Kurt Jensen Wil M.P. van der Aalst (Eds.)

Transactions on Petri Nets and Other Models of Concurrency II

Special Issue on Concurrency in Process-Aware Information Systems

Springer

Editor-in-Chief

Kurt Jensen
University of Aarhus
Faculty of Science
Department of Computer Science
IT-parken, Aabogade 34, 8200 Aarhus N, Denmark
E-mail: kjensen@cs.au.dk

Guest Editor

Wil M.P. van der Aalst
Technical University of Eindhoven
Department of Mathematics and Computer Science
Den Dolech 2, 5612 AZ Eindhoven, The Netherlands
E-mail: w.m.p.v.d.aalst@tue.nl

Library of Congress Control Number: Applied for

CR Subject Classification (1998): D.2.2-4, D.2, I.6, F.3-4, H.2.3, C.4

ISSN	0302-9743 (Lecture Notes in Computer Science)
ISSN	1867-7193 (Transactions on Petri Nets and Other Models of Concurrency)
ISBN-10	3-642-00898-4 Springer Berlin Heidelberg New York
ISBN-13	978-3-642-00898-6 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12631339 06/3180 5 4 3 2 1 0

Preface by Editor-in-Chief

This is the second volume of the Journal entitled "LNCS Transactions on Petri Nets and Other Models of Concurrency" (ToPNoC). This special issue of ToPNoC focuses on a particular topic: *Concurrency in Process-Aware Information Systems*. Like some of the volumes in the earlier "Advances in Petri Nets" series, this volume provides a comprehensive state-of-the-art overview on a more focused topic. Process-Aware Information Systems have become one of the most important application domains of Petri nets. For example, workflow technology is driven by languages closely related to Petri nets, and various analysis techniques ranging from workflow verification to process mining benefit from decades of concurrency research. This explains why this first special issue of ToPNoC is devoted to Process-Aware Information Systems.

As Editor-in-Chief of ToPNoC, I would like to thank the editor of this special issue: Wil van der Aalst. Moreover, I would like to thank all authors and reviewers whose efforts have contributed to this interesting and highly relevant ToPNoC volume.

January 2009

Kurt Jensen
Editor-in-Chief
LNCS Transactions on Petri Nets and Other Models of Concurrency (ToPNoC)

LNCS Transactions on Petri Nets and Other Models of Concurrency: Aims and Scope

ToPNoC aims to publish papers from all areas of Petri nets and other models of concurrency ranging from theoretical work to tool support and industrial applications.

The foundation of Petri nets was laid by the pioneering work of Carl Adam Petri and his colleagues in the early 1960s. Since then, an enormous amount of material has been developed and published in journals and books and presented at workshops and conferences.

The annual International Conference on Application and Theory of Petri Nets and Other Models of Concurrency started in 1980. The International Petri Net Bibliography maintained by the Petri Net Newsletter contains close to 10,000 different entries, and the International Petri Net Mailing List has 1,500 subscribers. For more information on the International Petri Net community, see: `http://www.informatik.uni-hamburg.de/TGI/PetriNets/`

All issues of ToPNoC are LNCS volumes. Hence they appear in all large libraries and are also accessible online in Springerlink (electronically). Simultaneously the ToPNoC volumes form a Journal, and it is possible to subscribe to ToPNoC without subscribing to the rest of LNCS.

ToPNoC contains:

– Revised versions of a selection of the best papers from workshops and tutorials at the annual Petri net conferences
– Special sections/issues within particular subareas (similar to those published in the *Advances in Petri Nets* series)
– Other papers invited for publication in ToPNoC
– Papers submitted directly to ToPNoC by their authors

Like all other journals, ToPNoC has an Editorial Board, which is responsible for the quality of the journal. The members of the board assist in the reviewing of papers submitted or invited for publication in ToPNoC. Moreover, they may make recommendations concerning collections of papers proposed for inclusion in ToPNoC as special sections/issues. The Editorial Board consists of prominent researchers within the Petri net community and in related fields.

Topics

System design and verification using nets; analysis and synthesis, structure and behavior of nets; relationships between net theory and other approaches; causality/partial order theory of concurrency; net-based semantical, logical and algebraic calculi; symbolic net representation (graphical or textual); computer tools for nets; experience with using nets, case studies; educational issues related to nets; higher level net models; timed and stochastic nets; and standardization of nets.

Applications of nets to different kinds of systems and application fields, e.g.: flexible manufacturing systems, real-time systems, embedded systems, defense systems, biological systems, health and medical systems, environmental systems, hardware structures, telecommunications, railway networks, office automation, workflows, supervisory control, protocols and networks, the Internet, e-commerce and trading, programming languages, performance evaluation, and operations research.

For more information about ToPNoC, please see: `www.springer.com/lncs/topnoc`.

Submission of Manuscripts

Manuscripts should follow LNCS formatting guidelines, and should be submitted as PDF or zipped PostScript files to ToPNoC@cs.au.dk. All queries should be addressed to the same e-mail address.

LNCS Transactions on Petri Nets and Other Models of Concurrency: Editorial Board

Preface by Guest Editor

Process-aware information systems support operational business processes by combining advances in information technology with recent insights from management science. Workflow management systems are typical examples of such systems. However, many other types of information systems are also "process aware" even if their processes are hard-coded or only used implicitly (e.g., ERP systems). The shift from data orientation to process orientation has increased the importance of process-aware information systems. Moreover, advanced analysis techniques ranging from simulation and verification to process mining and activity monitoring allow for systems that support process improvement in various ways.

Information technology has changed business processes within and between enterprises. More and more work processes are being conducted under the supervision of information systems that are driven by process models. Examples are workflow management systems such as FLOWer, FileNet, and Staffware; advanced middleware software such as WebSphere; enterprise resource planning systems such as SAP and Oracle; as well as many domain specific systems. It is hard to imagine enterprise information systems that are unaware of the processes taking place. Although the topic of business process management using information technology has been addressed by consultants and software developers in depth, more fundamental approaches towards such Process-Aware Information Systems (PAISs) have been rather uncommon. It wasn't until the 1990s that researchers started to work on the foundations of PAISs. Clearly, concurrency theory is an essential ingredient in these foundations as business processes are highly concurrent involving all types of routing logic and resource allocation mechanisms.

PAISs play an important role in Business Process Management (BPM). There exist many definitions of BPM. Here we will use the following definition: "Business process management (BPM) is a field of knowledge that combines knowledge from information technology and knowledge from management sciences and applies this to operational business processes". BPM can be seen as an extension of Workflow Management (WFM), which primarily focuses on the automation of business processes.

The term "business process" should be interpreted in a broad sense. It also encompasses other types of operational processes that need to be supported. For example, scientific computing using grid technology is definitely included when talking about process-aware information systems. The challenges in scientific workflows go beyond classical workflows and, e.g., grid technology will enable new forms of BPM.

Different models of concurrency have been used to model workflows, typically aiming at their analysis. For example, a particular variant of Petri nets, called workflow nets, has been widely used for the modeling and analysis of workflows. Moreover, many workflow management systems use a graphical notation close to Petri nets. There are interesting challenges in the modeling of PAISs, e.g., high-level Petri nets can be used to describe process languages and standards (EPCs, YAWL, BPMN, BPEL, etc.) and the architectures of the systems supporting these languages. The resulting models can be used for all kinds of analysis, e.g., verification and performance analysis. Moreover, there are also interesting challenges in the discovery of concurrent processes (cf. process mining techniques).

This special issue of ToPNoC focuses on concurrency in PAISs. The papers in this ToPNoC volume address the various problems related to designing, implementing, and analyzing PAISs. Many of the papers address the concurrency aspect by using Petri nets. Different papers address the issue of language transformations either for the purpose of interoperability or analysis. In many cases, Petri-net-based analysis techniques are used to verify a certain property. There is a special focus on interacting systems as these are more difficult to handle. This focus is triggered by the emerging service-oriented architectures. Some papers also address problems related to synthesis and process mining in a PAIS environment. Note that information systems increasingly produce enormous event logs that can be used for analysis purposes.

This volume of ToPNoC is interesting both for researchers working in concurrency theory and people working on business process management as it identifies and addresses the challenges posed by PAISs. Moreover, it is also valuable for practitioners involved in the development of new PAISs or the application of existing systems.

This topical volume of ToPNoC contains 16 papers. The authors of these papers were invited to submit a paper based on their expertise in this area. All invited papers were reviewed by four referees. In the first round of reviews, some papers were rejected, some were accepted (with minor revisions), and some were asked to submit a revised version. Based on a second round of reviewing (again by three or four referees), the final decisions were made. As Editor of the special issue, I would like to thank the reviewers and authors for doing an outstanding job. I would also like to thank the two secretaries involved in preparing the final version of this volume: Ine van der Ligt (Eindhoven University of Technology) and Dorthe Haagen Nielsen (University of Aarhus). I would also like to thank the Springer LNCS/ToPNoC team for handling things in an efficient and effective manner.

In the remainder of this preface, the contributions are briefly summarized.

The first paper titled "Process-Aware Information Systems: Lessons to Be Learned from Process Mining" serves two purposes. On the one hand, it provides an introduction to this special issue by giving an overview of the domain. On the other hand, it presents a rather personal view on the research in this area and the lessons that can be learned from recent insights provided by process mining.

The second paper "Model-Based Software Engineering and Process-Aware Information Systems" by Ekkart Kindler aims to bridge the gap between Model-Based Software Engineering (MBSE) and PAIS.

The paper “Petri Net Transformations for Business Processes: A Survey” by Niels Lohmann, Eric Verbeek, and Remco Dijkman focuses on the challenges related to translating one process language to another. The authors discuss transformations related to BPMN, YAWL, EPCs, Petri nets, and BPEL.

Frank Puhlmann and Mathias Weske discuss the π-calculus as a formal foundation for PAIS in their paper “A Look Around the Corner: The Pi-Calculus”. They use the workflow patterns to discuss the applicability of the π-calculus in this domain.

The paper “newYAWL: Towards Workflow 2.0” by Nick Russell and Arthur ter Hofstede focuses on a concrete workflow language: newYAWL. Using colored Petri nets, the semantics of newYAWL and architectural considerations are discussed.

Michael Köhler-Bußmeier, Matthias Wester-Ebbinghaus, and Daniel Moldt present a Petri-net-based organizational model called SONAR in their paper “A Formal Model for Organisational Structures behind Process-Aware Information Systems”. The goal is to devote more attention to organizational aspects in an integrated manner.

The paper “Flexibility in Process-Aware Information Systems” by Manfred Reichert, Stefanie Rinderle-Ma, and Peter Dadam focuses on one of the most important challenges for PAISs: flexibility. After formulating some requirements, it is shown how these are addressed in the ADEPT2 tool.

The paper “Business Grid: Combining Web Services and the Grid” by Ralph Mietzner, Dimka Karastoyanova, and Frank Leymann discusses requirements for the so-called “business grid”. While grids are studied in detail in the context of e.g., scientific workflows, the grid community is disconnected from the BPM community and this paper advocates the unification of results from both domains.

Karsten Wolf focuses on service interaction in his paper “Does My Service Have Partners?”. The paper provides results for both the controllability of single-port services and multiple-port services.

The paper “Deciding Substitutability of Services with Operating Guidelines” by Christian Stahl, Peter Massuthe, and Jan Bretschneider is related to the paper by Karsten Wolf. Using the so-called “operating guidelines” three substitutability notions are operationalized.

The paper “A Framework for Linking and Pricing No-Cure-No-Pay Services” by Kees van Hee, Eric Verbeek, Christian Stahl, and Natalia Sidorova also looks at services but now from a “pricing” point of view. It is shown how a Petri-net-based framework can be used to compute the price of a service orchestration.

In his paper “Empirical Studies in Process Model Verification” Jan Mendling addresses the need for more empirical research in this area. He discusses several large-scale verification studies and the lessons that can be learned from this.

While most of the papers discussed so far mainly focus on the design and analysis of process models or systems, the last four papers focus on process mining, i.e., the analysis of behavior observed in real life (e.g., based on event logs, example scenarios, or interactions).

The paper "Process Mining: Overview and Outlook of Petri Net Discovery Algorithms" by Boudewijn van Dongen, Ana Karla Alves de Medeiros, and Lijie Wen gives an overview of existing mining techniques. Using various quality notions, 13 Petri-net discovery algorithms (all available in ProM) are reviewed.

The paper "Construction of Process Models from Example Runs" by Robin Bergenthum, Jörg Desel, Sebastian Mauser, and Robert Lorenz proposes a novel approach for the automatic construction of Petri nets based on example runs. In this work, example scenarios rather than event logs are used as input.

Hong-Linh Truong and Schahram Dustdar address the need for the analysis of service interaction in their paper "Online Interaction Analysis Framework for Ad-Hoc Collaborative Processes in SOA-Based Environments". The paper presents the VOIA (Vienna Online Interaction Analysis) framework and discusses some experiments.

The paper "Exploiting Inductive Logic Programming Techniques for Declarative Process Mining" by Federico Chesani, Evelina Lamma, Paola Mello, Marco Montali, Fabrizio Riguzzi, and Sergio Storari concludes this special issue. The paper combines inductive logic programming with a declarative language to discover less structured processes. The approach is implemented as a ProM plug-in called DecMiner.

The papers in this ToPNoC volume provide a good overview of PAIS research. Some papers focus on the foundations of PAIS, while others try to apply existing (Petri-net-based) techniques to business process management. Therefore, this volume provides a useful blend of theory, practice, and tools related to concurrency and PAIS research.

January 2009 Wil van der Aalst
Guest Editor, Special Issue of ToPNoC on Concurrency in PAIS

Organization of This Issue

Guest Editor

Wil van der Aalst, The Netherlands

Referees

Table of Contents

Process-Aware Information Systems: Lessons to Be Learned from Process Mining

Wil M.P. van der Aalst

Department of Mathematics and Computer Science,
Eindhoven University of Technology
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
w.m.p.v.d.aalst@tue.nl

Abstract. A *Process-Aware Information System* (PAIS) is a software system that manages and executes operational processes involving people, applications, and/or information sources on the basis of process models. Example PAISs are workflow management systems, case-handling systems, enterprise information systems, etc. This paper provides a brief introduction to these systems and discusses the role of process models in the PAIS life-cycle. Moreover, it provides a *critical reflection on the state-of-the-art based on experiences with process mining.* Process mining techniques attempt to extract non-trivial and useful information from event logs. One aspect of process mining is control-flow discovery, i.e., automatically constructing a process model (e.g., a Petri net) describing the causal dependencies between activities. The insights provided by process mining are very valuable for the development of the next generation PAISs because they clearly show a mismatch between the models proposed for driving these systems and reality. On the one hand, models tend to oversimplify things resulting in systems that are too restrictive. On the other hand, models fail to capture important aspects of business processes.

1 Introduction

In the last two decades there has been a shift from "data-aware" information systems to "process-aware" information systems [24]. To support business processes, an enterprise information system needs to be aware of these processes and their organizational context. Early examples of such systems were called WorkFlow Management (WFM) systems [4,28,33,36,38,41,45,67]. In more recent years, vendors prefer the term Business Process Management (BPM) systems. BPM systems have a wider scope than the classical WFM systems and are not just focusing on process automation. BPM systems tend to provide more support for various forms of analysis (e.g., simulation) and management support (e.g., monitoring). Both WFM and BPM aim to support operational processes that are often referred to as "workflow processes" or simply "workflows". We will use the generic term *Process-Aware Information System* (PAIS) to refer to systems that manage and execute such workflows.

In a Service Oriented Architecture (SOA) the information system is seen as a set of connected services. A PAIS can be realized using such an architecture and in fact it is very natural to see processes as the "glue" connecting services. The fit between SOA and PAIS is illustrated by emerging standards such as BPEL [20] and BPMN [68]. The focus on web services and SOA has stirred up enthusiasm for process-orientation. As a result it is expected that in the future generic PAISs will start to play a more important role. However, at the same time it should not be forgotten that most PAISs are dedicated towards a particular application domain or even a specific company.

The flow-oriented nature of workflow processes makes the Petri net formalism a natural candidate for the modeling and analysis of workflows. Most workflow management systems provide a graphical language which is close to Petri nets. Although the routing elements are different from Petri nets, the informal semantics of the languages used are typically token-based and hence a (partial) mapping to Petri nets is relatively straightforward. This explains the interest in applying Petri nets to PAISs.

The purpose of this paper is twofold. On the one hand, we aim to provide an *introduction to PAISs and the role of models in the development and configuration* of such systems. On the other hand, we would like to share some *insights obtained through process mining*. Process mining exploits event logs of real processes and uses these to discover models or check the conformance of existing ones. Experiences with process mining show that there are typically large discrepancies between the idealized models used to configure systems and the real-life processes. Moreover, process mining has changed our perception of models. For example, there is no such thing as *the* model. In any situation different models are possible all providing a particular view on the process at hand. Based on our experiences using process mining, we would like to challenge some of the basic assumptions related to PAIS and business process modeling.

The remainder of this paper is organized as follows. Section 2 provides a definition and classification of PAISs. The role of process models is discussed in Section 3 and Section 4 briefly introduces the concept of process mining. Section 5 presents the lessons that can be learned from process mining. This section serves as a "reality check" for PAIS research. Section 6 concludes the paper.

2 Process-Aware Information Systems

In this paper we adopt the following definition of a *Process-Aware Information System* (PAIS): *a software system that manages and executes operational processes involving people, applications, and/or information sources on the basis of process models* [24]. Although not explicitly stated in this definition, it should be noted that the process models mentioned are usually represented in some graphical language, e.g., a Petri-net-like notation. The models are typically instantiated multiple times (e.g., for every customer order) and every instance is handled in a predefined way (possibly with variations).

Classical examples of PAISs are WorkFlow Management (WFM) systems and Business Process Management (BPM) systems. These systems support operational business processes and are driven by an explicit process representation. Given the above definition, one can see that a text editor is not "process aware" insofar as it is used to facilitate the execution of specific tasks without any knowledge of the process of which these tasks are part. A similar comment can be made regarding e-mail clients used to send and receive electronic mail. A task in a process may result in an e-mail being sent, but the e-mail client is unaware of the process it is used in. At any point in time one can send an e-mail to any person without being supported nor restricted by the e-mail client. Text editors and e-mail clients (at least contemporary ones) are applications supporting tasks, not processes. The same applies to a large number of applications used in the context of information systems.

Process awareness is an important property for information systems and the shift from task-driven to PAISs brings a number of advantages [24]:

– The use of explicit process models provides a means for communication between people.
– Systems driven by models rather than code have less problems dealing with change, i.e., if an information system is driven by process models, only the models need to be changed to support evolving or emerging business processes.
– The explicit representation of the processes supported by an organization allows their automated enactment. This may lead to a better performance.
– The explicit representation of processes enables management support at the (re)design level, i.e., explicit process models support (re)design efforts.
– The explicit representation of processes also enables management support at the control level. Generic process monitoring and mining facilities provide useful information about the process as it unfolds. This information can be used to improve the control (or even design) of the process.

A detailed introduction PAISs is beyond the scope of this paper. However, to provide an overview of the important issues, we summarize the classification given in [24]. In addition, we refer to the well-known *workflow patterns* [6,58,70].

2.1 Design-Oriented Versus Implementation-Oriented

Figure 1 summarizes the phases of a typical PAIS life-cycle. In the design phase, processes are designed (or re-designed) based on the outputs of a requirements analysis. In the configuration phase, designs are refined into an implementation, typically by configuring a generic infrastructure for a process-aware information system (e.g. a WFM system, a case handing system, or an EAI platform). After configuration, the enactment phase starts: the operational processes are executed using the configured system. In the diagnosis phase, the operational processes are analyzed to identify problems and to find aspects that can be improved.

Different phases of the PAIS life-cycle call for different techniques and types of tools. For example, the focus of traditional WFM systems is on the lower half of the PAIS life-cycle. They are mainly aimed at supporting process configuration and execution and provide little support for the design and diagnosis phase.

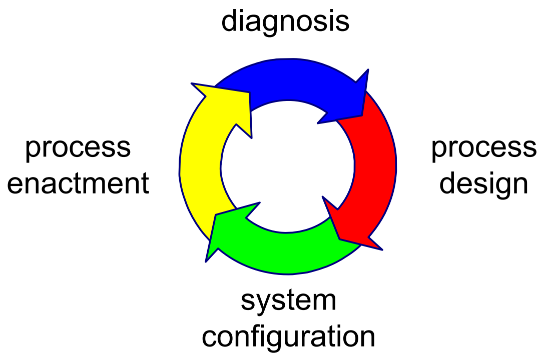


Fig. 1. The PAIS life-cycle [7]

Business process modeling tools are design-oriented and may use all kinds of analysis to evaluate designs. Besides classical analysis techniques such as simulation, more advanced techniques such as process mining come into play, i.e., process improvement by learning from running processes.

2.2 People Versus Software Applications

Another way of classifying PAISs is in terms of the nature of the participants (or resources) they involve, and in particular whether these participants are humans or software applications. In this respect, PAISs can be classified into human-oriented and system-oriented [28] or more precisely into *Person-to-Person* (P2P), *Person-to-Application* (P2A) and *Application-to-Application* (A2A) processes [24].

In P2P processes the participants involved are primarily people, i.e. the processes primarily involve tasks which require human intervention. Job tracking, project management, and groupware tools are designed to support P2P processes. Indeed, the processes supported by these tools usually do not involve entirely automated tasks carried out by applications. Also, the applications that participate in these processes (e.g. project tracking servers, e-mail clients, video-conferencing tools, etc.) are primarily oriented towards supporting computer-mediated interactions.

At the other end of the spectrum, A2A processes are those that only involve tasks performed by software systems. Such processes are typical in the area of distributed computing, and in particular distributed application integration. Transaction processing systems, EAI platforms, and Web-based integration servers are designed to support A2A processes.

P2A processes are those that involve both human tasks and interactions between people, and tasks and interactions involving applications which act without human intervention. Workflow systems fall in the P2A category since they primarily aim at making people and applications work in an integrated manner.

Note that the boundaries between P2P, P2A, and A2A are not crisp. Instead, there is a continuum of techniques and tools from P2P (i.e. manual, human-driven) to A2A (automated, application-driven).

2.3 Predictability of Processes

The degree of structure of the process to be automated (which is strongly linked to its predictability) is frequently used as a dimension to classify PAISs [28]. Structured processes are easier to support than unstructured processes. Moreover, it is also obvious that smaller processes are easier to support than larger ones. However, like in [13,24] we would like to elaborate on the predictability aspect. As Figure 2 shows, we distinguish between *unframed*, *ad hoc framed*, *loosely framed*, and *tightly framed* processes.

A process is said to be *unframed* if there is no explicit process model associated with it. This is the case for collaborative processes supported by groupware systems that do not offer the possibility of defining process models.

A process is said to be *ad hoc framed* if a process model is defined a priori but only executed once or a small number of times before being discarded or changed. This is the case in project management environments where a process model (i.e. a project chart) is often only executed once. It is also the case in grid computing environments, where a scientist may define a process model corresponding to a computation involving a number of datasets and computing resources, and then run this process only once.

A *loosely framed* process is one for which there is an a-priori defined process model and a set of constraints, such that the predefined model describes the "normal way of doing things" while allowing the actual executions of the process to deviate from this model within certain limits.

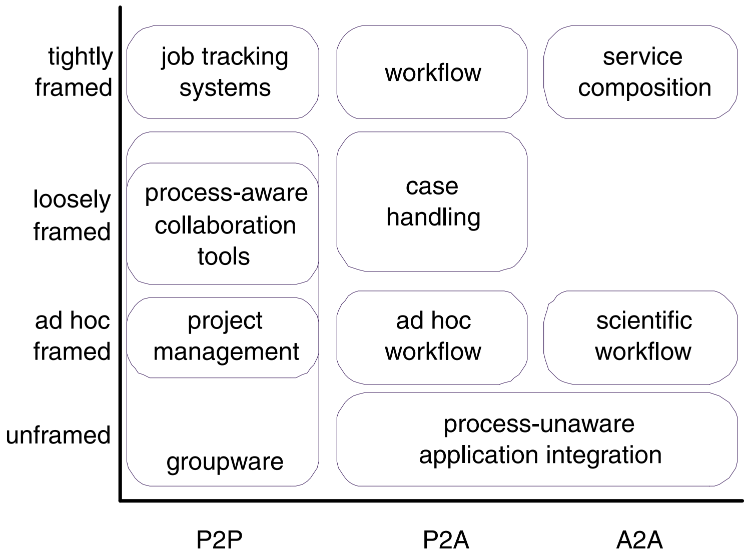


Fig. 2. Type of PAISs and associated development tools [24]

Finally, a *tightly framed* process is one which consistently follows an a-priori defined process model. This is the case of traditional workflow systems.

Figure 2 plots different types of PAISs and PAIS-related tools with respect the degree of framing of the underlying processes (unframed, ad hoc, loosely, or tightly framed), and the nature of the process participants (P2P, P2A, and A2A) [24].

As with P2P, P2A, and A2A processes, the boundaries between unframed, ad hoc framed, loosely framed, and tightly framed processes are not crisp. In particular, there is a continuum between loosely and tightly framed processes. For instance, during its operational life a process considered to be tightly framed can start deviating from its model so often and so unpredictably, that at some point in time it may be considered to have become loosely framed. Conversely, after a large number of cases of a loosely framed process have been executed, a common structure may become apparent, which may then be used to frame the process in a tighter way.

The topic of flexibility in PAISs attracted a lot of attention in the scientific community. Numerous researchers proposed ways of dealing with flexibility and change. Unfortunately, few of these ideas have been adopted by commercial parties. Moreover, it has become clear that there is no "one size fits all" solution, i.e., depending on the application, different types of flexibility are needed. In [60] a taxonomy is given where four types of flexibility are distinguished: (1) *flexibility by design*, (2) *flexibility by deviation*, (3) *flexibility by underspecification*, and (4) *flexibility by change* (both at type and instance levels). This taxonomy shows that different types of flexibility exist. Moreover, different paradigms may be used, i.e., even within one flexibility type there may be different mechanisms that realize different forms of flexibility [63]. All of these approaches aim to support ad hoc framed and/or loosely framed processes.

2.4 Intra-organizational Versus Inter-organizational

Initially, PAISs were mainly oriented towards intra-organizational settings. Focus was on the use of process support technologies (e.g. workflow systems) to automate operational processes involving people and applications inside an organization (or even within an organizational unit). Over the last few years, there has been a push towards processes that cross organizational barriers. Such inter-organizational processes can be one-to-one (i.e. bilateral relations), one-to-many (i.e. an organization interacting with several others) or many-to-many (i.e. a number of partners interacting with each other to achieve a common goal).

The trend towards inter-organizational PAISs is marked by the adoption of SOA and the emergence of web services standards such as BPEL et al.

3 Role of Models

In the previous section, we introduced PAISs and provided a classification. In this section, we focus more on the role of *models*. First of all, we elaborate on the different purposes of models (to provide insights, for analysis purposes, or for enactment). Second, we discuss differences between formal and informal models. Finally, we differentiate between man-made and derived models.

3.1 Purpose

Models can serve different purposes. In fact, the same model can be used for different objectives in the context of a PAIS.

Insight. When developing or improving a PAIS it is important that the different stakeholders get insight into the processes at hand and the way that these processes can or should be supported. Models can be used to discuss requirements, to support design decisions, and to validate assumptions. Moreover, the modeling process itself typically provides new and valuable insights because the modeler is triggered to make things explicit. It is interesting to use the metaphor of a construction drawing for a house. Only when people are confronted with concrete drawings they are able to generate requirements and make their wishes explicit. This holds for houses but also for other complex artifacts such as information systems.

Analysis. Using the metaphor of a construction drawing for a house, it is clear that models can be used to do analysis (e.g., calculating sizes, strengths, etc.). Depending on the type of model, particular types of analysis are possible or not. Moreover, in the context of a PAIS, analysis may focus on the business processes or on the information system itself. For example, the performance of a system (e.g., response times) is not the same as the performance of the processes it supports. Traditionally, most techniques used for the analysis of business processes originate from operations research. Students taking courses in operations management will learn to apply techniques such as simulation, queueing theory, and Markovian analysis. The focus mainly is on *performance analysis* and less attention is paid to the correctness of models. However, *verification* is needed to check whether the resulting system is free of logical errors. Many process designs suffer from deadlocks and livelocks that could have been detected using verification techniques. Notions such as soundness [1,30] can be used to verify the correctness of systems. Similar notions can be used to check interorganizational processes where deadlocks, etc. are more likely to occur [9,39,42].

Enactment. In the context of a PAIS, models are often used for enactment, i.e., based on a model of the process, the corresponding run-time support is generated. In a WFM system, a model of a process suffices to generate the corresponding system support. In other environments, the set of processes is often hard-coded. For example, although ERP systems like SAP have a workflow engine, most processes are hard-coded into the system and can only be changed by programming or changing configuration parameters. As a result, modifications are either time-consuming (because of substantial programming efforts) or restricted by the set of predefined configuration parameters.

3.2 Formality of Models

Related to the purpose of the model is the degree of formality.

Informal models. Informal models cannot be used for enactment and rigorous analysis. Their main purpose is to provide insight, support discussion, etc. We define a model to be informal if it is impossible to determine whether a particular scenario (i.e., a trace of activities) is possible of not.

Formal models. A formal model is able to tell whether a particular sequence of activities is possible or not. For example, given a Petri net it is possible to determine whether a trace corresponds to a possible firing sequence. Even declarative models may be formal. For example, given a model in Declare [47] or plain LTL or CTL [40], it is possible to check whether a trace is possible or not. Formal models typically allow for obtaining insights, analysis, and enactment. However, they may be more difficult to construct than informal models.

The boundaries between formal and informal models seem well-defined. However, in practice one can see many semi-formal models (e.g., BPMN, UML activity diagrams, EPCs, etc.). These models started out as informal models without any formal semantics. However, during the process, subsets have been formalized and are supported by tools that assume particular semantics. The problem is that some people interpret these models in a "formal way" while others use these notations in a rather loose manner. Consider for example the EPC models in SAP where at least 20 percent has serious flaws when one attempts to interpret them in a somewhat unambiguous manner [44]. Besides the differences in interpretation there is the problem that some of the informal concepts create conundrums. For example, the informal semantics of OR-join in EPCs and other languages creates the so-called "vicious cycle" paradox [2,34].

Figure 3 illustrates the relation between industry-driven languages, formal (science-driven) models, and analysis models. The industry-driven languages can be split into informal, semi-formal, and executable. Notations such as BPMN, EPCs, etc. can be seen as semi-formal, i.e., subsets can be interpreted in a precise manner. Languages like BPEL and many other workflow languages are executable because they are supported by concrete workflow engines. Note that these can be considered as formal although there may be different interpretations among different systems. However, in the context of a single execution engine, it is clear what traces are possible and what traces are not possible.

Languages like Petri nets and various process algebraic languages (CSP, CCS, ACP, π-calculus, etc.), are formal and mainly driven by the academic community. The focus is on a clear and unambiguous specification of the process and not on a particular analysis technique. However, such formal languages can often be mapped onto dedicated analysis models. For example, a Petri net can be mapped onto an incidence matrix to calculate invariants or onto a coverability graph to decide on reachability or boundedness.

Let us look at some examples bridging the three layers shown in Figure 3. Woflan is able to translate Staffware, COSA, Protos, and WebSphere models into

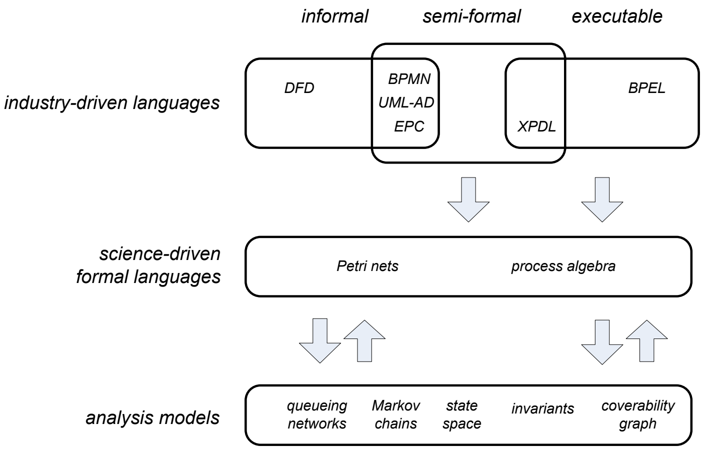


Fig. 3. Relationships among models

Petri nets and then analyze these using the coverability graph and incidence matrix [62]. The toolset BPEL2oWFN/Fiona/LoLA can be used to analyze BPEL models using open workflow nets as an intermediate format [39]. These examples show the possible interplay between various languages.

3.3 Construction Approach

Finally, we distinguish between man-made models and derived models.

Man-made Models. Traditionally, one thinks of models as man-made, i.e., some designer is constructing the model from scratch. When developing a new system or process, this is the only way to obtain models.

Derived Models. If there is already a process or system in place, it is also possible to "derive" models. There are basically two approaches. One approach is to try and reverse engineer models from the system itself, e.g., analyze the code or configuration parameters. Another approach is to extract models based on event logs, i.e., learn from example behavior observed in the past. The next section on process mining will elaborate on the latter type of derived models.

4 Process Mining

After an introduction to PAISs and discussing the various roles of models in the context of such systems, we now focus on a particular analysis technique: *process mining* [12,14,15,19,21,22,23,26,32,35,43,52,64,65]. The reason for elaborating on this particular analysis technique is that our experiences with process mining have dramatically changed our view on PAISs and the role of models in these systems. In fact, the goal of the paper is to *provide a critical reflection on the* *state-of-the-art based on experiences with process mining.* Therefore, we first provide a short introduction to process mining and then elaborate on the lessons learned.

Today's information systems are recording events in so-called event logs. The goal of process mining is to extract information on the process from these logs, i.e., process mining describes a family of *a-posteriori* analysis techniques exploiting the information recorded in the event logs. Typically, these approaches assume that it is possible to sequentially record events such that each event refers to an activity (i.e., a well-defined step in the process) and is related to a particular case (i.e., a process instance). Furthermore, some mining techniques use additional information such as the performer or originator of the event (i.e., the person/resource executing or initiating the activity), the timestamp of the event, or data elements recorded with the event (e.g., the size of an order).

Process mining addresses the problem that most organizations have very limited information about what is actually happening in their organization. In practice, there is often a significant gap between what is prescribed or supposed to happen, and what *actually* happens. Only a concise assessment of the organizational reality, which process mining strives to deliver, can help in verifying process models, and ultimately be used in a process redesign effort or PAIS implementation.

The idea of process mining is to discover, monitor and improve real processes (i.e., not assumed processes) by extracting knowledge from event logs. We consider three basic types of process mining (Figure 4).

Discovery. There is no a-priori model, i.e., based on an event log some model is constructed. For example, using the α-algorithm [15] a Petri net can be discovered based on low-level events. Many algorithms have been proposed to discover the control-flow [12,14,15,19,21,22,23,32,35,43,64,65] and few have been prosed to discover other aspects such as the social network [11].

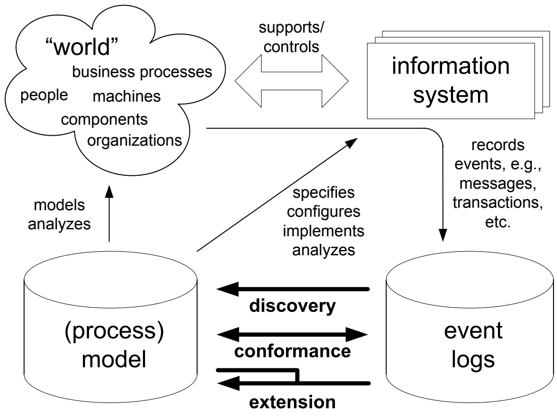


Fig. 4. Three types of process mining: (1) Discovery, (2) Conformance, and (3) Extension

Conformance. There is an a-priori model. This model is used to check if reality, as recorded in the log, conforms to the model and vice versa. For example, there may be a process model indicating that purchase orders of more than one million Euro require two checks. Another example is the checking of the four-eyes principle. Conformance checking may be used to detect deviations, to locate and explain these deviations, and to measure the severity of these deviations. For examples, we refer to the conformance checking algorithms described in [54].

Extension. There is an a-priori model. This model is extended with a new aspect or perspective, i.e., the goal is not to check conformance but to enrich the model. An example is the extension of a process model with performance data, i.e., some a-priori process model is used on which bottlenecks are projected. Another example is the decision mining algorithm described in [53] that extends a given process model with conditions for each decision.

Today, process mining tools are becoming available and are being integrated into larger systems. The ProM framework [3] provides an extensive set of analysis techniques which can be applied to real process enactments while covering the whole spectrum depicted in Figure 4. ARIS PPM was one of the first commercial tools to offer some support for process mining. Using ARIS PPM, one can extract performance information and social networks. Also some primitive form of process discovery is supported. However, ARIS PPM still requires some a-priori modeling. The BPM|suite of Pallas Athena was the first commercial tool to support process discovery without a-priori modeling. Although the above tools have been applied to real-life processes, it remains a challenge to extract suitable process models from event logs. This is illustrated by recent literature [12,14,15,19,21,22,23,32,35,43,64,65].

5 Lessons Learned

Now we would like to provide a critical reflection on the state-of-the-art in PAISs based on our experiences with process mining. The insights provided by process mining are very valuable for the development of the next generation PAISs because they clearly show a mismatch between the models proposed for driving these systems and reality. On the one hand, models tend to oversimplify things resulting in systems that are too restrictive. On the other hand, models fail to capture important aspects of business processes.

In the remainder we present some of the main lessons learned through our various process mining projects.

5.1 Models Do Not Reflect Reality

The first, and probably the most important, lesson is that models typically provide a very naive view of reality. Reality is typically much more dynamic and complex than what is captured in models. Models should abstract from details and aspects not relevant for the purpose of the model. However, the discrepancies

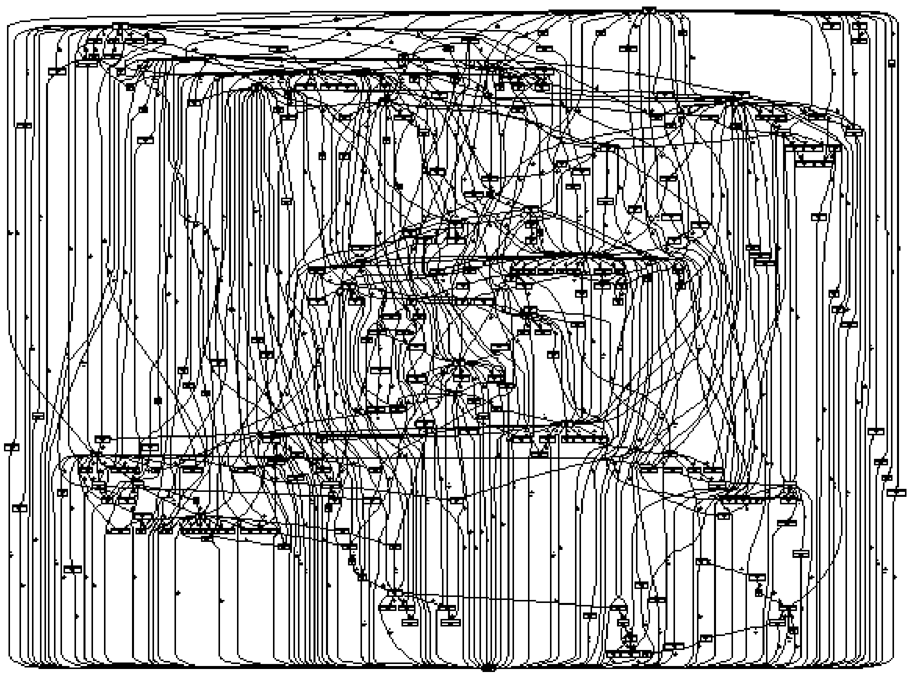


Fig. 5. Process discovered based on an event log with information about 2712 patients

that can be found between models and reality can typically not be justified by reasons of abstraction.

To illustrate this consider the process model shown in Figure 5. This model was discovered using ProM's Heuristics Miner [64] based on the data of 2712 patients treated in a Dutch hospital. The log contained 29258 events (i.e., +/- 10.8 events per case) corresponding to 264 activities. The discovered process model reflects the complexity of care processes. One may expect such "spaghetti-like processes" in a hospital. However, we have found similarly unstructured processes in many environments where one would expect more structured processes (e.g., municipalities, banks, insurance companies, etc.). It is important to note that the spaghetti-like process shown in Figure 5 is not due to limitations of the process mining techniques used, i.e., it is completely caused by the real complexity of the process.

Insights provided by process models such as the one shown in Figure 5 serve as a reality check for any PAIS implementation. Without a complete understanding of the processes at hand, the PAIS is destined to fail.

To illustrate the discrepancies between models and reality further, consider Figure 6 taken from [55]. These models have been obtained when analyzing one of the test processes of ASML (the leading manufacturer of wafer scanners in the world). ASML designs, develops, integrates and services advanced systems to produce semiconductors. In short, it makes the wafer scanners that print the chips. These wafer scanners are used to manufacture semi-conductors

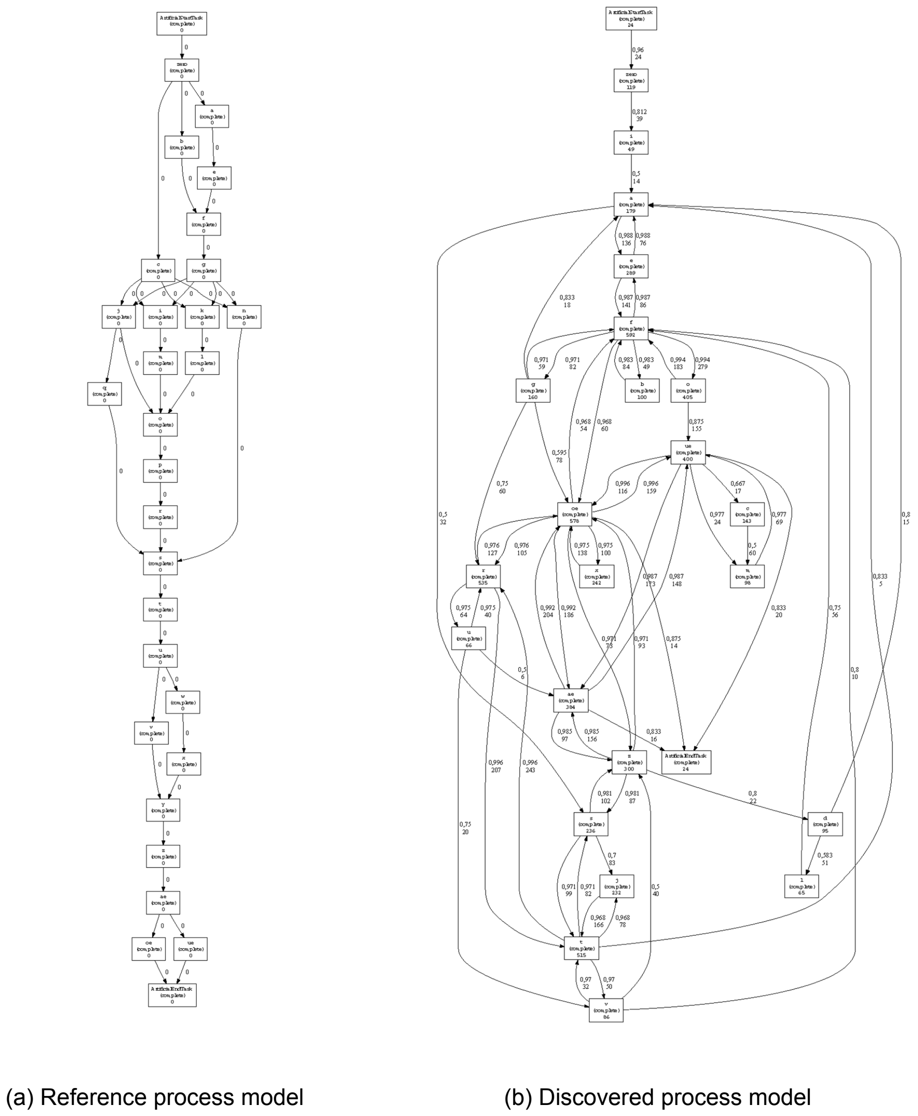
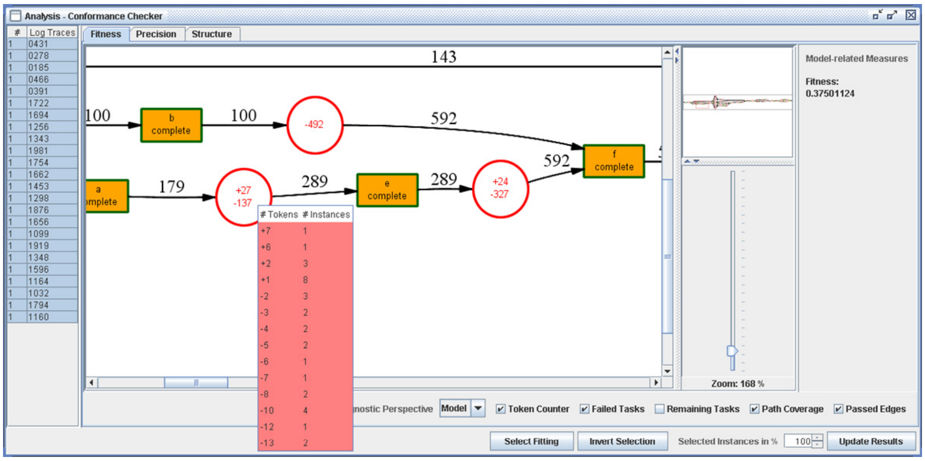


Fig. 6. Two heuristic nets [43,64] showing the difference between (a) the translated reference model for the test process on job-step level and (b) the discovered process model based on log which was mapped onto the job-step level [55]

(e.g., processors in devices ranging from mobile phones ad MP3 players to desktop computers). At any point in time, ASML's wafer scanners record events that can easily be distributed over the internet. Hence, any event that takes place during the test process is recorded. The availability of these event logs and the desire of ASML to improve the testing process, triggered the case study reported in [55]. If we apply ProM's discovery to the low-level logs, we obtain



Fitness:
0.37501124

#	Log Traces
1	0431
1	0278
1	0185
1	0466
1	0391
1	1722
1	1694
1	1256
1	1343
1	1981
1	1754
1	1662
1	1453
1	1298
1	1876
1	1656
1	1099
1	1919
1	1348
1	1596
1	1164
1	1032
1	1794
1	1160

# Tokens	# Instances
+7	1
+6	1
+2	3
+1	8
-2	3
-3	2
-4	2
-5	2
-6	1
-7	1
-8	2
-10	4
-12	1
-13	2

Fig. 7. Screenshot of ProM's Conformance Checker while analyzing the difference between the reference model and reality [55]

a spaghetti-like process similar to the one shown in Figure 5. However, using domain knowledge the low level log can be translated to an event log at the so-called job-step level. ASML also provided us with a reference model at the job-step level. This model was used to instruct the test engineers. Figure 6(a) shows the reference model. The discovered model is shown in Figure 6(b). It is interesting to note that the discovered model allows for much more scenarios than the reference model.

In Figure 7 we used ProM's Conformance Checker while analyzing the deviations in ASML's test process. As shown the fitness is only 37.5 percent, i.e., roughly one third of the events can be explained by the model indicating that "exceptions are the rule" [54]. By looking at the most frequent paths that appear in Figure 6(b) and not in Figure 6(a) and at the diagnostics provided in Figure 7 it is possible to pinpoint the most important deviations. Note that deviations are not necessarily a bad thing and may reflect (desirable) flexibility. We will elaborate on this in Section 5.3.

The results presented in this section are not exceptional, i.e., many processes turn out to be more spaghetti-like than expected. Nevertheless, most attention in both academia and industry is given to the analysis and use of models and not to the way to obtain them. Both sides take models as a starting point. Analysis techniques, process engines, etc. focus on what to do with models rather than obtaining faithful models. Therefore, we would like to stress the need for more emphasis on the faithfulness of models. For example, analysis results are only meaningful if the corresponding models are adequate.

5.2 A Human's Characteristics Are Difficult to Capture

In the previous section, we focused in discrepancies between the control-flow as modeled and the real control-flow. When it comes to resources similar problems emerge, especially if the resources are human. This mismatch becomes evident when comparing the behavior of humans observed when using process mining techniques and the behavior of humans assumed in simulation tools [10]. In the remainder, we focus on the problems encountered when modeling people for simulation purposes. However, the insights also apply to other analysis methods and enactment support (e.g., software for work distribution).

In practice there are few people that only perform activities for a single process. Often people are involved in many different processes, e.g., a manager, doctor, or specialist may perform tasks in a wide range of processes. However, simulation often focuses on a single process. Suppose a manager is involved in 10 different processes and spends about 20 percent of his time on the process that we want to analyze. In most simulation tools it is impossible to model that a resource is only available 20 percent of the time. Hence, one needs to assume that the manager is there all the time and has a very low utilization. As a result the simulation results are too optimistic. In the more advanced simulation tools, one can indicate that resources are there at certain times in the week (e.g., only on Monday). This is also an incorrect abstraction as the manager distributes his work over the various processes based on priorities and workload. Suppose that there are 5 managers all working 20 percent of their time on the process of interest. One could think that these 5 managers could be replaced by a single manager (5*20%=1*100%). However, from a simulation point of view this is an incorrect abstraction. There may be times that all 5 managers are available and there may be times that none of them are available.

Another problem is that people work at different speeds based on their workload, i.e., it is not just the distribution of attention over various processes, but also a person's absolute working speed influences his/her capacity for a particular process. There are various studies that suggest a relation between workload and performance of people. A well-known example is the so-called Yerkes-Dodson law [69]. The Yerkes-Dodson law models the relationship between arousal and performance as an inverse U-shaped curve. This implies that for a given individual and a given type of tasks, there exists an optimal arousal level. This is the level where the performance has its maximal value. Thus work pressure is productive, up to a certain point, beyond which performance collapses. Although this phenomenon can be easily observed in daily life, today's business process simulation tools do not support the modeling of workload dependent processing times.

