are notified. They can accept or reject the request to execute T.
3. Since T is the instantiation of an activity A and since for each activity A, there is exactly one task pattern P assigned to it, the workflow engine will next retrieve that task pattern P.
4. It then determines the context of T and uses it to retrieve relevant resources that should be added to abstractor services of the task pattern P. These resources will be added to the TDO corresponding to T.
5. Once the first person has accepted the task, the corresponding TDO is fetched, together with the task pattern P. P is enriched with the information in TDO and both the task details and the task pattern are displayed to the user in a task management application.

6. The user can then start working on the task, making use of the information provided in the task pattern P. Resources, but also problem/solution objects can be easily copied from the task pattern into the task, becoming attachments to it.
7. On the other hand, the user can also attach her own resources (that she considers useful in the context of T) to T.
8. Enhancements of a task pattern are stored locally. They will be available if the same user performs another task T' that corresponds to the same A (i.e. when the task pattern is loaded next time). However, if the user chooses to publish the enhancements, she can do so by a simple click, making them available to the others who have to perform a task of type A.
9. When the user has finished working on the task, she sets the status to "completed" and the workflow engine is notified of this.

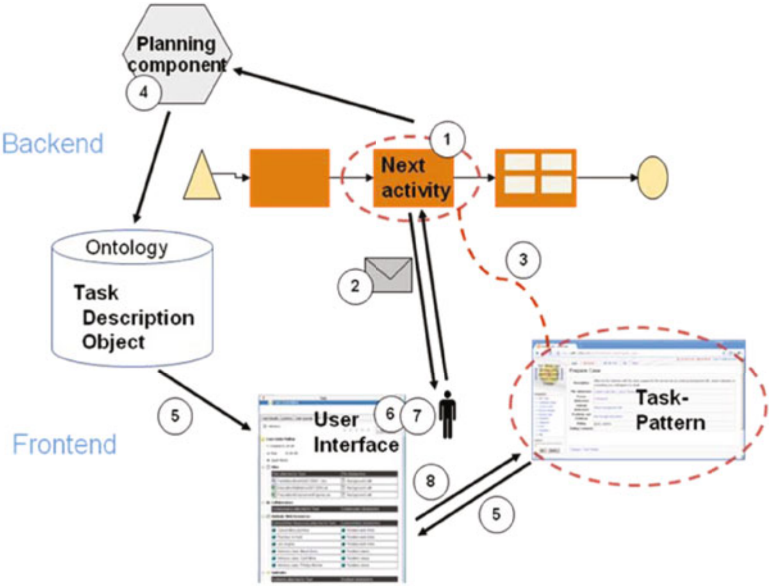


**Fig. 1.** Flow of action during runtime

### 2.3 Learning from Work Experience

As mentioned in the introduction task patterns should be thought of as seeds to aggregate information contributed by end-users. This participation becomes particularly relevant since we do not believe in fully automatic improvements of process models. Instead we envisage a tool that is able to detect deviations and analyse the collected data in order to make suggestions for process changes, in a way similar to work carried out in the area of process mining, e.g. [30]. Based on those blueprints the knowledge engineer will be able to decide on the suggested changes. For process improvement the following aspects will be analysed [3,27]:

**Subtasks** that are frequently added to a task (or as subtask abstractor to the corresponding task pattern): if many users add the same (kind of) subtask to a given task, this indicates that potentially the process model can be improved by including that subtask as a new activity.

**Delegation** of tasks can indicate that either the work balance is not correctly considered or the skills of the assigned persons are not appropriately evaluated. Rules for resource allocation should be adapted accordingly.

**Problem/solution** objects added to a task pattern can be included by other users in their tasks. If many users do so, it means that the problem occurs frequently and that it should be considered for process or task pattern improvements.

**Resources** such as documents or persons can be attached to abstractor services of task patterns, indicating the contexts in which they are useful (namely the process, and activity, task, respectively they are being used in). An analysis of these contexts is generally of interest as it may help to categorise the resources according to their domains of application. Similarly, an analysis of the set of all documents attached to task patterns can lead to a categorisation of documents based on the type of situation(s) in which they are consumed.

In all these cases, an initial step in the analysis is aligning the corresponding items with each other, e.g. to find out that two problem descriptions refer to the same (type of) problem in reality.