As indicated earlier, people may be involved in different processes. Moreover, they may work part-time (e.g., only in the morning). In addition to their limited availabilities, people have a tendency to work in batches (cf. Resource Pattern 38: Piled Execution [58]). In any operational process, the same task typically needs to be executed for many different cases (process instances). Often people prefer to let work-items related to the same task accumulate, and then process all of these in one batch. In most simulation tools a resource is either available or not, i.e., it is assumed that a resource is eagerly waiting for work and immediately reacts to any work-item that arrives. Clearly, this does not do justice to the way people work in reality. For example, consider how and when people reply to e-mails. Some people handle e-mails one-by-one when they arrive while others process all of their e-mails at fixed times in batch.

Also related is the fact that calendars and shifts are typically ignored in simulation tools. While holidays, lunch breaks, etc. can heavily impact the performance of a process, they are typically not incorporated in the simulation model.

All these observations show that it is very difficult to adequately capture human activity in simulation models. As a result it is not uncommon that the simulation model predicts a flow time of hours while in reality the average flow time is weeks. In [10] the effects of some of these incorrect assumptions on the simulation results are shown. Using process mining one can get insight into the way that humans actually work and use this to build more faithful simulation models.

Note that the difficulties encountered when characterizing humans is not only relevant for simulation but also for enactment support. It can be observed that only few of the resource patterns [58] are supported by contemporary PAISs. Moreover, insights such as the Yerkes-Dodson law are not used by today's PAISs and systems are unable to predict problems. The lack of understanding and limited functionality impairs the successfulness of PAISs.

5.3 Spaghetti and Flexibility: Two Sides of the Same Coin

The topic of flexibility in the context WFM systems has been addressed by many authors [16,18,25,27,47,48,51,66]. See the taxonomy in [60] or the flexibility patterns in [63] to get an overview of the different approaches proposed in literature. See also [8,17,28,31,46,50,59] for other classifications of flexibility. Researchers proposed numerous ways of dealing with flexibility and change. Unfortunately, few of these ideas have been adopted by commercial parties. Process mining can expose the need for flexibility. Spaghetti-like processes as shown in Figure 5 and the quantification of non-conformance illustrated by Figure 7 illustrate the need for flexibility.

When building a PAIS for existing processes, process mining can be used to identify the flexibility needs. When looking at the spaghetti-like processes discovered using process mining, it becomes evident that one has to decide on what kinds of variability are actually desired. Some deviations are good because they correspond to adequate responses to requests from the environment. Other deviations may be undesirable because they impair quality or efficiency.

It is easy to say that PAISs should provide more flexibility. However, process mining also shows that it is very difficult to actually do this. It seems that much of the research in this domain is rather naive. For example, it is ridiculous to assume that end-users will be able construct or even understand process models such as the one depicted in Figure 5.

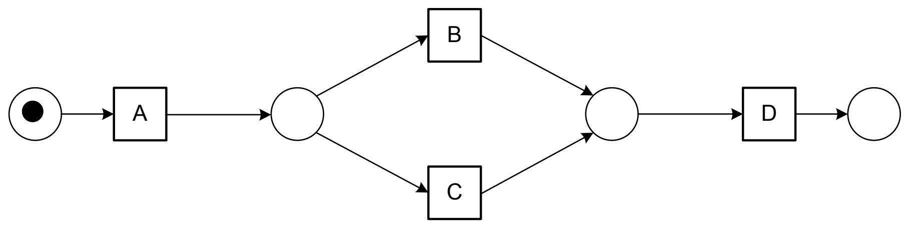
5.4 Process Models Should Be Treated as Maps

There are dozens of process modeling languages. Most of these languages provide some graphical notation with split and join nodes of particular types (AND, XOR, etc.). Although there are important semantical differences between these notations, the basic idea is always to draw a graph describing the routing of process instances (cases). This seems to be a good approach as it is adopted by most vendors and common practice in industry. Nevertheless, our experiences with process mining have revealed several weaknesses associated with this classical approach. Diagrams like Figure 5 show that automatically derived models are difficult to understand and lack expressiveness. This triggered us to look at process models as ordinary geographic maps. The "map metaphor" reveals some interesting insights.

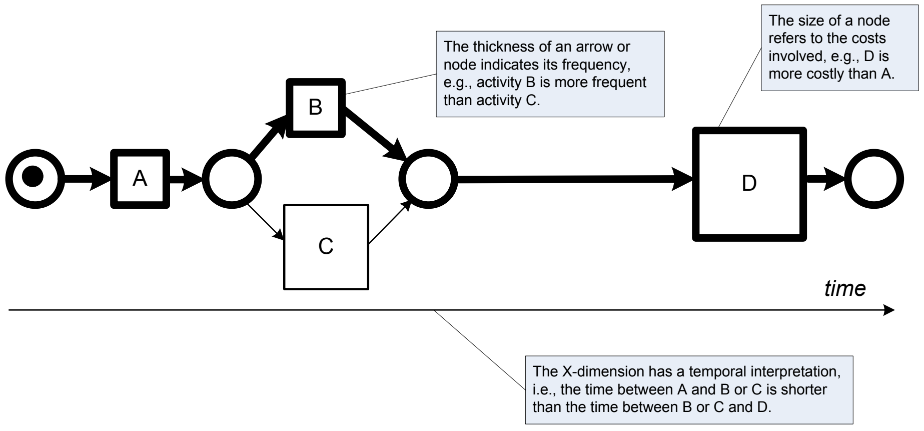
– There is no such thing as "the map". One may use city maps, highway maps, hiking maps, cycling maps, booting maps, etc. depending on the purpose for which it is intended to be used. All of these maps refer to the same reality. However, nobody would aim at trying to construct a single map that suits all purposes. Unfortunately, when it comes to processes one typically aims at a single map.
– Another insight is that process models do not exploit colors, dimensions, sizes, etc. It is remarkable that process models typically have shapes (nodes and arcs) of a fixed size, and, even if the size is variable, it has no semantical interpretation. Colors in process models are only used for beatification and not to express things like intensity, costs, etc. On a geographic map the X and Y dimension have a clear interpretation. These dimensions are not explicitly used when drawing process models.

To illustrate the above, consider Figure 8 showing two times the same Petri net. Although from a logical point of view the Petri nets are identical, the lower one also shows frequencies, costs, and time. For example, it is shown that the path (A, B, D) is much more frequent than the path (A, C, D). Moreover, activities C and D are more costly than A and B. The X-dimension is used to reflect time. The horizontal position corresponds to the average time at which activity takes places after the model is initiated by placing a token on the input place. It clearly shows that most time is spent waiting for the execution of D. Figure 8(b) is still very primitive compared to the drawing of maps. Maps typically also use colors and other intuitive annotations to indicate relevant information. Moreover, maps abstract and aggregate. Abstraction is used to leave out things that are less significant (i.e., dirt roads and small townships). Aggregation is used to take things together. For example, the roads of a city are taken together into a single shape. In terms of Figure 8, abstraction could mean that C is removed because it is too insignificant. Aggregation should be used to group A, B, and C into a single node because they typically occur together in a short time distance.

Today, electronic maps overcome some of the limitations of paper maps. As indicated above one may use different maps (city maps, highway maps, etc.) depending on the purpose. When using Google Maps or a car navigation system like TomTom it is possible to dynamically zoom-in, zoom-out, change focus, or change the type of information. Moreover, these electronic maps are increasingly used to project dynamic information on. For example TomTom is able to show



(a) Ordinary Petri net just showing the control-flow logic.


The thickness of an arrow or node indicates its frequency, e.g., activity B is more frequent than activity C.
The size of a node refers to the costs involved, e.g., D is more costly than A.
A
B
C
D
time
The X-dimension has a temporal interpretation, i.e., the time between A and B or C is shorter than the time between B or C and D.

(b) Petri net also showing other dimensions (frequency, time, and costs).

Fig. 8. Using the map metaphor for drawing process models

traffic jams, fuel stations with the lowest prices, weather information, etc. These ideas can also be used for process models.

– Process models should allow for different views, i.e., it should be possible to zoom-in and zoom-out seamlessly. The static decompositions used by contemporary drawing tools force the user to view processes in a fixed manner. Moreover, decompositions typically address the needs of a technical designer rather than an end-user. Hence the challenge is to be able to support easy navigation and seamlessly zooming-in/out when viewing process models.
– It should be possible to project dynamic information on top of process models. For example, it should be possible to view current process instances projected on the process model and to animate history by replaying past events on the same process model. This is similar to showing real-time traffic information by a car navigation system like TomTom.

The limitations related to the representation and visualization of process models mentioned above became evident based on experiences gathered in many process mining projects. It seems that the "map metaphor" can be used to present process models and process information in completely new ways [29,37]. Few

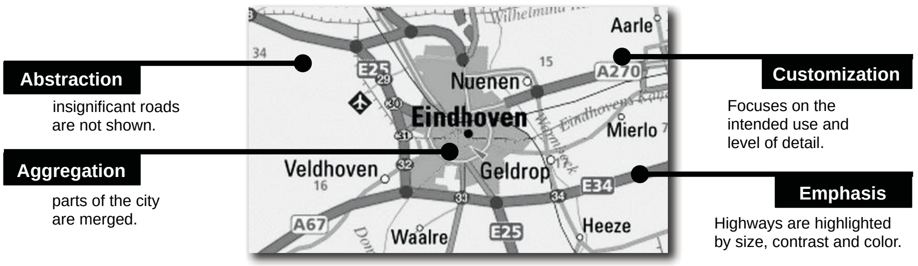

Abstraction
insignificant roads are not shown.
Aggregation
parts of the city are merged.
Customization
Focuses on the intended use and level of detail.
Emphasis
Highways are highlighted by size, contrast and color.
Aarle
Nuenen
Eindhoven
Mierlo
Veldhoven
Geldrop
Waalre
Heeze
A270
A67
E25
E34

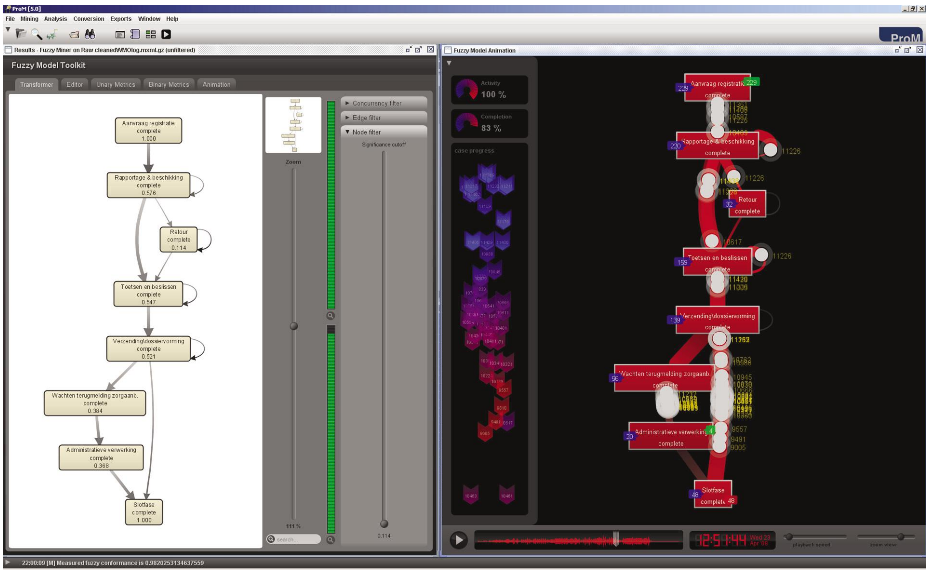
Fig. 9. ProM's Fuzzy Miner implements some of the ideas learned from (electronic) maps [29]

researchers have been investigating such ideas. Here we would like to point out two ideas we have been working on. In the context of YAWL [5,37,61,72], we showed that it is possible to show current work items on top of various maps. Work items can be shown on top of a geographic map, a process model, a time chart, an organizational model, etc. In the context of ProM, we have used the "map metaphor" to enhance the so-called Fuzzy Miner [29]. As presented in [29], four ideas are being combined in ProM's Fuzzy Miner to draw maps of process models.

- *Aggregation:* To limit the number of information items displayed, maps often show coherent clusters of low-level detail information in an aggregated manner. One example are cities in road maps, where particular houses and streets are combined within the citys transitive closure (e.g., the city of Eindhoven in Figure 9).
- *Abstraction:* Lower-level information which is insignificant in the chosen context is simply omitted from the visualization. Examples are bicycle paths, which are of no interest in a motorists map.
- *Emphasis:* More significant information is highlighted by visual means such as color, contrast, saturation, and size. For example, maps emphasize more important roads by displaying them as thicker, more colorful and contrasting lines (e.g., motorway "E25" in Figure 9).
- *Customization:* There is no one single map for the world. Maps are specialized on a defined local context, have a specific level of detail (city maps vs highway maps), and a dedicated purpose (interregional travel vs alpine hiking).

Figure 10 shows screenshot of ProM's Fuzzy Miner [29]. The left screen shows a discovered model. Note that the thickness of each arc is determined by the number of times this path is taken (i.e., frequency). Moreover, some nodes and arcs have been left out because they are insignificant. The left screen shows an animation based on historic information. This animation shows the actual execution of cases on top of the discovered model.

It is obvious that the ideas presented here are not limited to process mining. When developing, analyzing, or controlling a PAIS, such visualizations can be very useful.


ProM [5.0]
File Mining Analysis Conversion Exports Window Help
Results - Fuzzy Miner on Raw cleanedWMOlog.mxml.gz (unfiltered)
Fuzzy Model Toolkit
Transformer Editor Unary Metrics Binary Metrics Animation
Aanvraag registratie complete 1.000
Rapportage & beschikking complete 0.576
Retour complete 0.114
Toetsen en beslissen complete 0.547
Verzending\dossiervorming complete 0.521
Wachten terugmelding zorgaanb. complete 0.384
Administratieve verwerking complete 0.368
Slotfase complete 1.000
Zoom
111 %
search...
Concurrency filter
Edge filter
Node filter
Significance cutoff
0.114
ProM
Fuzzy Model Animation
Activity 100 %
Completion 83 %
case progress
playback speed
zoom view

22:00:09 [M] Measured fuzzy conformance is 0.9820253134637559

Fig. 10. ProM's Fuzzy Miner (left) and the corresponding animation facility (right)

5.5 Analysis Techniques Do Not Use the Information Available

The last lesson to be learned is related to the limited use of existing artifacts. People tend to model things from scratch and do not use information that is already recorded in information systems. In practice, it is time consuming to construct a good process model. For example, when constructing a simulation model one not only has to construct a model but also determine the input parameters. A pitfall of current approaches is that existing artifacts (models, logs, data, etc.) are not used in a direct and systematic manner. If a PAIS is used, there are often models that are used to configure the system (e.g., workflow schemas). Today, these models are typically disconnected from the simulation models and created separately. Sometimes a business process modeling tool is used to make an initial process design. This design can be used for simulation purposes when using a tool like Protos or ARIS. When the designed process is implemented, another system is used and the connection between the implementation model and the design model is lost. It may be that at a later stage, when the process needs to be analyzed, a simulation model is built from scratch. This is a pity as the PAIS contains most of the information required. As a result the process is "reinvented" again and again, thus introducing errors and unnecessary work. The lack of reuse also applies to other sources of information. For example, the PAIS may provide detailed event logs. Therefore, there is no need to "invent" processing times, arrival times, and routing probabilities, etc. All of this information can be extracted from the logs. Note that a wealth of information can be derived from event logs. In fact, in [56] it is demonstrated that complete simulation models can be extracted from event logs.

Contemporary simulation tools tend to support experiments that start in an arbitrary initial state (without any cases in the pipeline) and then simulate the process for a long period to make statements about the steady-state behavior. However, this steady-state behavior does not exist (the environment of the process changes continuously) and is thus considered irrelevant by the manager. Moreover, the really interesting questions are related to the near future. Therefore, it seems vital to also support transient analysis, often referred to as *short-term simulation* [49,57,71]. The "fast-forward button" provided by short-term simulation is a useful option, however, it requires the use of the current state. Fortunately, when using a PAIS it is relatively easy to obtain the current state and load this into the simulation model.

The above not only applies to simulation models. Also other types of analysis can benefit from the information stored in and recorded by the PAIS [57].

6 Conclusion

Workflow management systems, case-handling systems, enterprise information systems, etc. are all examples of PAISs. We introduced these systems by characterizing them in several ways. Moreover, we elaborated on the role of process models in the context of such systems. After this introduction, we focused on lessons learned from process mining. The goal of process mining is to extract information from event logs. These event logs can be used to automatically generate models (process discovery) or to compare models with reality (conformance checking).

Extensive experience gathered through various process mining projects, has revealed important lessons for the development and use of PAISs. The first lesson is that models typically provide a very naive view of reality. The second lesson is that it is far from trivial to adequately capture the characteristics of human actors. The third lesson is that the true need for flexibility can be seen by analyzing spaghetti-like process models. The fourth lesson is that the way we view processes can be improved dramatically by using the "map metaphor". The fifth lesson is that many artifacts (models and logs) remain unused by today's analysis approaches.

Although these lessons were triggered by the application of process mining to many real-life logs, they are useful for the whole PAIS life-cycle. It does not make any sense to talk about analysis or enactment, without a good and deep understanding of the processes at hand.

Acknowledgements

Parts of the work reported in this paper are the result of joint work with many other researchers from TU/e and QUT. Section 2 is based on joint work with Marlon Dumas and Arthur ter Hofstede [24]. The experiences reported in Section 5 were only possible because the maturity of our process mining tool ProM. Therefore, the author would like to thank the many people involved in the development of ProM.

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., Desel, J., Kindler, E.: On the Semantics of EPCs: A Vicious Circle. In: Nüttgens, M., Rump, F.J. (eds.) Proceedings of the EPK 2002: Business Process Management using EPCs, Trier, Germany, November 2002, pp. 71–80. Gesellschaft für Informatik, Bonn (2002)
3. van der Aalst, W.M.P., van Dongen, B.F., Günther, C.W., Mans, R.S., de Medeiros, A.K.A., Rozinat, A., Rubin, V., Song, M., Verbeek, H.M.W., Weijters, A.J.M.M.: 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)
4. van der Aalst, W.M.P., van Hee, K.M.: Workflow Management: Models, Methods, and Systems. MIT Press, Cambridge (2004)
5. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: Yet Another Workflow Language. Information Systems 30(4), 245–275 (2005)
6. 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)
7. 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)
8. van der Aalst, W.M.P., Jablonski, S.: Dealing with Workflow Change: Identification of Issues and Solutions. International Journal of Computer Systems, Science, and Engineering 15(5), 267–276 (2000)
9. van der Aalst, W.M.P., Lohmann, N., Massuthe, P., Stahl, C., Wolf, K.: From Public Views to Private Views: Correctness-by-Design for Services. In: Dumas, M., Heckel, R. (eds.) WS-FM 2007. LNCS, vol. 4937, pp. 139–153. Springer, Heidelberg (2008)
10. van der Aalst, W.M.P., Nakatumba, J., Rozinat, A., Russell, N.: Business Process Simulation: How to get it right? In: vom Brocke, J., Rosemann, M. (eds.) International Handbook on Business Process Management. Springer, Berlin (2008)
11. van der Aalst, W.M.P., Reijers, H.A., Song, M.: Discovering Social Networks from Event Logs. Computer Supported Cooperative work 14(6), 549–593 (2005)
12. van der Aalst, W.M.P., Reijers, H.A., Weijters, A.J.M.M., van Dongen, B.F., Alves de Medeiros, A.K., Song, M., Verbeek, H.M.W.: Business Process Mining: An Industrial Application. Information Systems 32(5), 713–732 (2007)
13. van der Aalst, W.M.P., Stoffele, M., Wamelink, J.W.F.: Case Handling in Construction. Automation in Construction 12(3), 303–320 (2003)
14. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow Mining: A Survey of Issues and Approaches. Data and Knowledge Engineering 47(2), 237–267 (2003)
15. 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)
16. van der Aalst, W.M.P., Weske, M., Grünbauer, D.: Case Handling: A New Paradigm for Business Process Support. Data and Knowledge Engineering 53(2), 129–162 (2005)
17. Adams, M.: Facilitating Dynamic Flexibility and Exception Handling for Workflows. Phd thesis, Queensland University of Technology (2007)

18. Adams, M., ter Hofstede, A.H.M., van der Aalst, W.M.P., Edmond, D.: Dynamic, Extensible and Context-Aware Exception Handling for Workflows. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 95–112. Springer, Heidelberg (2007)
19. Agrawal, R., Gunopulos, D., Leymann, F.: Mining Process Models from Workflow Logs. In: Schek, H.-J., Saltor, F., Ramos, I., Alonso, G. (eds.) EDBT 1998. LNCS, vol. 1377, pp. 469–483. Springer, Heidelberg (1998)
20. Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., Guízar, A., Kartha, N., Liu, C.K., Khalaf, R., Koenig, D., Marin, M., Mehta, V., Thatte, S., Rijn, D., Yendluri, P., Yiu, A.: Web Services Business Process Execution Language Version 2.0 (OASIS Standard). WS-BPEL TC OASIS (2007), `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html`
21. 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)
22. Datta, A.: Automating the Discovery of As-Is Business Process Models: Probabilistic and Algorithmic Approaches. Information Systems Research 9(3), 275–301 (1998)
23. van Dongen, B.F., van der Aalst, W.M.P.: Multi-Phase Process Mining: Building Instance Graphs. In: Atzeni, P., Chu, W., Lu, H., Zhou, S., Ling, T.-W. (eds.) ER 2004. LNCS, vol. 3288, pp. 362–376. Springer, Heidelberg (2004)
24. Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Process-Aware Information Systems: Bridging People and Software through Process Technology. Wiley & Sons, Chichester (2005)
25. Dustdar, S.: Caramba - A Process-Aware Collaboration System Supporting Ad Hoc and Collaborative Processes in Virtual Teams. Distributed and Parallel Databases 15(1), 45–66 (2004)
26. Dustdar, S., Gombotz, R.: Discovering Web Service Workflows Using Web Services Interaction Mining. International Journal of Business Process Integration and Management 1(4), 256–266 (2006)
27. Ellis, C.A., Keddara, K., Rozenberg, G.: Dynamic Change within Workflow Systems. In: Comstock, N., Ellis, C., Kling, R., Mylopoulos, J., Kaplan, S. (eds.) Proceedings of the Conference on Organizational Computing Systems, ACM SIGOIS, Milpitas, California, August 1995, pp. 10–21. ACM Press, New York (1995)
28. Georgakopoulos, D., Hornick, M., Sheth, A.: An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. Distributed and Parallel Databases 3, 119–153 (1995)
29. Günther, C.W., van der Aalst, W.M.P.: Fuzzy Mining: Adaptive Process Simplification Based on Multi-perspective Metrics. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 328–343. Springer, Heidelberg (2007)
30. van Hee, K.M., Sidorova, N., Voorhoeve, M.: Generalised Soundness of Workflow Nets Is Decidable. In: Cortadella, J., Reisig, W. (eds.) ICATPN 2004. LNCS, vol. 3099, pp. 197–215. Springer, Heidelberg (2004)
31. Heinl, P., Horn, S., Jablonski, S., Neeb, J., Stein, K., Teschke, M.: A Comprehensive Approach to Flexibility in Workflow Management Systems. In: Georgakopoulos, G., Prinz, W., Wolf, A.L. (eds.) Work Activities Coordination and Collaboration (WACC 1999), pp. 79–88. ACM press, San Francisco (1999)
32. Herbst, J.: A Machine Learning Approach to Workflow Management. In: Lopez de Mantaras, R., Plaza, E. (eds.) ECML 2000. LNCS, vol. 1810, pp. 183–194. Springer, Heidelberg (2000)

33. Jablonski, S., Bussler, C.: Workflow Management: Modeling Concepts, Architecture, and Implementation. International Thomson Computer Press, London (1996)
34. Kindler, E.: On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. Data and Knowledge Engineering 56(1), 23–40 (2006)
35. Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Inducing Declarative Logic-Based Models from Labeled Traces. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 344–359. Springer, Heidelberg (2007)
36. Lawrence, P. (ed.): Workflow Handbook 1997, Workflow Management Coalition. John Wiley and Sons, New York (1997)
37. de Leoni, M., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Visual Support for Work Assignment in Process-Aware Information Systems. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 67–83. Springer, Heidelberg (2008)
38. Leymann, F., Roller, D.: Production Workflow: Concepts and Techniques. Prentice-Hall PTR, Upper Saddle River (1999)
39. 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)
40. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer, New York (1991)
41. Marinescu, D.C.: Internet-Based Workflow Management: Towards a Semantic Web. Wiley Series on Parallel and Distributed Computing, vol. 40. Wiley-Interscience, New York (2002)
42. Massuthe, P., Reisig, W., Schmidt, K.: An Operating Guideline Approach to the SOA. Annals of Mathematics, Computing & Teleinformatics 1(3), 35–43 (2005)
43. de Medeiros, A.K.A., Weijters, A.J.M.M., van der Aalst, W.M.P.: Genetic Process Mining: An Experimental Evaluation. Data Mining and Knowledge Discovery 14(2), 245–304 (2007)
44. Mendling, J., Neumann, G., van der Aalst, W.M.P.: Understanding the Occurrence of Errors in Process Models Based on Metrics. In: Curbera, F., Leymann, F., Weske, M. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 113–130. Springer, Heidelberg (2007)
45. zur Muehlen, M.: Workflow-based Process Controlling: Foundation, Design and Application of workflow-driven Process Information Systems, Logos, Berlin (2004)
46. Pesic, M.: Constraint-based Workflow Management Systems: Shifting Control to Users. Phd thesis, Eindhoven University of Technology (May 2008)
47. Pesic, M., Schonenberg, M.H., Sidorova, N., van der Aalst, W.M.P.: Constraint-Based Workflow Models: Change Made Easy. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 77–94. Springer, Heidelberg (2007)
48. Reichert, M., Dadam, P.: ADEPTflex: Supporting Dynamic Changes of Workflow without Loosing Control. Journal of Intelligent Information Systems 10(2), 93–129 (1998)
49. Reijers, H.A., van der Aalst, W.M.P.: Short-Term Simulation: Bridging the Gap between Operational Control and Strategic Decision Making. In: Hamza, M.H. (ed.) Proceedings of the IASTED International Conference on Modelling and Simulation, pp. 417–421. IASTED/Acta Press, Anaheim (1999)
50. Rinderle, S., Reichert, M., Dadam, P.: Evaluation of Correctness Criteria for Dynamic Workflow Changes. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 41–57. Springer, Heidelberg (2003)

51. Rinderle, S., Reichert, M., Dadam, P.: Correctness Criteria For Dynamic Changes in Workflow Systems: A Survey. Data and Knowledge Engineering 50(1), 9–34 (2004)
52. Rozinat, A., van der Aalst, W.M.P.: Conformance Testing: Measuring the Fit and Appropriateness of Event Logs and Process Models. In: Bussler, C.J., Haller, A. (eds.) BPM 2005. LNCS, vol. 3812, pp. 163–176. Springer, Heidelberg (2006)
53. Rozinat, A., van der Aalst, W.M.P.: Decision Mining in ProM. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 420–425. Springer, Heidelberg (2006)
54. Rozinat, A., van der Aalst, W.M.P.: Conformance Checking of Processes Based on Monitoring Real Behavior. Information Systems 33(1), 64–95 (2008)
55. Rozinat, A., de Jong, I.S.M., Günther, C.W., van der Aalst, W.M.P.: Process Mining of Test Processes: A Case Study. In: BETA Working Paper Series, WP 220. Eindhoven University of Technology, Eindhoven (2007)
56. Rozinat, A., Mans, R.S., Song, M., van der Aalst, W.M.P.: Discovering Colored Petri Nets From Event Logs. International Journal on Software Tools for Technology Transfer 10(1), 57–74 (2008)
57. Rozinat, A., Wynn, M.T., van der Aalst, W.M.P., ter Hofstede, A.H.M., Fidge, C.: Workflow Simulation for Operational Decision Support Using Design, Historic and State Information. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 196–211. Springer, Heidelberg (2008)
58. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M., Edmond, D.: Workflow Resource Patterns: Identification, Representation and Tool Support. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 216–232. Springer, Heidelberg (2005)
59. 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)
60. Schonenberg, H., Mans, R., Russell, N., Mulyar, N., van der Aalst, W.M.P.: Process Flexibility: A Survey of Contemporary Approaches. In: Dietz, J., Albani, A., Barjis, J. (eds.) Advances in Enterprise Engineering I. LNBIP, vol. 10, pp. 16–30. Springer, Heidelberg (2008)
61. Streit, A., Pham, B., Brown, R.: Visualisation Support for Managing Large Business Process Specifications. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 205–219. Springer, Heidelberg (2005)
62. Verbeek, H.M.W., Basten, T., van der Aalst, W.M.P.: Diagnosing Workflow Processes using Woflan. The Computer Journal 44(4), 246–279 (2001)
63. Weber, B., Reichert, M., Rinderle-Ma, S.: Change Patterns and Change Support Features: Enhancing Flexibility in Process-Aware Information Systems. Data and Knowledge Engineering 66(3), 438–466 (2008)
64. Weijters, A.J.M.M., van der Aalst, W.M.P.: Rediscovering Workflow Models from Event-Based Data using Little Thumb. Integrated Computer-Aided Engineering 10(2), 151–162 (2003)
65. Wen, L., van der Aalst, W.M.P., Wang, J., Sun, J.: Mining process models with non-free-choice constructs. Data Mining and Knowledge Discovery 15(2), 145–180 (2007)
66. Weske, M.: Formal Foundation and Conceptual Design of Dynamic Adaptations in a Workflow Management System. In: Sprague, R. (ed.) Proceedings of the Thirty-Fourth Annual Hawaii International Conference on System Science (HICSS-34). IEEE Computer Society Press, Los Alamitos (2001)

67. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer, Berlin (2007)
68. White, S.A., et al.: Business Process Modeling Notation Specification (Version 1.0, OMG Final Adopted Specification) (2006)
69. Wickens, C.D.: Engineering Psychology and Human Performance. Harper (1992)
70. Workflow Patterns Home Page, `http://www.workflowpatterns.com`
71. Wynn, M.T., Dumas, M., Fidge, C.J., ter Hofstede, A.H.M., van der Aalst, W.M.P.: Business Process Simulation for Operational Decision Support. In: ter Hofstede, A.H.M., Benatallah, B., Paik, H.-Y. (eds.) BPM Workshops 2007. LNCS, vol. 4928, pp. 66–77. Springer, Heidelberg (2008)
72. YAWL Home Page, `http://www.citi.qut.edu.au/yawl/`

Model-Based Software Engineering and Process-Aware Information Systems

Ekkart Kindler

Technical University of Denmark, DTU Informatics
Richard Petersens Plads, DK-2800 Kgs. Lyngby, Denmark
eki@imm.dtu.dk

Abstract. Today, there are many graphical formalisms for modelling software—among others the *Unified Modeling Language* (*UML*). And there are different technologies that automatically generate code from such models. We call these as well as any approach that takes models more seriously than just a sketch or an illustration *Model-based Software Engineering* (*MBSE*). Most of today's code generation approaches, however, focus on standard behaviour; application specific behaviour often still needs to be programmed manually. The ultimate goal of MBSE, however, is to generate all code automatically.

In the field of *Process-aware Information System* (*PAIS*) and workflow management, models have been in the focus right from the beginning. What is more, the models were used to define and enact application specific behaviour or business logic by providing process models. This way, they are one of the successful precursors of Model-based Software Engineering.

In this paper, we will give an overview of the concepts and technologies in MBSE, and its main ideas, principles, and concepts. We will point out some differences to PAIS, but also some common ground, and we show how both fields could benefit from each other. This way, we want to start bridging the gap between MBSE and PAIS .

Keywords: Business Process Modelling, Model-based Software Engineering (MBSE), Model-driven Architecture (MDA), Process-aware Information Systems (PAIS), Process models.

1 Introduction

In the 90ties, *Workflow Management Systems* (*WfMS*) [16,4,14,35,22] promoted the idea that a system supporting the *business processes* of an enterprise or administration could be realized based on *process models*. These process models could be executed or, in the terminology of workflow management, *enacted* by a *workflow engine*. This helped decreasing the development costs and increasing the flexibility of the systems. What is more, the process models are much closer to the world of the users and the application domain than modelling notations and programming languages from classical software engineering. Today, many of the ideas from workflow-management live on—just under the more modern names *web services* and *service oriented architecture* (*SOA*).

Here, we cannot give a detailed evaluation of the success and the impacts of workflow management systems. For now, it should suffice that workflow management systems have shown that it is possible to build information systems by making use of models without any programming. Of course, this works only for a specific kind of information systems, which today are called *Process-aware Information Systems*[1] (*PAIS*) [7].

About ten years later, the *Object Management Group*[2] (*OMG*) promoted an idea that takes the next step: the *Model-driven Architecture* (*MDA*) [25,21]. The MDA suggests to take models much more seriously during the software development process and, ultimately, to get rid of any kind of programming when making software. In particular, the MDA is meant for any kind of software and not restricted to PAIS. Note that the MDA leaves some freedom on how to achieve these goals and visions. Fowler [9] gives a nice overview on the different directions, which he calls "camps". Still, the MDA is tightly related to other OMG standards and technologies such as the *Unified Modeling Language* (*UML*) [30,29], the *Meta-object Facility* (*MOF*) [27], the *XML Metadata Interchange* (*XMI*) [26], and the *Query/View/Transformation* (*QVT*) [28]. Moreover, MDA focuses on two specific kinds of models, which are called *Platform Independent Model* (*PIM*) and *Platform Specific Model* (*PSM*), and on the transformations between these models and to the final code. These kinds of models are, however, relatively technical already. The models that are used in workflow modelling are much closer to the domain and would be called *Computation Independent Models* (CIM) in the MDA. But, MDA is not very explicit on how it deals with CIMs; in fact, many publications on MDA do not even mention CIMs at all.

We believe that MDA had and will have a great impact on the way software will be developed: focusing on modelling rather than on programming. And it shares or carries on some visions of workflow management, and promotes them for systems that are not specifically process-aware. But due to the use of more technical models, the use of transformations for obtaining the final software, and the focus on related OMG technologies, MDA does not subsume technologies from workflow management and other modelling approaches. Therefore, we define a bit broader term here: *Model-based Software Engineering* (*MBSE*)[3]. We call any kind of software or system development in which models are taken more seriously than a nice sketch, illustration, or drawing MBSE. This can either be

[1] Actually, PAIS has two slightly different meanings: as a characteristics of information systems to be developed and as a technology for developing these kinds of systems. In order to keep these two meanings apart, we use PAIS for the kind of information systems only; we introduce the term *Process-centric Software Engineering* (*PCSE*) for the technology.

[2] See: `http://www.omg.org/`

[3] Sometimes, the term *Model-driven Development* (*MDD*) is used for MDA-like approaches that are not focused on the OMG technologies; but in other publications MDD is used as synonym for MDA; and, as MDA, MDD is an OMG trademark. Therefore, we use the more neutral term MBSE here.

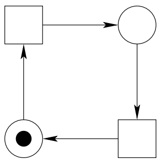


Fig. 1. A simple Petri net

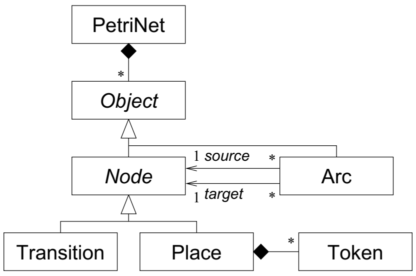


Fig. 2. A meta model for Petri nets

more or less automatic code generation, some kind of analysis or verification, or the execution of the models. In this sense, workflow management is Model-based Software Engineering.

In this paper, we will discuss how ideas of MBSE can be used for the development of PAIS, but also how concepts and lessons learned from workflow management systems—along with other concepts from software engineering—can help making the dream of developing software without any programming come true. To this end, we need to understand why this idea worked for PAIS already more than ten years ago, but why there are still some deficiencies in general.

2 Model-Based Software Engineering

In this section, we present some more details on MBSE, its ideas, concepts, visions, and its limitations. We do this by discussing a simple example project: a Petri net editor, which is realized in the *Eclipse Modeling Framework* (*EMF*) [5] and the *Graphical Modeling Framework*[4] (*GMF*). Without going into all details, this example illustrates, how software can be generated fully automatically from models: a class diagram capturing the domain of Petri nets and a model representing their graphical appearance.

2.1 Example: A Petri Net Editor

We start with explaining the syntax of a simple version of Petri nets: Figure 1 shows a very simple example. The Petri net consists of two types of *nodes*, which are called *places* and *transitions*, and *arcs* between them. The places are graphically represented as circles, the transition are graphically represented as squares, and the arcs are graphically represented as arrows pointing from one node to another. Moreover, there can be any number of *tokens* on a place, which are shown as black dots.

A Meta Model for Petri Nets. Above, we have explained the concepts of Petri nets in natural language. Figure 2 shows how the concepts of Petri nets can

[4] See `http://www.eclipse.org/modeling/gmf/`.

be expressed as a UML class diagram, as a so-called *domain model*. The *classes* (represented as boxes) define the different concepts of Petri nets, the *associations* (represented as arrows between the classes) represent the relationships between these concepts. For example, the two associations between the class Arc and Node say that each arc has exactly one source and exactly one target node. Moreover, there is an *inheritance* between class Node and the classes Transition and Place, which is graphically represented by a line starting with a triangle. This inheritance says that transitions and places are a special kind of node. Likewise, the class Object and the inheritance relation to Node and Arc say, that nodes (and with it places and transitions) and arcs are objects of a Petri net. A PetriNet consist of any number of such objects. This is represented by a special kind of association, which is called *composition*; a composition is graphically represented by a line that starts with a black diamond. The other composition in that diagram says that a place can contain any number of tokens.

This use of class diagrams is standard in software engineering. Since we do not assume that all readers are software engineers, we explained the diagram and its notations anyway. Note that we call the diagram of Fig. 2 a *meta model*. The reason is that it is a model of what Petri nets are. Since a Petri net itself is a model already, this is a "model of a model", which is called a meta model.

Actually, this meta model is a slightly simplified version of the meta model that is used in the definition of the *Petri Net Markup Language PNML* [17,3], an interchange format for different versions of Petri nets, which is currently standardised as International Standard ISO/IEC-15909-2 [15,18]. The PNML meta model was actually used to implement the *PNML Framework*, which helps Petri net tool makers implementing the PNML standard [13] by providing an *API*[5] for Petri nets. The PNML Framework was implemented using the EMF-technology, which generates the API and some code from the class diagram.

Use of EMF-models. In this section, we briefly discuss how such a model can be used for automatically implementing standard functionality by using the EMF-technology. To this end, we interpret the above model as an EMF-model, which technically is different from UML class diagrams, but conceptually means the same.

Once we have created this meta model in EMF, we can fully automatically generate program code that implements standard functionality. First and foremost, EMF generates an API, which consists of a set of classes and interfaces that provide methods for creating Petri nets, and for adding, deleting, and changing its objects and their relation. The generated program code will automatically take care that the restrictions of the meta model are maintained, and that the Petri net is consistent at all times.

In addition, EMF generates code for loading and saving Petri net models from and to XML-files. This relieves us from programming own routines for reading and writing XML-files, and in particular from using XML-parsers explicitly. The XML-format in which the models are stored is defined by the XMI standard [26].

[5] API stands for *Application Programming Interface*.

XMI defines for any UML- resp. EMF-model, how its instances are represented in XML. And EMF generates the code for loading and saving the instances in exactly this format.

In many cases, using the XMI syntax is an easy way for loading and saving models to files. PNML, however, comes with its own XML-format. Therefore, the PNML Framework needs to provide some extra information on how to represent the instances in PNML-syntax. This could be manually programmed code for explicitly reading and writing the files. But, there is a technology that helps us implementing these methods more easily: *Java Emitter Templates* (*JET*). As suggested by the name, this requires to write some templates for all the classes of the model, which define the XML-syntax and are then used to generate the code[6] for loading and saving models in that format. JETs where used for the realization of the PNML Framework (see [13] for details).

But, EMF provides even more: Often, different applications access and modify such a model more or less independently of each other. Then, the different applications need to inform each other about changes; this is typically done by some *notification mechanism* such as the *observer pattern* [10]. EMF generates all the code for this notification mechanism automatically. This mechanism can then be used for implementing an editor for the models using the *Model-View-Controler*-principle (*MVC*)[7]. Actually, EMF can even generate a simple editor for creating, viewing, and editing instances of the model in a tree-like structure based on this MVC pattern.

Generating the Editor. For Petri nets, this automatically generated tree-editor is of limited use. We rather want an editor in which Petri nets can be edited in the graphical syntax as shown in Fig. 1. This is where GMF comes in, which is meant to automatically generate graphical editors on top of EMF models and the code generated by EMF.

To understand the main idea, let us ask the following question: In addition to features of standard editors, what do we need to know for implementing a specialised graphical editor for Petri nets? Basically, we need to know how the different elements from the meta model from Fig. 2 should be represented graphically. We gave this information already when we explained Petri nets in the beginning of Sect. 2.1. Figure 3 illustrates this mapping a bit more formally: Transitions are mapped to squares, places are mapped to circles, arcs are mapped to arrows, and the tokens are mapped to filled circles. Now, Fig. 2 and Fig. 3 together give a full account of the concepts of Petri nets and their graphical representation as explained in the first paragraph of Sect. 2.1.

Based on this (and a little more) information, the GMF can generate a fully fledged Petri net editor. To this end, this information is entered via some GMF-specific tool (so Fig. 3 is just an illustration), from which we can generate the

[6] Here, we cannot go into the details of JET. For more details see the introductory tutorial on JET at
`http://www.eclipse.org/articles/Article-JET/jet_tutorial1.html`

[7] The MVC-Principle is attributed to Trygve Reenskaug back in 1979.

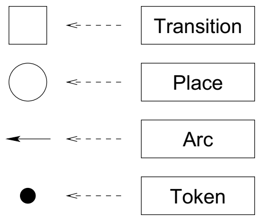


Fig. 3. Mapping to graphics

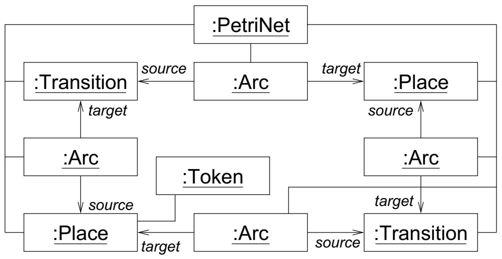


Fig. 4. The Petri net in abstract syntax

full code for the Petri net editor. But, we do not go into the technical details of GMF here. Here, it may suffice that it takes about 15 minutes to provide this information to GMF and to generate the graphical editor for Petri nets—with all nice features you would expect: zooming, overview view, properties view, etc. And of course, you can add, delete, and modify places, transition, and arcs—and load and save Petri nets to and from files.

Altogether, we have seen that the EMF- and GMF-technology can be used for modelling a complete Petri net editor; the editor can then be generated fully automatically without writing a single line of code. Of course, acquiring the skills in using this technology takes some time; and there are some caveats in using EMF and, even the more, GMF. Still EMF and GMF show that MBSE technologies made a big step forward. And we hope that EMF and GMF and their application help overcoming the big mental blockade that still keeps our way of thinking back at the programming level. In addition to EMF and GMF, there are many other technologies and commercial tools that support a similar kind of code generation: e. g. the *Enterprise Architect*, the *Rational Software Architect*, and *Together*.

2.2 More Concepts

In the previous section, we have seen an example of how MBSE can be used today. Next, we will discuss some more advanced concepts in order to better understand the field.

Abstract and Concrete Syntax. First, we have a look at our Petri net example again. The class diagram from Fig. 2 is a meta model for Petri nets, and the Petri net of Fig. 1 is an *instance* of that meta model. But, an instance of a class diagram can also be represented as an *object diagram* such as the one shown in Fig. 4: It shows the objects of the Petri net as instances of the classes, and it shows the links between them, which are instances of the associations of the class diagram. The object diagram from Fig. 4 represents exactly the Petri net from Fig. 1. The representation as object diagrams as shown in Fig. 4 is called *abstract syntax*, the representation in its graphical form as shown in Fig. 1 is called *concrete syntax*[8].

[8] Personally, I prefer the term *graphical syntax* since this term is a bit more suggestive.

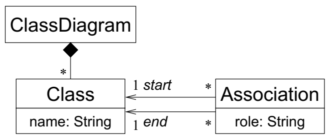


Fig. 5. A meta model for class diagrams

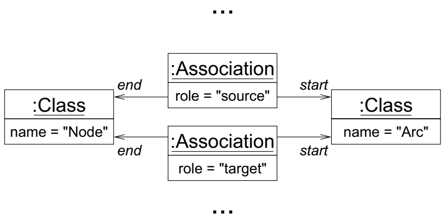


Fig. 6. Meta model in abstract syntax

The abstract syntax (object diagram) comes with the used technology, UML in our case. For a given notation, it is clear how its instances look like in abstract syntax. And for automatically generating the Petri net editor in Sect. 2.1, we needed to define a mapping from the abstract syntax to the concrete syntax.

The Meta Model Diagonal. At last, we change the perspective from someone who develops some piece of software, such as our Petri net editor, to someone who develops a tool that supports MBSE such as EMF and GMF: typically called a *CASE*-tool[9]. So we actually move up one meta-level further: In this perspective, the class diagrams are the models for which we need to define a meta model.

In Sect. 2.1, we started out from a Petri net example, identified its concepts and the relations between them, and then, in Fig. 2, came up with a meta model for Petri nets. Now, we do the same thing for class diagrams. Figure 2 shows an example of a class diagram: It consists of *classes* and *associations* between them. For simplicity, we do not consider inheritance and compositions here. The simplified[10] meta model for class diagrams is shown as a class diagram in Fig. 5.

Again, we can represent instances of this meta model in two different ways. In concrete syntax, such as the one shown in Fig. 2, or in abstract syntax as an object diagram. For example, Fig. 6 shows the two classes Node and Arc of Fig. 2 with the two associations between them in abstract syntax—the rest of the diagram is omitted in order not to clutter the graphics.

The step from an example of a class diagram to its meta model is exactly the same as from an example of a Petri net to its meta model. Still, it twists our brains a bit more. The reason is that, on the next meta-level, we have a class diagram, which very much looks like its instances in graphical syntax again—actually, it is its own instance. This twist occurs whenever we develop a CASE-tool that uses its own technology. And it is crucial conceptually as well as technically that at some level in this hierarchy the next higher level will exactly look like the level below. Conceptually, this means that we can express the concepts of a MBSE-technology in itself. This way, the stack of "turtles over turtles" does not continue indefinitely; rather it ends with a reference to itself. This is important, because the technology that should be promoted by the CASE-tool can be used to develop

[9] CASE stands for *Computer Aided Software Engineering.*
[10] The meta models of EMF and UML are much more complicated [29,5].

that very tool. Actually in the MOF, this would typically end on yet one level on top of the diagram shown in Fig. 5: the so-called MOF-level 3; but, we do not go into the details of MOF here (see [27] for details).

Figure 7 gives an overview on the models we introduced so far, and the different levels: Petri nets are at level 0, the meta model[11] for Petri nets is at level 1, and the meta model for class diagrams is at level 2.

Usually, all these models are arranged vertically on top of each other. This, however, does not leave room to distinguish between concrete and abstract syntax, and leaves no room for defining the concrete syntax for a model. Therefore, we arranged the models in concrete syntax on the main diagonal of a matrix. The vertical dimension corresponds to the MOF levels, which are indicated on the left side. The horizontal dimension is for the syntax. Each entry in the main diagonal of the matrix represents a model in concrete syntax. On the right side of such a model, we show the model represented in its abstract syntax, which actually is an instance of the model sitting exactly on top of it. On the left side of a model in the main diagonal, we show the mapping that defines the concrete syntax for the instances of that model, i. e. its view. The instance in concrete syntax immediately below it uses the graphical elements of that mapping.

Altogether the main diagonal shows models on the different levels, the upper secondary diagonal defines the graphical syntax for the instances of the model, and the lower secondary diagonal shows the abstract syntax of the models. Normally, we do not see much of the abstract syntax; this just reflects the internal representation of the objects in the main memory, on which the API is working. Moreover, this abstract syntax is used for externalising the models to a file. For example XMI defines, how to map this abstract syntax to XML.

Note that Fig. 7 does not show all details. For example, the abstract syntax representation of the Petri net meta model does only cover the Node and Arc classes and the associations between them. And we did not even discuss the definition of the concrete syntax for class diagrams, since it is quite sophisticated. But, GMF is shipped with an editor for UML and EMF class diagrams that is completely defined in GMF. This shows that editors for models that are a bit more complicated than Petri nets can be defined that way.