## 3 An Application Scenario

This section presents the scenario of an evaluation study that was performed within the University of Applied Sciences Northwestern Switzerland (FHNW) in the context of the EU-funded project MATURE to elicit application scenarios and requirements for the ICT system that is being built to realise the process-related knowledge maturing concepts presented in this paper. Therefore, a prototype (later also called "demonstrator") was built that implements the concepts described here.

The model for the business process of matriculation is shown in Fig 2. We can see that the student, the administration office and the dean are involved in the process; tasks can be assigned to the administration office or to the dean as they directly interact with the system. KIAs are highlighted.

The matriculation process starts with a student's application request. After the receipt of the request, several checks of the application have to be executed in a KIP. As it is shown, the KIAs will not be executed in a pre-defined order. The reason is the following: Depending on where the applicant comes from (but also further criteria), different activities have to be performed. E.g. the availability of a matriculation number has only to be checked if the applicant is from Switzerland. Therefore, a *variable process identification and selection service* automatically chooses the needed activities and assigns them to the possible executor of the activity. The determination of study fees is based on given regulations and can be supported using a *constraint checking service* for decision making. Further on, a *resource allocation service* assigns artefacts based on given criteria. For instance, when checking the approval of a university, appropriate websites or experts from a respective nation are attached to the activity.

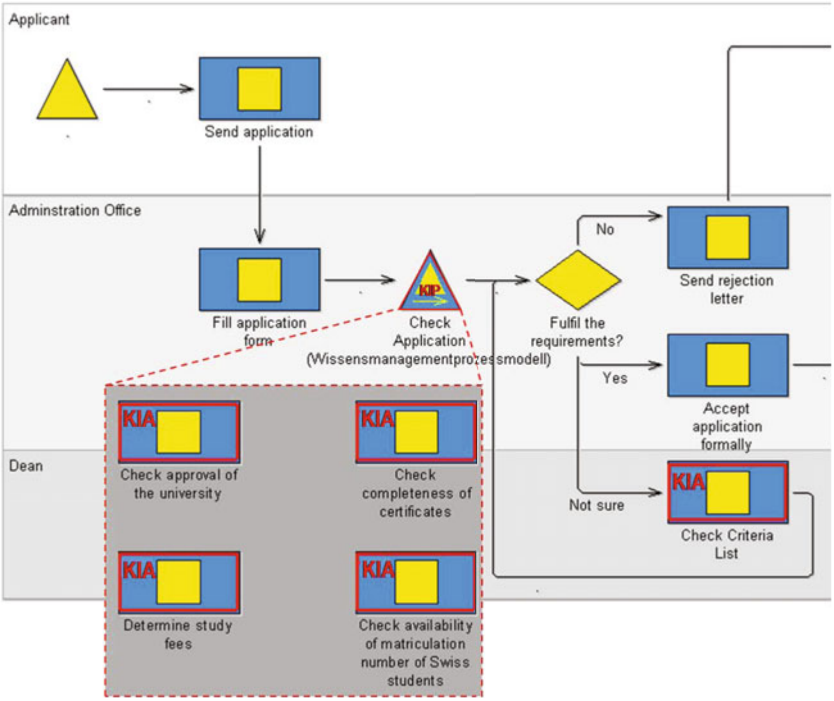
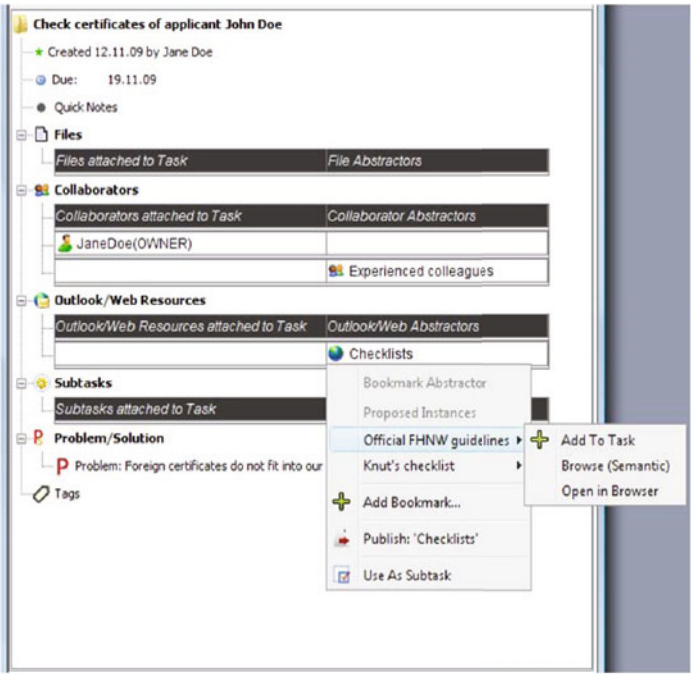