Transformation Technologies. In order to make all this work, MBSE needs some basis technologies. The most important ones are transformations between different kinds of models (such as the platform independent and the platform specific model in MDA) and from models to code. The first kind is called *Model-to-Model transformation* (*M2M*), the second kind is called *Model-to-Text transformation* (*M2T*). As indicated earlier, EMF makes use of Java Emitter Templates (JET) for M2T-transformations. OMG features a separate standard, which is called *QVT*, for the M2M-transformations in their MDA approach [28]. But, software companies have their own specific ways of doing their transformations and code generation (e. g. [33]). The CASE tool FUJABA[12] greatly benefits

[11] As mentioned earlier this would, typically, be called a model only.
[12] See `http://www.fujaba.de/`

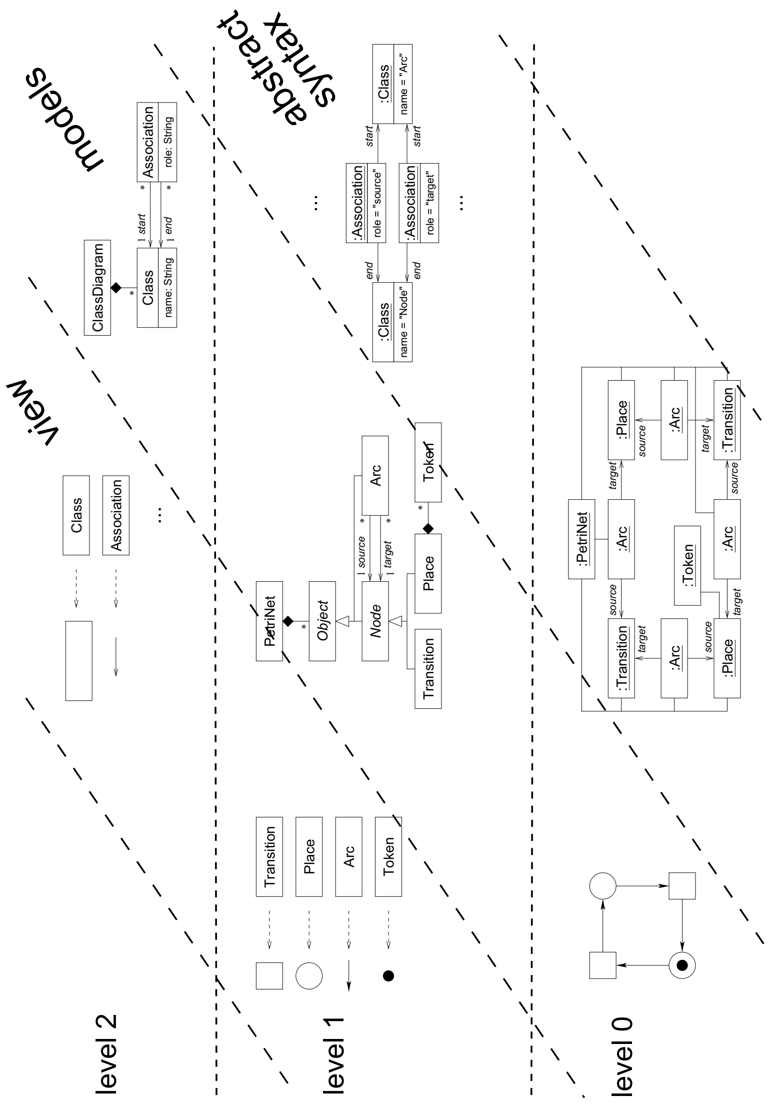


Fig. 7. The meta model diagonal

from the use of *Triple Graph Grammars* (*TGGs*) [32] for M2M-transformations. Actually, we believe that due to their conceptual clarity and its descriptive way of defining relations between models, TGGs have a great potential in MBSE for defining transformations and for keeping different models synchronized [20,11].

2.3 Limitations

Though our Petri net editor is a great example for demonstrating the power of MBSE, it also shows some of its limitations: The editor does not have any specific functionality in addition to being an editor. If we want to equip it with a simulation function or some analysis features, we need to write some program code for that. Of course, that is not a restriction of MBSE in general, but a limitation of the used EMF technology[13], which focuses on the structural parts of a model, and does not have concepts for modelling a specific behaviour. Still, we believe that modelling non-standard behaviour is one of the weaknesses of today's MBSE technologies in general.

Actually, there are different concepts and technologies that allow the modelling of behaviour and to generate program code from that. The most prominent example might be *Executable UML* [24], which is featured by the OMG. Another example are *story patterns* and *story diagrams* [8], which are at the core of the FUJABA tool. This shows that, in principle, it is possible to model non-standard or application-specific behaviour of some software.

The problem with these approaches is that they are often on a quite low level of abstraction and do not easily integrate with other approaches. We believe that one of the main reasons is that UML does not provide a canonical way for integrating behavioural models with structural models. In a class diagram, for example, the only concept relating to behaviour are methods, which are quite close to programming already and often not exactly what we need in a domain model. What is missing here is a concept of *events* (e. g. [23]), which can be used to identify interesting or relevant points of the behaviour from the domain's point of view, and which then can be used for defining synchronizing and adding different kinds of behaviour to these points.

This is why some application domains come with their own specific notation for modelling the behaviour. These approaches are called *domain specific modelling* (*DSM*) and *domain-specific languages* (*DSL*) [6]. One example is workflow management systems and PAIS. Another example is Triple Graph Grammars (TGGs) for defining the relation between different kinds of models in a very suggestive way; still these TGG models of a relation are executable, making a transformation from one model to the other either by generating code for the transformation or by interpreting the TGG [20]. But, by their very definition, these approaches are not universal.

Altogether, MBSE technologies are working today; they are able to generate code for standard behaviour. In some domains, it is possible to model application-specific behaviour. But, there is no approach that works universally. Therefore, some code still needs to be written manually. A universal approach for modelling behaviour still needs to be found.

[13] Note that we do by no means blame EMF for that. In fact, EMF might be so useful already today since it restricts to what can be really done fully automatically: generating all kinds of standard behaviour, which goes much beyond our Petri net editor example; for example, it covers transactionality (see EMF Transactions).

3 Process-Aware Information Systems

In the previous section, we have outlined the main idea of MBSE. One observation was that a universal approach for modelling behaviour (on the domain level) is yet to be found. Amazingly enough, this works pretty well in the area of Process-aware Information Systems (PAIS) and Workflow-Management Systems (WfMS). WfMS allow us to model business processes in a notation very close to the domain without forcing us to use artifacts just for the sake of the used tool or technology. Still, these models can be enacted.

In this section, we will give a brief account on business-process modelling and some reasons why we believe this idea worked.

3.1 Example: A Business Trip

We start with a simple Petri net model—a *workflow net* [34]—of a business trip, which is shown in Fig. 8. The model shows the different tasks of a business trip and the order in which they are executed. The tasks are represented as transitions of the net. Initially, only the task "apply for trip" is enabled. After starting and finishing this task, the tasks "support trip" (a superior countersigning the trip application), and the task "book trip" are enabled concurrently. After the trip, the employee may apply for reimbursement of the travel expenses. Note that, at this stage, there might be an iteration: If the bills for the trip are rejected, the billing activity will be repeated.

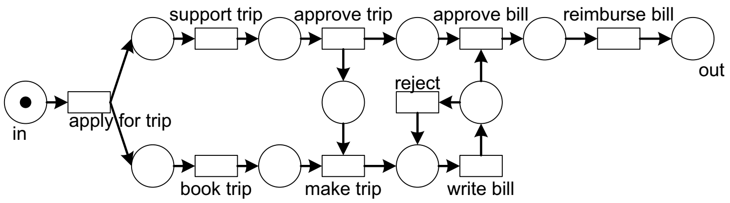


Fig. 8. A model of a business trip

Together, with some other models representing the data and resources of the process, this model can be used to enact the process. But, we do not discuss these other models here. Rather, we discuss the concepts and terminology independently of a concrete example.

3.2 Concepts of Business Process Modelling

This discussion follows the lines of AMFIBIA [2] but is roughly compatible with other terminology [14,34,22]. The original intention of AMFIBIA was to identify the concepts necessary for modelling business processes in such a way that they are independent from a particular modelling notation and that models for different aspects of a business process can be easily integrated with each other.

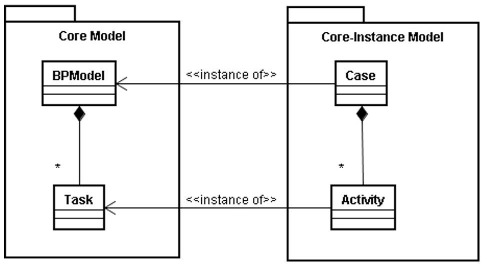


Fig. 9. The core concepts of BPM [2]

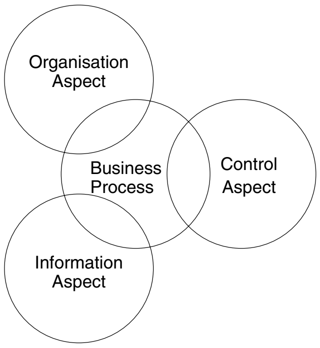


Fig. 10. The main aspects of BPM [2]

Hence we called it *A Meta-model For the Integration of BusIness process modelling Aspects*. But, it turned out that, based on these concepts, we could even implement a workflow engine independently of a particular modelling notation, which we call AMFIBIA too.

A *business process* involves a set of *tasks* that are executed in some enterprise or administration according to some rules in order to achieve certain *goals*. Though the *goals* and objectives are very important for designing and understanding a process, this is the part of business processes that, typically, is modelled only very informally or not modelled at all. The *business process model* is a more or less formal and a more or less detailed description of the persons and resources involved in the execution of a business process and its task and the rules governing their execution.

An execution of a business process model is an *instance* of the business process. Often, the instances of a business process are also called business processes. Since this easily results in confusion, we use the term *business process model* for the model and the term *case* for the instance. The same distinction, applies for tasks: The term *task* refers to a task in the model; an instance of a task in some case is called an *activity*. Note that, even within the same case, a task can be executed, i. e. instantiated, many times due to iterations. These core concepts of business processes are modelled in the class diagram of Fig. 9.

There are many different ways how business processes can be modelled, which we call *business process modelling notations*, and when they come with a precise meaning *business process modelling formalisms*. Independently, of the concrete modelling notation, it is well accepted that there are three main aspects of business processes that need to be modelled for a business process: *control*, *information*, and *organization* (see Fig. 10). The *control aspect* defines the order in which the different tasks of a business process are executed, where concurrent or parallel execution of different tasks is allowed. The *organization aspect* defines the organization structure and the *resources* and *agents*, and the way in which they may or need to participate in the different tasks. The *information aspect* defines the information and documents that are involved in a business process, how it is represented, and how it is propagated among the different tasks of the process.

Typical process modelling notations cover different aspects in a single notation. For conceptual clarity, however, we separated the concepts of the different aspects from each other in AMFIBIA [2]. In particular, AMFIBIA identifies the events that are relevant for coordinating the behaviour of the different aspects of a workflow engine. Typical examples of such events are "initialize process" or "intitialize task". Then, the different aspects of a workflow system can contribute their behaviour to the overall system by triggering or by synchronizing on these events—without knowing the details of other aspects. AMFIBIA uses *Aspect-oriented Modelling* (*AoM*) for modelling a workflow engine. The AMFIBIA engine was implemented by hand based on these models, but we have shown that these kind of extended UML resp. EMF models can also be interpreted and executed directly [19].

3.3 Workflow Management

Typically, business process models are modelled in a notation that is quite close to the domain and can be understood by managers and staff who are supposed to manage and perform them (cf. our model for the control flow of a business trip in Fig. 8). In order to enact these processes by a *Workflow-Management System* (*WfMS*), some more technical details need to be added to the models. This is sometimes called the *operational* or *technical aspect*. The most important operational issue is which *applications* can be used for actually performing a task, and how the information of the process model is mapped to such an application and how information for the workflow can be extracted from it again. Conceptually, this is a mapping from the tasks of a business process model to the applications that are used for executing the tasks. In some cases, these applications are very simple such as a simple form for entering some information; in other cases, these applications can be heavy weight like text processors or numerical software. The integration of such applications is quite a challenge and needs all kinds of technical and programming work, such as implementing adapters or wrappers. But this is independent from an individual task or a specific process.

3.4 Discussion

Altogether, the business process models together with the operational information can be used for executing them. In particular, the models fully define the dynamic behaviour of the business process. This rises the question, why there are still some problems with modelling the behaviour in MBSE? One might claim that, even in workflow management, the most interesting part of the business processes still needs to be programmed: the applications. But, we do not consider that as a valid argument because that is work that could be done for all "relevant" applications once and for all; at least it is not specific to a particular business process. Moreover, the techniques from SOA and web services should help achieving this more easily now.

Here, we give some reasons why we believe that modelling and enacting application specific behaviour worked for business processes:

1. There is a notation that is specific to a *domain*; actually in this context, it might be better to talk about a *field*, because workflow-management covers many different domains. Actually, there is a plethora of many different modelling notations for workflows; basically, every workflow system comes with its own notation. But, they are very similar, so that the existence of many different notations is not the actual reason for the success; the reasons for the many notations are the historical development of the field and marketing considerations.
2. Therefore, we believe that it is not so much the different notations that made workflow-management work. Rather, it was to make two concepts explicit: the *process* and the *task* (resp. their instances *cases* and *activities*) as shown in Fig. 9. This allows us to model, reason about, and coordinate tasks without going into the details of how a task is implemented. Even more, the business process notations do not even allow us to go into the details of a task because this is considered to be atomic within a process.
3. One consequence of making the concepts of *process* and *task* explicit is the natural distinction between coarse-grain and fine-grain behaviour. The coarse-grain behaviour is the process and the coordination of its task, the fine-grain behaviour is in the tasks. Actually, business processes modelling does not even bother about the fine-grain behaviour, since this is within the applications. Most importantly, there is a dedicated notation for the coarse-grain behaviour: the process model.
4. A second consequence of the distinction between process and task is the different time scale. Business processes and the scheduling of their tasks are in the time scale of minutes, hours, days, and even month and years. Reasonable response times on user interactions are in the area of seconds not in the area of milli- or even nanoseconds. Therefore, it is possible to interpret the models instead of generating code for executing them. When models are interpreted, they must be taken seriously since this is what is actually executed. This is an advantage over code generation since generated code can be executed independently from the model, and people could change it independently from the model—ultimately making the original models irrelevant.

4 The Full Picture

The separate discussion of MBSE and PAIS in the previous two sections pretty much reflects the separation of the two fields. Our definition of MBSE, however, makes PAIS a part of MBSE, and we hope that this helps to get the best of both fields for making modelling the main business when developing all kinds of software and information systems. In this section, we give an overview on the field and provide it with some more structure.

4.1 Overview of Approaches

Figure 11 gives an overview of MBSE. Within the rounded box of MBSE, you see some specific approaches. We use the two dimensions to characterize these

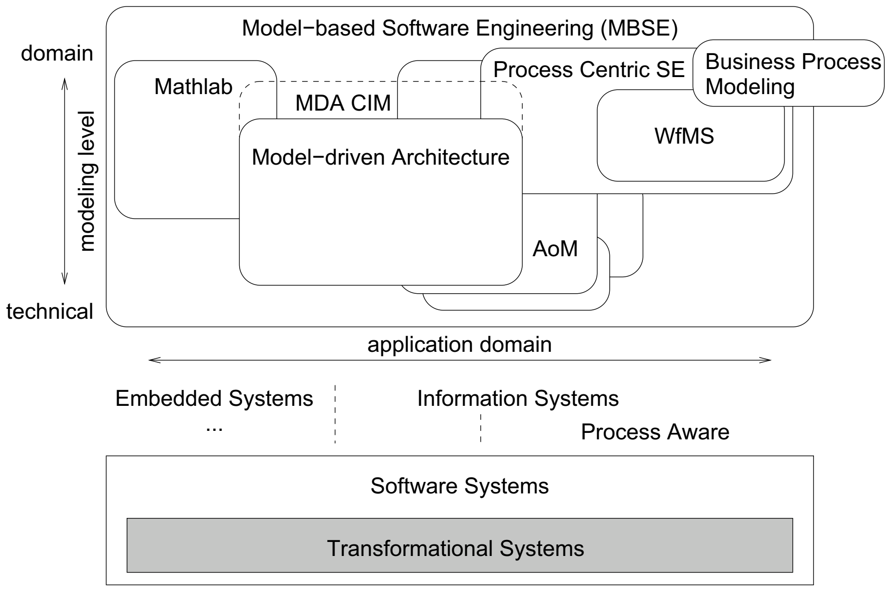


Fig. 11. Overview

approaches. The horizontal dimension indicates the characteristics of the systems that are typically developed by these approaches (application domain). The vertical dimension indicates how close the modelling notation or the set of modelling notations are to the *domain* or to the *technical* realization[14]. The position and dimension of the boxes for an approach indicate, which application domains are typically covered and which modelling levels are covered.

Let us first have a closer look at the application domains, which are indicated below the box for MBSE. In our context, the main distinction is between general *information systems* and *process-aware information systems*. But, there are other kinds of software and systems that are not considered as information systems at all. One example are *embedded systems*, which typically do not need to store data persistently but are highly interactive. Note that, according to our definition, a PAIS is an information system that in some way supports user processes explicitly. We call the technology for developing these kinds of systems by providing explicit process models *Process-centric Software Engineering* (*PCSE*) in order not to confuse the kind of systems to be developed with the technology for developing them.

Even within a single application domain and even within a single system, there are parts with completely different characteristics. This is indicated at the

[14] The idea of the "technical" side is similar to what the MDA calls platform. However, there are some modelling notations or programming languages that are independent from a platform; still they are very technical: In order to model some concepts of a domain, the notation forces us to introduce some artifacts that do not occur in the domain itself. Therefore, we used the terms domain and technical here.

bottom of Fig. 11: Most of today's software systems are *interactive* or *reactive*, which means that the system is in permanent interaction with its users or environment. This contrasts with classical theoretically approaches to capture computation as something that starts with some input, does some computation, and then terminates with some output. Therefore, Harel and Pnueli distinguished between *reactive* systems and *transformational* systems [12]. Information systems are reactive in nature. But, they have parts that are transformational in nature. For example, the transformational part could be a procedure for analysing some data, or software that calculates an image from the signals of a computer tomograph, or the simulation of some finite-element models, or the encryption or decryption of some text. The transformational parts are much more algorithmic, and are typically developed separately. Since algorithms are quite close to programming, it is quite natural to program these parts directly. Therefore, they are often out of the scope of MBSE, except that we need an interface for invoking the algorithms. These transformational parts of software are indicated by the shaded box in our figure.

Let us have a closer look at the different approaches within MBSE and their characteristics now. The typical *business process modelling* notations are close to the domain. That is why we placed them at the very top. Actually, we placed this box partly outside MBSE since business process modelling does not always have the purpose of developing or improving a software system. The notations for workflows are more technical (or add additional models that are more technical). Clearly, the WfMS approaches as well as all PCSE approaches are made for supporting some form of process. Therefore, they are typically used only for the development of PAIS.

The MDA covers all kinds of systems. And the MDA can be used for developing PAIS, even if the processes are not made very explicit in the notation. Actually, there are some specific *UML profiles* for processes, but we consider this as a half-hearted solution to PCSE. That is why MDA covers only a part of PAIS as application domain. We consider the models of MDA to be a bit more technical, so that we placed them a bit lower than PCSE. Actually, the MDA [25] defines so-called *computation independent models* (*CIM*) that are closer to the domain. But, the MDA itself and most of the realizations do not give much details on the use of CIM. That is why we have indicated that part of MDA by a dashed line.

We also included Mathlab in this diagram as an example of a modelling approach that is often used in the design of embedded systems. And we included *Aspect-oriented Modelling* (*AoM*) in this schema, since we believe that this might be one of the concepts that help to model behaviour close to the domain, and to integrate different forms of behaviour models with each other.

This was actually the idea of AMFIBIA, and we have shown that a complete workflow management system could be fully modelled (in particular covering its behaviour) that way [2,19]. The most important concept was to make all the relevant *events* explicit, so that the different aspects could contribute to the behaviour by synchronizing on these events. The actual behaviour of the different aspects was then defined in terms of automata that were synchronized with and triggered by the events.

4.2 Combining Powers

As pointed out earlier, PAIS resp. PCSE has its strength in modelling the coarse-grain behaviour, and it does not deal with the fine-grain behaviour at all. By contrast, classical MBSE does not have a universal concept for modelling behaviour, but the executable approaches work better on the fine-grain behaviour. The question, now is how these strengths can be combined. In this section, we briefly indicate some ideas, which of course need further research.

Coarse-grain and Fine-grain Behaviour. One lesson MBSE can learn from PCSE is the clear separation of coarse-grain and fine-grain behaviour and the way it is achieved. UML has different notations for modelling behaviour on different levels and there are some UML profiles for processes. Still, it is not clear how they work together and how they are integrated. A methodology for consistently using these concepts such that they can automatically be integrated is still missing. Making the concepts of process and task explicit in the structural as well as in the behavioural models might be a step in that direction.

Actually, there are technologies that allow us to realize such an integration of fine-grain and coarse-grain behaviour. Most prominently there is BPEL4WS and WSDL for defining processes on top of *web services* [1] or in the terminology of Peltz [31] for the *orchestration* and *choreography* of services. But, these are on the technical side already. What would be needed is something closer to the domain, that later allows a smooth transition to these technologies.

Modelling Events. As pointed out earlier, introducing an explicit concept of events in a domain helps better capturing the domain and its behaviour. Then, the events can be used for coordinating and synchronizing the behaviour of different aspects. Moreover, the events can be used to coordinate the fine-grain and the coarse-grain behaviour of a system, by synchronizing on them.

DSM only where Necessary. We pointed out already that there are many different notations for modelling behaviour in specific domains. One such domain are PAIS. But, this does not mean that everything needs to be domain specific. For example, the information aspect could use the very same notations as in MBSE. This would make it easier to use existing technologies from MBSE for the development of PAIS and to switch between the two worlds.

This way, we would be able to use what ever is best from the two fields.

5 Conclusion

In this paper, we have given an overview of MBSE and suggested that PAIS, resp. PCSE should be considered as a part of MBSE. We pointed out that one of the main factors for making WfMS and PAIS a success is the clear and enforced separation between coarse-grain and fine-grain behaviour by making the concepts of process and task explicit. In order to exploit that in general software engineering, we need to have such an enforced separation and a way to integrate the fine-grain and the coarse-grain behaviour in the final software.

References

1. Andrews, T., Cubera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarama, S.: Business process execution language for web services specification. Technical Report Version 1.1 (2003)
2. Axenath, B., Kindler, E., Rubin, V.: AMFIBIA: A meta-model for the integration of business process modelling aspects. International Journal on Business Process Integration and Management 2(2), 120–131 (2007)
3. Billington, J., Christensen, S., van Hee, K., 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)
4. Böhm, M., Schulze, W.: Grundlagen von Workflow-Managementsystemen. Wissenschaftliche Beiträge zur Informatik 8(2), 50–65 (1995)
5. Budinsky, F., Steinberg, D., Merks, E.: Eclipse Modeling Framework, 2nd edn. Addison-Wesley Professional, Reading (2006)
6. Cook, S.: Domain-specific modelling and model driven architecture. MDA Journal, pp. 2–10 (January 2004)
7. Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Process-Aware Information Systems: Bridging People and Software through Process Technology. Wiley & Sons, Chichester (2005)
8. Fischer, T., Niere, J., Torunski, L., Zündorf, A.: Story diagrams: A new graph rewrite language based on the Unified Modeling Language and Java. In: Ehrig, H., Engels, G., Kreowski, H.-J., Rozenberg, G. (eds.) TAGT 1998. LNCS, vol. 1764, pp. 296–309. Springer, Heidelberg (2000)
9. Fowler, M.: Language workbenches and model driven architecture (June 2006), `http://martinfowler.com/articles/mdaLanguageWorkbench.html`
10. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading (1995)
11. Greenyer, J., Kindler, E.: Reconciling TGGs with QVT. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F. (eds.) MODELS 2007. LNCS, vol. 4735, pp. 16–30. Springer, Heidelberg (2007)
12. Harel, D., Pnueli, A.: On the development of reactive systems. In: Apt, K.R. (ed.) Logics and Models of Concurrent Systems. Series F: Computer and System Science, vol. 13, pp. 477–498. Springer, Heidelberg (1985)
13. Hillah, L., Kordon, F., Petrucci, L., Trèves, N.: Model engineering on Petri nets for ISO/IEC 15909-2: API framework for Petri net types metamodels. Petri Net Newsletter 69, 22–40 (2005)
14. Hollingsworth, D.: The workflow reference model. Technical Report TC00-1003, The Workflow Management Coalition, WfMC (1995)
15. ISO/JTC1/SC7/WG19. Software and Systems Engineering – High-level Petri Nets – Part 2: Transfer Format. FCD 15909-2, v. 1.2.0, ISO/IEC (June 2007)

16. Jablonski, S.: Workflow-Management-Systeme: Modellierung und Architektur. Thomson Publishers (1995)
17. Jüngel, M., Kindler, E., Weber, M.: The Petri Net Markup Language. Petri Net Newsletter 59, 24–29 (2000)
18. Kindler, E.: The Petri Net Markup Language and ISO/IEC 15909-2: Concepts, status, and future directions. In: Schnieder, E. (ed.) Entwurf komplexer Automatisierungssysteme, 9. Fachtagung, pp. 35–55 (2006)
19. Kindler, E., Schmelter, D.: Aspect-oriented modelling from a different angle: Modelling domains with aspects. In: 12^{th} International Workshop on Aspect-Oriented Modeling, pp. 7–12 (2008)
20. Kindler, E., Wagner, R.: Triple Graph Grammars: Concepts, extensions, implementations, and application scenarios. Technical Report tr-ri-07-284, Department of Computer Science, University of Paderborn (2007)
21. Kleppe, A., Warmer, J., Bast, W.: MDA Explained: The Model Driven Architecture: Practice and Promise. Object Technology Series. Addison-Wesley, Reading (2003)
22. Leymann, F., Roller, D.: Production Workflow: Concepts and Techniques. Prentice-Hall PTR, Upper Saddle River (1999)
23. McNeile, A.T., Simons, N.: Protocol modelling: A modelling approach that supports reusable behavioural abstractions. Software and Systems Modeling 5(1), 91–107 (2006)
24. Mellor, S.J., Balcer, M.J.: Executable UML: A Foundation for Model-driven Architecture. Addison-Wesley, Reading (2002)
25. MDA guide v1.0.1. (2003), `http://www.omg.org/cgi-bin/doc?omg/03-06-01`
26. XML metadata interchange (XMI) specification, version 2.0. Technical Report formal/03-05-02, The Object Management Group, Inc. (2003)
27. Meta object facility (MOF) specification, version 1.4.1. Technical Report formal/05-05-05, The Object Management Group, Inc. (2005)
28. MOF QVT, final adopted specification. Technical Report ptc/05-11-01, The Object Management Group, Inc. (2005)
29. OMG. OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2. Object Management Group, 140 Kendrick Street, Needham, MA 02494, USA (2005), `http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF`
30. OMG. Unified Modeling Language: Superstructure Version 2.0. Object Management Group, 140 Kendrick Street, Needham, MA 02494, USA (2005), `http://www.omg.org/cgi-bin/doc?formal/05-07-04`
31. Peltz, C.: Web services orchestration and choreography. IEEE Computer 36(10), 46–52 (2003)
32. Schürr, A.: Specification of graph translators with triple graph grammars. In: Mayr, E.W., Schmidt, G., Tinhofer, G. (eds.) WG 1994. LNCS, vol. 903, pp. 151–163. Springer, Heidelberg (1995)
33. Unland, L., George, T.: MDA richtig einsetzen: Klassische und innovative Rezepte. OBJEKTspektrum 6, 41–44 (2005)
34. van der Aalst, W., van Hee, K.: Workflow Management: Models, Methods, and Systems. Cooperative Information Systems. MIT Press, Cambridge (2002)
35. Workflow Management Coalition: Terminology & glossary. Technical Report WFMC-TC-1011, The Workflow Management Coalition, WfMC (1999)

Petri Net Transformations for Business Processes – A Survey

Niels Lohmann[1], Eric Verbeek[2], and Remco Dijkman[3]

[1] Universität Rostock, Institut für Informatik, 18051 Rostock, Germany
niels.lohmann@uni-rostock.de
[2] Technische Universiteit Eindhoven
Department of Mathematics and Computer Science
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
h.m.w.verbeek@tue.nl
[3] Technische Universiteit Eindhoven
Department of Technology Management
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
r.m.dijkman@tue.nl

Abstract. In Process-Aware Information Systems, business processes are often modeled in an explicit way. Roughly speaking, the available business process modeling languages can be divided into two groups. Languages from the first group are preferred by academic people but shunned by business people, and include Petri nets and process algebras. These *academic* languages have a proper formal semantics, which allows the corresponding *academic* models to be verified in a formal way. Languages from the second group are preferred by business people but disliked by academic people, and include BPEL, BPMN, and EPCs. These *business* languages often lack any proper semantics, which often leads to debates on how to interpret certain *business* models. Nevertheless, business models are used in practice, whereas academic models are hardly used. To be able to use, for example, the abundance of Petri net verification techniques on business models, we need to be able to transform these models to Petri nets. In this paper, we investigate a number of Petri net transformations that already exist. For every transformation, we investigate the transformation itself, the constructs in the business models that are problematic for the transformation and the main applications for the transformation.

1 Introduction

Today, Business Process Management (BPM) is becoming more and more important to the business, which explains the increased popularity of business process modeling, and a plethora of similar but subtly different process modeling approaches has been proposed, including the Web Services Business Process Execution Language (BPEL) [1], the Event-driven Process Chains (EPCs) [2], the Yet Another Workflow Language (YAWL) [3], the Business Process Modeling Notation (BPMN) [4], process algebras [5] and Petri nets [6,7]. The resulting babel has raised the issue of comparing the relative expressiveness between languages and translating models defined in one language into *equivalent* models defined in another language.

Academic people prefer languages like process algebras and Petri nets, as these languages have proper formal semantics and, hence, one can check relevant and interesting properties on corresponding models. Business people, however, prefer languages like BPEL, EPCs, and BPMN, which more than often lack these proper formal semantics. As a result, business processes found in practice are modeled in a way that leaves room for interpretation. An exception to this is YAWL, which has its roots in the academic world, but is actually used in practice, and has a semantics in terms of reset nets (which will be explained further on in this paper). YAWL supports the most frequent control-flow patterns found in the current workflow practice. As a result, most workflow languages can be mapped onto YAWL without loss of control-flow details, even languages allowing for advanced constructs such as cancelation regions and OR-joins. If we want to combine the best of both worlds, that is, to combine the ability to calculate certain properties on a formal semantics and the actual, existing in practice, *business* models, we need to transform these business models to languages with a formal semantics. In this paper, we will cover a number of transformations from business models in BPEL, EPCs, YAWL, and BPMN, focusing on transformations onto Petri nets.

The remainder of this paper is organized as follows. Section 2 introduces the concepts (like Petri nets and relevant properties). These concepts are presented in an informal way, for a more formal description, we refer to the literature on the transformations itself. This section also explains 'workflow patterns' (frequently used constructs in business process modeling) and how these patterns can be transformed into Petri nets. Finally, this section presents a running example, which we use throughout the paper to illustrate the transformations. Sections 3, 4, 5 and 6 introduce the business modeling languages, their transformations into Petri nets and the constructs that are difficult or impossible to transform. Each transformation is explained by reference to the workflow patterns that the language supports. For additional details on these transformations, we refer to the existing literature on these transformations. Section 7 concludes the paper.

2 Preliminaries

This section first presents the different types of Petri net used to formalize the business process modeling languages. Second, it explains some well-known patterns from the area of business process modeling, their representation using Petri nets and possible difficulties to represent them. Third, it presents an example business process that we will use throughout this paper to illustrate the modeling languages.

2.1 Petri Net Classes

We assume the standard definition of *Petri nets* [6,7] to consist of two finite disjoint sets of places and transitions (graphically represented by circles and squares) together with a flow relation (represented as directed arcs).

A Petri net is called a *workflow net* [8] if it has a distinct source place, a distinct sink place, and if all nodes lie on some path from this source place to the sink place. Typically, a token in the source place signifies a new case, whereas a token in the sink place signifies a completed case. All transitions in the workflow net should contribute to forwarding some case from the new state to the completed state.

A workflow net net is called *sound* [8] if any case can always complete in a proper way (that is, without tokens being marooned) and if no transition is dead. Note that the workflow net requirement is a structural requirement, whereas the soundness requirement is a behavioral requirement. A workflow net is called *relaxed sound* [9] iff every transition can help in forwarding some case from the new state to the completed state. Note that this requirement is less strict than the soundness requirement, as the option to complete properly might not be guaranteed for every reachable marking.

A *reset net* [10] is a Petri net extended by reset arcs. A transition that is connected to a place with a reset arc removes *all* tokens on that place upon firing. Reset nets are more expressive than classical Petri nets: some forms of verification are undecidable in reset nets, while they are decidable in classical Petri nets.

An *open net* [11] is a Petri net extended with a set of interface places and a set of desirable final markings. The interface places are partitioned to input and output places. Open nets thereby extend classical workflow nets with an asynchronous interface to explicitly model message exchange. An important correctness criterion for open nets is *controllability* [12]. An open net is controllable if another open net exists such that their composition (where the communication places of both nets have been glued) always ends up in a desired final marking. Note that (relaxed) soundness does not imply controllability, or vice versa.

2.2 Workflow Patterns and Petri Nets

A collection of workflow patterns has been developed to analyze the expressive power of languages for workflow and business process modeling. Patterns with respect to the control-flow aspect, on which we focus in this paper, are described in [13,14]. The expressive power of modeling languages can be explained in terms of which patterns they support. (For an overview of support from languages in this paper, see [14].) Therefore, we use the workflow patterns from [14] as a frame of reference here. We show how some of them can be mapped to Petri nets, or what the problems are when they cannot be mapped. This provides the reader with information about which languages support which patterns and the particularities of mapping these patterns to Petri nets. Hence, it gives a good overview of the state-of-the-art in Petri net transformation of business process modeling languages.

Figure 1 shows the mapping of some of the workflow patterns to Petri nets. Transitions that represent tasks are given the labels A, B or C and transitions that affect the flow of control, but that do not represent tasks are given more

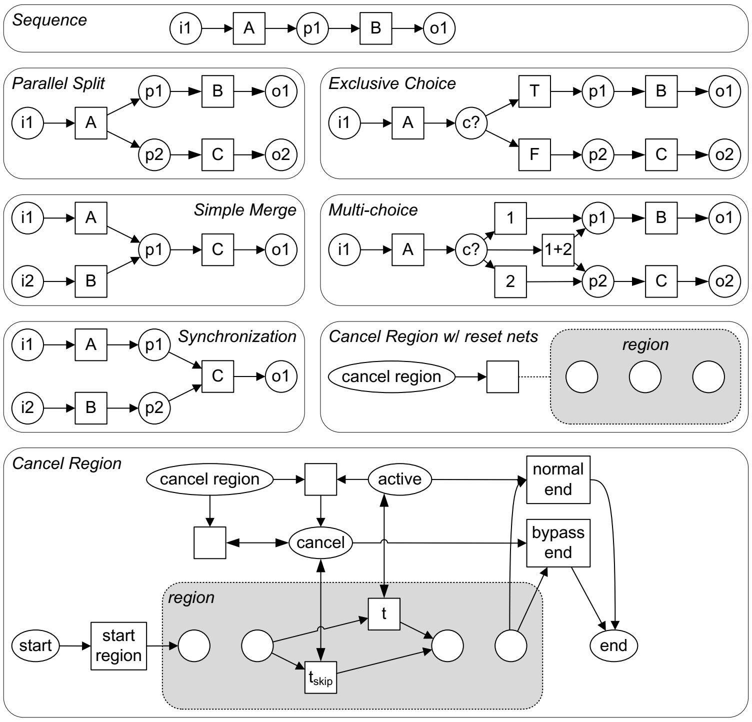


Fig. 1. Workflow patterns in Petri nets

descriptive labels. The so-called *simple patterns* can easily be represented in Petri nets. These are:

– 'Sequence', a task is enabled after another task is completed;
– 'Parallel Split' (also called 'AND-split'), all outgoing branches are enabled at the same time;
– 'Synchronization' (also called 'AND-join'), the process must wait for all incoming branches to complete before it can continue;
– 'Exclusive Choice' (also called 'XOR-split'), the execution of one out of a number of branches is chosen;
– 'Simple Merge' (also called 'XOR-join'), the process continues when one incoming branch completes.

Another pattern that can easily be represented by Petri net is the 'Deferred Choice' pattern, which differs from the 'Exclusive Choice' pattern in the fact that the branch is chosen by the environment rather than by the system itself.

Patterns that are harder to represent in Petri nets include the 'Multi-choice', in which the execution of a number of branches is chosen, and the 'Cancel Region', in which the execution of a set of tasks is disabled. These patterns lead to nets that become hard to read. For example, although the 'Multi-choice' in Fig. 1 is still readable, it becomes hard to read when there any number out of

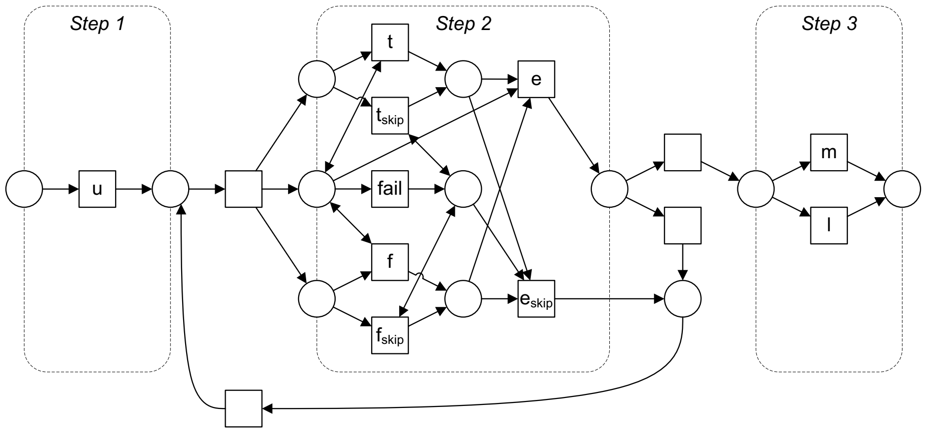


Fig. 2. Example process modeled as a Petri net

three or four outgoing branches can be chosen. To make these patterns more readable, another class of Petri net can be chosen. Figure 1 illustrates this for the 'Cancel Region', which is both represented as a classical Petri net and as a reset net.

Patterns that cannot be represented as a classical Petri net include the 'General Synchronizing Merge', which corresponds to a wait-and-see synchronizing construct. To represent such patterns, we would need very sophisticated classes of Petri nets, for which analysis might just be infeasible.

2.3 Example Process

We will illustrate each of the modeling techniques, using the same example process. As example process we take an image editing process. First, the customer uploads an image (u). Second, the following procedure is applied: The image is finished (f) and concurrently a thumbnail is created (t). Afterwards the results are evaluated (e). If a failure occurred or if the evaluation is negative, the procedure is repeated. Third, if the image is too big, it is stored temporarily and only a link is sent to the customer (l); otherwise the image is sent by e-mail (m). At any point during the second step, a failure occurs if the format in which the figure is stored cannot be imported by the tool that are used to process the image. The other steps are assumed to be infallible. Figure 2 shows the example modeled as a workflow net.

3 BPMN

The Business Process Modeling Notation (BPMN) [4] is developed as a standard for business process modeling. This section briefly explains the language, the main challenges when transforming BPMN models to Petri nets and the

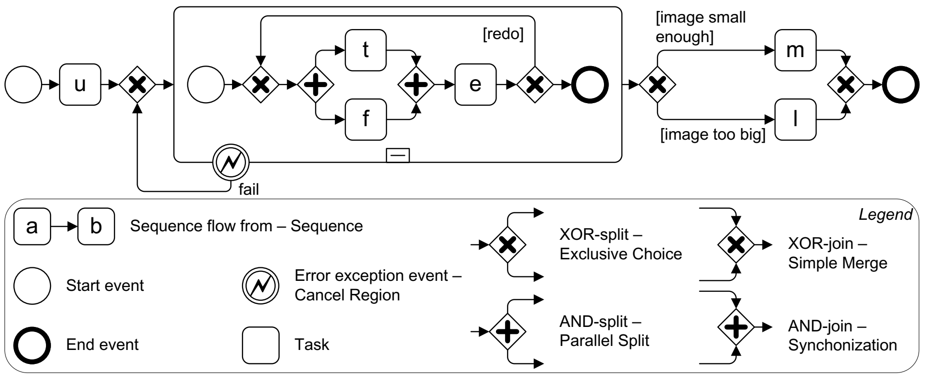

u
t
f
e
[redo]
fail
[image small enough]
m
l
[image too big]
Legend
a
b
Sequence flow from – Sequence
Start event
End event
Error exception event – Cancel Region
Task
XOR-split – Exclusive Choice
AND-split – Parallel Split
XOR-join – Simple Merge
AND-join – Synchonization

Fig. 3. The example process as a BPMN process

transformation itself. The section focuses on BPMN version 1.0, because at the time that the transformation was developed that was the current version. Therefore, comments on BPMN apply to version 1.0 only.

3.1 Language

BPMN is a rich language that provides the modeler with a large collection of object types to represent various aspects of a business process, including the control-flow, data, resources and exceptions. BPMN is mainly meant for modeling business processes at a conceptual level, meaning that it is mainly intended for drawing process models that will be used for communication between stakeholders in the processes. As a consequence, formal rigor and conciseness were not primary concerns when developing the BPMN specification.

The three types of BPMN objects that can be used to represent the control-flow aspect of a process are *activities*, *events*, *gateways*. Many subtypes of these objects exist. Control-flow objects can be connected by sequence flows, which are directed arcs that represent the flow of control from one object to the next. Figure 3 illustrates some of these objects, by representing the example process in BPMN and by relating the object types to the workflow patterns explained in Sect. 2.2.

3.2 Transformation Challenges

Due to the large number of object types that constitute BPMN it is hard to define a mapping and show (or prove) that the mapping works for all possible combinations of these object types. Especially, because the mapping of a composition of object types is not the same as the composition of the mapping of those object types. This complicates, for example, defining mapping rules for interruptions of sub-process invocations.

BPMN frequently introduces shorthands and alternatives for representing certain constructs. For example, an activity with multiple incoming flows will start as soon as the control is passed to one of these flows. Hence, an activity with multiple incoming flows behaves similar to an activity that is preceded by an XOR-join. This further complicates the mapping.

Version 1.0 of the BPMN standard contains inconsistencies and ambiguities. We uncovered several while defining the mapping [15]. This illustrates that defining a mapping to a formal language can be useful to uncover flaws in an informal language.

3.3 Transformation and Application

In prior work we defined a mapping from a restricted version of BPMN to workflow nets [15]. The restrictions include that the BPMN models must have a single start and a single end event. Also, activities with multiple concurrent instances and some types of gateways cannot be used (in particular OR-gateways, which represent the 'Multi-choice' and 'General Synchronizing Merge' patterns). The mapping focuses on the control-flow aspect of processes.

The mapping is developed by defining mappings for each activity, events and gateway object type, similar to the way in which Petri net mappings are defined for each workflow pattern in Sect. 2.2. When transforming a model, first each object is mapped onto a partial Petri net and second the partial Petri nets are composed into a complete model. Although this approach works for many constructs, some constructs cannot simply be mapped and then composed. The mapping from BPMN to workflow nets allows the soundness of these nets to be analyzed (see Sect. 2).

To the best of our knowledge the only other mapping from BPMN to a formal language is from BPMN to CSP [16].

4 EPCs

Event-driven Process Chains (EPCs) [2] were developed to provide an intuitive modeling language to model business processes. This section briefly explains ECPs, the main challenges when transforming EPCs to Petri nets and the transformation itself.

4.1 Language

Like BPMN, EPCs are meant for modeling business processes at a conceptual level: EPCs are not intended to be a formal specification of a business process. Instead, it serves mainly as a means of communication. There are, however, some conceptual differences between BPMN and EPCs:

- BPMN supports the 'Cancel Region', whereas EPCs do not, and
- EPCs supports the 'General Synchronizing Merge', whereas BPMN does not.

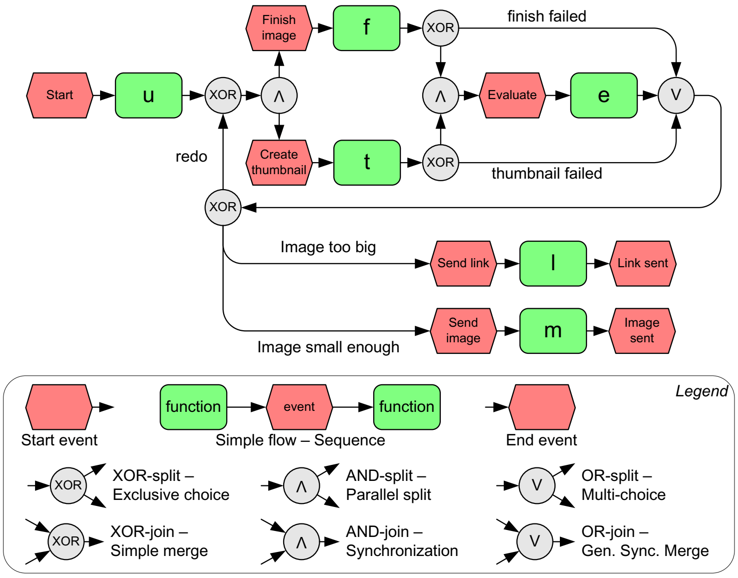


Fig. 4. The example process as an EPC

Three types of EPC objects can be used to model the control-flow aspect of a process: *functions*, *events*, and *connectors*. In a natural way, these types correspond to the BPMN *activities*, *events*, and *gateways*. However, EPCs do not allow for exceptions, and it supports only a limited set of connectors, which is shown by Fig. 4. Apart from the full set of connectors, this figure also shows an the example process as an EPC, and it relates the object types to the workflow patterns explained in Section 2.2.

4.2 Transformation Challenges

A main challenge in EPCs is the semantics of the constructs that support the 'Simple Merge' and 'General Synchronizing Merge' patterns, viz. the XOR-join connector and the OR-join connector. Everybody agrees that the XOR-join connector should be enabled if one of its inputs is enabled, but this agreement is lacking in case more than one inputs is enabled. Some say that the XOR-join should be executed for every single enabled input, while others say that the connector should block if multiple inputs are enabled. An even bigger problem is the OR-join connector, for which a definitive semantics has lead to extensive discussions in literature and to different solutions, all of which fail for some EPCs [17,18,19]. As a result, not everybody will agree on a given mapping, as not everyone will agree with the semantics used by it.

Furthermore, an EPC allows for multiple start events and multiple final events, but not all combinations of these events are possible. Although the process designer might know the possible combinations, an EPC does not contain this information.

4.3 Transformation and Application

The transformation to workflow nets as introduced in [20] explicitly targets the verification of EPCs, and assumes that an OR-join is enabled as soon as any of its inputs are enabled, and that an XOR-join with multiple inputs enabled will be executed multiple times. This transformation results in a safe[1] workflow net, as it introduces a so-called shadow place for every place, and uses a number of (optional) EPC reduction rules prior to transforming the EPC.

As mentioned above, an EPC allows for multiple start events and multiple final events. However, the EPC designer might be aware of the fact that certain combination of start events will not occur, and that the process will behave in such a way that certain combinations of final events are impossible as well. Clearly, this knowledge of the designer is vital for the verification, but unfortunately not included in the EPC. Therefore, the verification approach in [20] proposes to query the user for this information. First, the user has to select which combinations of start events are possible. Second, based on this information a state space is build that possibly contains multiple subsets of final events. Third, the user has to select the subsets of final events that indeed are possible. For the example process, three subsets of final events were detected (next to a number of deadlock states that also include non-final events, which are assumed to be ignored by default), from which one (the subset containing both Image sent and Link sent, which appears to be possible) needs to be ignored. Fourth, the soundness property is checked on the resulting state space. If the state space corresponds to a sound net, then the EPC is correct: A desired subset of final events will always be reachable. Otherwise, the relaxed soundness property is checked, where an OR-join (OR-split) transition is allowed to be non-relaxed sound if and only if its inputs (outputs) are covered by relaxed sound OR-join (OR-split) transitions. If the state space is relaxed sound, then the EPC can be correct, although it allows for undesirable behavior. Otherwise, the EPC is incorrect, as certain parts of the EPC cannot lead to any desired subset of final events, when executed.

The example EPC can be correct, but allows for undesired behavior. For example, if Finish image fails, then Create thumbnail should fail as well to be able to reach a desirable subset of final events (that is, {Link sent}, or {Image sent}).

5 BPEL

The Web Services Business Process Execution Language (BPEL) [1], is a language for describing the behavior of business processes based on Web services. That makes BPEL a language for the *programming in the large* paradigm. Its focus is — unlike modifying variable values in classical programming languages such as C or Java — the message exchange and interaction with other Web services. Advanced concepts such as instantiation, a complex exception handling,

[1] A net is safe if and only if every place in every reachable marking contains at most one token. As a result, the set of reachable markings is finite.

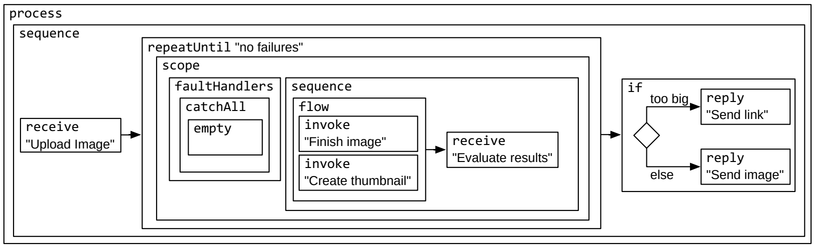


Fig. 5. The example process as a BPEL process

and long running transactions are further features that are needed to implement business processes.

5.1 Language

Activities organize the communication with partners, variable manipulation, etc. They can be ordered using structured activities which makes BPEL similar to a block-based language. To support the simple patterns depicted in Fig. 1, control links can be used to express splits, choices and merges. Due to restrictions, BPEL avoids the problems occurring with the OR-join.

Being an execution language, the exceptional behavior which also includes cancelation of parts of the process is described in great detail in the BPEL specification [1]. In addition, BPEL supports the concept of hierarchical scopes that model local units to which a local exception management (implemented by fault, termination, and compensation handlers) is bound.