Applicant
Send application
Adminstration Office
Fill application form
KIP
Check Application (Wissensmanagementprozessmodell)
Fulfil the requirements?
No
Yes
Not sure
Send rejection letter
Accept application formally
Dean
KIA
Check Criteria List
Check approval of the university
Check completeness of certificates
Determine study fees
Check availability of matriculation number of Swiss students

**Fig. 2.** First part of the matriculation process model and the KIP sub-process "Check application"

After these checks, the process goes on and it is decided whether the needed requirements are fulfilled or not. The *branching and decision making service* can be invoked in order to decide, whether it is already clear at this stage that the requirements cannot be fulfilled by the applicant and a rejection letter has to be sent. Otherwise the applicant is invited to an interview. Afterwards an interview will be held, the application dossier will be updated and a commission meeting will be held to decide about the acceptance or rejection of the applicant. The process continues with mainly administrative activities until it reaches the end.

### 3.1 Example of Task Pattern Application

Now, we illustrate the application of task patterns by giving an example from the matriculation scenario described above. Let us consider the task of checking the completeness of an applicant's certificates (part of the knowledge intensive sub-process "Check Application" displayed in Fig 2).


Check certificates of applicant John Doe

Created 12.11.09 by Jane Doe

Due: 19.11.09

Quick Notes

Files

| Files attached to Task | File Abstractors |
|---|---|

Collaborators

| Collaborators attached to Task | Collaborator Abstractors |
|---|---|
| JaneDoe(OWNER) | |
| | Experienced colleagues |

Outlook/Web Resources

| Outlook/Web Resources attached to Task | Outlook/Web Abstractors |
|---|---|
| | Checklists |

Subtasks

| Subtasks attached to Task | |
|---|---|

Problem/Solution

Problem: Foreign certificates do not fit into our

Tags

**Fig. 3.** An example of a task pattern and how to consume information from it

Fig 3 shows a task pattern that corresponds to this activity. More precisely, the details of the current task - namely "Check certificates of applicant John Doe" - are displayed (e.g. due date and owner of the task) on the left-hand side. The task pattern with its abstractor services (called "abstractors" in the UI) is displayed on the right-hand side. We can see two abstractor services: i) experienced colleagues: colleagues who have handled many applications; ii) checklists: lists that provide guidance as to what should be checked and how.

The figure also shows a context menu that appears when right-clicking on the latter abstractor service. It contains the resources that are offered by the abstractor service. Clicking on "Add to Task" will result in a resource being displayed on the left-hand-side, vertically aligned with the corresponding abstractor service. Imagine, for example, that a new colleague by the name of Jane has to work on the task at hand and needs to get acquainted with the official guidelines.

In that case, she would access the information in the abstractor service "Checklists" and consult the "Official FHNW guidelines" as depicted in the figure. A similar context menu exists for resources on the left-hand side, allowing these to be added to abstractor services and thus for the contribution of resources to task patterns. For example, Jane might discover – after having worked for some time on various student applications – that a few additional things usually need to be checked, which she documents in her own private checklist that she attaches to her tasks. Via the context menu, she can add this checklist to the "Checklist" abstractor service such that it becomes available for other users. Thus, using these context menus, end-users can easily consume information offered in task patterns and contribute to them.

### 3.2 Demonstrator Evaluation

In order to verify the automated process knowledge maturing, comprehensive use of the approach is necessary. Therefore an evaluation phase of one month where the demonstrator was used, including intermediate and post-evaluation interviews, was conducted.

The main aim of the evaluation was to use the demonstrator in a productive environment. Therefore it was of special interest whether KISSmir addresses the clear need and/or problem in the tester's context. The post-interview should give an insight about the use of all functionalities, suggestions, tasks patterns, knowledge sharing but also technology acceptance. Of further interest was the perceived degree of support of the demonstrator regarding maturing of process-related knowledge and how it could be improved.

The results showed first that process adaptations do actually surface as a result of the use of the demonstrator. Although the matriculation process was modeled together with the end users, some adaptations of it were detected to be necessary during productive use. Furthermore necessary changes in rules were identified. As a second point, the process support was experienced as big benefit of using the demonstrator. The end users liked to be reminded about the tasks needed to be executed and being guided by the sequence. As the number of tasks and their sequence varies for each process execution, they didn't have to think about which tasks need to be done and were able to accomplish the tasks more quickly. Thirdly, the resource recommender functionality was analysed. For each task, experts, historical cases and web-links were proposed (if available) by the demonstrator. The post-evaluation interview showed that with exception of web-links these suggestions have been used very seldom. The main reason for the limited use was mainly the unawareness of it and a (too) small knowledge base. Further features as quick notes which can be added to any task at any time and problem/solution objects were perceived as being useful. Last but not least the knowledge sharing functionalities were analysed. By adding resources to task patterns and publish this information, knowledge can be shared. The same can be done with problem/solution descriptions. However, similar as for the suggestions, the interviewed persons think these are nice functionalities as needed information could be found easier and faster, but did not use it extensively.

# 4 Implementation

## 4.1 Process Modelling and Execution

The modelling of an adaptive business process can be performed in a semantic modelling environment like ATHENE [9] or WSMO-Studio [2]. ATHENE provides the creation of several models (process model, organisational model, etc.) based on ontologies. The activities of the processes will be linked with the related task patterns and resources (files, roles, etc.) which are stored in semantic repositories (❶ in Fig 4). Further on, the process model can be enhanced with adaptivity services (❷). After modelling the process in ATHENE it can be transferred to the execution framework and stored in a semantic repository (❸). The model is represented in a knowledge representation language like RDF/S[1] or OWL[2] and will be transformed via a transformation service into a process execution language like BPEL[3] or XPDL[4].

The transformed process can be executed in the execution framework (e.g., BPEL workflow engine) (❹). The process can access the linked resources which are stored in the semantic repositories. During run time the process can invoke the defined adaptivity services. The "task management service", which is part of the process framework, invokes the task GUI (graphical user interface) (❺). The instance management service stores and holds the instances.

## 4.2 Task Pattern and Task Management

Tasks and task patterns are delivered to the user through a Personal Task Management infrastructure. That infrastructure is part of the NEPOMUK Social Semantic Desktop [7].

It consists of a semantic task management framework (STMF [19]) which offers task-related (web) services over the entire desktop and handles the manipulation, storage and retrieval of all task and task pattern-related information; information is stored locally in the RDF repository of the Social Semantic Desktop in a way that ensures seamless semantic integration of information objects and task representations.

As a user interface, the KASIMIR sidebar [6] has been developed, which builds on the STMF task services and makes Task Management functionality available to end- users. KASIMIR allows users to assign basic task properties and to attach involved persons, information objects and subtasks. It also allows users to view task patterns attached to a task, consume its resources and contribute In addition to the local storage of personal task (pattern) information, there is a server component that stores public task patterns. It is based on Semantic MediaWiki (SMW[5]), which allows the initial modelling and later adaptation

[1] http://www.w3.org/TR/rdf-schema/

[2] http://www.w3.org/TR/owl-features/

[3] http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[4] http://www.wfmc.org/xpdl.html