Example Process. BPEL is an XML-based execution language without standardized graphical representation. Figure 5 shows a possible implementation of the example process using BPEL, in a schematic way. The process "Image editing" contains a sequence, which in turn contains receive "Upload image", repeatUntil "no failures", and if "too big", etc.

5.2 Transformation Challenges

The positive control flow of a BPEL process (i. e., the sheer business process) can be straightforwardly mapped to Petri nets by defining a translation of each of BPEL's activity type. The biggest challenge is the transition from the positive to the negative control flow. The BPEL specification defines the following steps to be performed in case a fault occurs. (1) All running activities in the scope of the faulty activity have to be stopped. (2) The fault handler of the scope is called. (3) If the fault could be handled, the execution continues with the scope's successor. If the fault could not be handled, it is escalated to the parent scope.

This procedure requires a global state (i. e., all running activities are stopped) to be reached before invoking a fault handler. Petri nets naturally model distributed systems with concurrently acting *local* components. The formalization of the enforcement of a *global* state with Petri nets is therefore cumbersome, because it requires all components to synchronize. The stopping of originally independently running activities can be achieved on two ways.

(a) The request to stop is propagated from the scope to each running activity.
(b) A global "variable" modeled by a place describes the "mode" of the scope; that is, whether the scope's internal activities should be executed or stopped.

Option (a) has the advantage that the BPEL process's nature of a distributed system is mirrored in the Petri net model: Activities embedded into a flow are executed concurrently. However, this concurrency introduces a lot of intermediate states that model the situation in which a fault is detected by a scope yet not fully propagated. With a global state like in (b), stopping can be modeled more easily, but implicitly *schedules* originally concurrent activities.

A similar problem arises when modeling the dead-path elimination [1] which requires to skip activities than cannot be executed due to their join condition. Again, this can be achieved through propagation (of the information whether a branch is executed or skipped) or global places (a global status place which controls whether an activity is executed or skipped).

5.3 Transformation and Application

Though there exist many works to formalize BPEL using Petri nets (see [21] for an overview), only two Petri net transformations are *feature-complete*; that is, covering all mentioned activities and aspects of a BPEL process. These formalizations are from the Humboldt-Universität zu Berlin together with the University of Rostock (abbreviated with "HR"), and from the Queensland University of Technology ("QUT"). A detailed comparison between these semantics can be found in [22]. Here, it suffices to mention that the HR transformations uses propagation (see Sect. 5.2) and selectively results in either a normal Petri net or an open net, whereas the QUT transformation uses global places and results in a workflow net.

HR Transformation to Petri nets. The HR transformation implemented in the tool BPEL2oWFN [23] can be used to translate BPEL into a standard Petri net without interface places. This Petri net can be analyzed for deadlocks or other classical Petri net properties, soundness, as well as temporal logical formulas. A case study is presented in [24] shows that the internal behavior of large processes with nested scopes and complex exception handling can be analyzed using the model checking tool LoLA [25]. Furthermore, the semantics could be validated by proving deadlock-freedom of the patterns.

HR Transformation to Open Nets. With the explicitly modeled interface of the open net, the communication behavior of the BPEL processes can be analyzed. The tool Fiona [23] can check controllability [12], synthesize a partner process which can be translated back to BPEL [26], or calculate the operating guideline [27] of the net. This operating guideline characterizes all partners that communicate deadlock-freely with the original net and can be used for service

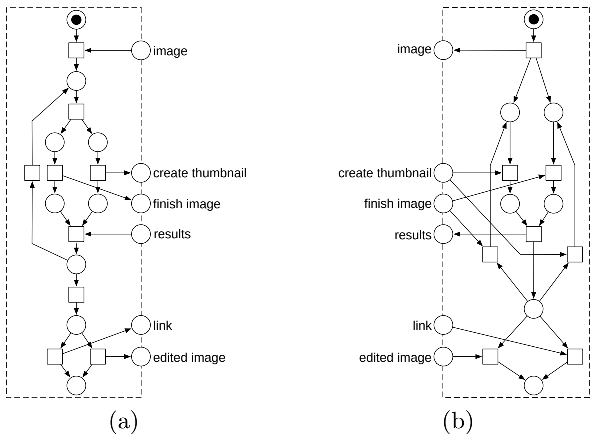


Fig. 6. The BPEL process transformed into an open net (a) and a synthesized partner open net (b). To increase legibility, fault handling is not depicted.

discovery. An extension to formalize choreographies [28] further allows to apply the mentioned analysis techniques to a choreography of many BPEL processes instead of just a single process.

Figure 6(a) shows the result of transforming the BPEL process to an open net. The net is controllable, and Fig. 6(b) shows a synthesized partner open net. The composition of the open nets is free of deadlocks, and a desired final marking of the composition always reachable.

QUT Transformation to Workflow Nets. The QUT transformation was developed to decide soundness on the resulting workflow net, using the WofBPEL tool [29], which is a spawn-of of the workflow verification tool Woflan [30,31]. However, the transformation does not allow for improper completion and BPEL processes by definition have the option to complete. Thus, the soundness check boils down to a check on dead transitions. Next to the soundness check, the WofBPEL tool can also check whether an incoming message can be handled by multiple elements (which is considered an anomaly in BPEL) and can augment the BPEL model with information on when to garbage collect queued messages. Based on this information, the BPEL garbage collector can decide to remove for a certain running instance certain incoming messages from the message queue as it is certain that these messages cannot be handled anymore by the instance.

6 YAWL

The Yet Another Workflow Language (YAWL) [3] was originally conceived as a workflow language that would support 19 of the 20 most frequent used patterns found in existing workflow languages. As such, YAWL supports the ‘Multiple Instance’ pattern, the ‘General Synchronizing Merge’ pattern, and the ‘Cancel

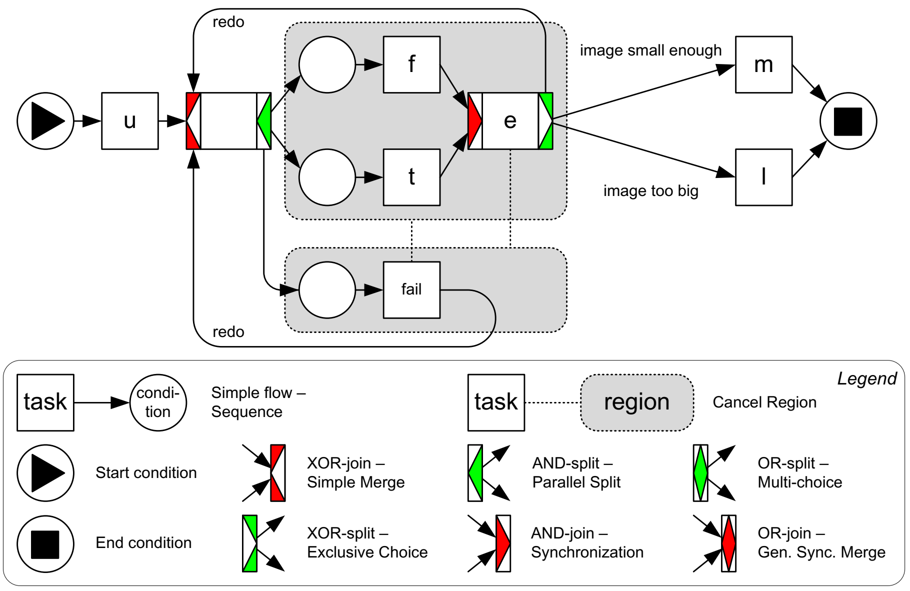


Fig. 7. The example process as a YAWL model

Region' pattern. The only pattern not supported by YAWL is the 'Implicit Termination' pattern (A process implicitly terminates when there is no more work to do and the process is not in a deadlock.), and the authors of YAWL deliberately chose not to support this pattern. Lately, the patterns have been revised and extended [14], and YAWL is being extended to support the new patterns

6.1 Language

In YAWL, two objects are used to model the control-flow aspect of a process: *tasks* and *conditions*. Loosely speaking the former correspond to *activities* (BPMN) and *functions* (EPC), and the latter to *events* (both BPMN and EPC). The BPMN *gateways* and EPC *connectors* are modeled by specifying the *join* and *split* behavior of a task. Like EPCs, YAWL supports AND, XOR, and OR splits and joins. Unlike EPCs, the semantics of the OR-join is well-defined, and an engine exists that supports the execution of any YAWL model. As such, a YAWL model can both act as a conceptual model and an IT model. Figure 7 shows a possible implementation of the example process using YAWL.

6.2 Transformation Challenges

The formalization of YAWL is straightforward, as it has a proper formal semantics. Challenges in YAWL include the OR-join and the cancelation regions. The YAWL OR-join comes with a semantics that includes backwards reasoning and coverability in reset nets, which is impossible to capture in a classical Petri nets. The cancelations regions are hard to capture (though possible) in classical Petri nets, but are straightforward to capture when using reset nets.

6.3 Transformation and Application

YAWL comes with a transformation to reset nets, which is straightforward except for the OR-join [32]. Furthermore, there is also a transformation to workflow nets that *covers* the behavior of the YAWL model [33]: any behavior exhibited by the YAWL model will also be present in the Petri net, but not vice versa. Finally, there is a transformation (see [34]) that is used to obtain a Petri-net-based simulation model (using CPN tools [35]) for an operational YAWL model. To keep things simple for the time being, this transformation assumes that there are no cancelation regions, and that an OR-join is enabled a soon as any of its inputs are enabled.

The transformation to reset nets that comes with YAWL is used by the YAWL engine to check which tasks are enabled [32]. For an AND-join task and an XOR-join task this check is quite simple (a task is enabled if and only if any of the corresponding transitions in the reset net is enabled), but for an OR-join task this check is quite complex and involves a coverability check on any corresponding input place in the reset net that is not marked. As coverability is decidable for reset nets, this procedure is decidable as well.

This transformation is also used to verify YAWL models [36]. In the absence of OR-joins, a YAWL model can be transformed to a reset net, which can (possibly) be verified for soundness. If the reset net is to complicated to be checked successfully, a set of reduction rules is given to simplify the reset net prior to checking soundness [37].

The transformation from [33] is also used to verify YAWL models, but is restricted to relaxed soundness. If the state space is too complex to be constructed, transitions invariants can be used to estimate relaxed soundness. This approach is correct (errors reported are really errors), but not necessarily complete (not every error might get reported).

The other transformation to workflow nets is used to transform an existing YAWL model into a colored Petri net that can be simulated by CPN Tools. If an event log from the given YAWL model is provided during the transformation, then relevant information such as organizational details and performance characteristics are included in the resulting simulation model.

7 Conclusion

Many transformations to Petri nets currently exist, and several of these transformations struggle with concepts that are hard to handle in Petri nets, like OR-joins and exceptions, but other transformations simply can abstract from these concepts, either because the source language does not support the concept as well, or because the application of the transformation allows for the abstraction. For example, YAWL and EPCs do not support exceptions, and some of the YAWL and EPC transformations can abstract from the OR-join because the relaxed soundness property allows this.

Our experience indicates that transforming an informal and complex language like BPEL to a low-level Petri net language is quite difficult without first having formalized the language in a proper way. Many BPEL constructs require a dedicated and possibly complex solution in Petri net terms, and to keep these solutions nicely orthogonal (we do not want one solution to obstruct a second) is not an easy task. Therefore, it seems sensible to:

– first, formalize the language using, for example, a high-level Petri net language, and
– second, transform the high-level formalization to the target low-level Petri net language.

Almost all transformations have been implemented in tools, and most of these tools are included in the ProM framework [38]. For example, the transformations from EPCs has been implemented in regular ProM conversion plug-ins, and the transformations from BPEL have been implemented in tools for which ProM conversion wrapper plug-ins have been implemented. The transformations from YAWL models to reset nets and the transformations from BPMN to workflow nets have been implemented in separate tools and it is expected that these transformations will be included in the ProM framework in the near future.

The applications of the different transformations differ. Several transformations are used to verify the business process at hand, others are also used for the actual execution of the business process (the transformation from YAWL to reset nets is a good example for this). Furthermore, some transformations exist that aim to simplify the source language, examples include the EPC reductions and the well-known Petri-net reduction rules by Murata [7].

Informal languages often describe alternatives and shorthands to represent process parts that have the same (formal) semantics. For the four languages described here this holds only for BPMN. Other OMG standards also make use of alternatives and shorthands. In our experience alternatives and shorthands are most efficiently dealt with by creating a 'normal form' version of the language and defining the mapping for this normal form. Alternatives and shorthands should first be translated to the normal form. They will then be mapped to the formalism of choice automatically.

From the transformation of a very detailed language such as BPEL into a simple formalism like Petri nets, we learned that the applications of techniques well-known in the field of compiler theory greatly systematize and simplify the transformation. In particular, using high-level Petri nets as *intermediate formalism* to explicitly model data aspects yields a better understanding of BPEL. Only when a low-level pattern is actually needed (e.g., for verification), we abstract from data aspects. In addition *static analysis* [39] allows for an improved translation by collecting information on the context of each activity. This information can be used to chose the best fitting pattern (e.g., depending on the presence of handlers or the chosen verification goal) from a pattern repository.

This *flexible model generation* [40] has been shown to yield very compact transformation results, and can be similarly applied to all presented source languages.

References

1. Alves, A., et al.: Web Services Business Process Execution Language Version 2.0. OASIS Standard, April 11, 2007, OASIS (2007)
2. Keller, G., Nüttgens, M., Scheer, A.: Semantische Processmodellierung auf der Grundlage Ereignisgesteuerter Processketten (EPK). Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89 (in German), University of Saarland, Saarbrücken (1992)
3. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: Yet Another Workflow Language. Information Systems 30(4), 245–275 (2005)
4. OMG: Business Process Modeling Notation (BPMN) Version 1.0. OMG Final Adopted Specification, Object Management Group (2006)
5. Baeten, J.C.M., Weijland, W.P.: Process Algebra. Cambridge tracts in theoretical computer science, vol. 18. Cambridge University Press, Cambridge (1990)
6. Reisig, W.: Petri Nets. In: EATCS Monographs on Theoretical Computer Science edn. Springer, Heidelberg (1985)
7. Murata, T.: Petri nets: Properties, analysis and applications. Proc. IEEE 77(4), 541–580 (1989)
8. 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)
9. Dehnert, J.: A Methodology for Workflow Modelling: from Business Process Modelling towards Sound Workflow Specification. PhD thesis, Technische Universität Berlin, Berlin, Germany (2003)
10. Dufourd, C., Finkel, A., Schnoebelen, P.: Reset nets between decidability and undecidability. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 103–115. Springer, Heidelberg (1998)
11. Massuthe, P., Reisig, W., Schmidt, K.: An operating guideline approach to the SOA. Annals of Mathematics, Computing & Teleinformatics 1(3), 35–43 (2005)
12. Wolf, K.: Does my service have partners? Transactions on Petri Nets and Other Models of Concurrency (accepted for publication) (2008)
13. van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow patterns. Distributed and Parallel Databases 14(3), 5–51 (2003)
14. Russell, N., ter Hofstede, A., van der Aalst, W., Mulyar, N.: Workflow control-flow patterns: A revised view. Report BPM-06-22, BPM Center (2006)
15. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Information & Software Technology (accepted for publication) (2008)
16. Wong, P.Y., Gibbons, J.: A Process Semantics for BPMN. In: Liu, S., Maibaum, T., Araki, K. (eds.) ICFEM 2008. LNCS, vol. 5256, pp. 355–374. Springer, Heidelberg (2008), `http://web.comlab.ox.ac.uk/oucl/work/peter.wong/pub/bpmnsem.pdf`
17. Rittgen, P.: Modified EPCs and their Formal Semantics. Technical report 99/19, University of Koblenz-Landau, Koblenz, Germany (1999)
18. van der Aalst, W.M.P., Desel, J., Kindler, E.: On the Semantics of EPCs: A Vicious Circle. In: EPK 2002, Trier, Germany, GI, pp. 71–80 (2002)
19. Kindler, E.: On the semantics of EPCs: A framework for resolving the vicious circle. Data and Knowledge Engineering 56(1), 23–40 (2006)

20. van Dongen, B.F., Jansen-Vullers, M.H., Verbeek, H.M.W., van der Aalst, W.M.P.: Verification of the SAP reference models using EPC reduction, state space analysis, and invariants. Computers in Industry 58(6), 578–601 (2007)
21. Breugel, F.v., Koshkina, M.: Models and verification of BPEL (2006), `http://www.cse.yorku.ca/~franck/research/drafts/tutorial.pdf`
22. Lohmann, N., Verbeek, H.M.W., Ouyang, C., Stahl, C., van der Aalst, W.M.P.: Comparing and evaluating Petri net semantics for BPEL. Computer Science Report 07/23, Eindhoven University of Technology, Eindhoven, The Netherlands (2007)
23. 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)
24. 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)
25. Schmidt, K.: LoLA: A low level analyser. In: Nielsen, M., Simpson, D. (eds.) ICATPN 2000. LNCS, vol. 1825, pp. 465–474. Springer, Heidelberg (2000)
26. Lohmann, N., Kleine, J.: Fully-automatic translation of open workflow net models into simple abstract BPEL processes. In: Modellierung 2008, GI. LNI, vol. P-127, pp. 57–72 (2008)
27. 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)
28. 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)
29. Ouyang, C., van der Aalst, W.M.P., Breutel, S., Dumas, M., ter Hofstede, A.H.M., Verbeek, H.M.W.: WofBPEL: A tool for automated analysis of BPEL processes. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 484–489. Springer, Heidelberg (2005)
30. Verbeek, H.M.W., van der Aalst, W.M.P.: Woflan 2.0: A Petri-net-based workflow diagnosis tool. In: Nielsen, M., Simpson, D. (eds.) ICATPN 2000. LNCS, vol. 1825, pp. 475–484. Springer, Heidelberg (2000)
31. Verbeek, H.M.W., Basten, T., van der Aalst, W.M.P.: Diagnozing workflow processes using woflan. The Computer Journal 44(4), 246–279 (2001)
32. 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)
33. Verbeek, H.M.W., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Verifying workflows with cancellation regions and OR-joins: An approach based on relaxed soundness and invariants. The Computer Journal 50(3), 294–314 (2007)
34. Rozinat, A., Wynn, M., van der Aalst, W.M.P., ter Hofstede, A.H.M., Fidge, C.: Workflow simulation for operational decision support using yawl and prom. BPM Center Report BPM-08-04, BPMcenter.org (2008)
35. CPN Group, University of Aarhus, Denmark: CPN Tools Home Page, `http://wiki.daimi.au.dk/cpntools/`

36. Wynn, M.T., van der Aalst, W.M.P., ter Hofstede, A.H.M., Edmond, D.: Verifying workflows with cancellation regions and OR-joins: An approach based on reset nets and reachability analysis. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 389–394. Springer, Heidelberg (2006)
37. Wynn, M.T., Verbeek, H.M.W., van der Aalst, W.M.P., ter Hofstede, A.H.M., Edmond, D.: Reduction rules for reset workflow nets. BPM Center Report BPM-06-25, BPMcenter.org (2006)
38. van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H.M.W., Weijters, A.J.M.M., van der Aalst, W.M.P.: The ProM framework: A new era in process mining tool support. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 444–454. Springer, Heidelberg (2005)
39. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. Springer, Heidelberg (1999)
40. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing interacting WS-BPEL processes using flexible model generation. Data Knowl. Eng. 64(1), 38–54 (2008)

A Look Around the Corner: The Pi-Calculus

Frank Puhlmann and Mathias Weske

Business Process Technology Group
Hasso Plattner Institute at the University of Potsdam
14482 Potsdam, Germany

Abstract. While Petri nets play a leading role as a formal foundation for business process management (BPM), other formalizations have been explored as well. This chapter introduces the π-calculus as a formal foundation for BPM. The approach presented is pattern-centric, thus allowing for direct comparisons between the π-calculus and different formalizations. In particular, selected basic and advanced control flow patterns as well as service interaction patterns are discussed. The chapter furthermore introduces the application of bisimulation techniques for proving soundness properties of business processes.

1 Introduction

This chapter introduces the formal specification and analysis of business processes using a process algebraic approach. Just as Petri nets (e.g. [15,11]), process algebra (e.g. [10,12,4]) can be defined in terms of transition system. In both approaches, states—and transitions between them—are in the center of attraction. Process algebra is based on formal terms, whereas Petri nets are based on bipartite graph structures. The combinations of process algebra and Petri nets in the area of business process management (BPM) [2] have already been investigated, for instance by Basten in [6].

By the end of the 20th century, however, new kinds of process algebra have been developed, focusing on mobile systems [14]. These new kinds of process algebra allow the representation of processes with dynamic structures, which is in contrast to Petri nets, that capture processes with static structures. While both approaches suffer from the inherent problem of transition systems, i.e. state space explosion, mobile process algebra brings a new point of view into play. This new point of view allows a direct representation of *dynamic binding*, as found in today's BPM systems realized as service-oriented architectures (SOA) [7]. Dynamic binding is represented via a technique called *link passing mobility*, that directly resembles the evolution of a system with dynamic structures. Consider for instance figure 1, that shows the classical SOA triangle in a different notation. In the beginning, a service broker has knowledge about a number of service providers. Also, the service requester has knowledge about the service broker. What is wanted, however, is a direct connection between the service requester and one single service provider. While in theory there could be static connections to all service providers, such as a classic Petri net representation would require,

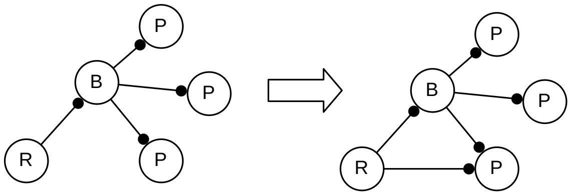


Fig. 1. Dynamic binding in service-oriented architectures

link passing mobility directly captures the initial idea. The following sections introduce the reader into a particular process algebra, the π-calculus [13], that handles link passing mobility as a first class citizen. The introduction is strongly focused on the area of business process modeling and specification. In particular, we shortly introduce the π-calculus as the process algebra of choice in section 2. The discussion of fundamental control flow and interaction patterns is given in section 3 and 4. Section 5 introduces a basic theory of reasoning in the process algebraic world—bisimulation—by discussing simple kinds of soundness.

2 The Pi-Calculus

The representation of business processes can be seen from different perspectives, such as the functional viewpoint that focuses on the activities of a process, the behavioral perspective that defines the ordering of the activities, the organizational perspective that specifies the required roles, and so on [8]. In this paper, we focus on the behavioral perspective.

2.1 Syntax and Semantics

This chapter discusses the representation of the behavioral perspective using the π-calculus. The π-calculus is a process algebra, that consists of names—representing the unification of channels and data—and agents that interact on names.

Definition 1 (Pi-Calculus). *The agents of the π-calculus are given by:*

$$\begin{aligned} P &::= M \mid P|P \mid \nu z\ P \mid A(x_1,\ldots,x_n) \\ M &::= \mathbf{0} \mid \pi.P \mid M+M\ . \\ \pi &::= \overline{x}\langle\tilde{y}\rangle \mid x(\tilde{z}) \mid \tau \mid [x=y]\pi\ . \end{aligned}$$

The informal semantics is as follows: $P|P$ is the concurrent execution of P and P, $\nu z\ P$ is the restriction of the scope of the name z to P, i.e. z is only visible in P and distinct from all other names, and $A(y_1,\cdots,y_n)$ denotes parametric recursion over the set of agent identifiers. $\mathbf{0}$ is inaction, a process that can do nothing, and $M+M$ is the exclusive choice between M and M. The prefixes of the calculus are given by π. The output prefix $\overline{x}\langle\tilde{y}\rangle.P$ sends a tuple of names

$\tilde{y}$ via the co-name $\overline{x}$ and then continues as P. The input prefix $x(\tilde{z})$ receives a tuple of names via the name x and then continues as P with $\tilde{z}$ replaced by the received names. Matching input and output prefixes of different components might interact, leading to an unobservable action (τ), e.g. $\overline{a}.A|a.B \xrightarrow{\tau} A|B$. An explicit representation of an unobservable action is given by the prefix $\tau.P$ and the match prefix $[x = y]\pi.P$ behaves as $\pi.P$ if x is equal to y.

The actions of the agents are given by the input, output, and unobservable prefixes, denoted as $Act = \{\overline{x}\langle\tilde{y}\rangle, x(\tilde{z}), \tau\}$. The evolution of the state of an agent to a succeeding state is denoted by a transition bearing the corresponding action, i.e. $\overline{a}\langle w\rangle.A \xrightarrow{\overline{a}\langle w\rangle} A$, where we assume an interleaved semantics.

Throughout this chapter, upper case letters are used for agent identifiers and lower case letters for names. We abbreviate a set of components as $\prod_{i=1}^{n} Pi$, i.e. $\prod_{i=1}^{3} Pi = P_1 \mid P_2 \mid P_3$. We also omit the trailing $\mathbf{0}$ and omit the enumeration of free names at agent identifiers if obvious. A detailed discussion on the semantics applied in this chapter can be found in [16].

2.2 Bisimulation

An interesting property of two different agent terms is their behavioral equivalence. Process algebra use the notion of bisimilarity or bisimulation equivalence for this purpose. It denotes that two agents are able to mimic all their actions in arbitrary directions, i.e. the first agent does an action, the second agent mimics it, and thereafter the second agent does another action that the first agent needs to mimic, etc. If there exists a bisimulation between two agents A and B, i.e. they are bisimilar, we denote this as $A \sim B$.

Of particular interest is the weak variant, i.e. weak bisimulation, that ignores unobservable τ-actions. Hence, internal steps that are explicitly denoted as τ as well as internal communication between components (also leading to a τ-transition) are not included. This yields an equivalence criteria, denoted as $A \approx B$, that only covers the external observable behavior of two agents. Formal details are discussed for instance in [13].

3 Workflow Patterns

The research on the different perspectives of business process is strongly driven by patterns, as for instance given by *Workflow Patterns* [1], *Data Patterns* [19], *Resource Patterns* [20] or *Service Interaction Patterns* [5]. We use selected Workflow and Service Interaction Patterns to describe the fundamentals for formally specifying interacting business processes in the π-calculus.[1]

From an abstract point of view, each business process can be seen as a directed graph with nodes and edges. Additionally, types and properties need to be added to the different nodes. While in our earlier work we referred to such structures

[1] The whole set of formalized Data, Workflow, and Service Interaction Patterns can be found in [16].

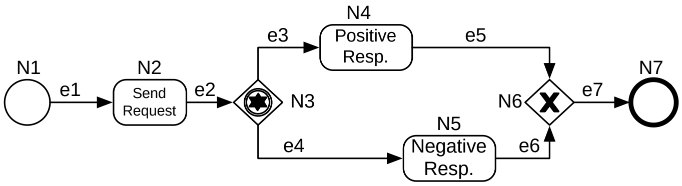


Fig. 2. Business process with enumerated elements in BPMN

as *process graphs* [18,17], we rather focus on the patterns themselves in this chapter. We provide a methodology for gluing the patterns together as required.

Basically, a business process in π-calculus is represented as an agent consisting of a set of restricted names and a number of parallel components:

$$N \overset{def}{=} (\nu e1 \ldots)(\prod Ni) \ . \qquad (1)$$

Each restricted name resembles an edge of a business process. Each component resembles an arbitrary node. The restricted names and identifiers for the components can be directly derived by labeling the elements of the graphical representation of the business process, as shown in figure 2. We stick to the *Business Process Modeling Notation (BPMN)* for illustration purposes. The approach, however, is not fixed to a certain notation. Any graph-based notation, including *UML Activity Diagrams* or *Event-driven Process Chains*, are applicable.

The components that represent the nodes of a business process are then refined according to different patterns as shown in figure 3. The names $e1, e2, \ldots$ are renamed according to the label of the corresponding edge. The identifiers of the components are also renamed according to the label of the corresponding node. The following subsections discuss the different patterns and provide an illustrating example.

3.1 Sequence Pattern

A basic pattern found in business processes is the sequential execution of activities, as described by the *Sequence Pattern.* An activity N is enabled, once the preceding activity has finished execution. The π-calculus representation of the node N is shown in figure 2. N waits for the preceding activity to finish, as denoted by the occurrence of the name $e1$, that represents the incoming edge. Afterwards, the functional part is executed—abstracted via τ. Finally, the name $e2$ is emitted to signal the successful execution of N. At the same time, via the parallel operator, the remainder of the term N resets itself via recursion. This recursive definition is required for loops, where multiple instances of an activity are required.

According to the pattern description, the node $N2$ of the business process shown in figure 2 is formalized as follows, where the agent $N2$ defines the semantics of the corresponding node $N2$.

$$N2 \overset{def}{=} e1.\tau.(\overline{e2} \mid N2) \ .$$

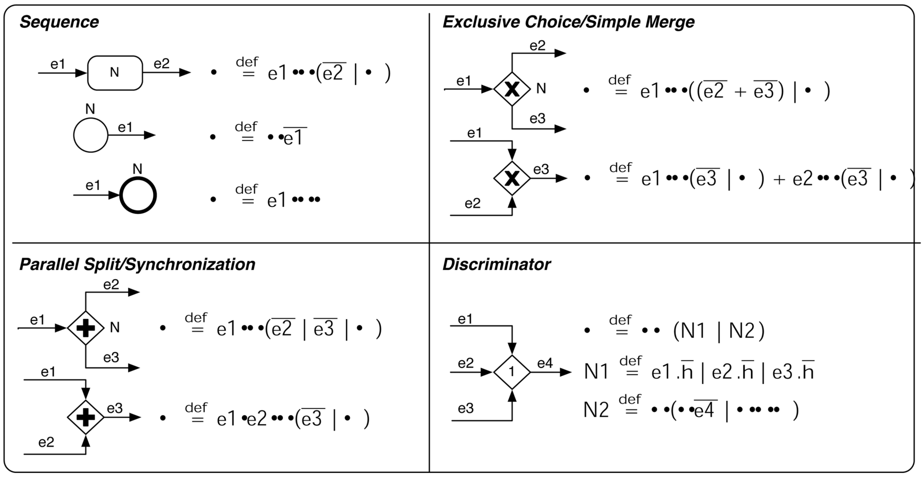

Sequence
e1
N
e2
N
e1
N
e1
Exclusive Choice/Simple Merge
e2
e1
N
e3
e1
e3
e2
Parallel Split/Synchronization
e2
e1
N
e3
e1
e3
e2
Discriminator
e1
e2
1
e4
e3
N1 def= e1.h̄ | e2.h̄ | e3.h̄
N2 def=

Fig. 3. Fundamental workflow patterns

Please note that *N2* contains a recursive definition of itself at the right hand side, allowing multiple instances to be created.

The initial and the final nodes of a business process can be seen as variants of the Sequence Pattern. An initial node–i.e. a node with only an outgoing edge—has no precondition for execution. It can simply start and emit its successful execution via the name *e1* afterwards. Since the initial node cannot be part of a loop, it does not use recursion to reset itself. An according formalization for the semantics of node *N1* of figure 2 is given by:

$$N1 \overset{def}{=} \tau.\overline{e1} \ .$$

The final node of a business process is represented similar to the initial node, with the distinction that recursion is required, since the final node might be reached multiple times. An example is given by the formalization of node *N7*:

$$N7 \overset{def}{=} e7.\tau.N7 \ .$$

3.2 Exclusive Choice/Simple Merge Pattern

Another common pattern is the exclusive choice between different paths of execution—and afterwards joining them together. The associated patterns are documented as *Exclusive Choice* and *Simple Merge*. The former pattern denotes that a node N makes an exclusive choice between a number of directly following nodes, whereas the latter enables a node N each time a directly preceding node is activated.

The *Exclusive Choice Pattern* makes use of the π-calculus summation operator to denote the exclusive choice between subsequent nodes. Otherwise, it resembles the *Sequence Pattern*. The *Simple Merge Pattern* waits for one of the preceding activities to finish. Once one of them finishes execution, it behaves just like the *Sequence Pattern.* Consider for instance the formalization of the node *N6* of the example shown in figure 2:

$$N6 \stackrel{def}{=} e5.\tau.(\overline{e7} \mid N6) + e6.\tau.(\overline{e7} \mid N6)\ .$$

The formalization of the *Simple Merge Pattern* furthermore resembles the semantics of the *Multiple Merge Pattern*, since the subsequent node is activated for each completion of a preceding node. Since both patterns only differ in their assumptions (single execution vs. multiple executions), this overlapping does not lead to problems. Instead, it provides the behavior required for looping parts of the process model.

3.3 Parallel Split/Synchronization Pattern

Besides choosing a certain path for execution, a node can activate all of its direct successors or wait until all of the directly preceding nodes have finished execution. The corresponding patterns are known as *Parallel Split* and *Synchronization.* The formalization of the former pattern is similar to the *Exclusive Choice* pattern, with the difference that the names *e2* and *e3* are emitted in parallel. The latter pattern—*Synchronization*—waits for the names *e1* and *e2* in a sequential order. This order captures the intended semantics accurately, since an interaction via *e2* can only occur after an interaction via *e1*.

3.4 Discriminator Pattern

The final *Workflow Pattern* that we want to discuss in this chapter is the *Discriminator.* A node with discriminator semantics waits for the completion of all its directly preceding nodes, where it is assumed that all of them are activated once. Once one of them finishes, the subsequent node is activated. The discriminator node, however, resets itself only after all of the other directly preceding nodes have finished execution.

The corresponding formalization in the π-calculus is achieved by mapping the names *e1*, *e2*, and *e3*—that denote the incoming edges from the preceding activities—to a restricted name h. The restricted name is only visible inside the agent terms of the discriminator node. It enables an interaction between the components of *N1* and *N2* each time an interaction on the names representing the incoming edges occurs. The component *N2* formalizes the intended semantics. After an interaction via the restricted name h, two components are enabled. The left-hand component enables a τ-action, that is followed by an emission of the name *e4*. The right-hand component collects the remaining interactions via h, before the terms are reset via recursion.

4 Interaction Patterns

After we discussed the basics of representing internal business processes in the π-calculus, we move on to interactions among them. Basically, an interaction

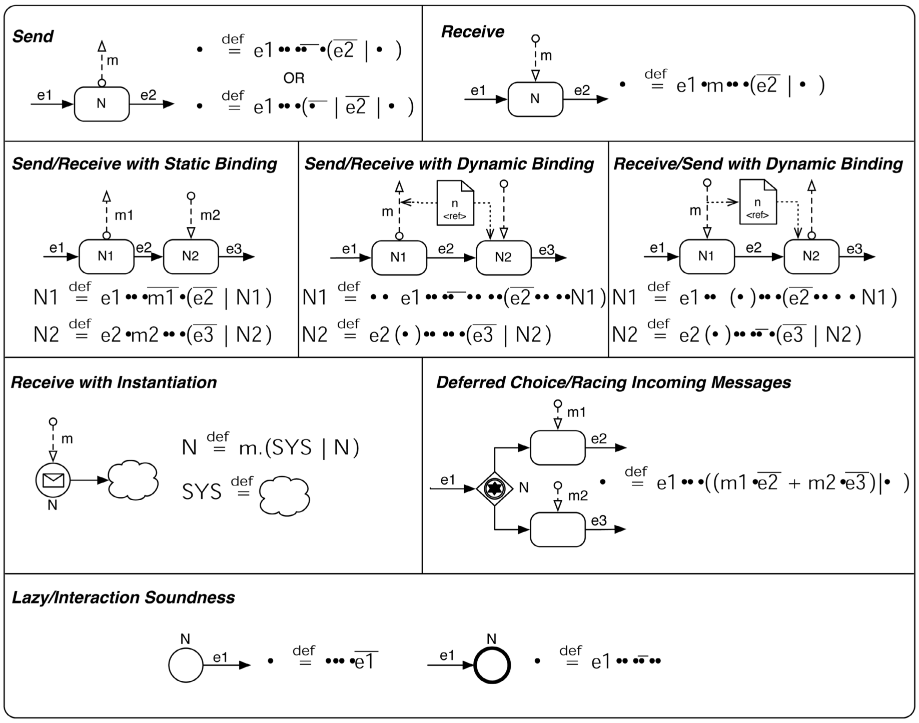

Send
m
e1
N
e2
OR
Receive
Send/Receive with Static Binding
m1
m2
N1
N2
e3
Send/Receive with Dynamic Binding
n
<ref>
Receive/Send with Dynamic Binding
Receive with Instantiation
N ≝ m.(SYS | N)
SYS ≝
Deferred Choice/Racing Incoming Messages
Lazy/Interaction Soundness

Fig. 4. Fundamental interaction patterns

consists of different business processes, that are represented by their according formalizations as described. The formalizations, however, need to be modified to capture the sending and reception of messages among the participants of the choreography. The discussion of interactions is based on two key concepts, namely *correlations* and *dynamic binding.*

Correlations between different business processes are required if more than one instance of a business process is required. A typical example would be a service—such as a stock broker—that starts a new instance each time a request arrives. The correlations between the different instances of the service and the requestor need to be captured formally. Besides capturing the correlations, also the creation of new instances needs to be specified.

Dynamic binding between interacting business processes is for instance required to correctly route the (response) messages of business processes acting as services. Thus, it provides a mechanism for supporting correlations via *callbacks.* Another use of dynamic binding is acquiring interaction partners at runtime via service brokers.

The π-calculus supports the direct representation of simple correlations and dynamic binding via the use of restricted names. A restricted name created inside an agent formalization of a business process can be exposed to the outside and

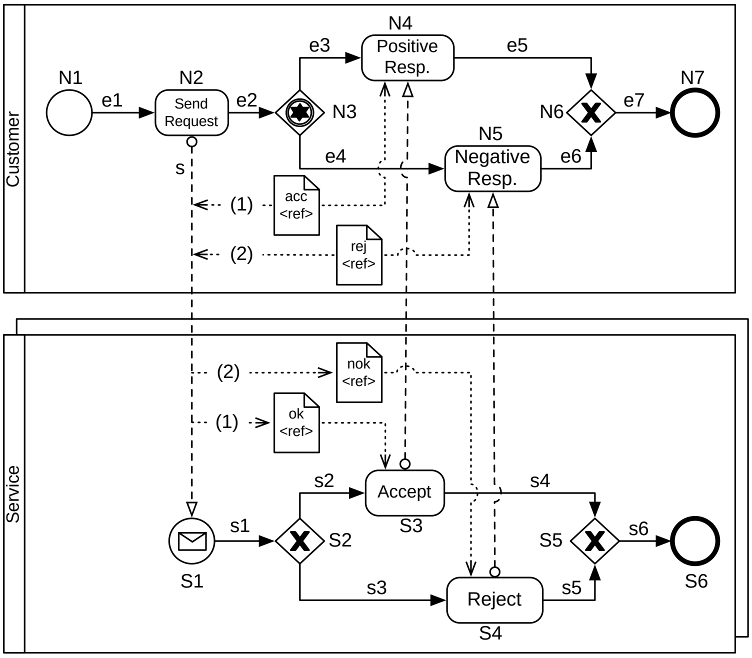


Fig. 5. Interacting processes in extended BPMN [9]

used as an exclusive response channel with built-in correlation. Furthermore, π-calculus names can be passed around using link passing mobility, providing the foundations for dynamic binding. A potential service could create a fresh, restricted name and pass it on to a service broker. The service broker, in turn, can forward it to potential requesters of the service.

An interaction between different business processes is given by a set $\mathcal{BP}$ of formalized business processes according to equation 1. The members of the set represent components of an agent wrapping them together. The *wrapping* agent furthermore restricts all free names of the members of the set (given by $\epsilon = \text{fn}(x)|x \in \mathcal{BP}$), thus creating an isolated system:

$$I \overset{def}{=} \nu\epsilon \prod \mathcal{BP} \ . \tag{2}$$

As done before, we refer to typical patterns between interacting business processes, shown in figure 4. An example is shown in figure 5. It shows two interacting business processes using an extended version of BPMN [9]. The extensions allow the representation of correlations and dynamic binding in BPMN. The upper pool of the figure is the same as discussed in section 3. The lower pool introduces a service, that complements the process of the upper pool. Notable, the pool is shown with a shadow, denoted that multiple instances could exist. While the example only shows one customer, there could be more without changing the service's pool. The correlations between the customer and the service are shown using associated references. Their semantics as well as their formalization in the π-calculus will be discussed in the next subsections.

4.1 Send Pattern

One of the two fundamental interaction patterns is the *Send Pattern*. The *Send Pattern* has different variants, e.g. synchronous vs. asynchronous, dynamic or static binding of the receiver, error handling, etc. We discuss dynamic and static binding in subsequent patterns and omit error handling in this chapter. Regarding synchronous and asynchronous messaging, the π-calculus formalization is able to support both.

Figure 4 contains the π-calculus formalization for nodes of a process that send messages. The upper formalization represents synchronous messaging. The message, m, has to be sent before the agent term continues. The lower formalization, in contrast, shown asynchronous messaging. The message is provided anytime after the functional part of the node—represented by τ—has finished. In the following, we consider only synchronous messaging. It can be adapted to asynchronous as required.

4.2 Receive Pattern

The other fundamental interaction pattern is the *Receive Pattern*. In the common case, it represents the blocking reception of a message. If a message should be received in parallel to other activities, a parallel path should be introduced in the corresponding business process. There are, however, exceptions such as the *Event-based Rerouting Pattern* [16]. In this chapter, we stick to the common case.

The formalization of the receive pattern that waits on a static, pre-defined channel is straightforward. It simply waits for an additional name—the message channel m—before enabling the τ-action.

4.3 Send/Receive with Static Binding Pattern

Using the *Send* and *Receive Patterns* as core building blocks, more elaborate interaction behavior can be constructed. A typical construct is a *Send/Receive Pattern with Static Binding*. In this case, the outgoing as well as the incoming channel are pre-defined. The sending and the receiving node are connected via the *Sequence Pattern*.

4.4 Send/Receive with Dynamic Binding Pattern

A more elaborate pattern is given given by *Send/Receive with Dynamic Binding*. In this case, a unique channel—represented by a fresh, restricted π-calculus name—is sent via an already known message channel. The sent channel is then used later on in the process as a response channel. Figure 4 shows a graphical depiction of this pattern. The data object shown represents a *channel reference* and the directed associations leaving it represent the points of use. The first association points to the outgoing message flow, meaning that the reference is sent via this channel. The second association points to activity *N2*, meaning that the reference represents the response channel. An extended discussion of these extension can be found in [9].

The formalizations of the patterns make use of two distinguishing features of the π-calculus: the creation of fresh, restricted names as well as link passing. In agent *N1*, a fresh, restricted name n is created first. This name is sent via the previous known name m. Afterwards, it is forwarded via *e2* to agent *N2*. In agent *N2*, it is used as a response channel.

By applying the techniques described—restriction of names and link passing—a correlation between the sending and the receiving nodes is given, regardless of environments that process the request. A possible environment needs to apply the *Receive/Send with Dynamic Binding Pattern.* This pattern first receives a name n to be used as a reference via m in agent *N1*. The name is forwarded via *e2* to agent *N2*, where it is used as a response channel.

An application of this pattern is shown in figure 5. The customer creates two references, *acc* and *rej*, that should be used as response channels for the service invoked via the static channel s. Furthermore, the references are sent in order, denoted by the numbers at the corresponding associations. First, the *acc* reference is sent, followed by the *rej* reference. Those references are received by the service, where they are stored in the placeholder references *ok* and *nok*. These placeholder references can be seen as variables, that are filled with actual values during the execution of the process. The references are then used to correlate the nodes *S3* and *N4* as well as *S4* and *N5*. We give a formalization of the node *N2*:

$$N2 \overset{def}{=} \nu acc, rej\ e1.\tau.\overline{s}\langle acc, rej\rangle.(\overline{e2}\langle acc, rej\rangle \mid N2)\ .$$

Node *N2* creates the fresh, restricted names *acc* and *rej* and transmits them via s. Afterwards, it forwards the restricted names via *e2* using link passing mobility. A corresponding formalization of the node *S3* is given by:

$$S3 \overset{def}{=} s2(ok).\tau.\overline{ok}.(\overline{s4} \mid S3)\ .$$

Node *S3* first receives the response channel (here denoted with the placeholder *ok* that will be replaced with *acc* in the example). Afterwards, it emits via *ok*. The reception of the initial message in *S1* as well as the deferred choice represented by the nodes *N3*, *N4*, and *N5*, are represented by different patterns that will be discussed below.

4.5 Receive with Instantiation

A common pattern between interacting business processes is the creation of new instances based on the arrival of a message, denoted as *Receive with Instantiation Pattern.* The representation of this pattern in the π-calculus is also shown in figure 4. This pattern is required to encapsulate a business process as a service, since different instances might exist in parallel.

Technically, the arrival of a message via channel m triggers the contained business process (represented by the cloud). In parallel, the agent N resets itself via recursion. Since a formalized business process represented by SYS has its own set of restricted names for internal routing, no interference can occur.

We give an example for the node $S1$ of figure 5:

$$SERVICE \overset{def}{=} s(ok, nok).(SYS \mid SERVICE) \text{ and } SYS \overset{def}{=} (\nu s1 \ldots s6\ )(\prod_{i=1}^{6} Si)$$

with

$$S1 \overset{def}{=} \tau.\overline{s1}\langle ok, nok\rangle\ .$$

The agent SYS is defined according to equation 1. The agent $S1$ forwards the names *ok* and *nok* that have been received in N.

4.6 Deferred Choice/Racing Incoming Messages Pattern

A last pattern that is typically required in interacting business processes is the *Deferred Choice/Racing Incoming Messages Pattern.* While the *Deferred Choice* is a *Workflow Pattern*, the *Racing Incoming Messages* is a *Service Interaction Pattern.* The former pattern is more generic, since it states that a choice is made on external events. These events are specified in the latter pattern, relating them to incoming messages.

Depending on the graphical notation used, the *Deferred Choice/Racing Incoming Messages Pattern* might be split across multiple nodes of a business process. In case of BPMN, the pattern is spread across at least three nodes—an *Event-based Gateway* and two *Receive Tasks*—, as shown in figure 4. The different nodes are formally specified in one agent term N. The formalization is similar to the *Exclusive Choice Pattern*, with the distinction that additional names, representing the message channels, are required.

We give an example for the nodes *N3*, *N4*, and *N5* of figure 5, that together resemble a *Deferred Choice/Racing Incoming Messages Pattern*:

$$N3 \overset{def}{=} e2(acc, rej).\tau.((acc.\overline{e5} + rej.\overline{e6}) \mid N3)\ .$$

When this pattern is applied, the nodes directly following the *Event-based Gateway* are left out in the definition of the process according to equation 1. Hence, the formal specification of the *Customer* from figure 5 is given by:

$$CUSTOMER \overset{def}{=} (\nu e1, e2, e5, e6, e7\ )(\prod_{i\in\{1,2,3,6,7\}} Ni)\ .$$

The complete specification of the interaction between the *Customer* and the *Service* can be constructed according to equation 2:

$$I \overset{def}{=} (\nu s\ )(CUSTOMER \mid SERVICE)\ .$$

The only free name according to *CUSTOMER* and *SERVICE* is s, that is used for initial interaction with the service. The specified system is robust, meaning that an arbitrary number of *CUSTOMER* agents might be added without interfering expected operation of the system.

5 Analysis

A typical use of formal specifications of business processes and interactions among them is a precise description of the behavior. This description can be used as a foundation for discussing and implementing the processes and interactions. Yet another important use case is the possibility of formally analyzing the specified behavior. In particular, we are able to prove invariants on the formalized business processes and interactions. Technically, these invariants are proved using bisimulation techniques for the π-calculus. We stick to basic techniques in this chapter and provide a more elaborate discussion in [17].

5.1 Structural Soundness

A first requirement for proving invariants on business processes is given by *Structural Soundness.* It can be defined informally as:

> A graph representing a business process is structural sound if it has exactly one initial node, exactly one final node, and all other nodes lie on a path between the initial and the final node.

A business process needs to be structural sound for further analysis. The property is derived from the definition of a *Workflow Net* [3].

5.2 Lazy Soundness

A business process can be analyzed by observing its behavior. The observation of the initial and the final node are of particular interest, since they document the begin and the end of a process instance. If the result is provided in the final node, *Lazy Soundness* can informally be defined as follows:

> A structural sound graph representing a business process is lazy sound if in any case a result is provided exactly once.

To prove *Lazy Soundness* formally, we need to be able to observe the initial and the final node of a business process. This is done by mapping the initial and the final nodes of the business process according to *Lazy/Interaction Soundness* of figure 4. Due to the addition of the free names i and o, we are able to observe the occurrence of the corresponding nodes. Regarding the *CUSTOMER* agent, the nodes *N1* and *N7* need to be specified as follows:

$$N1 \overset{def}{=} i.\tau.\overline{e1} \text{ and } N7 \overset{def}{=} e7.\tau.\overline{o}.N \ .$$

An agent that has been enhanced with the free names i and o can be compared with another agent representing the invariant via weak bisimulation. The wanted behavior is given by an agent

$$S_{LAZY} \overset{def}{=} i.\overline{o} \,.$$

This agent behaves as follows. First, it is able to interact via i once. Afterwards, it is able to emit via o once. If a more complex agent—representing a business process—has the same observable behavior according to weak bisimulation, then it fulfills the same invariant. Hence, after each occurrence of the initial node (enhanced with i), the final node (enhanced with o) of the complex agent—representing the business process—is observed.

Lazy Soundness is formally defined as follows:

Definition 2 (Lazy Sound). *A structural sound business process mapped to a π-calculus agent D according to the patterns shown in figure 4 is lazy sound, if $D \approx S_{LAZY}$ holds.*

The business process of figure 2 is lazy sound, if we assume the *Event-based Gateway* to be formalized using the *Exclusive Choice Pattern.* If we want to consider the interactions with the environment—and hence apply the *Deferred Choice/Racing Incoming Message Pattern*—we need to consider a possible environment, such as given by the *Service.*

5.3 Interaction Soundness

An invariant known as *Interaction Soundness* is derived by mapping a certain process of an interaction according to the *Lazy/Interaction Soundness Pattern.* When constructing the complete system I according to equation 2, the names i and o must not be contained in ϵ: $\epsilon \cap \{i, o\} = \emptyset$. Interaction soundness can be defined informally as:

> A structural sound process graph G representing a business process is interaction sound with respect to an environment E, if G is lazy sound inside the composition of E and G.