[5] http://semantic-mediawiki.org/

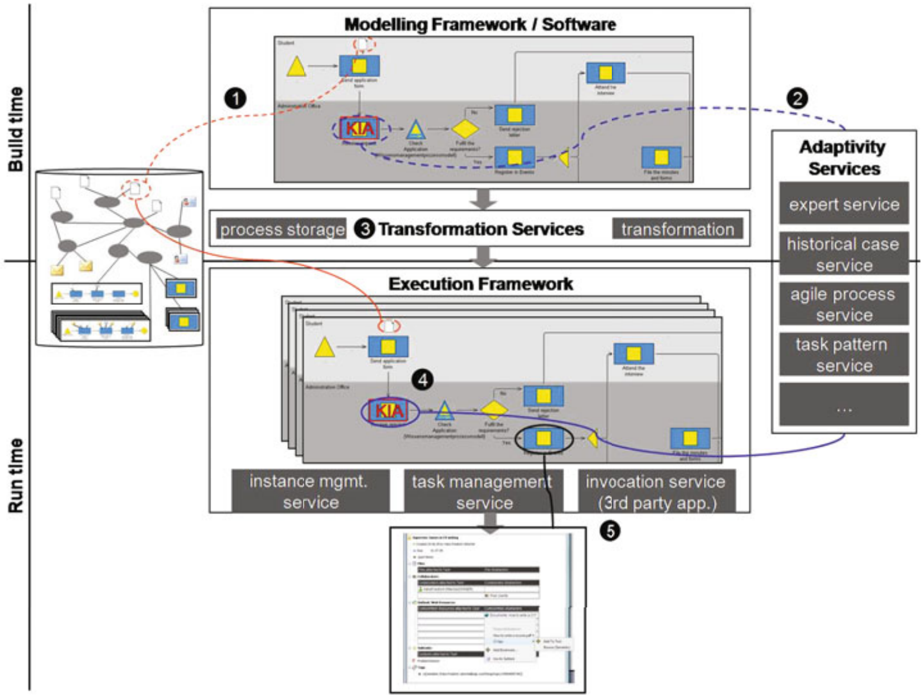


**Fig. 4.** Customer Information Portal Architecture

of public task patterns over the Wiki's user interface. Entities related to task patterns are modelled as SMW semantic templates.

## 5 Related Work

Making business processes agile to meet the requirements of knowledge intensive work and faster changing business environments is a topic that has already been addressed for some time. It is guided by the insight that in knowledge intensive processes the particular sequence of tasks is often variable and depends on the information at hand. Traditionally Workflow Management Systems (WfMS) distinguish between design time and run time [12] and it is the dependency of the process on input information that makes this distinction to become blurry. An overview of approaches to tackle this problem can be found in [22].

Usually a process model containing all activities and resources is created during design time. Flexibility is provided through modelling choices and merge-constructs. This can lead to highly complex models which are hard to maintain [25]. In addition, especially in the tertiary sector the processes are knowledge-intensive and cannot be foreseen for all exceptional situations and circumstances. During run time exceptional situations, unforeseeable events and unpredictable situations have to be dealt with. Therefore van der Aalst et al.

introduced case handling 'as a new paradigm for supporting flexible business processes' [31] in order to avoid predefined process execution.

However, supporting flexible process execution does not cover all of the dimensions of change in business processes (dynamism, adaptability, flexibility) as introduced by Sadiq et al. [25]. For this, tracking and mining of the actual process/task variations users perform are necessary.

To support agility, semantic technologies have been used in several approaches, amongst others by [8] for process implementation and querying by [14] to build their 'agent based business process management system' or by [22] to facilitate task patterns. Especially the pattern approach is tightly built on semantic technologies and an approach for combining it with Service-Oriented Architecture has been proposed [23]. Another semantic approach to process management based on Unified Activity Management has been suggested by Moran et al. [18]. A general overview is given by [17]. Recently, Feldkamp, Hinkelmann et al. introduced the 'KISS approach' combining semantically enriched process models with business rules ([18], [10]). Although this approach reaches the flexibility to execute agile processes, two aspects are not yet covered: a) how to share the knowledge gained through task handling without cumbersome publishing and b) how to automatically detect execution variances. In this paper we have shown how deviations between the actual process execution and the process model can be identified and how adaptations can be recommended automatically. Question a) is addressed in section 2.2, whereas question b) is detailed in section 2.3.

As far as the world of business process modelling is concerned, approaches to collect and mature process knowledge collaboratively are scarce. [15] has proposed an architecture that integrates knowledge management and business process management. However, this approach follows a traditional expert-driven way. Approaches in the field of process mining (e.g. [31]) - which try to extract process knowledge from implicit information contained in system event logs - exploit user interaction, but do not actually encourage explicit user contributions to an evolving process knowledge repository. With respect to knowledge work this is a complicated task since the nature of individual tasks and the associated experience cannot be identified properly enough to enable successful knowledge proliferation. Sharing process knowledge in a task management environment has been explored, for example, in [13], suggesting to copy information from previous related tasks. Task patterns, as a more elaborate way of mediating experience transfer have been proposed in [22] and elaborated further, cf. e.g. [3,20,27]. In [29], a more process-oriented view of task patterns has been introduced where users can exchange and collaboratively develop lightweight process models.