The system I that is composed out of the agents *CUSTOMER* and *SERVICE* is interaction sound according to the viewpoint of the *Customer.* This property also holds for a certain *Customer*, if multiple *Customers* are engaged in the interaction.

6 Conclusions

This chapter introduced the basic theory behind business process process management based on the π-calculus. We introduced the essentials of our earlier work in this area, providing an overview of the key concepts. The difference is given by the style of presentation. We focused on a pattern based approach, where we assume the knowledgeable reader to be able to arrange the different patterns in a meaningful manner. An extended discussion, as stated, can be found in [16].

The presented techniques might be compared with existing Petri net-based approaches. This comparison, however, can only be seen from an angle of expressiveness focusing on the concepts that are most important for a specific modeling purpose. Since the π-calculus is Turing-complete—such as many extended Petri net variants—, their formal modeling power is equal. What is different, however, is the mindset behind. Petri nets have a strong focus on the description of behavior in static systems. Correlations and dynamic binding can be introduced with extensions. The main point, however, is the flow of tokens from place to place by the firing of transitions. The π-calculus is an algebra, meaning that the current state of the system is represented as an algebraic term. Using several laws, the state can be transformed to a succeeding one. The approach is similar to a term rewriting system. And since the terms are rewritten with each transition, the structure represented by them is changed as well. Thus, the π-calculus describes behavior in dynamic systems.

Another difference of the π-calculus—according to Petri nets—is the possibility of describing interfaces with dynamic behavior, such as found in service-oriented architectures. Reconsider for instance the *Service* from section 4. The underlying formal specification is able to open new communication ports based on received names. Hence, the behavior of the interface is changing. Initially, only an interaction via a static channel is possible. Afterwards, several channels are added for each participating *Customer*.

Besides the positive aspects of the π-calculus, also its drawbacks in the area of BPM have to be considered. First of all, interactions between names always occur between two components. It is not possible to receive names from different components within the same interaction. For some settings, this might be a rather strong restriction, since atomic actions cannot be represented. Furthermore—such as in all state-space-relying approaches—the effort in deciding bisimulation based on π-calculus terms is high, so that restrictions need to be established.

References

1. 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)
2. 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)
3. van der Aalst, W., van Hee, K.: Workflow Management. MIT Press, Cambridge (2002)
4. Baeten, J.C.M., Weijland, W.P.: Process Algebra. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge (1990)
5. Barros, A., Dumas, M., ter Hofstede, A.H.M.: Service Interaction Patterns. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 302–318. Springer, Heidelberg (2005)

6. Basten, T.: In: Terms of Nets: System Design with Petri Nets and Process Algebra. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands (1998)
7. Burbeck, S.: The Tao of E-Business Services (2000)
8. Curtis, B., Kellner, M.I., Over, J.: Process Modeling. Communications of the ACM 35(9), 75–90 (1992)
9. Decker, G., Puhlmann, F.: Extending BPMN for Modeling Complex Choreographies. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 24–40. Springer, Heidelberg (2007)
10. Hoare, C.A.R.: Communicating Sequential Processes. Communications of the ACM 21(8), 666–677 (1978)
11. Jensen, K.: Coloured Petri Nets, 2nd edn. Springer, Berlin (1997)
12. Milner, R.: A Calculus of Communicating Systems. In: Jones, N.D. (ed.) Semantics-Directed Compiler Generation. LNCS, vol. 94. Springer, Heidelberg (1980)
13. Milner, R., Parrow, J., Walker, D.: A Calculus of Mobile Processes, Part I/II. Information and Computation 100, 1–77 (1992)
14. Nestmann, U.: Welcome to the Jungle: A Subjective Guide to Mobile Process Calculi. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 52–63. Springer, Heidelberg (2006)
15. Petri, C.A.: Kommunikation mit Automaten. PhD thesis, Institut für Instrumentelle Mathematik, Bonn (1962)
16. Puhlmann, F.: On the Application of a Theory for Mobile Systems to Business Process Management. Doctoral thesis, University of Potsdam, Potsdam, Germany (July 2007)
17. Puhlmann, F.: Soundness Verification of Business Processes Specified in the Pi-Calculus. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I, vol. 4803, pp. 6–23. Springer, Heidelberg (2007)
18. Puhlmann, F., Weske, M.: Investigations on Soundness Regarding Lazy Activities. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 145–160. Springer, Heidelberg (2006)
19. Russell, N., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Workflow Data Patterns. QUT Technical Report FIT-TR-2004-01, Queensland University of Technology, Brisbane (2004)
20. Russell, N., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Workflow Resource Patterns. BETA Working Paper Series WP 127, Eindhoven University of Technology, Eindhoven (2004)

*new*YAWL: Towards Workflow 2.0*

Nick Russell[1] and Arthur H.M. ter Hofstede[2]

[1]Eindhoven University of Technology,
PO Box 513, 5600MB, Eindhoven, The Netherlands
`n.c.russell@tue.nl`

[2]Queensland University of Technology,
PO Box 2434, QLD, 4001, Australia
`a.terhofstede@qut.edu.au`

Abstract. The field of workflow technology has burgeoned in recent years providing a variety of means of automating business processes. It is a great source of opportunity for organisations seeking to streamline and optimise their operations. Despite these advantages however, the current generation of workflow technologies are subject to a variety of criticisms, in terms of their restricted view of what comprises a business process, their imprecise definition and their general inflexibility. As a remedy to these potential difficulties, in this paper we propose a series of development goals for the next generation of workflow technology. We also present *new*YAWL, a formally defined, multi-perspective reference language for workflow systems.

1 Introduction

Workflow management technology provides support for the execution of business processes. A workflow management system routes work to the right people or software applications at the right time. While these types of systems have been around in one incarnation or another for several decades (e.g. office automation systems originated in the seventies [26,11]), it wasn't until fairly recently that they reached a level of maturity where their broader uptake was feasible. Workflow technology allows businesses to save time and money by providing them with a means of taking charge of their processes. Not only does it support the execution of business processes, but they can also be more easily analyzed, monitored, audited, and adapted. Increasingly now, the term Business Process Management (BPM) is used to indicate that the field has moved beyond mere process specification and execution to encompass a holistic view of the business process as a corporate asset that merits ongoing maintenance and refinement.

* This research was conducted in the context of the *Patterns for Process-Aware Information Systems (P4PAIS)* project which is supported by the Netherlands Organisation for Scientific Research (NWO).

A business process can be viewed from a number of different viewpoints (see e.g. [15]). The *control-flow* perspective deals with the control-flow dependencies that exist between the various tasks in a business process, the *data* perspective deals with the data required by and produced by these tasks, while the *resource* perspective deals with the way in which tasks are allocated to resources. Starting in the late 1990's, the Workflow Patterns Initiative[1] began cataloging patterns in these perspectives. These patterns identify recurrent concepts that arise when modelling business processes, and provide assistance with tool selection, process specification and language development. Over time, process modelling languages, workflow management systems, research prototypes, and (proposed) standards in the BPM field have been examined in terms of these patterns, revealing their relative strengths, weaknesses and opportunities for improvement.

Despite the rapid advances in BPM technology, there have been significant obstacles to full realisation of the benefits that it promises to deliver. To some degree, these obstacles can be explained through the lack of consensus in regard to the conceptual, formal, and technological foundations of BPM. Moreover none of the standards that have been proposed have achieved any degree of uptake. As an example, consider the Workflow Management's XPDL 1.0, introduced in the 1990's. Not only does it lack a formal foundation, allowing it to be interpreted in substantially different ways (cf. [17]), but it also has a very limited range of functionality as demonstrated by its minimalistic support for the workflow patterns (cf. [1]). More recent standards proposals also lack a formal foundation (cf. BPEL, BPMN) and while their overall capabilities have noticeably improved (e.g. as demonstrated by the extent of their workflow pattern support), they still exhibit minimal support for the resource perspective [21]. It is also worth remarking that the operation of a number of commonly utilised workflow constructs are actually quite complex to capture precisely (an interesting illustration of this is the concept of the OR-join in the control-flow perspective to which whole publications have been dedicated, see e.g. [2,25]). This also holds for interdependencies between constructs in different perspectives (consider e.g. concurrency issues in the control flow perspective and how these may affect resources). These considerations underscore the fact that it is extremely difficult, if not impossible, to describe a powerful business process language informally and be precise enough to avoid ambiguities with respect to its interpretation at runtime.

Many of the challenges in the BPM field stem from a lack of a proper conceptual and formal foundation. In this paper we will focus on this issue and demonstrate that it is possible to fully define a comprehensive reference language for both the specification and enactment of business processes. In doing so, in Sect. 2 we first examine the field of BPM in an historical context. Then in Sect. 3 we propose a fundamental set of requirements for the next generation of workflow languages. Section 4 proposes a concrete reference language, *new*YAWL [22,21], which offers powerful support for the workflow patterns and is formally defined using Coloured Petri nets. Finally Sect. 5 concludes the paper.

[1] www.workflowpatterns.com

2 Workflow 1.0: The Journey So Far

The introduction of workflow technology follows a historical trend where application considerations have been progressively separated from file management, data management and user interface considerations through the introduction of operating systems, database management systems, and GUIs. Workflow management systems take this trend a step further by making explicit the dependencies that exist between various applications and the activities conducted by human resources as well as the strategies that are associated with allocating work to resources. An in-depth description of these historical developments can be found in [24].

Another way of looking at modern workflow management from an historical perspective is to consider the various fields that have influenced its development. These include office automation, document management, advanced transaction models and groupware. Coordination between various participants and applications in work processes are a primary concern in office automation. Similarly in document management, information needs to be routed between a number of participants and may be modified along the way.

Yet another way of looking at developments in the BPM field is by considering the lifecycle of business processes. In the past the emphasis has been on process modelling and enactment, however in recent years there is the motivation to "close the loop" and include monitoring and diagnosis as part of the BPM lifecycle, thereby viewing processes as continually adapting to changes in their operating environment [10]. This trend is reflected in the emergence of the field of process mining, where process deployments are analysed through the events that they have generated during their execution [5].

It is striking that in the BPM field there has been an abundance of (proposed) standards over the years, involving a number of standardisation bodies such as WfMC, OASIS and OMG, however their impact has not been as profound or long-lasting as it should have been. One explanation for this is that standards were proposed before a sufficient understanding had been achieved of the concepts involved. Another reason is the fact that these proposals were informally defined, leaving scope for ambiguities and resulting in implementations of the same standard exhibiting fundamental differences. In some cases the standards were not sufficiently powerful and vendors defined their own extensions to address this. For example Oracle BPEL's support for the resource patterns [18] is significantly stronger than what is proposed by the standard itself.

The informal definition of standards poses a significant problem with distinct implementations choosing varying interpretations of individual constructs and most recently there has been an increased effort (cf [19,9]) to provide a formal semantics for widely used languages such as BPMN and BPEL. Even within widely utilised offerings such as Staffware, the lack of a precise operational semantics can result in the same design-time model yielding different runtime outcomes for the same data inputs (cf [21]).

Increasingly, information systems need to be able to operate in dynamic environments where interactions are required with distributed, autonomous and evolving components. Technological developments in the BPM field need to follow suit. Therefore, if tasks can be assigned to web services, web service composition can be achieved through the use of a workflow management system, and Service-Oriented Architectures provide the principles upon which open and flexible systems can be built. Ultimately aspects of this type of interaction also need to be reflected at the modelling level, paving the way for new types of powerful communication primitives (e.g. [8]).

There are a series of important lessons that have been learnt during the first phase of workflow development activities. The MOBILE [15] and WIDE [12] projects have demonstrated the importance of taking multiple perspectives into account when designing and enacting business processes, yet most contemporary standards and offerings tend to be control-flow centric. Process flexibility continues to be a focus of many workflow initiatives and a recent survey [23] identified four distinct approaches (design, deviation, adaptation and change) to its facilitation, however most offerings are only able to demonstrate capabilities in one or two of these areas. The ADEPT [20] project is one of the most mature research initiatives in terms of the flexibility support that it provides and is one of the few research endeavours to approach commercial strength. Whilst many process technologies tend to focus on the normal or expected sequence of execution events, it is the ability to handle the exceptional cases that marks their effectiveness and not surprisingly there has been a wide body of research in this area (cf [13,7,6,14]) much of which has still not made its way into mainstream offerings.

The aforementioned issues point to the need for a reference language that can offer guidance for future technology initiatives on the breadth of capabilitities that they need to embody. Such a language should be able to precisely capture the broad range of concepts which underpin contemporary business processes and be formally defined in order to avoid any ambiguities in their interpretation. YAWL [3] (Yet Another Workflow Language) was designed to provide comprehensive support for the original control-flow patterns [4] however it is increasingly clear that it needs to be comprehensively overhauled in order to ensure that it provides an appropriate level of support for the range of concepts described above. In this paper we propose a radical extension, termed *new*YAWL, that aims to achieve this goal and show that a comprehensive formalisation of its operation is possible using Coloured Petri nets.

3 Workflow 2.0: Requirements for the Next Generation

It is clear from the preceding discussion that despite the inherent benefits offered by workflow technology, they only deliver part of the solution required to effectively automate contemporary business processes. Existing offerings tend to focus on providing support for control-flow aspects and provide significantly less facilities for managing other important aspects such as the data and resource perspectives. Moreover they have a narrow view of the overall business process management lifecycle and in many cases limit their area of operation to business process enactment with somewhat less consideration of other design-time and ongoing operational issues (e.g. monitoring, refinement) that are associated with the deployment of such processes. In an attempt to lay down a clear vision for future workflow technology, in this section, we identify a series of core requirements that the next generation of workflow tools should seek to address.

REQ1: Multi-perspective support. Existing tools offer a control-flow centric view of a business process both in terms of the way in which they are defined and also in the manner in which they are deployed. Whilst the control-flow perspective is central to business processes, markedly more focus needs to be given to other significant aspects. The data and resource perspectives in particular need to be considered as first-class citizens and direct support for modelling and enacting the data repesentation, data passing, resource definition and work routing issues encountered in the context of business processes is required.

REQ 2: Integrated modelling and enactment. Traditionally, the modelling and enactment of workflows were considered to be distinct activities. Workflows were described using high-level business process modelling formalisms that focused on capturing the "spirit" of the overall business objective. Often this activity was undertaken by business analysts and the results of this work were passed to technical staff who mapped it to an equivalent workflow definition that contained sufficient detail in order for the process to actually be enacted. Often the modelling and enactment activities utilised differing technologies. Obviously the gap between these technologies leaves open the potential for ambiguities and inconsistencies to be introduced into the resultant automated business process. For this reason, an approach to modelling workflow processes is required which involves their specification in sufficient detail that they can be directly enacted.

REQ 3: Support for flexible process design and enactment. One of the ongoing criticisms of production workflow systems is that they enforce rigid processes on users that hamper rather than assist them in reaching their end goals hence obviating any potential benefits in process automation. Consequently there is now an increased focus on what process flexibility means and how it can be facilitated. In order to offer material benefit in this area, an offering needs to provide a wealth of design-time constructs for embodying flexibility in processes as well as offering runtime facilities that allow for controlled deviation from the prescribed process model, execution of underspecified process models, and adaptation and change of processes during execution.

REQ 4: Language constructs mirroring those encountered in practice. One of the fundamental difficulties associated with many workflow modelling languages is the fact that their core constructs are developed in isolation from the business processes that they are ultimately intended to facilitate. This leads to difficulties in capturing many of the actual situations that are encountered in practice. The only practical solution to resolving this impasse is to actually derive the range of constructs within a workflow language from those encountered in the real world. In order to do so, a comprehensive catalogue of the actual modelling considerations encountered in practice is required.

REQ 5: Deterministic runtime model. One of the common issues arising in workflow languages stems from their informal definition. In most cases, the constructs which make up the workflow modelling language are not formally defined and consequently do not have a precise operational semantics. This means that there is a degree of inherent ambiguity associated with their usage. In order for this difficulty to be resolved, it is necessary to provide a precise operational definition for each of the constructs in a workflow language.

REQ 6: Comprehensive concurrency support. One of the early drivers for workflow technology was to provide more efficient ways to distribute the activities associated with business processes throughout the resources within an organisation. However in many cases current workflow offerings demonstrate a surprising lack of support for managing the concurrency inherent in these processes. There are considerations that arise at a number of levels including: providing a means of facilitating concurrent task execution within a process instance, managing the use of data elements by multiple concurrent activities, handling timing issues associated with the trafficking of data elements between concurrent activities both within and between a workflow and the broader operational environment, and managing the advertisement and distribution of work between multiple resources in a predictable and reliable way. The next generation of workflow technology needs to provide broad support for all of these needs.

REQ 7: Graceful handling of expected and unexpected exceptions. One of the great benefits offered by widespread adoption of workflow technology is that it offers organisations the opportunity to move towards a management by exception regime. In this scenario, the handling of normal process instances is automated as far as possible and only deviations from the expected behaviour are subject to manual scrutiny. In order to acccomplish this, a process needs to embody support for as much exception handling as possible in order to deal with *expected* errors and similarly, it also needs to be resilient when experiencing *unexpected* exceptions and provide users with the ability to intervene in a process instance in order to instigate appropriate corrective action.

REQ 8: Recognition of the full BPM lifecycle. The first generation of workflow tools essentially focused on the automation of business processes. The emphasis being on the quantity rather than the quality of the automation achieved. The next generation of workflow technology needs to refine this approach to business process enablement and provide an increased range of options for analysing the resultant automation both to ensure its correctness and consistency from a design-time standpoint and also to examine its efficiency and effectiveness from a runtime perspective. In addition to a broader range of design-time aids to assist process developers, this also necessitates the recording of a much richer range of execution results for subsequent analysis and reflection.

In the following section, we will present an overview of *new*YAWL, a workflow reference language which aims to address these requirements and shows the form that automated support for business processes may take in the future.

4 *new*YAWL: A Blueprint for Workflow 2.0

This section provides an overview of the operational capabilities of *new*YAWL, a reference language for workflow systems based on the workflow patterns. *new*YAWL aims to lay a foundation for the next generation of workflow technology and describes a workflow modelling and execution language that incorporates comprehensive support for the control-flow, data and resource perspectives. It has a deterministic execution model whose operational semantics are defined in terms of Coloured Petri nets. This approach to characterising *new*YAWL means that it not only has a precise static definition but also that its approach to dealing with the dynamic issues that arise during execution is fully specified.

4.1 Language Overview and Format

*new*YAWL provides a comprehensive reference language for describing business processes that are to be enacted as workflows. The language constructs in *new*YAWL are informed by the various workflow patterns, hence they have a direct correspondence with the fundamental elements which are actually encountered in real-world business processes and consequently have general applicability. The *new*YAWL language is specified in two parts. It has a complete abstract syntax which identifies the characteristics of each of the language elements and their configuration. Associated with this is an executable, semantic model — presented in the form of Coloured Petri nets — which defines the runtime semantics of each of the language constructs.

The abstract syntax for *new*YAWL provides an overview of the main concepts that are captured in a design-time business process model. It is composed of five distinct schemas, each of which is specified on a set-theoretic basis. Fig. 1 summarises the content captured by each of the individual schemas and the relationships between them. Each process captured using the *new*YAWL abstract syntax has a single instance of the *new*YAWL specification associated with it. This defines elements that are common to all of the schemas and also captures

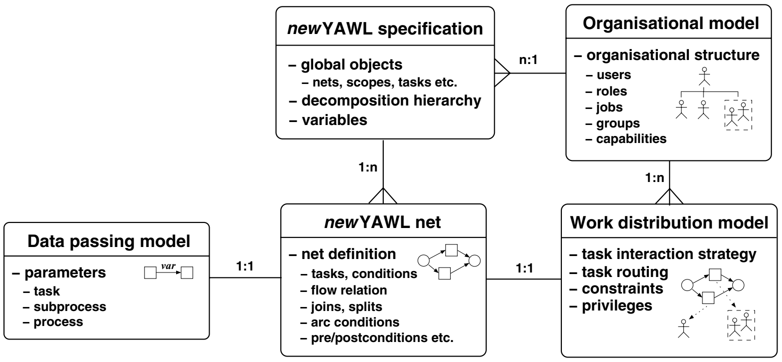


Fig. 1. Schema definition for *new*YAWL abstract syntax

the decomposition hierarchy. Each *new*YAWL specification is associated with an instance of the organisational model that describes which users are available to undertake tasks that comprise the process and the organisational context in which they operate.

A *new*YAWL process can be made up of a series of distinct subprocesses (where each subprocess specifies the manner in which a composite task is implemented) together with the top-level process. For each of these (sub)processes, there is an instance of the *new*YAWL net which describes the structure of the (sub)process in detail in terms of the tasks that it comprises and the sequence in which they occur. Associated with each *new*YAWL net is a data passing model which defines the way in which data is passed between elements in the process in terms of formal parameters operating between these elements. There is also a work distribution model that defines how each task will be routed to users for execution, any constraints associated with this activity and privileges that specific users may have assigned to them. The collective group of schemas for a specific process model is termed a *complete new*YAWL specification.

One of the virtues of specifying the operational semantics of *new*YAWL in terms of Coloured Petri nets is that the CPN Tools [16] offering provides an executable environment for models developed in this formalism. This means a candidate *new*YAWL model can actually be executed in order to verify its consistent operation. There is a two stage process for mapping a *new*YAWL specification defined in terms of the abstract syntax to an initial marking of the semantic model in CPN Tools. In the interest of brevity, this process is not discussed here but full details of its operation can be found in [21].

The complete operational semantics for *new*YAWL is based on a series of 55 CPN models [2]. Fig. 3 shows the top level model which illustrates the main events during the lifecycle of a process instance as transitions and the information elements required to support its execution as places. Each of the events takes the form of a substitution transition indicating that it has a more detailed underlying definition, however in summary, the `start case` transition is responsible for initiating a new process instance. Then there are a succession of `enter`→`start`→`complete`→`exit` transitions which fire as individual task instances are enabled, the work items associated with them are started and completed and the task instances are finalised before triggering subsequent tasks in the process model. Each atomic work item is distributed to a suitable resource for execution via the `work distribution` transition. This cycle repeats until the last task instance in the process is complete, at which point the `end case` transition terminates the process instance. Data interchange with the operating environment is faciliated by the `data management` transition and the `add` transition enables additional task instances to be dynamically instantiated for designated (multiple instance) tasks.

The places in the *new*YAWL CPN model divide into two main groups: (1) *static* places which capture the various components of a *new*YAWL process model such as the flow relation, task details, variable declarations, parameter

[2] Available from `http://www.yawl-system.com/newYAWL`.

mappings, preconditions, postconditions, scope mappings and the hierarchy of processes and subprocesses. These correspond to the design-time information captured about a *new*YAWL process as illustrated in Fig. 1 and remain unchanged during the execution of a process. (2) *dynamic* places which capture the *state* of a process instance during its execution and include items such as the current marking of each place in the flow relation, variable instances and their associated values, locks which restrict concurrent access to data elements, details of subprocesses currently being enacted, folder mappings (identifying shared data folders assigned to a process instance) and the current execution state of individual work items (e.g. *enabled*, *started* or *completed*).

An indication of the information content of an actual instance of a *new*YAWL process model is illustrated in Fig. 2 which shows a simple process for responding to a customer request for foreign exchange services. On the basis of a customer enquiry about a prospective deal, a quote is prepared and forwarded to them. They have 24 hours to respond to the quote and confirm they wish to proceed, otherwise it is withdraw. No later than 24 hours after the quote is issued, the deal is finalised, either as a result of a specific customer response or because of a timeout. For each of the tasks in the process, there is a specific routing strategy describing who should undertake the task and a specific interaction strategy indicating the basis on which it should should be distributed to a potential resource for subsequent execution. For example, the `quote` task is to undertaken by the resource that demonstrates the capability of being the advisor for the customer and it is directly allocated to (only) them for execution where as the `finalise deal` task is offered to all resources who are part of the back office role with the expectation that one of them will elect to execute it at a future time. The relevant data elements and data passing strategy are also part of a *new*YAWL process model and the *tvar* entries indicate that certain task-level data elements are passed between tasks. For example, the `quote` task receives the *cust-id* and *req-type* parameters from the `enquiry` task and passes on the *cust-id*, *quote-id* and *timeout* parameters to subsequent tasks. Despite the relative simplicity of this example, it gives an insight into the breadth of information in

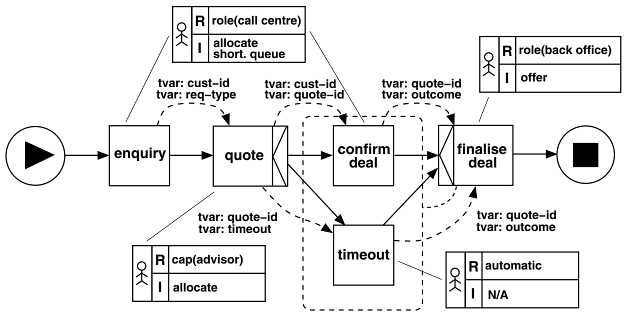


Fig. 2. Indicative example of a *new*YAWL process model: customer request for foreign exchange services

various perspectives that potentially can be captured in a *new*YAWL process model. Further details on the specifics of each of the perspectives are included in the following sections.

4.2 Control-Flow Perspective

A process model in *new*YAWL is analogous to a Petri net (although they are not the same and *new*YAWL includes some additional constructs such as the cancellation region that are not available in Petri nets). It consists of tasks (which correspond to executable activities) and conditions (that correspond to states) connected in the form of a directed graph. There is a designated start and end node for a process and there are two fundamental requirements pertaining to model structure: (1) all nodes (i.e. tasks and conditions) must be on a path from the start to the end node and (2) a condition may not be connected to another condition. Processes may be hierarchical in form with block tasks mapping to a corresponding subworkflow to which they pass control when invoked. Where multiple arcs enter or exit a task, various forms of join and split conditions are supported which describe the state requirements for task initiation and the effects of completion. In Fig. 3 the `flow relation` and `process hierarchy` places capture the details of individual *new*YAWL workflow models and the hierarchy that they form.

Control-flow in *new*YAWL is managed in an analogous manner to that in a Petri net and is based on the traversal of tokens through a process model. Each control-flow token identifies the process model and process instance to which it applies. In Fig. 3 the various control-flow tokens reside in the `process state` place. Tokens are removed from this place by enabled tasks and returned to it when they complete. The lifecycle of a task is illustrated by the `enter`, `start`, `complete and terminate block` and `exit` transitions which identify the various stages through which an enabled task passes as it progresses from initiation to completion.

Each process has a unique identifier known as a `ProcessID` and each process model has a unique `BlockID` (this is necessary as the hierarchy within a process means it may contain several distinct process models defining subworkflows in addition to the top-level model). Each task within a process is identified by a unique `TaskID`. In order to allow for and differentiate between concurrent execution instances, it is necessary to introduce some additional notions. First an executing instance of a process is termed a case. It has a case identifier `CID` which is unique for a given `ProcessID`. Hence the tuple `(ProcessID,CID)` uniquely identifies all cases. Similarly an enabled task instance is known as a work item. It has a more complex identification scheme denoted by the five-tuple `(ProcessID,CID,TaskID,Inst,TaskNr)` where `Inst` identifies the specific instance of the task that is being executed (thus allowing for distinct instances of a task as may occur if it is in a loop for example) and `TaskNr` which allows distinct concurrent execution instances of a multiple instance task to be differentiated. By adopting this identification scheme, it is possible for the semantic model to

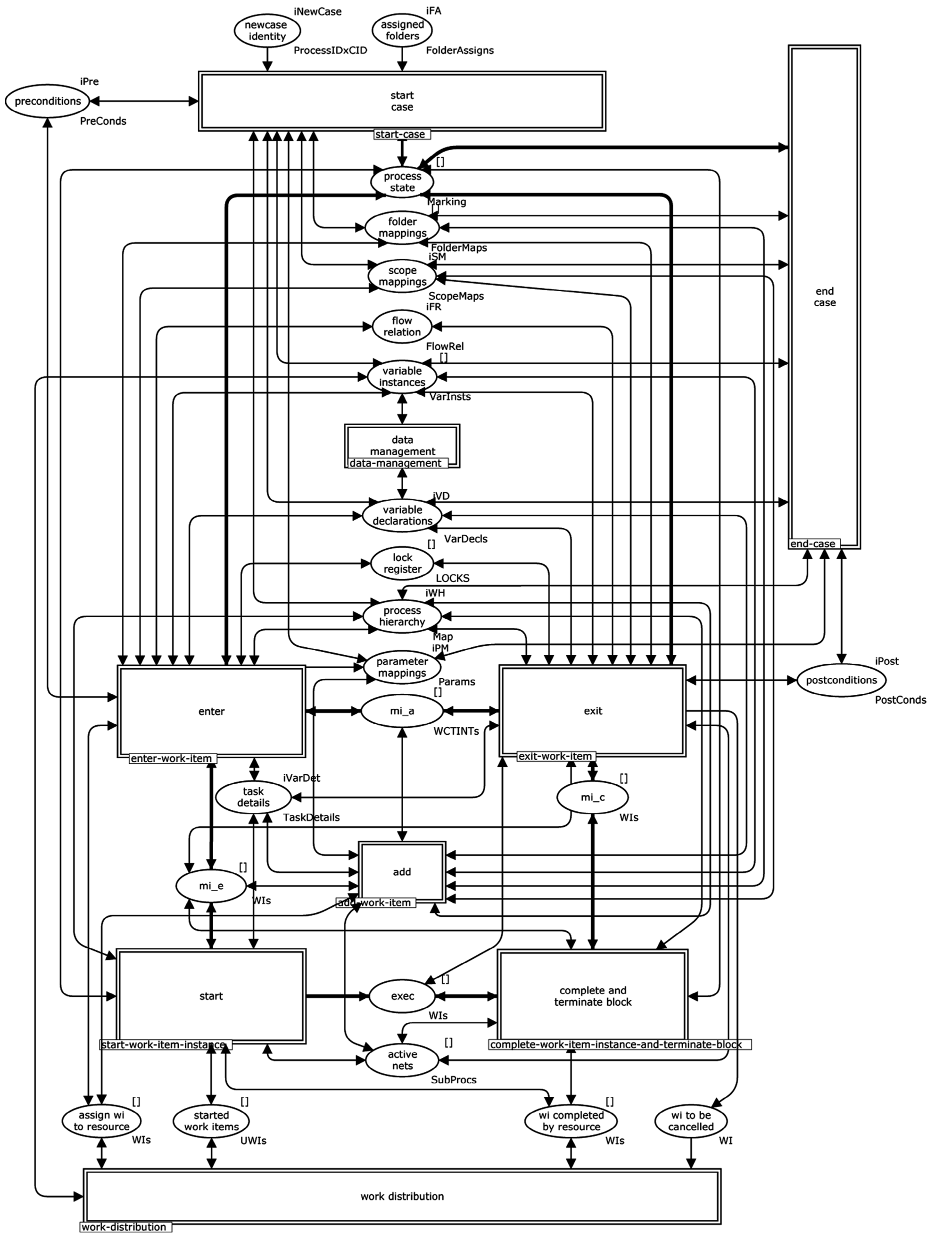


Fig. 3. Overview of the operational semantics for *new*YAWL

cater for multiple concurrent processes, process instances and task instances in a common environment.

4.3 Data Perspective

*new*YAWL incorporates a series of features derived from the data patterns, providing coverage of issues such as persistence, concurrency management and complex data manipulation which are often absent from workflow languages. It provides support for a variety of *distinct scopes* to which data elements can be bound e.g. task, block, case etc. These are encoded in the `variable declarations` place shown in Fig. 3. Data passing between process constructs is based on the use of *formal parameters* which take a function-based approach to data passing thus supporting inline transformations during data passing events. These parameters are encoded in the `parameter mappings` place. A similar functional approach is taken to specifying *link conditions* for OR-splits and XOR-splits that allow the determination of which outgoing branches should be activated and *preconditions* and *postconditions* for tasks and processes. These are encoded in the `task details,` and `preconditions` and `postconditions` places respectively. Finally, the use of *locks* allows concurrent data usage to be managed through an approach which requires tasks to specify data elements that they require exclusive access to (within a given process instance) in order to commence. A task instance can commence execution if it can acquire locks on all required data elements. It retains these locks until it has completed execution preventing any other task instances from using the locked data elements concurrently. The locks are recorded in the `lock register` place.

4.4 Resource Perspective

The resource perspective is responsible for describing the resources who undertake a given business process and the manner in which associated work items are distributed to them and managed through to completion. For each task, a specific *interaction strategy* is specified which describes how the associated work item will be distributed to users, and what degree of autonomy they have in regard to choosing whether they will undertake it or not and when they will commence exeuting it. Similarly, a detailed *routing strategy* can be defined which identifies who can undertake the work item. Users can be specified by name, in terms of roles that they perform, based on capabilities that they possess, in terms of their job role and associated organisational relationships or based on the results of preceding execution history. The routing strategy can be further refined through the use of constraints that restrict the potential user population. Indicative constraints may include: *retain familiar* (i.e. route to a user that undertook a previous work item), *four eyes principle* (i.e. route to a different user than one who undertook a previous work item), *random allocation* (route to a user at random from the range of potential users), *round robin allocation* (route to a user from the potential population on an equitable basis such that all users receive the same number of work items over time) and *shortest queue allocation* (route the work item to the user with the shortest work queue). *new*YAWL also supports two advanced operating modes *piled execution* and *chained execution* that are designed to expedite the throughput of work by imposing a defined protocol on the way in which the user interacts with the system and work items are allocated to them.

4.5 *new*YAWL: Progress Towards Workflow 2.0

One of the major objectives of the *new*YAWL initiative was to advance the state of the art in workflow systems by providing a reference language for multi-perspective business processes (i.e. requirement REQ 1 in Sect. 3). In doing so, it has also addressed many of the goals identified in Sect. 3 for the next generation of workflow technology. The language constructs in *new*YAWL are based on the fundamental requirements for business processes identified by the 126 workflow patterns [21]. The experiential approach taken to characterising these patterns, based on a comprehensive survey of commercial offerings, standards, modelling formalisms and programming language theory, ensures a close correlation between the language constructs in *new*YAWL and those encountered in practice (REQ 4). Moreover, the fact that these constructs span multiple perspectives of a business process is also directly recognised in the breadth of the abstract syntax for *new*YAWL.

*new*YAWL is formalised in terms of a series of CPN models, which provide a precise interpretation for each of its language constructs and facilitates deterministic execution (REQ 5) of business processes captured in terms of its abstract syntax. One of the major advantages of the approach taken to the language definition for *new*YAWL is that there is a clear mapping from the abstract syntax to the runtime environment and a business process captured in terms of the abstract syntax can be directly executed without requiring further information (REQ 2). This means that business process models specified using *new*YAWL are applicable throughout the BPM lifecycle. They are used directly in the modelling, implementation and enactment phases of workflow processes and, in conjunction with the logging information recorded during execution, they provide the basis for comprehensive analysis of processes in retrospect. Moreover, the comprehensive description contained in a *new*YAWL business process when utilised in conjunction with the proposed execution logs (which include not only details of task execution, but also data transfer and the various stages of the lifecycle for individual work items) means that the basis exists for comprehensive monitoring and analysis of business processes during execution (REQ 8).

A significant consideration during the design of *new*YAWL was in ensuring that the resultant language ultimately provided a broader range of facilities at runtime than has been demonstrated by preceding workflow technology. The broad range of workflow patterns it supports (118 fully and 1 partially out of the complete set of 126 patterns) gives an indication of the breadth of its overall capabilities, and it provides a range of useful features including support for complex data structures in workflow data elements, a variety of integration facilities with the operational environment, support for configurable exception handling

(REQ 7), rich resource definition and the ability to specify a wide range of work item distribution mechanisms for routing work to users and managing it through to completion. It also demonstrates a range of facilities for flexible process enactment (REQ 3) particularly in the areas of flexibility by design and underspecification although less so in the areas of deviation, adaptation and change. Further information on *new*YAWL can be found in [22] and [21] including illustrative examples of its usage and detailed discussions of the language design and validation. One area of the *new*YAWL feature set that merits special attention, is the manner in which it facilitates concurrency in business processes (REQ 6). The following section discusses this issue in detail with reference to four specific issues that arise in the control-flow, data and resource perspectives.

4.6 *new*YAWL: Better Concurrency Support for Processes

In this section, we discuss the concurrency support provided by *new*YAWL with reference to four specific examples: task enablement, concurrent data element usage, work item distribution and deferred choice.

Task Enablement. One of the most complex activities in workflow execution is managing the enablement of a task. This entails two distinct steps: (1) determining if the various prerequisites that apply to task enablement have been met and (2) facilitating the actual enablement as a single atomic activity.

There are five requirements for a task to be enabled: (1) the precondition associated with the task must evaluate to true, (2) all data elements which are inputs to mandatory input parameters must exist and have a defined value, (3) all mandatory input parameters must evaluate to defined values, (4) all locks which are required for data elements that will be used by the work items associated with the task must be available and (5) if the task is a multiple instance task, the multiple instance parameter when evaluated must yield a number of rows that is between the minimum and maximum number of instances required for the task to be initiated. Only when all of these prerequisites have been met can the actual enabling of a task occur. The `enter` transition, illustrated in Fig. 3, is responsible for managing task enablement which involves the simultaneous completion of the following actions:

1. Removing the control-flow tokens marking input conditions to the task corresponding to the instance enabled from the `process state` place;
2. Determining which instance of the task this is;
3. Determining how many work item instances should be created. For an atomic or composite task this will always be a single work item, however for a multiple instance or composite multiple instance task, the actual number started will be determined from the evaluation of the multiple instance parameter contained in the `parameter mappings` place together with the current data state in `variable instances` which will return a composite result containing a number of rows of data indicating how many instances are required. In all of these situations, individual work items are created which share the

same `ProcessID`, `CID`, `TaskID` and `Inst` values, however the `TaskNr` value is unique for each work item and is in the range 1...number of work items created;

4. For all work items corresponding to composite tasks, distinct subprocess `CID`s need to be determined from the `process hierarchy` place to ensure that any variables created for subprocesses are correctly identified and can be accessed by the work items for the subprocesses that will subsequently be triggered;
5. Creating variable instances for data elements associated with the task using the variable definitions corresponding to the task in the `variable definition` place. Data elements are added to the `variable instances` place;
6. Mapping the results of any input parameters for the task instance as identified in the `parameter mappings` place to the relevant task data elements. This uses data values from the `variable instances` place and updates any required input variables in this place created in step 5;
7. Recording any variable locks that are required for the execution of the task instance in the `lock register` place;
8. Creating work item distribution requests for the work item to be allocated to a specific resource. These are added to the `assign wi to resource` place for subsequent routing to resources; and
9. Finally, work items with an *enabled* status need to be created for this task instance and added to the `mi_e` place which identifies work items corresponding to enabled but not yet started tasks.

Concurrent Data Element Usage. An issue that was addressed many years ago by the database community but which is surprisingly lacking in many workflow solutions is the ability to manage concurrent usage of data elements within a process instance. *new*YAWL addresses this issue by introducing the notion of locks which allow exclusive access to a data element to be retained by a specific task instance during its execution. Locks are evaluated at the time of task enablement and where a task requires a data element to which it cannot acquire a lock, then its enablement is deferred until it can do so. In Fig. 3, the `lock register` place holds details of locks that are currently pending.

Work Distribution. The main motivation for workflow systems is achieving more effective and controlled distribution of work. Hence the actual routing of work items to specific resources and managing the interaction with the resource as they progress individual work items to completion are of particular importance. The process of managing the distribution of work items is summarised by Fig. 4 which shows how the `work distribution` transition in Fig. 3 is implemented. This transition coordinates the interaction between the workflow engine, and the `work item distribution`, `worklist handler`, `management interven- tion` and `interrupt handler` transitions.The specific functions provided by these transitions are as follows:

– the `work item distribution` transition identifies the resources to whom work items should be routed and manages the interaction(s) with individual resources as work items are offered, allocated, started and completed;

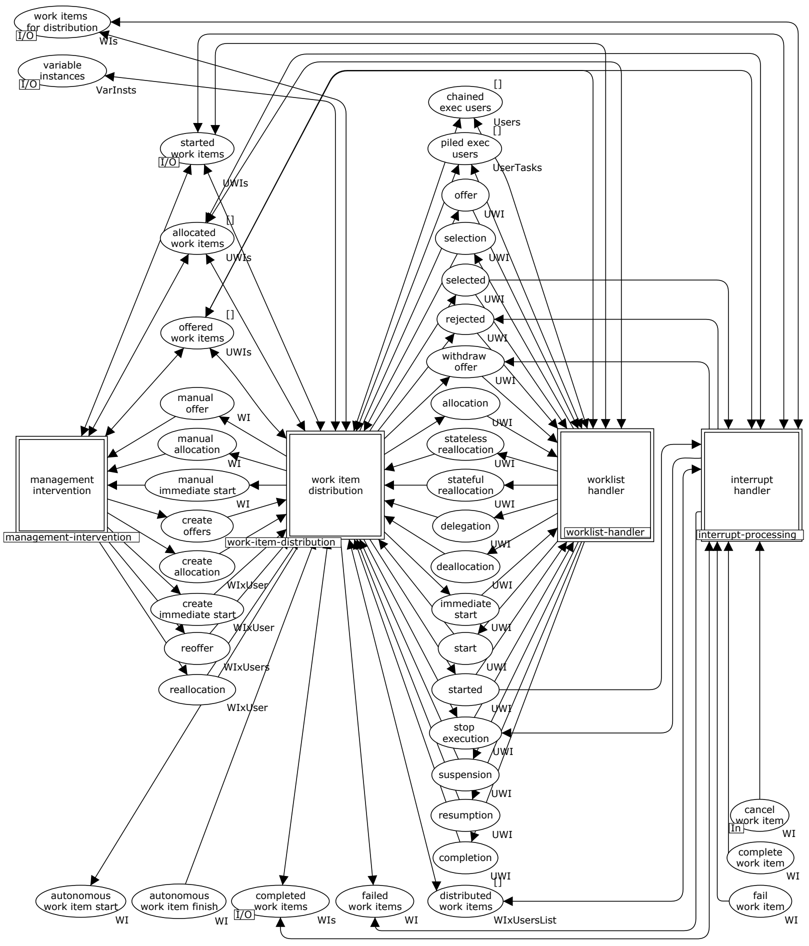


Fig. 4. Top level view of the `work distribution` transition

– the `worklist handler` transition corresponds to the user-facing client software that advises users of work items requiring execution and manages their interactions with the main `work item distribution` transition in regard to committing to execute specific work items, starting and completing them;
– the `management intervention` transition provides the ability for a workflow administrator to intervene in the `work distribution` process and manually reassign work items to users where required; and
– the `interrupt handler` transition supports the cancellation, forced completion and forced failure of work items as may be triggered by other components of the workflow engine (e.g. the control-flow process, exception handlers).

The distribution of work items to users from the workflow engine is initiated via the `work distribution` transition which forwards work items to the `work` `items for distribution` place. The `work item distribution` transition then determines how they should be routed to users. This may involve the services of the workflow administrator in which case they are sent to the `management intervention` transition or alternatively they may be forwarded directly to one or several users via the `worklist handler` transition. The various places between these three transitions correspond to the range of requests that flow between them.

The status of work items in progress is maintained in the `offered work items`, `allocated work items` and `started work items` places which are shared between the `work item distribution`, `worklist handler`, `management inter- vention` and `interrupt handler` transitions. Although much of the information about the state of work items is shared, the determination of when a work item is actually complete rests with the `work item distribution` transition. It inserts a token in the `completed work items` place when a work item is complete. Similarly, work item failures are notified via the `failed work items` place. Work items that are subject to some form of interrupt (e.g. an exception being detected and handled) are handled by the `interrupt handler` transition which manages cancellation, forced completion and failure requests received in the `cancel work item`, `complete work item` and `fail work item` places respectively. The complexity of the activities comprising the `work distribution` transition is underscored by the fact that each of them are also substitution transitions and in each case have a relatively complex underlying implementation.

Deferred Choice. The implementation of the deferred choice construct is problematic for many workflow systems that do not have a notion of state. An example of such a situation in a process is where a commuter defers the choice as to how to get to work until after they have left the house. The actual choice is made when they either decide to *walk to work* or *take the bus,* and the selection occurs at the instigation of the commuter when they actually commence on their chosen mode of travel. At this point, the other travel option is abandoned and ceases to be a possible alternative course of action. In *new*YAWL this construct is facilitated by offering all of the tasks subject to the deferred choice to the user(s) responsible for making the choice. Once one of them is selected by a resource, then the work items corresponding to the other tasks are removed from resources' work lists via a cancellation action.

5 Conclusions

Workflow technology offers great promise as a general purpose means of automating business processes, however in its current incarnation it is dogged by a series of criticisms including its narrow view of what constitutes a business process, the lack of formal foundations and its inability to characterise real-world business scenarios. This paper has examined the capabilities of the current generation of workflow technology and proposed a series of development goals for the next generation of workflow tools. As a first step towards these objectives, it has also presented *new*YAWL, a formally defined workflow reference language founded on the workflow patterns, that meets the proposed development goals and provides a yardstick against which the capabilities of future workflow offerings can be assessed. *new*YAWL is currently being used as the design blueprint for the next generation of the YAWL open-source workflow offering.

References

1. van der Aalst, W.M.P.: Patterns and XPDL: A Critical Evaluation of the XML Process Definition Language. QUT Technical report, FIT-TR-2003-06, Queensland University of Technology, Brisbane, Australia (2003)
2. van der Aalst, W.M.P., Desel, J., Kindler, E.: On the semantics of EPCs: A vicious circle. In: Rump, M., Nüttgens, F.J. (eds.) Proceedings of the EPK 2002: Business Process Management using EPCs, Trier, Germany, pp. 71–80. Gesellschaft fur Informatik (2002)
3. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: Yet another workflow language. Information Systems 30(4), 245–275 (2005)
4. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases 14(3), 5–51 (2003)
5. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow mining: A survey of issues and approaches. Data and Knowledge Engineering 47(2), 237–267 (2003)
6. Alonso, G.: Transactional Business Processes. In: Process-Aware Information Systems, pp. 257–278. John Wiley & Sons, Chichester (2005)
7. Casati, F., Ceri, S., Paraboschi, S., Pozzi, G.: Specification and implementation of exceptions in workflow management systems. ACM Transactions on Database Systems 24(3), 405–451 (1999)
8. Decker, G., Kopp, O., Leymann, F., Weske, M.: BPEL4Chor: Extending BPEL for modeling choreographies. In: Proceedings of the IEEE 2007 International Conference on Web Services (ICWS), pp. 296–303. IEEE Computer Society, Los Alamitos (2007)
9. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Information and Software Technology (IST) (accepted for Publication) (2008)
10. Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Process-Aware Information Systems: Bridging People and Software through Process Technology. Wiley-Interscience, Hoboken (2005)
11. Ellis, C.A.: Information control nets: A mathematical model of office information systems. In: Proceedings of the Conference on Simulation, Measurement and Modeling of Computer Systems, Boulder, CO, USA, pp. 225–240. ACM Press, New York (1979)
12. Grefen, P., Pernici, B., Sanchez, G.: Database support for workflow management: the WIDE project. Kluwer Academic Publishers, Boston (1999)
13. Grefen, P.W.P.J., Vonk, J.: A taxonomy of transactional workflow support. International Journal of Cooperative Information Systems 15(1), 87–118 (2006)
14. Hagen, C., Alonso, G.: Exception handling in workflow management systems. IEEE Transactions on Software Engineering 26(10), 943–958 (2000)
15. Jablonski, S., Bussler, C.: Workflow Management: Modeling Concepts, Architecture and Implementation. Thomson Computer Press, London (1996)

16. Jensen, K.: Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use, Basic Concepts. Monographs in Theoretical Computer Science, vol. 1. Springer, Berlin (1997)
17. Kiepuszewski, B., ter Hofstede, A.H.M., van der Aalst, W.M.P.: Fundamentals of control flow in workflows. Acta Informatica 39(3), 143–209 (2003)
18. Mulyar, N.: Pattern-based evaluation of Oracle BPEL. Technical Report BPM-05-24 (2005), `www.BPMcenter.org`
19. Ouyang, C., Verbeek, H.M.V., van der Aalst, W.M.P., Breutel, S., Dumas, M., ter Hofstede, A.H.M.: Formal semantics and analysis of control flow in WS-BPEL. Sci. Comput. Program. 67(2-3), 162–198 (2007)
20. Reichert, M., Rinderle, S., Dadam, P.: ADEPT workflow management system. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 370–379. Springer, Heidelberg (2003)
21. Russell, N.: Foundations of Process-Aware Information Systems. PhD thesis, Faculty of Information Technology, Queensland University of Technology (2007), `www.yawl-system.com/documents/RussellThesisFinal.pdf`
22. Russell, N., ter Hofstede, A.H.M., van der Aalst, W.M.P.: newYAWL: Specifying a workflow reference language using Coloured Petri Nets. In: Proceedings of the Eighth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, Department of Computer Science, University of Aarhus, Denmark, vol. DAIMI PB-584, pp. 107–126 (2007)
23. Schonenberg, H., Mans, R., Russell, N., Mulyar, N., van der Aalst, W.M.P.: Process flexibility: A survey of contemporary approaches. In: Dietz, J.L.G., Albani, A., Barjis, J. (eds.) CIAO! / EOMAS. LNBIP, vol. 10, pp. 16–30. Springer, Heidelberg (2008)
24. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer, Heidelberg (2007)
25. 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)
26. Zisman, M.D.: Representation, Specification and Automation of Office Procedures. PhD thesis, Wharton School of Business, University of Pennsylvania, PA, USA (1977)

A Formal Model for Organisational Structures behind Process-Aware Information Systems

Michael Köhler-Bußmeier, Matthias Wester-Ebbinghaus, and Daniel Moldt

University of Hamburg, Department of Informatics
Vogt-Kölln-Str. 30, D-22527 Hamburg
{koehler,wester,moldt}@informatik.uni-hamburg.de

Abstract. We present a formal model for the organisation of process-aware information systems. Our approach focuses on the structures "behind" business processes, like e.g. team formation and coordination. Our Petri net based model directly integrates organisational concepts like roles, teams etc. – allowing an alignment of business and IT. The benefit of this modelling overhead is that business reorganisation processes are carried out as formal model transformations. Additionally, the automated mapping of our models to multi-agent systems – in the spirit of the model driven architecture idea – is directly supported by our approach.