## 6 Conclusion and Future Work

In this article, we have outlined a new paradigm of knowledge and experience sharing that enhances agile business processes. This is done through connecting activities in formal process models with loosely regulated task patterns emerged from our everyday work. The former gives guidance to the business target while the latter allows us to proceed in a way that best suits the end-users' needs. Task patterns capture how people carry out a task (process knowledge) and how people leverage resources in supporting their solutions (functional knowledge). They thus help to avoid the problem of rigidity inherent in traditional process modelling approaches since it involves the end-users in shaping the support that the model offers and since it eventually adapts to the reality of end-users' process execution.

The results of our evaluation of the prototype with the University of Applied Sciences Northwestern Switzerland are promising: both the conceptual framework and the prototype were well accepted even though some non-technical barriers were identified.

In our approach we have followed the idea of knowledge maturing as a process that understands "[...]learning activities as embedded into, interwoven with, and even indistinguishable from everyday work processes[...]" [26]. According to the knowledge maturing concept learning is seen as a social and collaborative activity, in which individual and organisational learning processes are dynamically interlinked among each other [24]. This approach has been applied to knowledge intensive processes where the continuous collaborative enhancement appears as particularly important due to continuously changing work targets and situations.

Furthermore, we envisage the following improvement to our approach. For the future, it is planned to implement and deploy the agile business process (together with appropriate task patterns) at the project application partners and to observe if and how the intended process knowledge maturing takes place. Compared to other types of knowledge, the evolution of process knowledge is less transparent and thus more difficult to analyse. The interplay between task patterns and process model provides valuable insights. Indeed, after having the process productive over a certain period, a reasonable amount of real-life usage data of tasks and task patterns can be accumulated. Such data provide the ground for automatic or customised business process model updates.

## Acknowledgements

This work is supported by the European Union IST fund through the EU FP7 MATURE Integrating Project (Grant No. 216356).

## References

1. Byström, K., Hansen, P.: Conceptual framework for tasks in information studies. Journal of the American Society for Information Science and Technology 56(10), 1050–1061 (2005)
2. Dimitrov, M., Simov, A., Momtchev, V., Konstantinov, M.: Wsmo studio—a semantic web services modelling environment for wsmo. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, pp. 749–758. Springer, Heidelberg (2007)