1 Organisations as the Structure behind PAIS

In this paper we deal with the organisational structure relevant for process-aware information systems (PAIS). Let us consider some simple scenario: Workflows consist of (partially) ordered *activities*. Activities usually have to be executed by some (human or artificial) *agent*. The capabilities, that an agent has to provide in order to carry out an activity, is described by *roles*: Each activity is mapped to one role. If an agent implements some role R then it is responsible for all the activities mapped to R. One agent can of course implement several roles at the same time (even all the roles of a given workflow) and several agents may implement the same role.

For most scenarios this level of detail is sufficient when dealing with business processes of PAIS. For our approach, however, we like to look a little bit more behind the scenes and consider the assignment process. The process of assigning roles to agents is called *team formation*. Each role of a workflow, that is not yet assigned to some agent, is considered as a *task* in the team formation process. *Task implementation* is two-fold: In typical team-frameworks (like the contract net [1]) agents do not have to *execute* each task they are assigned to by themselves – they may *delegate* this task to some other agents. Starting with an initial task this iterated delegation constructs a directed tree with agents as nodes. The root is the agent that considers the initial task as relevant. The leaves are the agents that implement roles in the workflow (usually called executive agents or contractors) and the inner nodes are agents which serve coordination purposes (manager agents). A *team* consists of this delegation tree together with the workflow and its state. The processes of team formation, coordination, negotiation within one team is called *teamwork*.

When an agent A delegates a task to the agents $A_1, \ldots, A_n$ it generates one subtask for each agent A_i $(1 \leq i \leq n)$. Each agent A_i only has to implement some "smaller" portion of the role R of the original task, but it is possible that A_i implements this portion in a different way, i.e. they implement different roles R_i. But when all the agents $A_1, \ldots, A_n$ are working together (coordinated by A) their implementation of the roles $R_1, \ldots, R_n$ must be "as good" (or more technical: bisimilar) as if A would have implemented R itself.

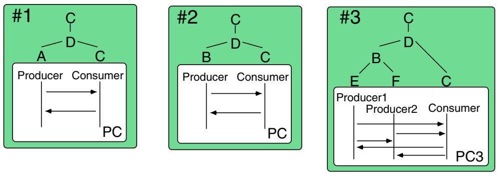


Fig. 1. A Workflow System

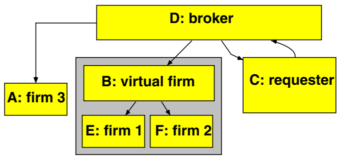


Fig. 2. An Organisation Chart

As our working example cf. Figure 1. It shows a workflow system with three workflow teams: #1, #2, and #3. All these instances describe some producer/consumer (P/C) interaction. Instance #1 and #2 are based on the same workflow PC while #3 is based on a more complex one named PC_3. (We are not interested in the details of this interaction for this moment.)

Above the workflows we have shown the delegation trees. All trees have agent C as root, i.e. it is C that signals the need for P/C interaction. In all trees C delegates to agent D which delegates to two agents, one for the producing and one for the consuming part, but which agents are chosen is different: In instance #1 we have agent A implementing the role Producer and C implementing Consumer (which are also the leaves of the tree, i.e. the executers) while in instance #2 we have agent B and C. In instance #3 we have the further complication that B does not implement the role Producer itself, but rather delegates to E and F which have the roles Producer_1 and Producer_2, respectively. To implement the original Producer/Consumer-task started by C we have to make sure that E and F (and the roles Producer_1 and Producer_2) together may serve the same purpose as B when acting in the role of Producer.

Obviously we can drive this point a little bit further and ask for the structure behind the teamwork itself. Following [2], this structure must provide enough information to describe which agent may delegate to whom, but may include additional information about the current workload of agents, their trustworthiness, their prices (whenever market mechanisms are applied) etc. This structure is called an (agent) *organisation*. The agents of a system occupy *positions* within the organisation. For our working example we might guess that the delegation network must look somehow similar to the chart in Figure 2, but we will soon find out that this structure is not rich enough to explain e.g. why C may act as a manager and executer while all other agents don't; or why it is possible for D

to choose between A and B (as in #1 and #2), while B has to delegate to both E and F (as in #3).

When dealing with such scenarios we can rely on a rich literature how to model workflows including the concepts like roles, activities, communication, refinement, observation equivalence etc. Workflow nets [3] and their descendants can be seen almost as the canonical approach both for theoretical and practical purposes as they provide a simple, intuitive, and elegant model.

While the process perspective is sufficiently clear (at least for the scope of this paper) the area of teamwork has not reached such a level. Teamwork has its origin in the area of *distributed artificial intelligence* (DAI) where problem solving is studied from the perspective of coordinating agents [4], in the area of *computer supported cooperative work* (CSCW) [5], and in the area of *decision support systems* [6]. For a recent survey on teamwork cf. [7]. There is also a relationship to interorganisational workflows [12] and to web services [14].

When research on teams can be characterised as ongoing then research on organisations has just started to attract attention from researchers. *Computational organisation theory* [8] can be characterised as the field at the borders of social and economic sciences and DAI. In recent years organisation theory has had an impact on the organisation centred design of (open) software systems, like e-markets or web services [9]. Reorganisational aspects of workflow management systems are studied in [10,11].

In this paper we propose a Petri net based organisational model, called SONAR (Self Organising Net ARchitecture), that (a) should be simple enough to be easily understood and analysed and (b) at the same time rich enough to capture the interplay of concepts like workflows, teamwork, delegation etc. in such a detail that we can automatically generate (i.e. compile) code from these organisational models.

Moreover, our SONAR-model is reflective in the sense that the organisation does not only shape the business processes by teamwork as described above (first order processes), but also reorganises itself as a reaction to the business processes (second order processes).

The paper gives a formal model for organisations and teams and is structured as follows: Section 2 gives a formal model of organisations based on Petri nets. The set of possible teams is modelled as a so-called R/D net. Section 3 describes organisation nets and their behaviour, i.e. the team formation and planning processes (which are the main activities in the early phase which precede the 'core' business processes). Section 4 describes how this formal model is used for transformation on the meta-level, i.e. for business reorganisation processes. The paper closes with a conclusion and an outlook.

2 Formal Organisations

A formal organisation describes the interplay of the delegation network, roles, positions, workflows, task delegation etc. In the following we introduce a formal model for organisations based on Petri nets.

2.1 Distributed Workflow Nets

At the basis we define workflows with several participants. The model used here is a multi-party version of workflow nets [3] where the parties are called *roles*. Roles are used to abstract from concrete agents. For example, the two roles *Producer* and *Consumer* have the same form of trading interaction no matter which agent is producing or consuming. Figure 3 shows the Petri net *PC* that describes the interaction between both roles: First the producer executes the activity *produce*, then *sends* the produced *item* to the consumer, who *receives* it. The consumer sends an *acknowledge* to the producer before he *consumes* the item. Technically speaking roles are some kind of type for an agent describing its behaviour. Note, that agents usually implement several roles.

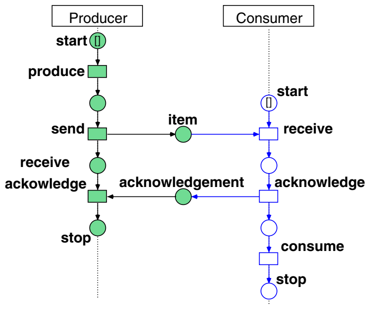


Fig. 3. The DWF Net *PC* (Producer/Consumer)

Each transition t of a *distributed workflow (DWF) net* D, which models an activity, is assigned to the (atomic) role $r(t) \in Rol$ with the meaning, that only agents that implement this role r are able to execute the activity t. In Figure 3 all transitions are drawn below the role that they are assigned to (here: Producer and Consumer). One can see from this figure that practical DWF nets are very similar to UML interaction diagrams.

Roles are names which obtain behavioural meaning via a DWF net. Given a set of atomic roles Rol the set $\mathcal{R} := 2^{Rol}$ is the *role universe*. Each $R \in \mathcal{R}$ is called a *role*. The singleton roles of the form $R = \{r\}$ are identified with the atomic role r itself.

A *Petri net* is a tuple $N = (P, T, F)$ where P is a set of places, T is a set of transitions, disjoint from P, i.e. $P \cap T = \emptyset$, and $F \subseteq (P \times T \cup T \times P)$ is the flow relation. A marking $\boldsymbol{m}$ is a multiset of places, i.e. a mapping $\boldsymbol{m} : P \to \mathbb{N}$. A Petri net $K = (B, E, \lessdot)$ is called a *causal net* whenever $\lessdot^*$ is a partial order. The *preset* of a node y is ${}^\bullet y := (\_ \ F \ y)$ and *postset* is $y^\bullet := (y \ F \ \_)$. The set of places with empty preset is ${}^\circ N = \{x \in P \cup T \mid {}^\bullet x = \emptyset\}$. The set of places with empty postset is $N^\circ = \{x \in P \cup T \mid x^\bullet = \emptyset\}$. For causal nets ${}^\circ N$ are the minimal and N° the maximal elements.

Workflow nets [3] are an established means to model workflow processes. A *workflow net* is a Petri net $N = (P, T, F)$ with two distinguished places i and

f with $^\bullet i = f^\bullet = \emptyset$ and each node lies on a path between i and f. To model workflows involving several roles we generalise this notion for multiple start and end points. Despite this, our model of DWF nets is very similar to the *Interorganizational Workflows* of [12]. (We do not deepen on this connection here because the interaction model itself does not lie in our main focus.) By adding an initial transition, that removes one token from a fresh place i and adding one tokens to all the places in $^\circ N$, and one final transition, that removes one tokens from all the places in N° and adds one to the fresh place f, we obtain a workflow net from a DWF net.

Definition 1. *A* DWF net $D = (N, r)$ *is a is a Petri net* $N = (P, T, F)$ *(where* $^\circ N$ *and* N° *are non empty sets and each node lies on a path between* $^\circ N$ *and* N°*) together with a role labelling* $r : T \to Rol$.

The canonical initial marking is m_i *and the final marking is* m_f *with:*

$$m_i(p) = \begin{cases} 1, & \text{if } p \in {^\circ N} \\ 0, & \text{otherwise} \end{cases} \quad \text{and} \quad m_f(p) = \begin{cases} 1, & \text{if } p \in N^\circ \\ 0, & \text{otherwise} \end{cases}$$

Define the mapping R *on DWF nets as* $R(D) := r(T)$.

We require that all transitions in the preset of a place p are assigned to the same role: $r(t_1) = r(t_2)$ for all $t_1, t_2 \in {^\bullet p}$ (and similarly for the postset). Whenever a place p connects two transitions t_1 and t_2 with $r(t_1) = r(t_2) = r$ it models some local state of the agent that implement the role r. Whenever $r(t_1) \neq r(t_2)$ then the place p models a *message buffer* which is drawn horizontally (here: *item* and *acknowledgement*).

The role inscriptions induce certain subnets, called role-components. The role component $D[R]$ is a Petri net module with interface places for in- and output similar to [13] and successors, e.g. [14].

Definition 2. *Given a DWF net* D *and a role* $R \subseteq R(D)$ *we can restrict* D *to the subnet* $D[R] = (P_R, T_R, F_R)$*, called the* R*-component of* D*, defined by the nodes related to* R*:* $T_R := r^{-1}(R)$ *and* $P_R := ({^\bullet T_R} \cup {T_R}^\bullet)$.

In Figure 3 the role component $PC[\mathsf{Producer}]$ is indicated by the filled nodes. In the introduction (cf. Figure 1), agent A executes $PC[\mathsf{Producer}]$ in #1 and B in #2 while C executes $PC[\mathsf{Consumer}]$ in #1 and #2.

The composition $N_1 \parallel N_2$ of two nets is defined by place fusion and disjoint union of transitions. For role components we have the following composition property.

Theorem 1. *For a DWF net* $D = (N, r)$ *and roles* $R_1, R_2 \subseteq R(D)$ *with* $R_1 \cap R_2 = \emptyset$ *we have* $N[R_1] \| N[R_2] = N[R_1 \cup R_2]$.

A Petri net processes (K, ϕ) consists of a causal net K and a locality preserving net morphism ϕ – cf. [15] and the appendix. $Proc^{wf}(D)$ denotes the set of all Petri net processes of D such that the minimal elements $^\circ K$ represent the initial marking: $\phi^{\oplus}({}^{\circ}K) = m_i$ and the maximal elements K° represent the final marking: $\phi^{\oplus}(K^{\circ}) = m_f$ (where $\phi^{\oplus}$ is the multiset extension of ϕ).

Given two DWF nets D_1 and D_2 and roles $R_1 \subseteq R(D_1)$ and $R_2 \subseteq R(D_2)$ the *interaction refinement* relation

$$D_1[R_1] \simeq D_2[R_2] \tag{1}$$

holds iff the subnet $D_1[R_1]$ can be replaced by $D_2[R_2]$ (and vice versa) in any context without changing the behaviour at the interface.[1] So, $D_1[R_1]$ and $D_2[R_2]$ are indistinguishable from their input/output behaviour, i.e. their abstract message processing.

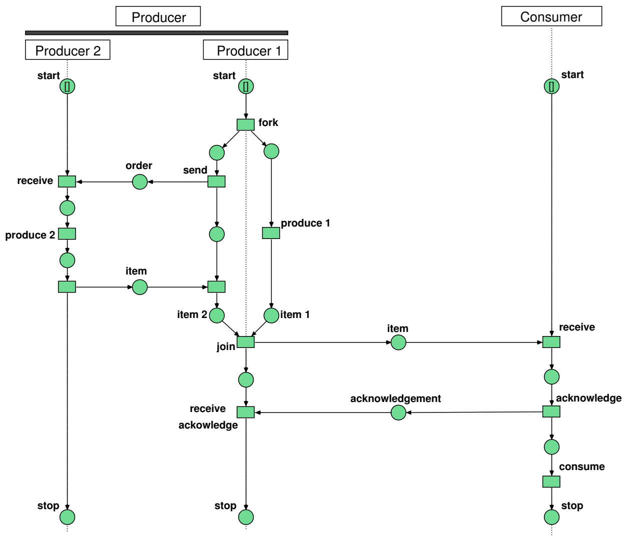


Fig. 4. Refined Producer/Consumer DWF Net PC_3

For example, Figure 4 presents the DWF net PC_3 that refines (with respect to communication behaviour) the role Producer in the net PC of Figure 3 by the roles $\mathsf{Producer}_1$ and $\mathsf{Producer}_2$ ($\mathsf{Prod}_1, \mathsf{Prod}_2$ for short):

$$PC[\mathsf{Prod}] \simeq PC_3[\mathsf{Prod}_1, \mathsf{Prod}_2]$$

A set of DWF nets $\mathcal{D}$ is called *DWF universe*, whenever $R \in R(D_1) \cap R(D_2)$ for $D_1, D_2 \in \mathcal{D}$ then $D_1[R]$ cannot be distinguished from $D_2[R]$ with respect to their communication behaviour.

In the instance #3 of our working example of Figure 1 agent B implements its task, which is described by $PC[\mathsf{Prod}]$ by delegating subtasks to the agents E

[1] This is formalised as a bisimulation where each message buffer place p is refined into a subnet $p' \rightarrow t_p \rightarrow p''$ and the bisimulation is defined only on the fresh transitions t_p; the original transitions are silent τ steps.

and F which together implement $PC_3[\mathsf{Prod}_1] \| PC_3[\mathsf{Prod}_2] = PC_3[\mathsf{Prod}_1, \mathsf{Prod}_2]$ which is (as required) a proper refinement of $PC[\mathsf{Prod}]$.

For a given DWF universe $\mathcal{D}$ we denote the set of all DWF nets which contain R as $\mathcal{D}(R)$:

$$\mathcal{D}(R) = \{D \in \mathcal{D} \mid R \subseteq R(D)\} \tag{2}$$

2.2 Role/Delegation Nets and Teams

For the rest of this paper we assume a fixed DWF universe $\mathcal{D}$ and a fixed role universe $\mathcal{R}$. Teams are modelled as so called *Role/Delegation (R/D) nets*. A R/D net is a Petri net (P, T, F) where each task is modelled by a place p and each task implementation (delegation/execution) is modelled by a transition t. A place p with ${}^\bullet p = \emptyset$ models an *initial task*, while ${}^\bullet p \neq \emptyset$ models a *subtask*. Transitions $t \in T$ with $t^\bullet \neq \emptyset$ are called delegative, transitions with $t^\bullet = \emptyset$ are called executive. Each place p is labelled by a role $R(p)$ and each transition t with a DWF net $D(t)$. Since teams are trees it is a natural restriction to allow exactly one place in the preset of a delegation: $|{}^\bullet t| = 1$. Also we assume for teams that the model is a causal net, which guarantees that the delegation process is conflict free. An example for an R/D net is given Figure 5 (the named boxes around places and transitions will be explained in the following section).

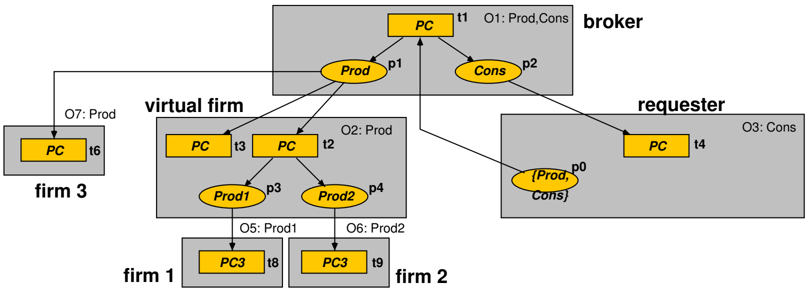


Fig. 5. Producer/Consumer Organisation

Definition 3. *A* R/D net *is the tuple* (N, R, D) *where:*

1. $N = (P, T, F)$ *is a Petri net with* $|p^\bullet| > 0$ *for* $p \in P$ *and* $|{}^\bullet t| = 1$ *for* $t \in T$.
2. $R : P \to \mathcal{R}$ *is the role assignment.*
3. $D : T \to \mathcal{D}$ *is the DWF net assignment.*

An R/D net is called a team net *if* N *is a connected causal net with exactly one minimal node:* $|{}^\circ N| = 1$.

In a team net all maximal nodes (i.e. the leaves) are transitions: $N^\circ \subseteq T$. Since $|{}^\bullet t| = 1$ by definition there exists a place p with ${}^\bullet t = \{p\}$.

In a well-formed organisation the roles of the DWF net $D(t)$ are consistently related to the roles of the places in the preset and the postset such that no role behaviour is lost or added during the delegation.

Definition 4. *A R/D net (N, R, D) is* well-formed *if we have for all $t \in T$:*

1. *Static role compatibility. The roles in the preset belong to the DWF net: $R(^\bullet t) \subseteq R(D(t))$*
2. *Role partition. Whenever $|t^\bullet| > 0$ holds, then the roles $R(p)$ are disjoint for all $p \in t^\bullet$ and there is a DWF net implementing $R(t^\bullet)$, i.e. $\mathcal{D}(R(t^\bullet)) \neq \emptyset$.*
3. *Behaviour refinement. Whenever $|t^\bullet| > 0$ holds, then there exists for each $D_t \in \mathcal{D}(R(t^\bullet))$ a partition $(P_r)_{r \in R(^\bullet t)}$ of the postset $t^\bullet$, such that for all $r \in R(^\bullet t)$ we have the behaviour refinement $D_t[R(P_r)] \simeq D(t)[\{r\}]$.*

The R/D net given in Figure 5 is well formed, since the initial place p_0 requests for an interaction of Prod and Cons. Activity t_1 delegates both roles via p_1 and p_2 to t_2 and t_4, respectively. The role Prod is refined properly by t_2 since the roles in its postset are Prod_1 and Prod_2 which are together behaviour-equivalent to the role Prod.

2.3 Organisations

In the following we like to characterise team nets as the result of a team formation which is controlled by an organisation (cf. Theorem 3 below). The organisation structure is built up by *organisational positions.*[2] Each position is responsible for the delegation/execution of several tasks and can delegate subtasks to other positions. Positions are modelled as disjoint subsets of $P \cup T$ of a R/D net.

Since each position decides autonomously which subtasks it generates we request that whenever t belongs to a position, then all generated subtasks $p \in t^\bullet$ belong to it, too, and whenever p belongs to a position, so are all $t \in {}^\bullet p$. As in R/D nets the transitions are related to DWF nets and the places with roles.

The initial tasks (i.e. the places p with $^\bullet p = \emptyset$) are the starting points of organisational activity.

Definition 5. *A* (formal) organisation net *is the tuple $Org = (N, \mathcal{O}, R, D)$ where (N, R, D) is a R/D net with $N = (P, T, F)$ and $\mathcal{O}$ is a partition on the set $P \cup T$ where all $O \in \mathcal{O}$ satisfy the following delegation constraint (with $\bar{O} := (P \cup T) \setminus O$):*

$$\forall p \in O \cap P : {}^\bullet p \subseteq O \wedge p^\bullet \subseteq \bar{O} \quad \wedge \quad \forall t \in O \cap T : {}^\bullet t \subseteq \bar{O} \wedge t^\bullet \subseteq O$$

An element $O \in \mathcal{O}$ is called position. *For each node $n \in (P \cup T)$ the uniquely defined position to which n belongs is denoted $O(n)$.*

The set of all positions in the postset of a place p is denoted by $\mathcal{O}_{dlg}(p)$. It is the set of all positions to which $O(p)$ may delegate p to. For example in Figure 5 the position O_3 may delegate p_0 only to O_1, i.e. $\mathcal{O}_{dlg}(p_0) = \{O_1\}$.

[2] It is useful to distinguish between the type and the implementation to keep the system architecture clean. In our concrete MAS architecture derived from a SONAR-model each position is implemented by two agents: One models the *formal position* and its duties and another models the *"employee"* implementing them.

There is a close connection between organisation nets and the commonly used organisation charts. In fact, organisation charts are a special sub-case of our model. Organisation nets encode the information about delegation structures – similar to charts – and also about the delegation/execution choices of tasks, which is not present in charts. If one fuses all nodes of each position $O \in \mathcal{O}$ into one single node, one obtains a graph which represents the organisation's chart. Obviously, this construction removes all information about the organisational processes. The chart in the introduction (Figure 2) represents the result when applying the construction to the organisation net in Figure 5.

Let $(N, \mathcal{O}, R, D)$ with $N = (P, T, F)$ be an organisation net. Those firing sequences $w \in T^+$ that fire a marking m to the empty marking $\mathbf{0}$ are called *task delegation sequences*, because after w has fired the net is empty, i.e. all tasks have been assigned to positions. Thus, each task delegation sequences models a team formation.

For the net in Fig. 5 we have $t_1t_3t_4$, $t_1t_6t_4$, and $t_1t_2t_8t_9t_4$ as task delegation sequences (modulo permutation of concurrent transitions). Since we like to know whether it is possible to generate teams for all tasks, it is a natural question to ask whether markings enable task sequences, or equivalent, whether $\mathbf{0} \in RS(m)$ holds (where $RS(m)$ denotes the set of markings reachable from m).

Definition 6. *Let (N, R, D) be a R/D net.*

– *The marking m is called* processable *iff $\mathbf{0} \in RS(m)$ holds.*
– *The marking m is called* strongly processable *iff all $m' \in RS(m)$ are processable.*
– *(N, R, D) is called* (strongly) processable *iff all its markings are (strongly) processable.*

Note, that R/D nets are unbounded in general and reachability may become a complex question to decide. Fortunately, due to the special tree-like structure we have the following properties (cf. Theorem 3.3 in [16]).

Theorem 2. *1. If the marking m is strongly processable, then it is processable.*
2. (N, R, D) is strongly processable iff it is processable.
3. A marking m is processable iff the markings $\{p\}$ with $m(p) > 0$ are.
4. It is decidable in linear time in the size of the R/D net N whether the marking m is (strongly) processable.
5. If all markings $\{p\}$ with $p \in P$ are processable, then N is strongly processable.

3 The Organisation in Action

It is obvious that the organisational SONAR-model can be seen as a distributed actor network where the positions $O \in \mathcal{O}$ are the actors/agents. This agent is denoted by O, too, in the following. The subnet $(P \cap O, T \cap O, F \cap O^2)$ represents the functionality of each agent O.

The actor network generated from the organisation net in Figure 5 is the system described in the introduction (cf. Figure 2). E.g. the position O_2 has to implement the role Prod which we obtain by computing $R({}^\bullet O_2)$. The activities O_2 has to implement is $PC[\mathsf{Prod}]$, computed from $D(t_3)[R(t_3)]$ where t_3 is the only executive transition of O_2.

These position agents are implemented within the Mulan framework [17] since Mulan is a Petri net based formalism which narrows the gap between model and implementation, but in principle any agent-oriented programming language could be adapted. The position agents are responsible mainly for three basic kinds of organisational coordination: team formation, distributed planning, and (on the meta-level) organisational transformation.

3.1 Team-Formation

The first phase of teamwork deals with team formation. Team nets can be characterised as Petri net processes of an organisation net. Petri net processes (cf. [15]) are a recognised alternative for describing the behaviour of Petri nets by firing sequences. Processes are themselves Petri nets from the class of *causal nets*, where no branching is allowed for the places. A *process* of a net N is defined as a causal net K together with a net morphism $\phi = (\phi_P, \phi_T)$.

A process has the *progress property* iff no transition is enabled in K°, i.e. for each subset $X \subseteq K^\circ$ there is no transition $t \in T$ such that $\phi(X) = {}^\bullet t$. The set of all finite processes with the progress property is denoted $\mathcal{K}(N, m)$.

The following theorem states that each process with the progress property introduces a team net. Given a process (K, ϕ) we define the mappings D and R as $R_G(b) = R(\phi(b))$ and $D_G(e) = D(\phi(b))$ (cf. Theorem 4.2 in [16]).

Theorem 3. *Let $Org = (N, \mathcal{O}, R, D)$ be a formal organisation. Then for each $p \in P$ and each process $(K, \phi) \in \mathcal{K}(N, \{p\})$ we have that $G(K, \phi) := (K, (R \circ \phi), (D \circ \phi))$ is a team net. If Org is well formed, then $G(K, \phi)$ is, too.*

This theorem allows us to characterise team formation processes in terms of the theory of Petri net processes: Each Petri net process of a formal organisation generates a team net.

Alternatively, a process (K, ϕ) can be constructed from the possible firings, i.e. the enablement of transitions, of the net N. The construction is inductively defined for a process net, by adding transitions according to the enabling condition of the net N. The starting point is given by the initial marking, which defines a simple process without any transitions, but only a place for each token in the initial marking. For the progress property this unfolding is continued until no transition is enabled.

This construction leads to the implementation of a *distributed team formation algorithm.* In our model, team nets are formed through an iterated delegation process. Whenever an agent O has been chosen as a team member to implement the role belonging to a place b with $\phi(b) \in {}^\bullet O$ it selects one of the implementation possibilities, i.e.

$$t \in (\phi(b)^\bullet \cap O)$$

This choice is equivalent to the extension of the process already constructed by an event e with $\phi(e) = t$. After choosing one element t such that $t^\bullet \neq \emptyset$ holds, the agent O further chooses one team member $O^{b'}$ from the set $\mathcal{O}_{dlg}(\phi(b'))$ for each $b' \in e^\bullet$. This delegation process as a whole corresponds to the construction of the possible firings of an organisation net and thus constructs the net's processes and hence team nets. The distributed team formation algorithm is given in pseudo-code in Figure 6. Given an initial task the agents in the algorithm compute a team net G, that assigns tasks to positions in a distributed manner.

```
when agent $O$ receives $(K, \phi, b)$ do
    choose $t \in (\phi(b)^\bullet \cap O)$
    $B'_t = \{b_p \mid p \in t^\bullet\}$
    $(K', \phi') := ((B', E', F'), \phi')$ where // extend process by $t$:
        $B' = B_K \uplus B'_t, \quad F' = F_K \cup \{(b, e)\} \cup (\{e\} \times B'_t)$
        $E' = E_K \uplus \{e\}, \quad \phi' = \phi \cup \{(e, t)\} \cup \{(b_p, p) \mid p \in t^\bullet\}$
    for each $b' \in B'_t$ do
        choose $O^{b'} \in \mathcal{O}_{dlg}(\phi(b'))$
        send $(K', \phi', b')$ to $O^{b'}$

initially choose $p \in P$ with $R(p) = R$ and ${}^\bullet p = \emptyset$
    $(K_0, \phi_0) := ((\{b_0\}, \emptyset, \emptyset), \{(b_0, p)\})$
    send $(K_0, \phi_0, b_0)$ to $O(p)$
```

Fig. 6. Team-formation as a distributed algorithm

Example 1. If we like to implement the initial task p_0 of Figure 5 we have $O_3 = O(p_0)$ as the starting agent. O_3 passes control to O_1, the only neighbour: $\mathcal{O}_{dlg}(p_0) = \{O_1\}$. The agent O_1 generates two subtasks: p_1 and p_2 according to t_1. For each of these subtasks one neighbour agent has to be activated. For p_1 the agent O_1 has to choose between O_7 and O_2, for p_2 we have only O_3. O_1 chooses O_7 and O_3 according to t_6 and t_4 which become activated but cannot delegate any further. So the algorithm terminates. The generated team corresponds to the task delegation sequence $t_1 t_6 t_4$ (modulo permutation): $G_{t_1 t_6 t_4}$. The team net equals the net that is obtained from the net in Figure 5 by restricting it to P and t_1, t_6, and t_4.

3.2 Team-Planning: Distributed Coordination

The local planning of an agent A has to be consistent with all other agents' plans, the team G, and sometimes with the team's constraints (if any). This process is usually called *distributed coordination.*

The set of maximal nodes K° of a team $G = (N, R, D)$ induces a special DWF net, called the *team-DWF* $D(G)$. Each transition $t \in N^\circ$ is associated to the DWF net $D(t)$ and the role $R({}^\bullet t)$. The team-DWF is the composition of all role components $D(t)[R({}^\bullet t)]$:

$$D(G) := \Big\|_{t \in N^\circ} D(t)[R({}^\bullet t)] \tag{3}$$

For well formed R/D nets this composition is always well-defined.

If we look at the team $G_{t_1t_6t_4}$ from above, we obtain the team-DWF:

$$D(G_{t_1t_6t_4}) = PC_3[\mathsf{Prod}_1] \| PC_3[\mathsf{Prod}_2] \| PC[\mathsf{Cons}] = PC_3$$

For each node $n \in P \cup T$ of the team G we define a set of *compromise plans* $CP(n)$. These sets are defined recursively over the structure of the team net: The maximal nodes $t \in N^\circ$ of a team G have no further constraints. Therefore each Agent $O(t)$ may choose any process of the set of all workflow processes $Proc^{wf}(D(G))$ as its plan: $CP(e) := Proc^{wf}(D(G))$ for $t \in N^\circ$.

For each place p the compromise set is defined as equal to that of its postset: $CP(b) := CP(b^\bullet)$. Each agent corresponding to an inner transition t of the team G chooses a subset of the intersection of all the compromise sets in its postset:

$$\xi_{O(t)}\Big(\{CP(p) \mid p \in t^\bullet\}\Big) \subseteq \bigcap_{p \in t^\bullet} CP(p)$$

The construction of compromise sets therefore relies on the family of mappings $(\xi_O)_{O \in \mathcal{O}}$ to reduce the space of compromises. Formally, we have:

$$\begin{aligned} CP(p) &:= CP(p^\bullet) \\ CP(t) &:= \begin{cases} Proc^{wf}(D(G)), & \text{if } t \in N^\circ \\ \xi_{O(t)}\big(\{CP(p) \mid p \in e^\bullet\}\big) & \text{if } t \in T_N \setminus N^\circ \end{cases} \end{aligned} \tag{4}$$

The team compromise set $CP(G)$ is the set finally computed for the root: $CP(G) := CP({}^\circ N)$. The team planning has been successful whenever there is at least one plan as a team compromise, i.e. $CP(G) \neq \emptyset$. Whenever there is no such compromise for the team, the agents have to adapt the mappings $(\xi_O)_{O \in \mathcal{O}}$ in a distributed negotiation phase (we omit details here).

```
when agent $O(t)$ receives $CP$ from agent $O(p), p \in t^\bullet$ then
    $CP_{O,p} := CP$

when agent $O(t)$ has received $CP$ from all agent $O(p), p \in t^\bullet$ then
    $CP_{O,t} := \begin{cases} Proc^{wf}(D(G)), & \text{if } t \in N^\circ \\ \xi_{O(t)}(\{CP(p) \mid p \in e^\bullet\}) & \text{if } t \in T_N \setminus N^\circ \end{cases}$
    send $CP_{O,t}$ to $O({}^\bullet t)$
```

Fig. 7. Team-planning as a distributed algorithm

This calculation directly leads to a simple distributed algorithm (cf. Figure 7) which can be seen as an abstract version of the well known partial global planning protocol [18]. Each agent has the variables $CP_{O,n}$ for all nodes belonging to O, i.e. for all $n \in O$. The messages flow from the leaves to the root of the team.

Example 2. Look at the team $G_{t_1 t_6 t_4}$ from above. The leaves of $G_{t_1 t_6 t_4}$ are the transitions t_8, t_9, and t_4. The composition of all these components is the team-DWF $D(G_{t_1 t_6 t_4}) = PC_3 = PC_3[\mathsf{Prod}_1] \| PC_3[\mathsf{Prod}_2] \| PC[\mathsf{Cons}]$.

The tree of the team has the following structure (with leaves at the left and the root at the right):

t_8	$PC_3[\mathsf{Prod}_1]$	t_2	$PC_3[\mathsf{Prod}_1, \mathsf{Prod}_2]$	t_1	PC_3
t_9	$PC_3[\mathsf{Prod}_2]$				
		t_4	$PC[\mathsf{Cons}]$		

Coordination processes this trees from the leaves to the root: The agent O_2, which implements the delegation t_2, integrates the compromise sets $CP(t_8)$ and $CP(t_9)$ of its child nodes into $CP(t_2)$. Due to Theorem 1 the set $CP(t_2)$ contains only processes of the DWF net

$$PC_3[\mathsf{Prod}_1] \| PC_3[\mathsf{Prod}_2] = PC_3[\mathsf{Prod}_1, \mathsf{Prod}_2].$$

In the second step $CP(t_2)$ and $CP(t_4)$ are combined by the agent O_1 into $CP(t_1) \subseteq CP(t_2) \cap CP(t_4)$.

3.3 Teamwork Parameters

Both aspects – formation and planning – can be realised in different ways which can be combined in an independent manner. *Team formation* can be parameterised by the degree of obligation. Possible alternatives are:

1. Each position agent has to accept each task assigned to him.
2. Before assigning task to others each agent asks his neighbours to evaluate the costs. After this biding phase the delegating agent chooses the cheapest agent who must not refuse.
3. Agents even have the freedom to reject an assignment after the bidding provided that there is at least one agent willing to receive the assignment.

Similarly, *compromise planning* can be parameterised in different ways:

1. Agents have the possibility to restrict the plans of agents of lower authority, i.e. restriction is related to the organisation structure.
2. Agents have the possibility to restrict the plans of those agents which are in lower branches of the team.
3. No agent has the possibility to restrict the other agents' plans. Compromise negotiation is a peer to peer process.

The team formation process presented in this paper is based on direct assignment while the team planning relies on team based restriction where the parent positions restrict the options of the child positions.

These parameters are independent from each other leading to a two dimensional scheme for organisational teamwork:

planning / formation	organisation based	team based	peer to peer
direct assignment	hierarchies		
bidding		federations	
reject option			markets

The common modes of teamwork – like hierarchies, federations, and markets (cf. [7] for other) – can be identified as instances of this scheme.

4 Transformation: Business Reorganisation Processes

So far we have considered a formal organisation as a context that frames the behaviour of the actors (i.e. software agents) occupying positions in the organisational framework. Now we turn to processes that target at the organisation itself. In this respect, we distinguish organisational processes of first and of second order. First-order processes have already been examined: Teams are formed to accomplish some task and the organisation is referred to as a static context. Second-order processes, on the other hand, are reorganisation processes that transform the organisation, which is consequently referred to as a variable. These second-order processes introduce the reflective character into our model.

Second-order processes fit neatly into our approach of teamwork-based organisational processes. Organisational transformations are carried out by *transformation teams*. Just as teams for first-order processes they are generated by Petri net processes of organisation nets. Each transition of *second order DWF-nets* additionally carries a *transformation instruction*.

We cannot give full details on the formal apparatus behind transformation and transformation teams in our model. Instead, we return to our running example of the production scenario in order to illustrate the basic concepts. Up to now, the requester part has no inner substructure. Requesting some product as the starting point and consuming this product upon reception is subsumed under one single position. Figure 8 illustrates a new evolution stage in the production scenario where the requester has been reorganised.

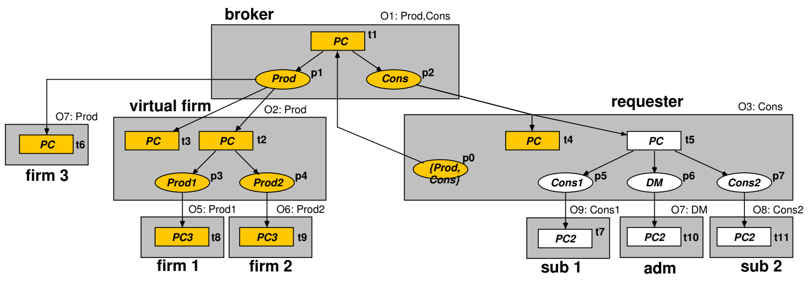


Fig. 8. Refined Organisation

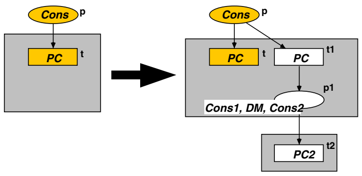


Fig. 9. Alternative Refining: ρ_1

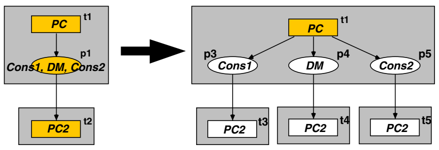


Fig. 10. Delegation Splitting: ρ_2

First of all, the original requester may now decide whether it consumes the requested product himself or whether it delegates the consumption (transformation step *alternative refining*, cf. Figure 9). For example, imagine the case where the supplier delivers micro processors that the requester incorporates into personal computers as end-products. Now the requester has decided to expand its business and order different kinds of micro processors for assemblage. But as it cannot accomplish the higher workload itself it has to delegate some of the work to subordinates.

As a second step, the delegated part of the work in Figure 8 comprises different subtasks and thus induces different subordinates in different roles (transformation step *delegation splitting*, cf. Figure 10). Referring to the micro processor example, one particular type might be delivered in varying production quality (Does anyone remember the 486SX which was the 486DX with a malfunctioning float pointing unit?). The quality of a particular processor decides upon the means of assemblage.[3] Thus, in addition to different positions for assemblage there exists one position to judge the quality of incoming products.

We assume that the DWF net PC_2 exists (not presented here) that refines the role Consumer in the net PC of Figure 3: The role Decision Maker (DM) decides whether $\mathsf{Consumer}_1$ or $\mathsf{Consumer}_2$ (short: Cons_1 and Cons_2, resp.) receives the item. The DWF net PC_2 has to be a refinement of PC with respect to communication behaviour: $PC[\mathsf{Cons}] \simeq PC_2[\mathsf{Cons}_1, \mathsf{DM}, \mathsf{Cons}_2]$.

The transformation of the organisation is based on net rewriting as a special form of graph transformation [19]. We demand our transformation rules to preserve the well-formedness of organisations nets. Figure 9 and 10 show two such rules ρ_1 and ρ_2 which preserve well-formedness. The first rule ρ_1 takes a delegation from p to t and introduces an alternative delegation transition t_1 which implements the role Cons by delegating to the roles Cons_1, DM, and Cons_2 which are implemented by the new transition t_2 which is labelled by the protocol PC_2. The well-formedness is preserved since the role-component $PC_2[\mathsf{Cons}_1, \mathsf{DM}, \mathsf{Cons}_2]$ is an interaction refinement of $PC[\mathsf{Cons}]$. The second rule ρ_2 splits the delegation from a transition t_1 to a place labelled with the three roles $\mathsf{Cons}_1, \mathsf{DM}, \mathsf{Cons}_2$ to three places labelled with one of the three nodes: $R(\{p_1\}) = R(\{p_3, p_4, p_5\})$. The three activities are executed independently by the same DWF net that has to equal to the original one $D(t_2)$, i.e. $D(t_2) = D(t_3) = D(t_4) = D(t_5)$. The

[3] Of course there has to exist an agreement between the requester and supplier as to whether and to what conditions the supplier is willing to accept lower quality products. We do not address such subtle issues here.

well-formedness is preserved since the role-component $PC_2[\mathsf{Cons}_1, \mathsf{DM}, \mathsf{Cons}_2]$ is interaction equivalent to the composition $PC_2[\mathsf{Cons}_1] \| PC_2[\mathsf{DM}] \| PC_2[\mathsf{Cons}_2]$.

The refined organisation of Figure 8 is generated by applying ρ_1 and then r_2 to the organisation in Figure 5 and is therefore well formed too.

5 Conclusion and Outlook

We have presented a Petri net-based approach to model formal organisations. The resulting formal organisational model, called Sonar, integrates structural as well as functional and interactional features of organisations. In this respect, we provide a structuring metaphor for distributed information systems and at the same time a formal model of distributed business processes that comprise participants in different roles and contexts.

In addition, each model of a formal organisation implicates a network of actors/agents to actually carry out the organisational specifications. We establish a close link between organisational specifications and their deployment as multi-agent systems. Not only business processes for task accomplishment themselves but also their genesis in the form of team formation and team planning are described by distributed algorithms that operate on the underlying Petri net model of the corresponding formal organisation.

Business processes are carried out by teams. The organisation may be seen as a network of actors (embodied by its associated network of position agents) that carries out first-level business processes to accomplish business tasks as well as second-level business processes (i.e. business reorganisation) to *reflectively* transform itself. Thus, we arrive at a self-contained model for the organisation of process-aware information systems.

Turning to future work, we consider different issues related to our intention towards the network of agents as an effective distributed and domain-independent level of implementation (middleware). In the context of this paper it was sufficient to associate each position with exactly one agent. But in the case of an *open system* environment with agents belonging to different stakeholders entering and leaving on a continual basis, we propose to refine this agent into a position/member pair. The organisation contracts each position to a member agent that carries out actions and makes decisions. But member agents must connect to the corresponding position agents and use them as (controlling and coordinating) *proxies* to the organisation. Consequently, this approach distinguishes between the organisational and the domain layer of applications. In addition our model allows for further refinement of positions to (sub)organisations.

References

1. Smith, R.G.: The contract net: A formalism for the control of distributed problem solving. In: 5th Conference on Artificial Intelligence (IJCAI 1977) (1977)
2. zur Mühlen, M.: Evaluation of workflow management systems using meta models. In: International Conference on System Sciences, vol. 5. IEEE CS, Los Alamitos (1999)

3. van der Aalst, W.V.D.: Verification of workflow nets. In: [20], pp. 407–426
4. Durfee, E.H., Lesser, V.R., Corkill, D.D.: Trends in cooperative distributed problem solving. Transactions on Knowledge and Data Engineering 1, 63–83 (1989)
5. Ellis, C.A., Wainer, J.: Groupware and computer supported cooperative work. In: Weiß, G. (ed.) Multiagent systems, pp. 425–458. MIT Press, Cambridge (1999)
6. Ossowski, S.: Co-ordination in Artificial Agent Societies. Springer, Heidelberg (1999)
7. Horling, B., Lesser, V.: A survey of multi-agent organizational paradigms. The Knowledge Engineering Review 19(4), 281–316 (2005)
8. Prietula, M.J., Carley, K.M., Gasser, L. (eds.): Simulating Organisations. Computational Models of Institutions and Groups. AAAI/MIT-Press (1998)
9. Boissier, O., Hübner, J.F., Sichman, J.S.: Organization oriented programming: From closed to open organizations. In: O'Hare, G.M.P., Ricci, A., O'Grady, M.J., Dikenelli, O. (eds.) ESAW 2006. LNCS, vol. 4457, pp. 86–105. Springer, Heidelberg (2007)
10. Rinderle-Ma, S., Reichert, M.: A formal framework for adaptive access control models. In: Spaccapietra, S., Atzeni, P., Fages, F., Hacid, M.-S., Kifer, M., Mylopoulos, J., Pernici, B., Shvaiko, P., Trujillo, J., Zaihrayeu, I. (eds.) Journal on Data Semantics IX. LNCS, vol. 4601, pp. 82–112. Springer, Heidelberg (2007)
11. Klarmann, J.: A comprehensive support for changes in organizational models of workflow management systems. In: 4th International Conference on Information Systems Modelling (2001)
12. van der Aalst, W.V.D.: Interorganizational workflows Systems Analysis - Modelling. Simulation 34(3), 335–367 (1999)
13. Kindler, E.: A compositional partial order semantics for Petri net components. In: [20]
14. Martens, A.: Analyzing web service based business processes. In: Cerioli, M. (ed.) FASE 2005. LNCS, vol. 3442, pp. 19–33. Springer, Heidelberg (2005)
15. Goltz, U., Reisig, W.: The non-sequential behaviour of Petri nets. Information and Control 57, 125–147 (1983)
16. Köhler, M.: A formal model of multi-agent organisations. Fundamenta Informaticae 79(3-4), 415–430 (2007)
17. Köhler, M., Moldt, D., Rölke, H.: Modeling the behaviour of Petri net agents. In: Colom, J.-M., Koutny, M. (eds.) ICATPN 2001. LNCS, vol. 2075, pp. 224–241. Springer, Heidelberg (2001)
18. Durfee, E.H., Lesser, V.R.: Partial global planning. IEEE Transactions on Systems, Man, and Cybernetics 21(5), 1167–1183 (1991)
19. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of algebraic graph transformation. Springer, Heidelberg (2006)
20. Azéma, P., Balbo, G. (eds.): ICATPN 1997. LNCS, vol. 1248. Springer, Heidelberg (1997)

Flexibility in Process-Aware Information Systems

Manfred Reichert, Stefanie Rinderle-Ma, and Peter Dadam

Institute of Databases and Information Systems, Ulm University, Germany
{manfred.reichert,stefanie.rinderle,peter.dadam}@uni-ulm.de

Abstract. Process-aware information systems (PAIS) must be able to deal with uncertainty, exceptional situations, and environmental changes. Needed business agility is often hindered by the lacking flexibility of existing PAIS. Once a process is implemented, its logic cannot be adapted or refined anymore. This often leads to rigid behavior or gaps between real-world processes and implemented ones. In response to this drawback, adaptive PAIS have emerged, which allow to dynamically adapt or evolve the structure of process models under execution. This paper deals with fundamental challenges related to structural process changes, discusses how existing approaches deal with them, and shows how the various problems have been exterminated in ADEPT2 change framework. We also survey existing approaches fostering flexible process support.

1 Introduction

In many application domains process-aware information systems (PAIS) will be not accepted by users if rigidity comes with them [1,2,3,4]. Instead, it should be possible to quickly implement new processes, to enable on-the-fly adaptations of running ones, to defer decisions regarding the exact process logic to runtime, and to evolve implemented processes over time. Consequently, process flexibility has been identified as one of the fundamental needs for any PAIS and different enabling technologies have emerged [5,6,7,8]. They support adaptive processes [9,10,11], declarative models [7], late modeling [12,13], and data-driven processes [14,15]. Basically, we need to be able to deal with uncertainty, to cope with exceptions, and to evolve processes over time:

– *Ability to deal with uncertainty.* The implemented process is based on a loosely or partially specified model, where the full specification is made during runtime and may be unique to each process instance. Rather than enforcing control through a rigid, or highly prescriptive model, that attempts to capture every aspect, the model is defined in a more declarative or incomplete way that allows individual instances to determine their own processes.
– *Ability to adapt processes.* The implemented process is able to react to exceptions, which may or may not be foreseen and which affect one or a few instances. Generally, it must be possible to adapt the structure and/or state of the process model of a particular instance. Respective adaptations, however, must not affect other instances being executed on this model as well.

– *Ability to evolve processes.* A process model has to be changed when the business process evolves. One challenge concerns the handling of long-running, active instances, which were initiated based on the old model, but now need to comply with the new specification. Potentially, thousands of active instances may be affected.

This paper focuses on structural adaptations of process models at different levels. Adaptations of single process instances (e.g., to add, delete or move activities) become necessary to deal with exceptional situations and often have to be accomplished in an ad-hoc manner [11]. Model changes at the process type level, in turn, have to be continuously conducted to evolve the PAIS [9,5]. It must be also possible to dynamically migrate running process instances to new model versions. Important challenges are to perform instance migrations on-the-fly, to guarantee compliance of migrated instances with the new model version, and to avoid performance penalties. Our ADEPT2 change framework addresses these challenges and explicitly covers the latter two kinds of flexibility; i.e., the adaptation and evolution of processes. However, through its ability to support late binding of sub-processes and to dynamically evolve or define these sub-processes, ADEPT2 is also able to support late modeling, and thus to deal with certain kinds of uncertainty.

The ultimate ambition of structural process adaptations during runtime is to ensure correctness of the modified instances afterwards. First, structural and behavioral soundness have to be guaranteed already at the model level (i.e., without considering instance states). Second, when performing instance adaptations this must not lead to flaws (e.g., deadlocks); i.e., none of the guarantees ensured by formal checks at build time must be violated due to the runtime adaptation. As example consider Fig. 1 where the model on the left-hand side is structurally modified by arranging parallel activities `B` and `C` in sequence afterwards. The instance running on the old model (with `B` being enabled and `C` being completed) does not comply with the new model version since its marking cannot be transferred to it (`B` must be completed before `C` may start). Such undesired runtime situations are denoted as *dynamic change bug* [16]. To exterminate them adequate correctness criteria are needed; e.g., to decide whether a given process instance is compliant with a modified process model and – if yes – how to adapt instance states when migrating the instance to the new model version.

In the following we deal with different correctness notions for dynamic process changes and discuss the strengths and weaknesses of the approaches relying on

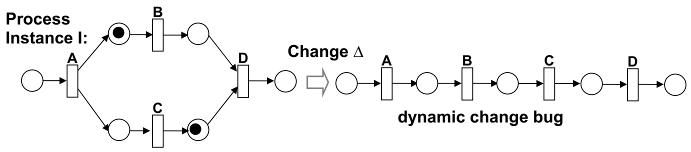


Fig. 1. Dynamic change bug

them. Based on these considerations we show how we deal with respective issues in ADEPT2, which constitutes one of the very few adaptive PAIS which allows for structural process changes during runtime at instance and type level. Section 2 introduces fundamental challenges emerging in the context of dynamic process changes and discusses existing approaches for dealing with them. Section 3 shows how ADEPT2 tackles the different challenges and exterminates dynamic change bugs. This includes both the control and the data flow perspective as well has the viewpoint of users. We survey alternative solutions for process flexibility in Section 4 and conclude with a summary in Section 5.

2 Fundamental Challenges of Dynamic Process Changes

2.1 Basic Notions

When implementing a new process in a PAIS its logic has to be explicitly defined based on the provided *process meta model.* For each *business process* to be supported, a *process type* represented by a *process schema* (i.e., *process model*) is defined. For one particular process type several schemes may exist representing the different versions and the evolution of this type over time. Based on a process schema an arbitrary number of new *process instances* can be created and executed. The PAIS orchestrates them according to the defined process logic.

For defining structural process adaptations two options exist. On the one hand, respective schema adaptations can be defined based on a set of change primitives (e.g., to add or delete edges). Following this approach, realization of a particular structural adaptation usually requires the application of multiple change primitives. To specify structural adaptations at this low level of abstraction, however, is a complex and error-prone task. Another, more favorable option is to base structural adaptations on high-level change patterns [6,17], which abstract from the concrete schema transformations to be conducted (e.g., to add a process fragment parallel to an activity or to move a fragment to a new position). Instead of specifying a set of change primitives the user applies one or few high-level change operations to define the required structural change.

Definition 1 (Process change). *Let $\mathcal{PS}$ be the set of all process schemas and let S, S' $\in \mathcal{PS}$. Let further Δ = $<op_1, \ldots, op_n>$ denote a process change which applies change operations op_i $i = 1, \ldots, n,\ n \in \mathbb{N}$ sequentially. Then:*

1. *$S[\Delta> S'$ if and only if Δ is correctly applicable to S. S' is the process schema resulting from the application of Δ to S (i.e., S' $\equiv$ S + Δ). We call a change Δ correctly applicable to a schema S if all formal pre-conditions of Δ are met for S or resulting schema S' is a correct process schema according to the correctness criteria set out by the process meta model of interest.*
2. *$S[\Delta>S'$ if and only if there are process schemas $S_1, S_2, \ldots, S_{n+1} \in \mathcal{PS}$ with $S = S_1$, $S' = S_{n+1}$ and for $1 \leq i \leq n$: $S_i[\Delta_i>S_{i+1}$ with Δ_i = $<op_i>$*

We assume that change Δ is applied to a *sound* process schema S [18]; i.e., S obeys the specific correctness constraints set out by the used process meta model (e.g., bipartite graph structure for Petri Nets). We denote this as *structural soundness*. We further claim that S' must obey *behavioral soundness*; i.e., any instance executed on S' must not run into a deadlock or livelock. This can be achieved in two ways. Either Δ itself preserves soundness based on pre-/post-conditions of the applied change patterns [11], or Δ is first applied on a schema copy and soundness of the resulting schema version S' is checked afterwards.

Another basic notion used in the following is *process trace*. Such trace sequentially logs the entries about the start and completion of process activities.

Definition 2 (Trace). *Let $\mathcal{PS}$ be the set of all process schemas and let $\mathcal{A}$ be the total set of activities (or more precisely activity labels) based on which process schemas $S \in \mathcal{PS}$ are specified (without loss of generality we assume unique labeling of activities). Let further $\mathcal{Q_S}$ denote the set of all possible traces producible on process schema $S \in \mathcal{PS}$. A particular trace $\sigma_I^S \in \mathcal{Q_S}$ of instance I on S is defined as $\sigma_I^S = < e_1, \ldots, e_k >$ (with $e_i \in$\{Start(a), End(a)\}, $a \in \mathcal{A}$, $i = 1, \ldots, k,\ k \in \mathbb{N}$). The temporal order of e_i in σ_I^S reflects the order in which activities were started and/or completed over S.*[1]

2.2 Under Which Conditions May Process Instances Be Adapted?

Most approaches dealing with structural instance adaptations [16,19,10,5,8] focus on correctness; i.e., applying a change to a running instance must neither violate its structural nor behavioral soundness. The correctness criteria used by adaptive PAIS vary and have led to different implementations [9]. Basically, there are *structural* and *behavioral* correctness criteria. While criteria from the former group try to structurally relate the process schema before the change to the resulting schema version [16,8] (e.g., using inheritance relations for realizing the schema mapping), the latter are based on execution traces; i.e., they compare which traces are producible on a process schema before and after its change.

Structural criteria. One approach relying on *structural criteria* in connection with dynamic changes exists for *WF Nets* [16]. A WF Net is a labeled place/transition net representing a control flow schema [16,20]. A *sound* WF Net has to be connected, safe, and deadlock free as well as free of dead transitions. Furthermore, sound WF Nets always properly terminate. Behavior of a process instance is described by a *marked WF net*. Core idea of the corresponding change framework is as follows: An instance I on schema S (represented by a marked WF Net) is considered as *compliant* with the modified schema $S' := S + \Delta$, if S and S' are related to each other under given *inheritance relations*; i.e., either S is a subclass of S' or vice versa. The following two kinds of inheritance relations are used [16]: A schema S is a subclass of another schema S' if one cannot distinguish behaviors of S and S' anymore either (1) when only executing activities of S which also belong to S' or (2) when arbitrary activities of S are executed, but only effects of activities being present in S' as well are taken into account. Thus, Inheritance Relation (1) works by *blocking* and Inheritance Relation (2) can be

[1] An entry of a particular activity can occur multiple times due to loopbacks.

realized by *hiding* a subset of the activities from S. More precisely, *blocked* activities are not considered for execution. *Hiding* activities implies that they are renamed to the silent activity τ. (A silent activity τ has no visible effects and is used, for example, for structuring purposes.) Consider the example from Fig. 2 where the newly inserted activities `X` and `Y` are hidden by labeling them to the silent activity τ. Thus, S' is a subclass of S. Further inheritance relations can be obtained by combined hiding and blocking of activities. Based on these inheritance relations we can state the following correctness criterion:

Criterion CC 1 (Compliance under inheritance relations) *Let S be a process schema which is correctly transformed into another schema S' by applying change Δ. Then instance I on S is compliant with S' if S and S' are related to each other under inheritance (see [16] for a formal definition).*

CC 1 ensures structural and behavioral soundness of instance I after applying change Δ to it. The question remains how to ensure CC 1; i.e., how to check whether Δ is an inheritance preserving change and therefore S and S' are related under inheritance. [16] defines precise *conditions* with respect to S and S'. When inserting a new net N into S, S and S' will be related under inheritance if N and S have exactly one place in common. This will be the case, for example, if a cyclic structure N_c is inserted into S (resulting in S') as shown in Fig. 2. Since S' is a subclass of S when hiding `X` and `Y` in N_c, soundness of I on S' is guaranteed. Checking inheritance of arbitrary process schemes is PSPACE-complete [16].

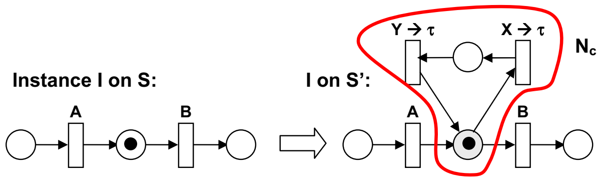


Fig. 2. Inheritance preserving change: insertion of cyclic structure

Using inheritance relations restricts the set of applicable changes to additive and subtractive ones. There is no adequate relation based on hiding/blocking activities in connection with order-changing operations. Nevertheless, this approach covers many relevant changes and copes with them <u>without</u> need for accessing instance states. It can be used for both correctness checks on single instances and on instance collections (e.g., WF Nets with colored tokens). It is debatable whether it also works with *concurrent changes*. Assume, for example, that instance I on S is changed resulting in instance-specific schema S_I, which is related to S under inheritance. Assume further that at process type level S is changed to S' (which is again under inheritance with S). Then it has to be analyzed whether S_I and S' are also related to each other under inheritance.

Behavioral criteria. A widely-used correctness property is the trace-based *compliance criterion* introduced by [19]. Intuitively, change Δ on schema S (i.e.,

$S[\Delta > S')$ can be correctly applied to instance I on S iff the execution of I, taken place so far, can be "simulated" on the new schema version S' as well. [19] bases compliance on *trying to replay trace* σ_I^S of I on S'. If this is possible, behavioral soundness can be guaranteed when migrating I to S' [19,5]. In summary, compliance is fundamental for changing both, single instances and instance collections. Basically, it also allows for concurrent changes. We discuss respective extensions in Section 3.5. Finally, the idea of preserving traces by structural changes based on Petri Nets is described in [21,10].

2.3 How to Adapt Instance States After Dynamic Changes?

In addition to decide whether change Δ can be correctly applied to an instance, it becomes necessary to properly and correctly adapt instance states afterwards.

Structural approaches. [16] provides *transfer rules* based on the aforementioned inheritance relations (cf. Criterion CC 1) to cope with marking adaptations in the context of WF net changes. After applying change Δ to schema S (i.e., $S[\Delta > S')$, necessary marking adaptations are realized by mapping markings of instances running on S onto markings on S'. Adapting markings after inserting parallel branches, for example, is complicated since in some cases we have to insert additional tokens to avoid deadlocks. Fig. 3 shows an example. By just mapping the token of s_3 on S to place s_3 on S', a deadlock is produced. [16] proposes to insert an additional token on s_5 (*progressive transfer rule*). Though the resulting marking on S' is correct, the semantics of newly inserted tokens is debatable, particularly, if colored tokens (i.e., data flows) are considered as well.

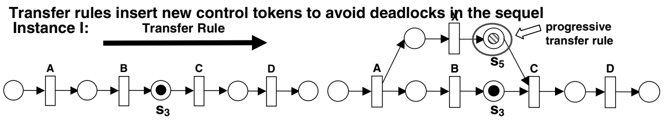

Transfer rules insert new control tokens to avoid deadlocks in the sequel
Instance I:
Transfer Rule
A
B
C
D
s3
A
X
B
s5
progressive transfer rule
C
D
s3

Fig. 3. Marking adaptation policy in [16,20]

Another approach has been proposed for Flow Nets [10] for which an explicit mapping between the markings of the net before and after the change has to be specified. This is done manually by adding *flow jumpers*; i.e., transitions mapping tokens from the old to the new net (cf. Fig. 4). Both single instances or instance collections can be migrated. The handling of concurrent changes at instance and type level, however, is cumbersome, since several new net versions have to be merged with the old net via flow jumpers. Manually specifying mappings between instance markings is not a realistic option in practice. As it can be see from Fig. 4 respective mappings already become complex for simple scenarios.

Behavioral approaches. Checking compliance means to replay instance traces on the changed process schema. Thus, marking adaptations come for free. However, at the presence of thousands of running instances, replaying whole traces becomes too expensive. In Sect. 3 we introduce a more sophisticated approach

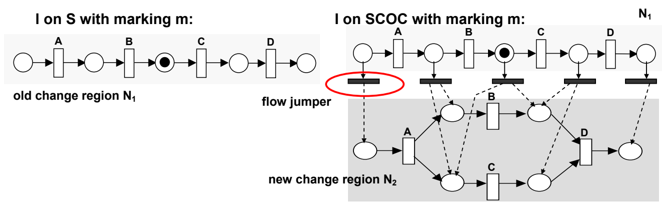


Fig. 4. Marking mapping (Synthetic Cut Over Change) [10]

for automatically checking compliance and adapting instance markings. It has been realized in ADEPT2 [5] and utilizes specific properties of the ADEPT2 meta model as well as the semantics of the ADEPT2 change patterns.

2.4 Discussion

Generally, a correctness criterion is needed which preserves structural and behavioral soundness of the dynamically adapted instances (cf. Fig. 5). This criterion should be valid independent from the used process meta model. Nonetheless, it is always applied in the context of a concrete meta model and change framework. Like serializability in database systems, defining a proper correctness notion is only one side of the coin. The other is to check it efficiently, particularly at the presence of a multitude of instances. When applying the criterion for a particular meta model, logical optimizations for checking it can be based on exploiting meta model properties as well as the semantics of the applied change operations. Additional optimizations are conceivable at the implementation level.

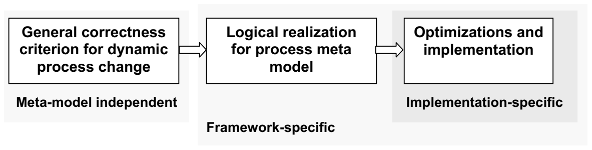

General correctness criterion for dynamic process change
Meta-model independent
Logical realization for process meta model
Framework-specific
Optimizations and implementation
Implementation-specific

Fig. 5. Correctness of process change – general view

3 Dynamic Process Changes in ADEPT2

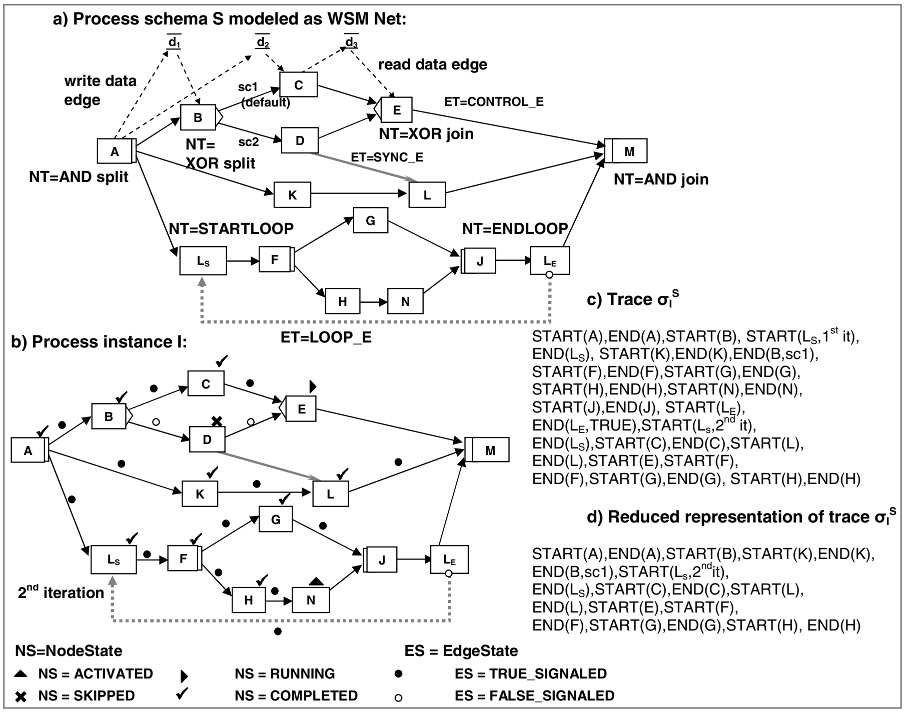
We now elaborate compliance as meta model independent correctness criterion in the context of a concrete process meta model (i.e., ADEPT2 WSM Nets [5]). We show how compliance can be efficiently checked and instance markings be automatically adapted when performing dynamic instance changes.

3.1 WSM Nets

Well-Structured Marking-Nets (WSM Nets) as applied in ADEPT2 can be used to represent process schemes by attributed serial-parallel graphs (cf. Fig. 6a).

Consider Fig. 6a, which depicts an example of a WSM Net. A WSM Net S is structurally sound if the following constraints hold: S has a unique *start* and a unique *end node*. Except for these start and end nodes each activity node has at least one incoming and one outgoing *control edge* $e \in CtrlE$[2]. Structuring nodes such as AND-Splits, XOR-Split, AND-Joins, and XOR-Joins can be distinguished based on their node type (6a). Loop backs can be explicitly modeled via loop edges $e \in LoopE$ (cf. Fig. 6a). Basically, WSM Nets are *block-structured*, where control blocks (sequences, branchings, loops) can be nested, but must not overlap. We additionally allow to relax this block structure and to synchronize the execution order of activities from parallel branches by means of so-called *sync links* $e \in SyncE$ if required. Such sync links must not cross the boundary of a loop block; i.e., an activity from a loop block must not be connected with an activity from outside the loop block via a sync link (and vice versa). Furthermore, $S_{fwd} = (N, CtrlE, SyncE)$ constitutes an acyclic graph which allows to exclude deadlocks due to cyclic "wait-for" dependencies.

For WSM Nets, data flow is realized by associating process data elements to activities by read and write edges (cf. Fig. 6a). For activities with mandatory



c) Trace σ_I^S

START(A),END(A),START(B), START(L_S,1st it),
END(L_S), START(K),END(K),END(B,sc1),
START(F),END(F),START(G),END(G),
START(H),END(H),START(N),END(N),
START(J),END(J), START(L_E),
END(L_E,TRUE),START(L_s,2nd it),
END(L_S),START(C),END(C),START(L),
END(L),START(E),START(F),
END(F),START(G),END(G), START(H),END(H)

d) Reduced representation of trace σ_I^S

START(A),END(A),START(B),START(K),END(K),
END(B,sc1),START(L_s,2nd it),
END(L_S),START(C),END(C),START(L),
END(L),START(E),START(F),
END(F),START(G),END(G),START(H), END(H)

Fig. 6. WSM Net with running instance, traces, and marking rules

[2] i.e.; S is connected.

input parameters linked to global data elements, it has to be ensured that respective data elements are always written by a preceding activity at runtime independent of which execution path is chosen.

Taking the WSM Net S from Fig. 6a new process instances can be created and executed (cf. Fig. 6b). Thereby, the execution state of an instance I is captured by marking function $\mathrm{M}^{S_I} = (\mathrm{NS}^{S_I}, \mathrm{ES}^{S_I})$ where S_I denotes the instance-specific schema of I. M^{S_I} assigns to each activity n its current status $NS(n)$ and to each edge e its current marking $ES(e)$. Markings are determined according to well defined firing rules. Based on the local context of an activity (i.e., incoming and outgoing edges), the activity marking can be determined [11]; markings of already passed regions and skipped branches are preserved (except loop backs). Activities marked as `Activated` are ready to fire (i.e., enabled) and can be worked on. Their status then changes to `Running` and afterwards to `Completed`. Activities belonging to non-selected execution branches obtain marking `Skipped` and can no longer be selected for execution (e.g., activity D in Fig. 6). Concerning data elements, different versions of a data object can be stored, which is important for the handling of partial rollback operations.

To cope with exceptional situations, instances can be individually modified by applying high-level change patterns (e.g., to insert or move activities). For such individually modified instances the *instance-specific* schema deviates from the original one they were started on. Respective instances are denoted as *biased*. To capture information about instance-specific changes, logically, each instance I runs on an instance-specific schema S_I with $S[\Delta_I > S_I$; Δ_I denotes the instance-specific *bias*. For unbiased instances, $\Delta_I = <>$ and consequently $S_I \equiv S$ hold. According to the change patterns framework presented in [6,22], Tab. 1 presents some *high-level change operations*, which can be used to define or structurally modify process schemes. A high-level change operation realizes a particular variant of a change pattern (e.g., serial or parallel insertion of activities). In ADEPT2 these change operations include formal pre- and post-conditions. They automatically perform necessary schema transformations while ensuring structural soundness. One typical example of such a change operation is the insertion of an activity and its embedding into the process context.

Currently, we are working on an extension of the ADEPT2 meta model to further increase expressiveness and to cover frequent workflow patterns (see [23] for details). Generally, there exists a trade-off between expressiveness of a meta model and support for structural adaptations in imperative approaches. ADEPT2 has been designed with the goal to enable the latter, i.e., to allow for the efficient implementation of adaptation patterns, restrictions on the process meta model are made. Similar restrictions in terms of expressiveness hold for other approaches supporting structural adaptations [24,8]. On the other hand, YAWL is a reference implementation for workflow patterns and therefore allows for a high degree of expressiveness [25]. Structural adaptations have not yet been addressed in YAWL and their implementation would be more difficult due to the higher expressiveness (see Section 4 for more details).

Table 1. A selection of high-level change operations on process schemas

Change pattern	**Design choice**	**Effects on schema S**
AP1: Insert activity		
	serial	inserts the activity between directly succeeding ones
	insert between node sets	
	without condition	inserts the activity parallel to existing ones
	with condition	conditional insert of the activity
AP2: Delete activity		deletes the activity from schema S
AP3: Move activity		
	serial	moves the activity to position between directly succeeding activities
	move between node sets	
	without condition	moves the activity parallel to existing ones
	with condition	conditional move of the activity

3.2 Checking Compliance in ADEPT2

In Section 2 two approaches for ensuring correctness of dynamically adapted instances are presented. CC 1 enables correctness checks for process changes without taking instance state into account. However, this comes for the price of a restricted set of change patterns (e.g., no order-changing operations). On the other side, traditional compliance [19] uses full instance information as captured by execution traces. Doing so allows for all kinds of change patterns. However, traditional compliance has turned out to be too restrictive (e.g., in conjunction with loops). Apart from this it is expensive to check. ADEPT2 follows an elegant compromise between these two compliance criteria abolishing their particular limitations. This is achieved by extending traditional compliance to overcome its restrictiveness. Furthermore, precise conditions for ensuring compliance are elaborated, which only take dedicated instance information into account. First of all, we formalize traditional compliance criterion CC 2:

Criterion CC 2 (Compliance of unbiased instances) *Let S be a sound process schema and let I be an unbiased process instance running on S with associated execution trace σ_I^S. Assume that change Δ transforms S into another sound process schema S' (i.e., $S[\Delta > S'$). Then: I will be compliant with S' (i.e., it can migrate to S') if its execution trace σ_I^S can be correctly replayed on S'.*

CC 2 depends on the representation (i.e. *view*) of trace σ_I^S. One is the `Start/End` view on σ_I^S. It logs both start and end events of executed activities (cf. Fig. 6c). Taking this view on σ_I^S we obtain an instance with correct marking when replaying it on S' [9]; i.e., I can continue execution based on S' afterwards while structural and behavioral soundness are preserved. However, this view is too restrictive in conjunction with changes of cyclic process structures [5]. If a loop is affected by a change, but has already undergone some iterations, the respective instance will be always considered as non-compliant with S' (i.e., trace entries related to finished iterations cannot be replayed on the adapted schema) though a migration of this instance would not lead to errors in the sequel. ADEPT2

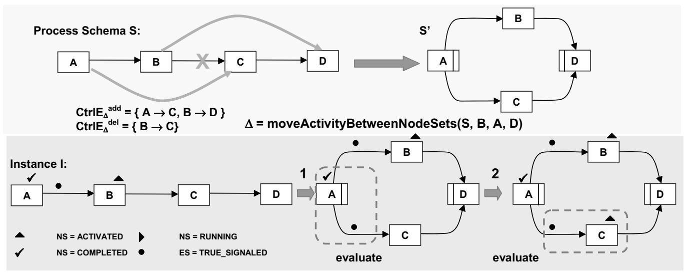

Process Schema S:
A
B
C
D
CtrlE$_\Delta^{add}$ = { A → C, B → D }
CtrlE$_\Delta^{del}$ = { B → C}
S'
Δ = moveActivityBetweenNodeSets(S, B, A, D)
Instance I:
1
2
evaluate
evaluate
NS = ACTIVATED
NS = COMPLETED
NS = RUNNING
ES = TRUE_SIGNALED

Fig. 7. Adapting markings for WSM Nets

therefore applies a *reduced representation* $\sigma_{I\,red}^{S}$ of σ_I^S, which corresponds to a (logical) projection of σ_I^S only on current loop iterations; i.e., for loop activities we only consider entries written during the last iteration of the respective loop (cf. Fig. 6d). Note that this approach is fostered by the block-structuring of WSM Nets. In addition, *data flow correctness* can be ensured by enriching execution traces with information about data access; i.e., read and write access on data elements. This is crucial in connection with dynamic process changes [5]. We dig into data flow correctness in Section 3.4.

CC 2 constitutes a logical correctness notion similar to serializability in database systems. Another challenge is to efficiently check it. A naive solution would be to try to replay instance traces on S' and to verify whether resulting instance states on S' are correct. Obviously, this can cause a performance penalty if a multitude of instances shall be migrated. Generally, a change framework has to provide methods which ensure CC 2 and can be efficiently checked. ADEPT2 provides methods which make use of the semantics of the applied change operations (cf. Tab. 1) and the model-inherent markings of WSM Nets for all change patterns supported [5,6]. Contrary to many approaches (e.g., [26,16]), ADEPT2 is able to deal with order-changing operations as well. (A discussion on the complexity of compliance checking for different change patterns and a comparison with other approaches can be found in [9].) As example consider Fig. 7 where activity B is moved to the position between activities A and D. Instead of replaying complete trace σ_I^S of I on S', according to the ADEPT2 *compliance conditions* for moving activities, the following has to be checked: I is compliant with S' if for all newly inserted control edges in $CtrlE_\Delta^{add}$ their destination activities are not running or completed yet. In the latter case (i.e., state of respective activities is `Running` or `Completed`), the state of the associated source node is `Completed` and compliance can be only ensured if the entries of source and destination node within trace $\sigma_{I\,red}^{S}$ have the right order (i.e., `END` entry of source node before `START` entry of destination node). For our example from Fig. 7, the destination activities of edges in $CtrlE_\Delta^{add}$ (i.e., C and D) have not been started yet. Consequently, activity B can be moved as described for instance I.

As can be seen from this example, moving activities is one of the few cases, where we might have to exploit additional information from trace $\sigma_{I\,red}^{S}$. In connection with newly added control edges, the associated orders must be already reflected by the entries of the trace. If the destination activities of the new control edges have not been started yet, the right order will be always guaranteed. Otherwise, the actual order has to be checked based on the execution trace. For inserting and deleting activities, checking node states is sufficient (see [27,28] for a complete summary of compliance conditions and respective proofs).

3.3 Adapting Instance Markings in ADEPT2

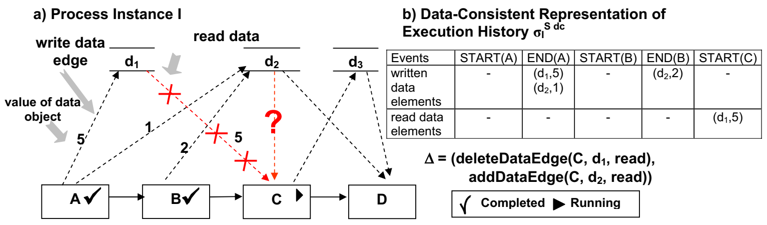
We have described how CC 2 can be ensured and which information is needed. Our goal was to prevent access to whole instance traces. By holding this maxim we now discuss how compliant instances can be automatically migrated to an adapted schema. One challenge, not adequately solved by other approaches, constitutes the efficient and correct adaptation of instance markings. According to CC 2, the marking of a migrated instance must be the same as it can be obtained when replaying its (reduced) trace on the new schema version. How extensive marking adaptations turn out depends on the kind and scope of the change. Except from initialization of newly added nodes and edges, no marking adaptations become necessary if the instance has not yet entered the change region. In other cases more extensive marking adaptations are required. An activated activity `X`, for example, will have to be deactivated if control edges are inserted with `X` as target activity. Conversely, a newly added activity will have to be activated or skipped if all predecessors already have marking `COMPLETED` or `SKIPPED`.

We utilize information on the change context to decide on marking adaptations. We illustrate this by means of an example. Consider Fig. 7 where B is moved to the position between A and D. The algorithm first determines which nodes and edges have to be potentially (re)marked. In the given case these sets can be determined based on the inserted and deleted control edges. Then, the algorithm steps through the initial node and edge sets and adapts instance markings step by step. In Fig. 7, these steps are denoted as intermediate steps. First of all, for newly inserted control edge $A \longrightarrow C$, an adaptation has to be done; since source node A is already completed, $A \longrightarrow C$ is marked as `True_Signaled`. Consequently, in the next step, destination node C has to be marked as `Activated` since all incoming edges have marking `True_Signaled`. For the other newly inserted edge $B \longrightarrow D$ and deleted edge $A \longrightarrow B$ no marking adaptation becomes necessary. Thus, the algorithm terminates with the desired marking of I on S'.

Based on the compliance criterion, the **dynamic change bug** as discussed in literature (e.g. [29,16] is not present anymore in ADEPT2. More precisely, the application of the change operation as depicted in Fig. 1 would be rejected in ADEPT2 based on the corresponding compliance conditions. Furthermore, even for order-changing operations, markings can be automatically adapted without need for interacting with users. Basically, the described approach for ensuring compliance can be transfered to other process meta models as well. We have shown this for BPEL [30] and for Activity Nets [31].

3.4 Data Flow Correctness

So far, we have not considered data flow correctness in connection with process changes. Basically, we have to ensure correctness of the modeled data flow when directly changing it (e.g., by adding or deleting data edges) as well as when adapting the associated control flow structure. Regarding the latter, ADEPT2 will only allow for control flow changes if data flow correctness can be preserved afterwards [28]. As example take process schema S from Fig. 7 and assume that B writes data element d and C reads it afterwards. Regarding this scenario, data flow correctness would be not preserved if we conducted the depicted adaptation (i.e., to move B from its position between A and C to the position parallel to C). Since B would then be ordered in parallel to C, we could not guarantee any longer that B writes d before C reads this data element. As another scenario, assume that change Δ inserts two activities A and B in an arbitrary schema S, where A is writing data element d, which is read by B afterwards. In this case, Δ would not be correctly applicable, if A is inserted within one branch of an alternative branching. In this case, it cannot be ensured that A is activated during runtime and d is written accordingly.


a) Process Instance I
write data edge
read data
value of data object
d_1
d_2
d_3
1
2
5
5
?
A ✓
B ✓
C ▶
D
b) Data-Consistent Representation of Execution History $\sigma_I^{S\ dc}$
Δ = (deleteDataEdge(C, d_1, read), addDataEdge(C, d_2, read))
✓ Completed ▶ Running

Events	START(A)	END(A)	START(B)	END(B)	START(C)
written data elements	-	(d_1,5) (d_2,1)	-	(d_2,2)	-
read data elements	-	-	-	-	(d_1,5)

Fig. 8. Data Consistency Problem

Altogether, to avoid such structural flaws, a given sequence of change operations can be only applied in ADEPT2 if structural and behavioral soundness is guaranteed afterwards. In the given example, the structural pre-conditions of the *move* operation would disallow the application of the intended change since in schema in S' the read access of C to d has no preceding write access to this data element.

Another challenge is to preserve the correctness of the data flow schema when changing it. As example consider the scenario depicted in Fig. 8. Activity C has been started and has already read value 5 of data element d_1. Assume that, due to a modeling error, read data edge $(C, d_1, read)$ shall be deleted and read data edge $(C, d_2, read)$ be inserted instead. Consequently, C should have read value 2 of data element d_2 (instead of data value 5). Such inconsistent read behavior has to be prohibited since it can lead to errors and inconsistencies in the sequel (e.g., if instance execution is aborted and therefore has to be rolled back). Using any representation of execution trace σ_I^S as introduced so far, this erroneous case cannot be detected, i.e., the the instance would be classified as compliant.

We need an adapted form of σ_I^S considering data flow as well. We denote ${\sigma_I^S}^{dc}$ as data-consistent trace representation of σ_I^S with ${\sigma_I^S}^{dc} = <e_1, \ldots, e_k>$: $e_i \in \{\text{START(a)}^{(d_1,v_1),\ldots,(d_n,v_n)}\ \text{END(a)}^{(d_1,v_1),\ldots,(d_m,v_m)}\}$, a $\in\ \mathcal{A}$ where tuple $(d_i,\ v_i)$ describes a read/write access of activity a on data element $d_i\ \in\ \mathcal{D}_S$ with associated value v_i $(i = 0, \ldots, k)$. Using this data-consistent representation of σ_I^S the problem illustrated in Fig. 8a is resolved.

3.5 Concurrent Process Adaptations

Being able to cope with changes of single instances or a collection of instances in isolated manner is crucial to meet practical needs. However, changes do not always occur separately from each other. Assume that instance I on schema S is modified due to an exception resulting in instance-specific schema S_I (i.e. $S[\Delta_I > S_I)$. If later S is changed as well due to new legal regulations resulting in S' (i.e. $S[\Delta > S')$, the challenge is to decide how to cope with *concurrent* changes Δ_I and Δ (cf. Fig. 9): May I migrate to S' and - if yes - how does the instance-specific change (i.e., bias) turn out on S'? The latter question is particularly interesting if Δ_I and Δ are *overlapping*; i.e., they have some or all change effects in common (e.g., deletion of same activity). Then Δ_I has to be adapted on S' since S' already reflects parts of Δ_I. Fig. 9 illustrates the different cases in connection with dynamic change. If S is transformed into S', the user might want to exclude some of the instances due to specific constraints. For all others, migration to S' is desired. First, we have to distinguish between instances still running according to S (unbiased instances) and those individually modified (biased instances). For biased instances it is further important to know whether concurrent schema and instances changes are *disjoint* or *overlap* since further migration strategy depends on that. For all instances we need adequate correctness criteria (see [28,32] for respective extensions of compliance and migration strategies).

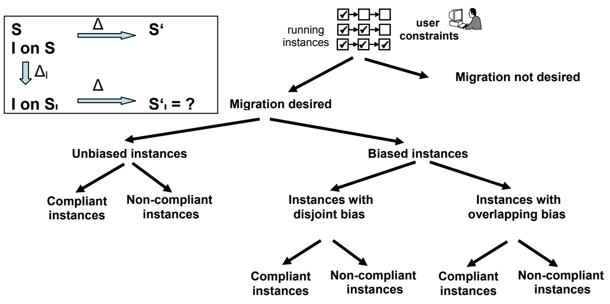


Fig. 9. Instance migration – big picture

3.6 How Users Interact with ADEPT2?

So far, ADEPT2 has been applied in several domains including healthcare, automotive development, construction engineering, logistics, and e-negotiation [1,2,33,34]. While for some applications the provided ADEPT2 clients were sufficient to adequately assist users in adapting their processes [34,1], in other cases specific client components were implemented based on the application programming interfaces offered by ADEPT2. AgentWork [33], for example, provides a rule-based planning component for the healthcare domain that automatically derives adaptations of patient treatment processes to be applied in a given context. Here, users only have to approve the suggested instance changes, which are then automatically carried out by the system; i.e., ADEPT2 serves as engine to implement the changes. CONSENSUS [1], in turn, uses the existing ADEPT2 clients to realize the flexibility and dynamism needed to accommodate to the various contingencies and obstacles that can appear during e-negotiations.

In all these case studies the provision of high-level change patterns and the change framework described were considered as strong points in favor of ADEPT2. Based on the lessons learned we are currently extending the meta model for WSM nets with additional workflow patterns [23]. Furthermore, we developed techniques targeting at improved user assistance. In [35], for example, we present an approach which uses conversational case-based reasoning to allow for the reuse of previously applied ad-hoc changes in similar problem context. We are also developing mechanisms to incorporate semantical constraints into adaptive PAIS in order to prohibit semantically counterproductive changes [36]. ADEPT2 expresses semantic constraints in terms of rules and verifies them during buildtime, runtime, and in connection with process changes. We further provide an authorization component, which allows to restrict process changes to authorized users, but without nullifying the advantages of a flexible PAIS by handling authorizations in a too rigid way [37]. Finally, we are investigating the concept of process views in connection with dynamic process changes [38]. Basic idea is to provide abstract views to users and to allow them to apply changes to these views and to propagate the view updates back to the underlying process.

4 Discussion

To effectively deal with exceptions through structural process adaptations and to enable process evolution have been major design goals of the ADEPT2 technology. In the previous sections we have presented basic issues and concepts to attain these goals and to enable dynamic structural changes of different process aspects. This section provides a survey on the state-of-the art (see also [9,17]), but extends it with a summary of approaches dealing with uncertainty as well. Furthermore we discuss alternative solutions for enabling process flexibility including declarative approaches [7] and case handling [14]. For a discussion of techniques for process evolution we refer to [9,17].

Dealing with Exceptions. While *expected exceptions* are usually considered during buildtime by specifying exception handlers to resolve the respective exceptions during runtime [39], non-anticipated situations, in turn, may require structural adaptations of single process instances [3,11]. A comprehensive overview of exception handling mechanisms is provided by [40]. Depending on the type of exception different handling strategies can be pursued (e.g., to roll back parts of the process), which are described as *exception handling patterns* in [40]. Exception handling often requires combined use of such patterns resulting in rather complex routines. The Exlet approach [41], for example, addresses this problem by allowing for the combination of different exception handling patterns to an exception handling process called Exlet. Similarly, [42] suggests the usage of meta workflows for coordinating exception handling. While exception handling patterns are well suited for dealing with expected exceptions, non-anticipated situations, in turn, often require *structural adaptations* of individual process instances as well [39]. Besides ADEPT2, several other approaches support ad-hoc changes [8,24,6], however, only the ADEPT2 framework allows for high-level change patterns (e.g., to insert, move or delete activities and process fragments, respectively) instead of change primitives (e.g., to add or delete nodes and edges in the process graph) [6]. To ensure correctness of run-time changes, soundness needs to be guaranteed. When conducting instance-specific changes, using change primitive (e.g., WASA2 [8] or CAKE2 [24]), soundness of the resulting process schema cannot be guaranteed and correctness of a process schema has to be explicitly checked after applying the respective set of primitives. ADEPT2, in turn, associates pre-/ post-conditions with the high-level change patterns, which allows to guarantee soundness. Finally, PAISs supporting instance-specific adaptations should be able to cope with concurrent changes as well. While many system prohibit concurrent process instance changes (e.g., FLOWer [14], WASA2 [8]), ADEPT2 supports them based on optimistic concurrent change techniques; CAKE2, in turn, supports concurrent process instance changes using pessimistic locking [24].

Dealing with Uncertainty. Flexible PAIS must be also able to cope with *uncertainty.* Common to existing approaches is the idea to defer decisions regarding the exact control-flow to runtime [13,17]. Instead of requiring a process model to be fully specified prior to execution, parts of the model can remain unspecified and be refined during run-time when more information is available. Examples for such techniques are Late Binding, Late Modeling and Late Composition of Process Fragments. Finally, data-driven processes provide for some flexibility regarding the exact control-flow as well [15,14,43].

Late binding allows to defer the selection of activity implementations to runtime; i.e., the implementation of the respective activity is chosen out of a set of process fragments at runtime either based on rules or user decisions [17]. As example consider Worklets [13], which allow for late binding of sub-process fragments to activities. At buildtime, the respective activity is modeled as a placeholder. *Late Modeling and Composition*, in turn, are techniques which go one step beyond by allowing parts or whole of the process to be defined during runtime [12,44]. Late Modeling allows for modeling selected parts of a process schema at runtime. At buildtime a placeholder activity as well as constraints for modeling the respective sub-process are defined. Usually, the modeling of the placeholder activity needs to be completed before its execution can start. Even more flexibility is provided by Late Composition. It allows users to compose existing process fragments on-the-fly; e.g., by dynamically introducing control dependencies between them. There is no predefined schema, but the (sub-)process instance is created in an ad-hoc way by selecting from the available fragments and obeying the predefined constraints. For both techniques, the model being dynamically defined may or may not be controlled by constraints. Complete lack of constraints can defeat the purpose of a PAIS, where as too many constraints may introduce rigidity that compromises the dynamic process [45].

[46,12] propose an approach for the late modeling of process fragments. A part of the process (termed *Pocket of Flexibility*) is deemed to be of a dynamic nature and is defined through a set of activities and a set of constraints defined on them. At runtime, the undefined part is detailed for a given process instance based on tacit knowledge and obeying the prescribed constraints. In contrast, the approach provided by DECLARE [44,7] enables late composition of process fragments. Basically, the whole process is defined in a declarative way. However, DECLARE can also be used in combination with imperative languages (e.g., YAWL). In this scenario, not the entire process model is described in a declarative way, but only sub-processes. Like in the *Pocket of Flexibility* approach a process model is defined as a set of activities and a set of constraints. During runtime process instances can be composed whereby any behavior is allowed which is not prohibited by any constraints. *Data-driven processes* as supported by the *case handling* tool FLOWer [14] do not predefine the exact control-flow, but orchestrate the execution of activities based on the data assigned to a case. Thereby, different kinds of data objects are distinguished. Mandatory and restricted data objects are explicitly linked to one or more activities. If a data object is mandatory, a value will have to be assigned to it before the activity can be completed. If a data object is restricted for an activity, this activity needs to be active in order to assign a value to the data object. Free data objects, in turn, are not explicitly associated with a particular activity and can be changed at any time during a case execution and consequently provide for flexibility during run-time. [47] compares workflow management and case handling with means of a controlled experiment. Recently, additional paradigms for the data-driven modeling and adaptation of large process structures have emerged. In particular, they allow for the transformation of data model changes to process adaptations as well as for sophisticated exception handling procedures [48,15].

5 Summary and Outlook

We have provided a general discussion on flexibility issues in adaptive PAIS and we surveyed the state-of-the-art. As core of any approach enabling dynamic process changes, adequate correctness notions are needed. When implementing them within a PAIS and making use of the formal properties of the underlying process meta model as well as change framework, different optimizations can be realized. Similarly, optimized techniques for the automated adaptation of instance states can be provided when migrating process instances to a modified schema. Along these challenges, we have discussed different correctness criteria and their application to specific process meta models. On one side we have considered structural criteria and their (logical) realization within Petri-net based PAIS. On the other side, we have analyzed approaches using traces for deciding whether an instance is compliant with a modified schema. Since both kinds of approaches come along with limitations, we have presented the ADEPT2 approach. ADEPT2 uses consolidated instance data and exploits the semantics of the applied change operations in order to abolish the limitations of pure structural and behavioral approaches. Finally, we have addressed issues related to concurrent changes, data flow correctness, and use of ADEPT2. Future work will extend our analysis of correctness criteria for dynamic process change. We will elaborate to what degree existing correctness notions can be relaxed to increase the number of compliant process instances [32]. Furthermore, there are still many open questions regarding the realization of concurrent process changes (e.g., how to deal with partly overlapping changes) and the management of the process variants resulting from instance changes. In this context, we are developing intelligent analysis techniques to learn from process changes [49,50,51]. Finally, we are currently working on issues related to the dynamic adaptation of organizational rules and access constraints [52,53], to process variant management [54], and to process model refactoring [55].

References

1. Bassil, S., Keller, R., Kropf, P.: A workflow–oriented system architecture for the management of container transportation. In: Desel, J., Pernici, B., Weske, M. (eds.) BPM 2004. LNCS, vol. 3080, pp. 116–131. Springer, Heidelberg (2004)
2. Lenz, R., Reichert, M.: IT support for healthcare processes - premises, challenges, perspectives. Data and Knowledge Engineering 61, 39–58 (2007)
3. Müller, R., Greiner, U., Rahm, E.: AgentWork: A workflow system supporting rule–based workflow adaptation. Data and Knowlege Engineering 51, 223–256 (2004)
4. Müller, D., Herbst, J., Hammori, M., Reichert, M.: IT support for release management processes in the automotive industry. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 368–377. Springer, Heidelberg (2006)
5. Rinderle, S., Reichert, M., Dadam, P.: Flexible support of team processes by adaptive workflow systems. Distributed and Parallel Databases 16, 91–116 (2004)
6. Weber, B., Rinderle, S., Reichert, M.: Change patterns and change support features in process-aware information systems. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007. LNCS, vol. 4495, pp. 574–588. Springer, Heidelberg (2007)
7. Pesic, M., Schonenberg, H., Sidorova, N., van der Aalst, W.M.P.: Constraint-based workflow models: Change made easy. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 77–94. Springer, Heidelberg (2007)
8. Weske, M.: Workflow management systems: Formal foundation, conceptual design, implementation aspects. University of Münster, Habilitation Thesis (2000)
9. Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems – a survey. Data and Knowledge Engineering 50, 9–34 (2004)

10. Ellis, C., Keddara, K., Rozenberg, G.: Dynamic change within workflow systems. In: COOCS 1995, pp. 10–21 (1995)
11. Reichert, M., Dadam, P.: ADEPT$_{flex}$ - supporting dynamic changes of workflows without losing control. Journal of Intelligent Information Systems 10, 93–129 (1998)
12. Sadiq, S., Sadiq, W., Orlowska, M.: Pockets of flexibility in workflow specifications. In: Kunii, H.S., Jajodia, S., Sølvberg, A. (eds.) ER 2001. LNCS, vol. 2224, pp. 513–526. Springer, Heidelberg (2001)
13. Adams, M., Hofstede, A., Edmond, D., van der Aalst, W.: Worklets: A service-oriented implementation of dynamic flexibility in workflows. In: Meersman, R., Tari, Z. (eds.) OTM 2006. LNCS, vol. 4275, pp. 291–308. Springer, Heidelberg (2006)
14. van der Aalst, W., Weske, M., Grünbauer, D.: Case handling: a new paradigm for business process support. Data and Knowledge Engineering 53, 129–162 (2005)
15. Müller, D., Reichert, M., Herbst, J.: A new paradigm for the enactment and dynamic adaptation of data-driven process structures. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 48–63. Springer, Heidelberg (2008)
16. van der Aalst, W., Basten, T.: Inheritance of workflows: An approach to tackling problems related to change. Theoretical Computer Science 270, 125–203 (2002)
17. Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features - enhancing flexibility in process-aware information systems. Data and Knowledge Engineering 66, 438–466 (2008)
18. Dehnert, J., Zimmermann, A.: On the suitability of correctness criteria for business process models. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 386–391. Springer, Heidelberg (2005)
19. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Workflow evolution. Data and Knowledge Engineering 24, 211–238 (1998)
20. van der Aalst, W., Weske, M., Wirtz, G.: Advanced topics in workflow management. Int'l Journal of Integrated Design and Process Science 7 (2003)
21. Haddad, S., Pradat-Peyre, J.: New efficient petri nets reductions for parallel programs verification. Parallel Processing Letters 16, 101–116 (2006)
22. Rinderle-Ma, S., Reichert, M., Weber, B.: On the formal semantics of change patterns in process-aware information systems. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (eds.) ER 2008. LNCS, vol. 5231, pp. 279–293. Springer, Heidelberg (2008)
23. Wolz, J.: New control and data flow concepts in ADEPT2. Master's thesis, Ulm University (2008)
24. Minor, M., Schmalen, D., Koldehoff, A., Bergmann, R.: Structural adaptation of workflows supported by a suspension mechanism and by case-based reasoning. In: WETICE 2007 (2007)
25. van der Aalst, W., ter Hofstede, A.: Yawl: Yet another workflow language. Information Systems 30, 245–275 (2005)
26. Agostini, A., De Michelis, G.: Improving flexibility of workflow management systems. In: BPM 2000, pp. 218–234 (2000)
27. Rinderle, S., Reichert, M., Dadam, P.: Supporting workflow schema evolution by efficient compliance checks. Technical Report UIB2003-02, Ulm University (2003), `http://www.uni-ulm.de/in/iui-dbis/forschung/publikationen.html`
28. Rinderle, S.: Schema Evolution in Process Management Systems. PhD thesis, Ulm University (2004)
29. van der Aalst, W.: Exterminating the dynamic change bug: A concrete approach to support worfklow change. Information Systems Frontiers 3, 297–317 (2001)

30. Reichert, M., Rinderle, S.: On design principles for realizing adaptive service flows with BPEL. In: EMISA 2006, pp. 133–146 (2006)
31. Reichert, M., Rinderle, S., Dadam, P.: On the common support of workflow type and instance changes under correctness constraints. In: Meersman, R., Tari, Z., Schmidt, D.C. (eds.) CoopIS 2003, DOA 2003, and ODBASE 2003. LNCS, vol. 2888, pp. 407–425. Springer, Heidelberg (2003)
32. Rinderle-Ma, S., Reichert, M., Weber, B.: Relaxed compliance notions in adaptive process management systems. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (eds.) ER 2008. LNCS, vol. 5231, pp. 232–247. Springer, Heidelberg (2008)
33. Müller, R.: Event-Oriented Dynamic Adaptation of Workflows. PhD thesis, University of Leipzig, Germany (2002)
34. Golani, M., Gal, A.: Optimizing exception handling in workflows using process restructuring. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 407–413. Springer, Heidelberg (2006)
35. Weber, B., Reichert, M., Wild, W., Rinderle-Ma, S.: Providing integrated life cycle support in process-aware information systems. Int'l Journal of Cooperative Information Systems (IJCIS) 18 (2009)
36. Ly, L., Rinderle, S., Dadam, P.: Integration and verification of semantic constraints in adaptive process management systems. DKE 64, 3–23 (2008)
37. Weber, B., Reichert, M., Wild, W., Rinderle, S.: Balancing flexibility and security in adaptive process management systems. In: Meersman, R., Tari, Z. (eds.) OTM 2005. LNCS, vol. 3760, pp. 59–76. Springer, Heidelberg (2005)
38. Bobrik, R., Reichert, M., Bauer, T.: View-based process visualization. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 88–95. Springer, Heidelberg (2007)
39. Reichert, M., Dadam, P., Bauer, T.: Dealing with forward and backward jumps in workflow management systems. Software and Systems Modeling (SOSYM) 2, 37–58 (2003)
40. Russell, N., van der Aalst, W., ter Hofstede, A.: Exception Handling Patterns in Process-Aware Information Systems. In: CAiSE 2006, pp. 288–302 (2006)
41. Adams, M., ter Hofstede, A., van der Aalst, W., Edmond, D.: Dynamic, extensible and context-aware exception handling for workflows. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 95–112. Springer, Heidelberg (2007)
42. Kumar, A., Wainer, J.: Meta workflows as a control and coordination mechanism for exception handling in workflow systems. Dec. Support Sys. 40, 85–105 (2004)
43. Rinderle, S., Reichert, M.: Data-driven process control and exception handling in process management systems. In: Dubois, E., Pohl, K. (eds.) CAiSE 2006. LNCS, vol. 4001, pp. 273–287. Springer, Heidelberg (2006)
44. Pesic, M.: Constrained-based Workflow Management Systems – Shifting Control to Users. PhD thesis, TU Eindhoven (2008)
45. Wainer, J., de Lima Bezerra, F.: Constraint-Based Flexible Workflows. In: Favela, J., Decouchant, D. (eds.) CRIWG 2003. LNCS, vol. 2806, pp. 151–158. Springer, Heidelberg (2003)
46. Mangan, P., Sadiq, S.: A constraint specification approach to building flexible workflows. J of Research and Practice in Inf Technology 35, 21–39 (2002)
47. Mutschler, B., Weber, B., Reichert, M.: Workflow management versus case handling: Results from a controlled software experiment. In: SAC 2008, pp. 82–89 (2008)
48. Müller, D., Reichert, M., Herbst, J.: Data-driven modeling and coordination of large process structures. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 131–149. Springer, Heidelberg (2007)

49. Li, C., Reichert, M., Wombacher, A.: Discovering reference process models by mining process variants. In: ICWS 2007, Beijing, pp. 45–53 (2008)
50. Guenther, C., Rinderle-Ma, S., Reichert, M., van der Aalst, W., Recker, J.: Using process mining to learn from process changes in evolutionary systems. Int'l Journal of Business Process Integration and Management 3, 61–78 (2008)
51. Li, C., Reichert, M., Wombacher, A.: On measuring process model similarity based on high-level change operations. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (eds.) ER 2008. LNCS, vol. 5231, pp. 248–264. Springer, Heidelberg (2008)
52. Rinderle-Ma, S., Reichert, M.: Managing the life cycle of access rules in CEOSIS. In: Proc. EDOC 2008, Munich, pp. 257–266 (2008)
53. Rinderle-Ma, S., Reichert, M.: A formal framework for adaptive access control models. In: Spaccapietra, S., Atzeni, P., Fages, F., Hacid, M.-S., Kifer, M., Mylopoulos, J., Pernici, B., Shvaiko, P., Trujillo, J., Zaihrayeu, I. (eds.) Journal on Data Semantics IX. LNCS, vol. 4601, pp. 82–112. Springer, Heidelberg (2007)
54. Hallerbach, A., Bauer, T., Reichert, M.: Managing process variants in the process lifecycle. In: ICEIS 2008, Barcelona, pp. 154–161 (2008)
55. Weber, B., Reichert, M.: Refactoring process models in large process repositories. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 124–139. Springer, Heidelberg (2008)

Business Grid: Combining Web Services and the Grid

Ralph Mietzner, Dimka Karastoyanova, and Frank Leymann

University of Stuttgart, Institute for Architecture of Application Systems
Universitätsstr. 38, 70569 Stuttgart, Germany
{mietzner,karastoyanova,leymann}@iaas.uni-stuttgart.de

Abstract. The common overarching goal of service bus and Grid middleware is "virtualization" – virtualization of business functions and virtualization of resources, respectively. By combining both capabilities a new infrastructure called "Business Grid" results. This infrastructure meets the requirements of both business applications and scientific computations in a unified manner and in particular those that are not addressed by the middleware infrastructures in each of the fields. Furthermore, it is the basis for enacting new trends like Software as a Service or Cloud computing. In this paper the overall architecture of the Business Grid is outlined. The Business Grid applications are described and the need for their customizability and adaptability is advocated. Requirements on the Business Grid like concurrency, multi-tenancy and scalability are addressed. The concept of "provisioning flows" and other mechanisms to enable scalability as required by a high number of concurrent users are outlined.