3. Du, Y., Riss, U.V., Ong, E., Chen, L., Patterson, D., Wang, H.: Work experience reuse in pattern based task management. In: 9th International Conference on Knowledge Management (I-KNOW), pp. 149–158 (2009)
4. Feldkamp, D., Hinkelmann, K., Thönssen, B.: Kiss: Knowledge-intensive service support: An approach for agile process management. In: Paschke, A., Biletskiy, Y. (eds.) RuleML 2007. LNCS, vol. 4824, pp. 25–38. Springer, Heidelberg (2007)
5. Fischer, G., Grudin, J., McCall, R., Ostwald, J., Redmiles, D., Reeves, B., Shipman, F.: Seeding, evolutionary growth and reseeding: The incremental development of collaborative design environments (2001)
6. Grebner, O., Ong, E., Riss, U.V.: Kasimir: Work process embedded task management leveraging the semantic desktop. In: Multikonferenz Wirtschaftsinformatik (MKWI 2008), Berlin, pp. 715–726. GITO-Verlag (2008)
7. Groza, T., Handschuh, S., Möller, K., Minack, E., Jazayeri, M., Mesnage, C., Reif, G., Gudjónsdóttir, R.: The nepomuk project- on the way to the social semantic desktop (2007)
8. Hepp, M., Leymann, F., Domingue, J., Bussler, C., Wahler, A., Fensel, D.: Semantic business process management: A vision towards using semantic web services for business process management. In: IEEE International Conference on e-Business Engineering (ICEBE), pp. 535–540 (2005)
9. Hinkelmann, K., Nikles, S., von Arx, L.: An ontology-based modeling tool for knowledgeintensive services. In: 1st International Conference on Methodologies, Technologies and Tools Enabling E-Government, pp. 43–56 (2007)
10. Hinkelmann, K., Probst, F., Thönssen, B.: Agile process management framework and methodology. In: AAAI Spring Symposium on Semantic Web Meets e-Government (2006)
11. Hinkelmann, K., Thönssen, B., Probst, F.: Referenzmodellierung für e-government-services. Wirtschaftsinformatik 5, 356–366 (2005)
12. Hollingsworth, D.: The workflow reference model. Workflow Management Coalition (1993)
13. Holz, H., Rostanin, O., Dengel, A., Suzuki, T., Maeda, K., Kanasaki, K.: Task-based process know-how reuse and proactive information delivery in tasknavigator. In: CIKM 2006: Proceedings of the 15th ACM International Conference on Information and Knowledge Management, pp. 522–531. ACM, New York (2006)
14. Jennings, N.R., Norman, T.J., Faratin, P., O'Brien, P., Odgers, B.: Autonomous agents for business process management. Applied Artificial Intelligence: An International Journal 14(2), 145–189 (2000)
15. Jung, J., Choi, I., Song, M.: An integration architecture for knowledge management systems and business process management systems. Computers in Industry 58(1), 21–34 (2007)
16. Karagiannis, D., Junginger, S., Strobl, R.: Introduction to business process management systems. In: Scholz-Reiter, B., Stickel, E. (eds.) Business Process Modelling, pp. 81–106. Springer, Heidelberg (1996)
17. Lautenbacher, F., Bauer, B.: A survey on workflow annotation & composition approaches. In: Workshop on Semantics for Business Process Management, SBPM (2007)
18. Moran, T.P., Cozzi, A., Farrell, S.P.: Unified activity management: supporting people in e-business. ACM Commun. 48(12), 67–70 (2005)
19. Ong, E., Grebner, O., Riss, U.V.: Pattern-based task management: Pattern lifecycle and knowledge management. In: 4th Conference of Professional Knowledge Management (WM 2007), vol. 2, pp. 357–364 (2007)

20. Riss, U.V., Cress, U., Kimmerle, J., Martin, S.: Knowledge transfer by sharing task templates: two approaches and their psychological requirements. Knowledge Management Research & Practice 5(4), 287–296 (2007)
21. Riss, U.V., Grebner, O., Du., Y.: Task journals as means to describe temporal task aspects for reuse in task patterns. In: 9th European Conference on Knowledge Management, pp. 721–730 (2008)
22. Riss, U.V., 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)
23. Riss, U.V., Weber, I., Grebner, O.: Business process modelling, task management, and the semantic link. In: AAAI Spring Symposium AI Meets Business Rules and Process Management, pp. 99–104 (2009)
24. Riss, U.V., Witschel, H.F., Brun, R., Thönssen, B.: What is organizational knowledge maturing and how can it be assessed? In: 9th International Conference on Knowledge Management (I-KNOW), pp. 28–38 (2009)
25. Sadiq, S., Sadiq, W., Orlowska, M.: 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)
26. Schmidt, A., Hinkelmann, K., Ley, T., Lindstaedt, S.N., Maier, R., Riss, U.: Conceptual foundations for a service-oriented knowledge and learning architecture: Supporting content, process and ontology maturing. In: Pellegrini, T., Auer, S., Tochtermann, K., Schaffert, S. (eds.) Networked Knowledge - Networked Media, ch. 6, vol. 221, pp. 79–94. Springer, Heidelberg (2009)
27. Schmidt, B., Riss, U.V.: Task patterns as means to experience sharing. In: Spaniol, M., Li, Q., Klamma, R., Lau, R.W.H. (eds.) ICWL 2009. LNCS, vol. 5686, pp. 353–362. Springer, Heidelberg (2009)
28. Workflow Management Coalition Specification. Workflow Management Coalition, Terminology & Glossary (Document No. WFMC-TC-1011). Workflow Management Coalition Specification (February 1999)
29. Stoitsev, T., Scheidl, S., Spahn, M.: A framework for light-weight composition and management of ad-hoc business processes. pp. 213–226 (2007)
30. van der Aalst, W., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering 16(9), 1128–1142 (2004)
31. 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)

# Author Index