1 Introduction

Traditionally the scientific computing and enterprise computing communities are two distinct groups. These communities have both developed architectures of middleware and applications to meet the demands of their fields in a domain-specific manner. As a result two different fields with corresponding infrastructures have evolved. The current trend we observe is that researchers in both fields investigate the applicability of approaches, technologies and techniques from the other field. With the advent of Web services the need for common solutions becomes apparent, since both fields use extensively the Web service stack [44]. Both communities identify needs that can be addressed by approaches from the other field. For example, the deficiency of the infrastructures used for scientific workflows are mainly related to reliability and scalability. The infrastructures for enterprise applications on the other hand lack the dynamic provisioning of resources and data needed for new service delivery models as well as for data and computing intensive applications. Both fields, however, impose strict requirements on their respective infrastructure concerning the need for concurrency, i.e. the ability to support a large number of simultaneous users without diminishing the performance of the applications. This is aided by the virtualization of resources in the Grid [8] and of functionalities in the enterprise service bus (ESB) [3,22]. Virtualization standardizes the access to resources and functionality exposed as services respectively, thus hiding the idiosyncrasies that heterogeneous systems otherwise exhibit. The Grid therefore offers homogeneous and standardized access to resources, whereas the ESB offers homogeneous and standardized access to applications, by exposing them as services.

We argue that the better approach to meet the requirements of both domains is to leverage existing approaches instead of creating isolated infrastructures featuring redundant and repeating functionalities. We therefore propose a unified infrastructure that is composed of the infrastructures developed by both the scientific computing and the enterprise computing communities. We call this infrastructure the "Business Grid". The Business Grid virtualizes services and resources in a unified manner and thus enables interchangeability of services and resources transparently for the applications. It is a composable infrastructure which can be developed as a mix of existing implementations of middleware from both domains. The proposed middleware is reliable, scalable, secure and exhibits high availability. Applications for both scientific and enterprise computing make transparent use of the Business Grid. The Business Grid does not discriminate between either type of applications thus allowing scientific applications to be a part of an enterprise application and vice versa. Further motivation for the Business Grid is the current trend towards new business models in dynamic markets such as Software as a Service (SaaS) and Cloud computing. As emerging approaches towards support for SaaS and Cloud computing already make use of technologies of scientific and enterprise computing we see the Business Grid as the intuitive middleware choice to support those new models. This combination of Grid and conventional enterprise service middleware, technology standards and infrastructures is novel and fosters reusability and standardization.

In this paper we begin with an overview of Web service technology and Grid to facilitate the discussion. Following this we identify the similarities in requirements towards the respective middleware in the two domains and reveal the differences in coverage of needed middleware functionality in both domains. Based on this comparison we list requirements for the Business Grid and investigate to what extend they are met by related projects and approaches from to the enterprise and Grid community. In Section 5 we introduce an architecture designed to address the requirements to the Business Grid. Summary and conclusions are published in the last section.

2 Web Services and Grid – Background Information

To ease the following discussion, in this section we present a summary of necessary background information. We describe existing research results that combine Web services and Grid and thus demonstrate the mutual interest of both communities to collaborate towards meeting the demands in their respective domains.

2.1 Web Services

Web services are the technology currently used to implement Service Oriented applications; it has been built right from the beginning to enable seamless integration of applications. The Web service technology provides a component model for using applications, rather than programming them. The technology is specified in terms of a stack of modular and composable specifications, called the "WS stack" [44] or simply WS*. Web services are about virtualization of software/applications. Applications are exposed as services at network endpoints (a.k.a. *ports*). Web services are described in WSDL[1] in terms of the messages they consume and produce, while the messages are grouped into operations and interfaces (a.k.a. *port types*). The interface description excludes any kind of platform or implementation specific information. One interface description may be implemented by multiple applications, accessible at different ports/endpoints over various transport protocols and using different message encoding; this is reflected in the so-called *binding*. The binding is the novelty WSDL introduces in comparison to other component models. It is defined separately from the service interface description. Thus, not only implementation and platform specifics are abstracted away but also the access mechanisms. Any combination of an abstract interface and binding is possible. The location of an implementation of an interface description and a binding are combined by the so-called *services*, while applications are made available at multiple ports (endpoints) of a service. The piece of middleware supporting virtualization in service-oriented applications is called the service bus [3,22]. The service bus is a composable middleware and supports numerous communication and transport protocols as needed by the applications using it. All services are exposed on the bus via *virtual endpoints* and described in terms of WSDL interfaces descriptions. The concrete implementations of the virtual endpoints are in general wrappers or adapters of the applications that expose the application as a service on a bus. There are infrastructure features on the bus like service discovery, routing, data mediation, service composition, and service invocation. These features are instrumental in supporting the major operations a bus implements: (1) publish, (2) find and bind, (3) invoke. Services use the bus for communication and use its infrastructure features; infrastructure features themselves are also exposed on the bus as services. There are already multiple non-commercial and commercial service bus implementations; the later are extensively used in enterprise-strength applications.

2.2 Grid

Originally, in the mid-1990s the term “the Grid” emerged as a name for a distributed IT infrastructure for advanced science and engineering applications [7,8]. The intent of the Grid was then to virtualize computing resources to make them available to scientific collaborations that needed more computing power than the computing resources available in their domains (their local IT infrastructures). Applications that run on such Grids are mainly focused on scientific applications that perform computations on and produce large amounts of potentially distributed data. In [7] Foster et al. define the Grid as a solution to the problem of “flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions and resources.” These dynamic collections of individuals, institutions and resources are referred to as “virtual organizations” [7]. Virtual organizations can be different groups of scientists working on a common goal but can also be different companies that join forces or are in a supplier-requestor relationship in order to reach a certain (mostly computing intensive) business goal. The fundamental paradigm behind these virtual organizations is that they not only share files or documents but actually share access to programs and even hardware resources. The sharing of computing resources from

[1] Web Service Description Language.

multiple organizations that form a virtual organization requires a standardized architecture for the Grid as well as interfaces, to enable interoperability among Grid resources. Therefore the Grid community introduced the Open Grid Services Architecture OGSA [9]. OGSA follows the principle of service oriented architecture (SOA) providing a standardized set of services important in a Grid environment (such as registry, authorization monitoring, data access and others) [8]. The technology to realize the OGSA is Web services, thus allowing Grid services to be described and used in a standardized and interoperable manner. The Open Grid Services Infrastructure (OGSI) and its successor the Web Service Resource Framework (WSRF) [5,32] are specifications that define how to specify stateful Web services that represent stateful resources (such as computers or storage) as they are present in Grid environments.

As the internet transforms into a ubiquitous integration platform and more increasingly powerful computing resources become available, more and more organisations (both from science and business) see the potential of the Grid to improve the utilization of idle resources from other (parts of their) organizations to perform computing intensive tasks or even sell otherwise idle computing cycles to other organizations and thus increase the utilization rates of their data centres. The Grid therefore transforms from an infrastructure for scientists into an infrastructure for enterprises.

2.3 Web Services and Grid

Web services used in a business context are usually stateless. This means that a Web service does not remember the messages that have been previously sent to it by a certain requestor and that a Web service has no observable properties. But in order to virtualize Grid resources as Web services, the notion of state is needed for Web services. Since WSDL does not provide support to explicitly describe state of a Web service, the Web Service Resource Framework has been created. The basic idea behind WSRF is to provide a standardized framework that allows specifying how stateful resources are virtualized using Web services. The combination of a resource and a Web service is therefore called a WS-Resource. WSRF defines the life time of resources, the notion of resource properties, the grouping of WS-Resources, and a standardized means to report faults in a set of composable specifications.

In order to compose Grid services and in particular WSRF services the BPEL service orchestration language has been utilized in several research projects [6, 24, 37, 39]. These examples emphasize the need and the willingness of the two communities to combine technologies from Grid and SOA-based enterprise computing. Even though Grid technologies and the WS technology are already applied in a combination to provide solutions for enterprises or scientists, for example in the EGEE project and its predecessor the DataGrid[2] project or the IBM WebSphere Business Grid components, the Grid infrastructures are still considered as different from the service buses. ESBs and Grids are similar in the functionality they provide, but they target different user groups and application domains. Therefore concepts and techniques in both field repeat and overlap, but some features of the one environment are missing in the other environment and vice versa. For instance, scientific computations need transactionality and to be reliable, which has been

[2] http://www.eu-egee.org , http://eu-datagrid.web.cern.ch

successful addressed by ESBs; on the other hand, businesses also need to perform data intensive computations and also rely on provisioning of resources that are outside of their organizational boundary. Therefore it is completely natural to reuse existing concepts, approaches and technology to address the needs of both domains.

3 The Need for Business Grid

We argue for the creation of one infrastructure that can serve the needs of both communities – business and scientific computations, so that both fields can benefit from the advances in the state-of-the-art of the one or the other area. In this work and in this section in particular, we identify the need of such an infrastructure - the *Business Grid* that addresses the demands of the two worlds.

3.1 Business World Requirements to the Service Bus/Middleware

Service-based business applications are typically compositions of services. They rely on the service bus as middleware. Additionally, business users depend on the existence of tools to support them during application/service development, deployment and execution. Human participants need to be seamlessly involved, especially in business processes. Business critical applications need to be reliable and scale with an increasing number of concurrent users. In business scenarios quality of services (QoS) are extremely important as they must be ensured in order to satisfy service level agreements (SLAs) between provider and consumers. Violation of SLAs might result in loss of customers and/or even hefty penalties. These features require support on behalf of both the platforms on which the actual service implementations run and the service bus. Reliability, scalability and availability of the service bus are of utmost importance. These are ensured by dynamic service discovery and composition, reliable messaging, fault and exception handling, load balancing, message routing [3], transactional support [34], coordination protocol support, message transformation and correlation, SLA negotiation, QoS-aware composition of services [41], monitoring of services and infrastructure as well as auditing [16] – all enabled by the infrastructure services of the bus. New computing paradigms such as utility computing and resulting application delivery models such as SaaS impose further requirements on the bus. Dynamic provisioning and de-provisioning of both hardware and software resources must be handled by the bus as new customers of SaaS applications can subscribe and unsubscribe dynamically to and from these applications. The SaaS delivery model heavily relies on the multi-tenancy of SaaS applications [4,26]. Multi-tenancy means that several customers (tenants) use the same instance of the application. This enables SaaS providers to offer the same application to multiple customers thus increasing the revenue of the application. From a tenant's point of view a multi-tenant enabled application is perceived and behaves as if that tenant was the sole user of the application. In order to ensure the multi-tenancy of SaaS applications these applications as well as the underlying middleware must be highly scalable, reliable and ensure a very high degree of concurrency.

3.2 Scientific Computing Requirements to the Grid

Scientific computations are generally based on large amounts of data and are long-running and computing intensive. They can be executed on the Grid using computing and storage resources. Thus scientific experiments analysis can be performed on resources borrowed from other organizations whose resources are currently idle. This implies that resources are inherently transient, as resource providers can dynamically add and remove resources from the Grid and can lend them only for a particular time period. This requires migration of computations from one machine to another, transferring data sets and even integrating data sets from different resources for single computational task. Scientists define the sequence of computation units and rely on the Grid to provide the resources, do the scheduling of computation tasks, workload management, life cycle management, discovery of services, and exposing the services with the corresponding meta-data on the Grid. The Grid infrastructure deals with all issues related to the use of transient resources. This is typically done using resource brokers. There are techniques for instrumentation of Grids for monitoring, which helps auditing, analysis, troubleshooting and fault detection, and facilitate improvement of program executions. Currently, no monitoring infrastructure satisfies all requirements imposed on Grid monitoring systems [8].

Also, still missing is a general accepted way of composing scientific tasks and resources [11]. The ability to mix Grid resources and services in a single composition is also needed; for this, automatic unified discovery of Grid services, which may be either resources or computing applications, must be supported. Reliability and robustness are not yet achieved completely. There is no standardized way to include humans in Grid applications [19,10]. Granting access of one application to multiple distinct users is also not supported. Additionally, users need to be able to control the applications functionality and to flexibly change it even during its execution. Grid services expose resources via interfaces providing much more details than the users can cope with, which renders the Grid infrastructure cumbersome to use for non-IT personnel. Cloud computing is a current development in the area of SOC and is considered a layer on top of the existing Grid and Service Bus infrastructures. “A distinguishing feature of Cloud interfaces are that instead of exposing the largest possible amount of semantics for a specific domain, Clouds tend to expose a minimal amount of semantics to the end-user, while still being useful” [14].

3.3 Requirements on the Business Grid and Existing Approaches

In this section we present the combined requirements on the Business Grid and identify approaches for each of the requirements. We thus show that the existing approaches need to be combined and extended to fully address all the expectations.

Reliability and Scalability
Requirements for a Business Grid include the reliability and scalability of the whole computing infrastructure. Companies, executing business critical applications in a Business Grid will expect and require providers to scale up to a large amount of concurrent users reliably. Providers therefore must ensure that their Grid as a whole actually meets the agreed upon reliability and availability levels despite of single entities

of the Grid that may be transient, i.e. become available or disappear from the Grid dynamically. This becomes even more critical as different customers might actually use the same resources in the same Grid and thus create even more concurrent load on the whole system. Load-balancing and clustering are some of the approaches applied by both, business and Grid infrastructures. Scalable middleware (such as application servers) that provides infrastructure services such as load-balancing, failover management, virtualization, hot-pools and clustering is a result of enormous effort in the business domain. Similarly, a lot of effort has been employed in the Grid domain to create Grid resource brokers with scheduling functionality.

Quality of Service Aware Discovery

Quality of services plays a role in both business and scientific domains. Several Business Grid users might have the same functional requirements (such as run application x) with different non-functional requirements. For example, a small company might be satisfied if the infrastructure that runs its application only supports one concurrent user, while a big company demands that the infrastructure scales up to 100s of concurrent users. Therefore resources in the Business Grid must be annotated with non-functional properties, usually called policies in the business domain, so that the Business Grid middleware can discover the appropriate resources for the computing tasks. The contracts on QoS between Business Grid users and providers are Service Level Agreements (SLAs) [41]. Standards to describe agreements and SLAs are WS-Agreement, WS-Policy and WSLA[3]. Most of them are already implemented by the bigger middleware providers.

In the business domain the middleware (the ESB [3]) uses policies used for service selection. The ESB selects a concrete implementation of a virtual service to handle a request based on the policies [43]. This mechanism can also be used to increase the reliability of the whole system since non-available services can be substituted by available ones or highly concurrent load can be balanced between different services. The Business Grid must support similar features for computing resources and thus serve the function of a combined ESB and resource broker [8, 42,20].

A supporting role for enacting these features is played by the discovery components that are also exposed as services on the bus. Discovery components classify services according to their functional and non-functional properties and may also maintain ranking of services according to multiple criteria. Since the Business Grid exposes all services and resources in a homogeneous way (through standard interfaces) it is a must to be able to expose services and processes as resources, and also provide information about their state. This will improve discovery additionally, because of the capability to discover them according to state information, not only functional and non-functional properties. Existing approaches already involve the use of semantics for describing functional and non-functional properties of services and resources and thus improve discovery. Examples include the work in [22, 28,37] and the projects like SUPER, ASG and DIP[4]; these projects address various additional issues that are out of the scope of this paper.

[3] Web Service Level Agreement.

[4] www.ip-super.org , www.asg-platform.org , http://dip.semanticweb.org/

Flexible Service and Resource Composition
The Business Grid must be able to compose atomic resources into composite resources that support more advanced functionality and qualities of service; composing them may need a business/integration logic definition. In the business domain service orchestration languages, such as BPEL, are used to compose services into higher-level services. Service orchestration techniques are adapted to orchestrate Grid services and resources in several projects [6, 24, 37, 39]. Furthermore such a collection must support the dynamic addition and removal of resources as they register or deregister from the Grid while maintaining a constant service level. Therefore a composite resource must be highly dynamic/flexible but act as a stable single resource to the client.

Flexibility of resource compositions can be improved by means of approaches that already exist in the business domain, such as process evolution and ad-hoc changes, fault and exception handling, as well as dynamic deployment/provisioning [27] and discovery of services on the bus [14]. Several approaches based on models from concurrency theory (such as Petri nets) allow the modeling and verification of service compositions [1] and can be applied for the verification of adaptations in service compositions [35]. A subset of the above mentioned flexibility approaches are supported partly or to the full by industry vendors and research prototypes [16,25]. However, flexibility of resource compositions using workflows, although required, is not yet addressed by the Grid community.

Human Involvement
Furthermore the management of human tasks that integrate humans in Business Grid applications is a requirement for the Business Grid. Human participants/users therefore need to be represented on the Business Grid as Grid resources [36]. The Business Grid should not make any assumptions on whether a specific resource/service is implemented by a computer program or is performed by or with the aid of a human. Therefore the Business Grid must support advanced task management capabilities as well as dynamic staff queries that can be used to assign work to a dynamically changing group of humans from different virtual organizations. In the business domain the WS-HumanTask specification describes a standardized way to perform task management across different business processes and services. Human tasks can be integrated in service orchestrations using the BPEL4People extension for BPEL.

Monitoring and Management
In order to ensure traceability of the execution of applications and maintain the required service levels monitoring and auditing capabilities on all levels – processes, services and resources, infrastructure, SLAs, policies, KPIs (Key Performance Indicators) – are necessary. Thus reaction to unexpected situations or fluctuations in the levels of service quality can be enabled. Business Grid applications can also be adapted to include feedback from the analysis of monitoring and audit data. In the ideal case the Business Grid includes elements of self-healing, self-managing autonomous systems [18], for example the Business Grid must detect and react to critical thresholds and events so that violation of SLAs can be prevented. Debugging of Business Grid applications must be enabled by capturing events published from all parts of the overall distributed infrastructure. Monitoring and Management is a must in the business domain and is often required by law.

Research projects like SUPER, MASTER, COMPAS[5] strive to ensure support for this, but Grid applications are out of their scope.

Tool Support

Since businesses concentrate on business processes the IT infrastructure needs to support them seamlessly and enable development automation and a low barrier to entry. Therefore the tools for modeling and/or development of business applications need to be provided as part of the Business Grid distributions. These include service interface generation, service discovery when composing applications and processes, message mapping support, routing procedures modeling and generation. Semantics can be involved to fill in the gap between the business world and the IT support [14, 28]. Additionally, composition and decomposition of SLAs, policies and mapping to KPIs also need to be supported by tools. Tools exist in both domains that allow the modelling of either general-purpose workflows (such as BPEL or Petri net editors) or domain-specific workflows (such as in biology, mechanics, manufacturing etc.); they need to be integrate into the Business Grid.

Concurrency

Both business and Grid applications and infrastructures are inherently required to support concurrency. The use of the term concurrency is manifold. Business applications need to support a large number of users simultaneously, whereas Grid infrastructures need to deal with applications that are concurrently run on several resources in a distributed environment.

The Business Grid must support all dimensions of concurrency. The infrastructure must support the concurrent access to resources via resource sharing and virtualization techniques. Business Grid applications must additionally support a large number of concurrent users. An example where both dimensions are necessary are Software as a Service applications with large numbers of concurrent users that run on a large set of shared resources and must guarantee service level agreements.

In the business domain several approaches to achieve concurrency exist. One approach is to determine the expected amount of concurrent users a system should support up front and then scale the system and infrastructure accordingly. Therefore performance-modeling and capacity-planning methods for certain middleware components such as application servers, or Web servers exist. Another approach is to use dynamic provisioning to dynamically adapt the infrastructure to new concurrency requirements. Techniques from autonomic computing such as autonomic feedback loops [18] that trigger the provisioning of new resources can be used. Neither of the domains supports all concurrency dimensions completely and combining the existing approaches and extending them is a must.

4 The Business Grid Architecture

To meet the above requirements the Business Grid must virtualize both resources and services. Hence the underlying Business Grid middleware must be a combination of an ESB and a Grid, and therefore the Business Grid virtualizes both resources and

[5] www.master-fp7.eu, www.compas-ict.eu

services as Business Grid Resources. Business Grid Resources may be conventional Web services or WS-Resources. Applications comprised of several services (that can also be processes) therefore are seen as a collection of Business Grid Resources and may also be made available on the Business Grid as Business Grid Resources. These Resources are visible on the Business Grid only as virtual resources but each of them is implemented by a multitude of alternative concrete resources. A service requestor sends a request including the addressing information, or non-functional properties for the appropriate service, to the virtual service endpoint, which then is distributed to the right resource.

The Business Grid is a composable infrastructure in terms of the functions it supports. Additionally, multiple Business Grid implementations using different technologies and programming languages can be composed to form a more complex infrastructure. The communication among Business Grid Resources is enabled by the communication backbone that supports any kind of communication and transport protocols, and expose stable Web Service interfaces (see Fig. 2). The type of the Business Grid Resource (Web service or WS-Resource) remains transparent for the applications. The implied resource pattern of WSRF specifies that for resources virtualized with WSRF, a Web service (e.g. a server service) serves as a front-end to a set of similar resources (i.e. a set of servers) and distributes messages sent to one of the resources to the right resource by examining the addressing information contained in the message (e.g. EPR [44]). Having examined the addressing information the message is either forwarded to the appropriate resource or service. The Business Grid Resources are classified into two main groups: infrastructure/middleware resources and application resources. Infrastructure resources have a specific function to carry out on the Business Grid.

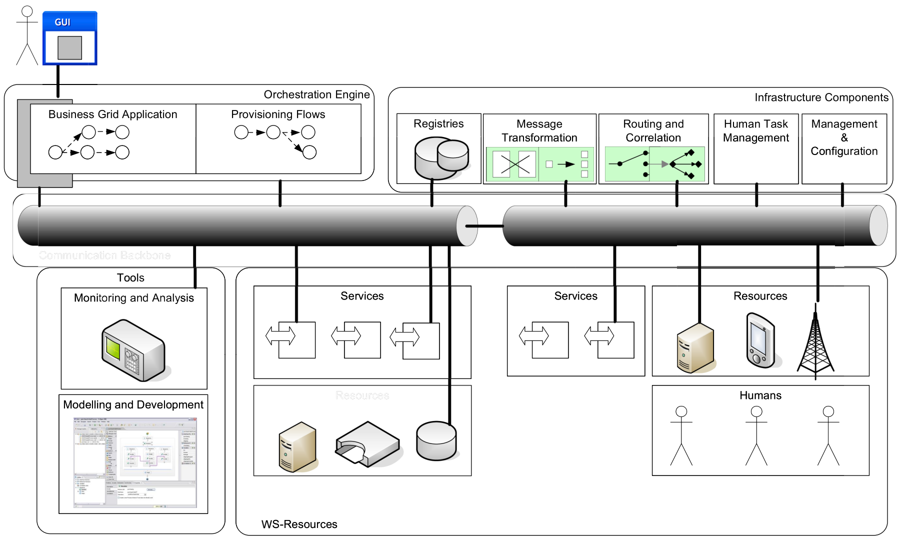


Fig. 1. Business Grid architecture

The Business Grid takes over the *discovery and invocation* of Business Grid Resources on behalf of clients. For this it uses the communication backbone and a discovery component. Discovery components are used to register services and resources and their characteristics. Workflows can also be registered as services and discovered by the Business Grid in a similar manner. Sometimes the request message does no match the messages a Business Grid Resource can consume. In such a situation the so-called *message transformation* service is used, which performs the transformation from one format to the other. Message *routing and correlation* are also part of the middleware services. All infrastructure components may be implemented in terms of multiple concrete applications and may be of distributed nature.

Composite Business Grid Resources are executed on *orchestration engines*; in the case of BPEL processes this would be a BPEL engine. In this case the BPEL engine plays the role of a service container [3, 14,22] and exposes orchestrations (processes) as Business Grid Resources. Since provisioning flows (presented in detail in the next section) can also be implemented in BPEL [17,27] the engine executing them can be considered to play the role of an infrastructure service, too.

Composite Business Grid Resources may involve human participants, for which facilities for *human task management* must be made available.
Monitoring and auditing facilities enable real time tracing of status of services, resources, the infrastructure and human participants. Additionally, the data collected for monitoring is stored persistently to enable auditing and analysis.

In addition to the monitoring tools and dashboard *analysis tools* are also of great value. Analysis can be performed on the data about the performance of the infrastructure, the infrastructure services, and the scientific and business processes. Complex data and process mining algorithms can be applied to provide insight to the functioning of the Business Grid to users and developers.

Management tools for configuring and fine tuning the infrastructure are also a must; these are usually under the control of authorized administrators.

Process *modeling tools* are needed to support business users in creating business processes and scientists in crating scientific workflows. Similarly provisioning flow modeling needs to be supported. These tools need to enable intuitive modeling of processes in any of the domain specific languages, e.g. BPMN for business experts.

Service implementation support should be enabled in terms of *automatic generation* of interfaces for services, service deployment, generation of code skeletons based on service interfaces, etc. Generation of resource interfaces upon their subscription to the Business Grid is also a must. Additionally, tools need to implement algorithms for *composing and decomposing policies and SLAs*; these tools need to be extensible and allow for the definition/implementation of new algorithms that consider new QoS characteristics and even KPIs. These tools are of great importance for modeling QoS-aware service compositions.

5 Dynamic Provisioning of Business Grid Resources

Similar to an ESB that distributes service request to services that meet the required service level agreements the Business Grid middleware must distribute computing requests to resources that meet the required non-functional properties. As the Grid is a dynamic infrastructure to which computing resources dynamically register and deregister, the dynamic provisioning of Business Grid Resources is a must for a Business Grid middleware. This improves the flexibility of applications which is demanded by both domains. Additionally it enables users to dynamically subscribe and unsubscribe to Business Grid applications as it is required for the realization of the SaaS delivery model.

Dynamic provisioning of new Business Grid Resources is done in the following cases: (a) if it is detected that the quality of services of current resources falls below a certain threshold and thus will violate the service levels agreed by customer and provider; (b) because an application has not been deployed on the Grid.

For example, if a new customer subscribes to an application that runs on the Business Grid, the middleware examines if the user can be served by the currently available resources. Methods from concurrency theory can be used to determine the dynamic scheduling of resources in advance. If this is not the case the middleware must invoke a so-called provisioning flow that can provision the application on more resources so that it meets the service level the user requires (Fig. 2).

Provisioning services are (Grid) Web services for resource management that are provided by a Grid infrastructure (such as the Globus toolkit) or a provisioning engine, such as IBM's Tivoli Provisioning Manager [13] or OpenQRM [33]. These provisioning services must be exposed for use on the Business Grid. This allows integrating the infrastructure of arbitrary providers/vendors into the Business Grid. For example, the operations provided by third party cloud providers (such as Amazon EC2 [2]) can be utilized as provisioning services on the Business Grid, too.

Provisioning flows [27] are orchestrations of provisioning services. Since provisioning services are Business Grid Resources, provisioning flows can be realized in WS-BPEL [30]. Furthermore, WS-BPEL allows the integration of people into the provisioning flow via the BPEL4People [31] extension.

Provisioning flows themselves are again provided as provisioning services so that they can be composed into higher-level provisioning flows. Provisioning services are annotated with provider policies that describe their non-functional properties and the guaranteed quality of services. For example, a provisioning service that can set up an

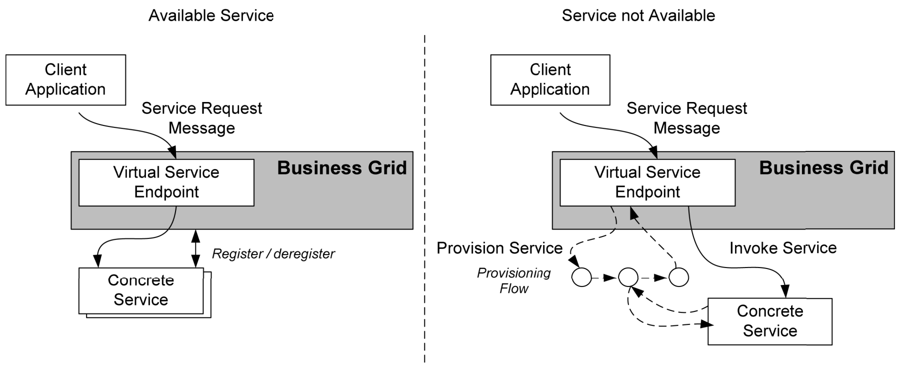


Fig. 2. Virtualization and dynamic provisioning on the Business Grid

application on a single server might have a policy attached that does not make any statement about the availability of the application, while a provisioning service provided by a sophisticated provisioning engine that installs the application on a cluster of servers guarantees an availability of 99.9999%. Depending on the requirements of the user, expressed through a policy in the request, the Business Grid middleware then chooses the provisioning service that fulfills these requirements by matching the requestor and the provider policies.

The indirection via the Business Grid infrastructure (which itself can be distributed as a Grid of enterprise service buses) allows to deal with *transient resources* that are sometimes available and sometimes not. When a resource leaves the Business Grid the Business Grid infrastructure detects this (either by polling or because the resource deregisters) and automatically runs a corresponding provisioning flow that provisions another resource so that it can replace the leaving resource. Additionally, through dynamic provisioning the Business Grid infrastructure can automatically add more resources to an application when needed and thus increase the amount of concurrent users that can use the application. The infrastructure can balance the load on resources by dynamically redirecting requests to resources that have available computing power. Therefore Business Grid Resource are not only annotated with policies that describe static quality of services such as that it can be run under transactions or that it must be invoked with encrypted messages, but also with dynamic properties that may be derived from underlying infrastructure, such as its utilization or available storage. This combination of static and dynamic properties is advertised as a policy by the resource and is used by the Business Grid middleware to decide to which resource the request is routed.

6 Business Grid Applications

Applications for business as well as for science can be deployed on the Business Grid. Applications are exposed as WS-Resources, i.e. via abstract endpoints [3], regardless of their implementation paradigm. The applications play the role of a service requester in SOA terms when they use services via the Business Grid. However, applications may also play the role of a service provider if used by other applications. The Business Grid is the gateway for the applications, which means that the whole communication between the application and any other WS-Resources is tackled by the middleware; any service calls are sent to a single logical entry point implemented by the middleware. Users can interact with applications via their user interfaces that may be utilizing the service interface of the application. Applications may choose to use the Business Grid for discovery of WS-Resources only.

Business Grid applications may possess their own management and monitoring facilities, possibly provided by the platform they are executed on, however for the purposes of configuration, management and monitoring on the Business Grid, they must expose such functionalities via management and monitoring interfaces in addition to the interface through which the service operations are made available. Applications describe their requirements toward partner services using policies, which may be derived from KPIs [45]. The fact that other applications use the Business Grid remains transparent for each application; applications are built with the assumption that they are the only users of the infrastructure. It is up to the Business Grid to maintain proper operation and enable reliability, scalability and emulate the isolation of applications when used by them. Flexible Business Grid applications rely heavily also on dynamic binding to services via the Business Grid as well as on dynamic staff query resolution across several virtual organizations in the case humans are the resources responsible for carrying out a function.

7 Conclusions

In this paper we argue that there is a need for a new kind of infrastructure that addresses the requirements of both scientific applications and enterprise applications. We call this infrastructure the Business Grid: it combines and unifies approaches for Grids and ESBs. It allows for the development, execution and monitoring of both scientific applications, which are data and computation intensive and current business applications that are reliable and scalable. The Business Grid leverages existing techniques from both fields to combine them in a powerful computing infrastructure with superior characteristics.

One contribution of this work is that we identify the similarities in requirements towards Grids and ESBs and the similarities and overlaps, as well as differences in addressing problems in both areas. The main requirements a Business Grid is expected to meet are: (i) robustness and reliability, a requirement that comes from both Business and Grids, and (ii) support for data and computing intensive applications, a need not yet met by conventional ESBs. Solutions to these demands draw on hardware and software provisioning techniques. We have also introduced the architecture of the Business Grid, which we devised considering existing middleware and Grid architectures. A Business Grid is a composable infrastructure that virtualizes both services and resources in a unified manner. It can be implemented by a combination of Grid and ESB middleware implementations and in fact virtualizes these implementations. Any application from the business world and scientific computing can be run on this infrastructure.

In this work we do not argue that there is a need of a totally new infrastructure to address the needs of two domains, but rather we promote reuse of research results and technologies that if applied in combination enable a powerful abstraction applicable in both business and scientific applications.

References

1. van der Aalst, W.M.P.: Verification of Workflow Nets. In: Proceedings of the 18th International Conference on Application and Theory of Petri Nets, June 23-27, pp. 407–426 (1997)
2. Amazon.com, Amazon Elastic Computing Cloud, `http://aws.amazon.com/ec2`
3. Chappell, D.: Enterprise Service Bus. O'Reilly Media, Inc., Sebastopol (2004)
4. Chong, F., Carraro, G.: Building Distributed Applications Architecture Strategies for Catching the Long Tail (2006), `http://msdn2.microsoft.com/enus/library/aa479069.aspx`

5. Czajkowski, K., et al.: From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring & Evolution, Global Grid Forum Draft Recommendation (2004)
6. Emmerich, W., et al.: Grid Service Orchestration Using BPEL. J. of Grid Computing (2005)
7. Foster, I., Kesselman, C., Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Int. J. High Perform. Comput. Appl. 15, 200–222 (2001)
8. Foster, I., Kesselman, C.: The Grid 2: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, San Francisco (2004)
9. Foster, I., et al.: The Open Grid Services Architecture, Version 1.0. GFD-I, Vol. 30 (2005)
10. Foster, I., et al.: Brain Meets Brawn: Why Grid and Agents Need each Other. In: Proc. Conference on Autonomous Agents and Multi-Agent Systems, New York (2004)
11. Fox, G.C., Gannon, D.: Workflow in Grid Systems. Concurrency and Computation: Practice and Experience 18, 1009–1019 (2006)
12. Gannon, D.: A Service Architecture for eScience Grid Gateways. In: GADA 2007 (2007)
13. IBM Tivoli Provisioning Manager, `http://www.ibm.com/software/tivoli/products/prov-mgr/`
14. Jha, S., Fox, A.G.: Using Clouds to Provide Grids Higher-Levels of Abstraction and Explicit Support for Usage Modes. In: OGF 2008 (2008)
15. Karastoyanova, D., et al.: Semantic Service Bus: Architecture and Implementation of a Next Generation Middleware. In: Proceedings of SEIW at ICDE 2007 (2007)
16. Karastoyanova, D., et al.: Parameterized BPEL Processes: Concepts and Implementation. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 471–476. Springer, Heidelberg (2006)
17. Keller, A., Badonnel, R.: Automating the Provisioning of Application Services with the BPEL4WS Workflow Language. In: Sahai, A., Wu, F. (eds.) DSOM 2004. LNCS, vol. 3278, pp. 15–27. Springer, Heidelberg (2004)
18. Kephart, J., Chess, D.: The vision of autonomic computing. Computer 36 (2003)
19. Kephart, J.: Research Challenges of Autonomic Computing. In: Proc. of ICSE 2005 (2005)
20. Krauter, K., et al.: A Taxonomy and Survey of Grid Resource Management Systems for Disitributed Computing. Software – Practice and Experience 32(2), 135–164 (2002)
21. Kuropka, D., Weske, M.: Implementing a Semantic Service Provision Platform — Concepts and Experiences. Journal Wirtschaftsinformatik, Issue 1/2008 (2008)
22. Layaida, O., et al.: A Framework for Dynamically Configurable and Reconfigurable Network-based Multimedia Applications. Journal of Internet Technology (October 2004)
23. Leymann, F.: The (Service) Bus: Services Penetrate Everyday Life. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 12–20. Springer, Heidelberg (2005)
24. Leymann, F.: Choreography for the Grid: towards fitting BPEL to the Resource Framework: Journal of Concurrency and Computation: Pract. & Experience 18 (2006)
25. van Lessen, T., et al.: An Execution Engine for Semantic Business Process. In: Proceedings of SeMSoC at the ICSOC 2007, Vienna, Austria (September 2007)
26. Mietzner, R., et al.: Defining Composite Configurable SaaS Application Packages Using SCA, Variability Descriptors and Multi-Tenancy Patterns. In: ICIW 2008 (2008)
27. Mietzner, R., Leymann, F.: Towards Provisioning the Cloud: On the Usage of Multi-Granularity Flows and Services to Realize a Unified Provisioning Infrastructure for SaaS Applications. In: SERVICES 2008 (2008)
28. Srinivasan, N., et al.: An Efficient Algorithm for OWL-S based Semantic Search in UDDI. In: Cardoso, J., Sheth, A.P. (eds.) SWSWPC 2004. LNCS, vol. 3387, pp. 96–110. Springer, Heidelberg (2005)

29. Nitzsche, J., et al.: BPEL for Semantic Web Services. In: Proceedings of AWeSome 2007 (2007)
30. OASIS BPEL TC, `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel`
31. OASIS BPEL4People TC, `http://www.oasis-open.org/committees/bpel4people/charter.php`
32. OASIS WSRF TC, `http://www.oasis-open.org/committees/documents.php?wg_abbrev=wsrf`
33. OpenQRM, `http://www.openqrm.org/`
34. Pottinger, S., Mietzner, R., Leymann, F.: Coordinate BPEL scopes and processes by extending the WS-business activity framework. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 336–352. Springer, Heidelberg (2007)
35. Reichert, M., Dadam, P.: ADEPT$_{flex}$-Supporting Dynamic Changes of Workflows Without Losing Control. J. Intell. Inf. Syst. 10(2), 93–129 (1998)
36. Reichwald, J., et al.: Model-Driven Process Development Incorporating Human Tasks in Service-Oriented Grid Environments. In: Multikonferenz Wirtschaftsinformatik (2008)
37. Shafiq, M.O., et al.: Autonomous Semantic Grid: Principles of Autonomous Decentralized Systems for Grid Computing. In: Proc. of IEICE 2005 (2005)
38. Slomiski, A.: On using BPEL extensibility to implement OGSI and WSRF Grid workflows. Concurrency and Computation: Practice & Experience 18 (2006)
39. Taylor, I.J., et al.: Workflows for e-Science: Scientific Workflows for Grids (2006)
40. Tuecke, S., et al.: Open Grid Services Infrastructure (OGSI) Version 1.0 (2003)
41. Unger, T., Leymann, F., Mauchart., S., Scheibler, T.: Aggregation of Service Level Agreements in the Context of Business Processes. In: Proc. EDOC 2008 (2008)
42. Venugopal, S., Buyya, R., Winton, L.: A Grid service broker for scheduling e-Science applications on global data Grids. Concurr. Comput. Pract. Exper. 18, 6 (2006)
43. W3C, Web Services Policy Framework, W3C Member Submission
44. Weerawarana, S., et al.: Web Services Platform Architecture. Prentice Hall, Englewood Cliffs (2005)
45. Wetzstein, B., Karastoyanova, D., Leymann, F.: Towards Management of SLA-Aware Business Processes Based on Key Performance Indicators. In: Proc. of BPMDS 2008 (2008)

Does My Service Have Partners?

Karsten Wolf

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

Abstract. Controllability for service models is a similar criterion as soundness for workflow models: it establishes a necessary condition for correct behavior of a given service model. Technically, controllability is the problem to decide, for a given service, whether it can interact correctly with at least one other service. Parameters to the problem are the established correctness criterion (e.g. deadlock freedom, livelock freedom, quasi-liveness), the shape of partners (centralized partners versus independently acting partners), or the shape of communication (asynchronous versus synchronous).

In this article, we survey and partly extend various recent results concerning the verification of controllability for Petri net based service models. Significant extensions include the study of livelock freedom as correctness criterion as well as the new results on autonomous multi-port controllability.

1 Introduction

Service oriented computing [15,29,14,1] is a paradigm that can be applied in the management of interorganisational workflows, for the programming-in-the-large, for loosely coupled interaction and aggregation over the web, and probably for many more use cases. It is centered around the concept of a *service*, i.e. a self-contained and self-explaining software unit that offers an encapsulated functionality over a well-defined interface.

These days, the language WS-BPEL [2] is one of the most important languages for the specification of services in practice. WS-BPEL specifications can be transformed into formal models using one of the various formal semantics, among which some [33,24,13] are feature complete, i.e. cover all exceptional behavior including fault handling, compensation handling, and termination handling. Petri nets are particularly useful as a formal model of services as there is even a translation from Petri nets back to WS-BPEL [20].

A service is controllable if it has at least one partner such that the composition of both is well-behaving. This question can be asked for several notions of "well behaving" as well as for various settings concerning the shape of services and their mutual communication.

In this article, we collect results concerning controllability in different settings. In particular we vary the correctness criterion (deadlock freedom, livelock freedom, quasi-liveness), the shape of partners (single partners, several partners with different degrees of coordination), the shape of communication (asynchronous versus synchronous), and study the incorporation of additional constraints. The results include published work, results from a number of diploma (master) theses, and some original work. Original contributions include the study of controllability for livelock freedom and the generalization of results on autonomous controllability.Status and origin of the results is reported in the respective sections.

Controllability may be used as a tool for verifying service models. An uncontrollable service is certainly malfunctioning (while a controllable service may or may not be correct). Our decision procedures are able to synthesize a particular correctly interacting partner. This partner can be transformed into executable code. This possibility may have interesting applications, e.g. in the synthesis of adapters between incorrectly interacting services [5,6,11,4]. Additionally, the synthesized partner may form the basis for a characterization of *all* partners of a services and resulting applications for service discovery [23] or substitutability considerations [35]. In Sect. 5, we show how additional constraints can be placed in the construction of a correctly interacting partner. Section 6.1 studies nonstandard models of communication. Finally, we discuss related work and summarize our results.

In the remainder of this article, we first introduce notations for Petri net models for services and define basic properties. Section 3 investigates controllability for services with just one partner under different correctness criteria. In Sect. 4, services with multiple partners are considered.

2 Petri Net Models of Services

We start from place/transition nets $N = [S, T, F, m_0]$ with the usual meaning of the constituents and the standard firing rule. We denote with $m \xrightarrow{t}_N$ that t is enabled in marking m of net N, with $m \xrightarrow{t}_N m'$ that firing t in m of net N leads to m', and with $m \xrightarrow{*}_N m'$ that m' is reachable from m in net N using some firing sequence, including the empty sequence. Let $R_N(m) = \{m' \mid m \xrightarrow{*}_N m'\}$ be the set of markings reachable from m in N. For modeling services, we refine place/transition nets to *open nets*—the net class used in [33,24] for defining a formal semantics of WS-BPEL.

Definition 1 (Open net). *$N = [S, T, F, m_0, S_i, S_o, M_F, \mathcal{P}]$ is an* open net *iff $[S, T, F, m_0]$ is a place-transition net, $S_i \subseteq S$ is the set of* input channels *such that $F \cap (T \times S_i) = \emptyset$ (no arc has its sink in S_i), $S_o \subseteq S$, $S_i \cap S_o = \emptyset$ is the set of* output channels *such that $F \cap (S_o \times T) = \emptyset$ (no arc has its source in S_o), M_F is a finite set of markings called* final markings *such that no $m \in M_F$ enables any transition, for all markings in $M_F \cup \{m_0\}$, all places in the* interface *$I := S_i \cup S_o$ are unmarked, and $\mathcal{P}$ is a partition of I with elements called* ports*. An open net with empty interface ($|I| = 0$) is called a* closed net*. An open net with just one port ($|\mathcal{P}| = 1$) is called* single-port net*. An open net with more than one port ($|\mathcal{P}| > 1$) is called a* multi-port net*. Open net N is called* normal *iff every transition of N is connected to at most one interface place. For a normal* *open net N, define mapping $l : T \rightarrow I \cup \{\tau\}$ such that $l(t)$ is the unique interface place adjacent to t if one exists, and $l(t) = \tau$ if t is not adjacent to any interface place.*

Input and output channels represent the interface of a service which is assumed to have an asynchronous message-passing nature. Synchronous communication shall be studied in Sect. 6.1. Channels are grouped into ports. A port represents the interface of the service to a particular partner. It corresponds to the concept of partner links in WS-BPEL which is typically specified using the language WSDL. For single-port nets, we usually omit $\mathcal{P}$. In the sequel, we shall only study normal open nets. This restriction is not significant, as every open net can be transformed into an equivalent normal open net [23]. Final markings model the successful completion of the modeled service. In the sequel, we shall use indices to distinguish the constituents of different open nets.

For an open net, its inner structure is defined by removing the interface places. We define some properties of an open net by referring to its inner structure. For a mapping f, let $f\mid_M$ be the restriction of f to arguments in M.

Definition 2 (Inner structure, Boundedness, Responsiveness). *Given an open net $N = [S, T, F, m_0, S_i, S_o, M_F, \mathcal{P}]$, the corresponding* inner structure *$Inner(N)$ is the closed net $Inner(N) = [S \setminus I, T, F \setminus ((I \times T) \cup (T \times I)), m_0\mid_{S\setminus I}, \emptyset, \emptyset, M_F\mid_{S\setminus I}, \emptyset]$. N is* bounded *iff $R_{Inner(N)}(m_0\mid_{Inner(N)})$ is finite. N is* responsive *iff, from each reachable marking in $Inner(N)$, a marking is reachable in $Inner(N)$ which is final or which enables a transition t which is connected to some interface place $((\{t\} \times I_N) \cup (I_N \times \{t\}) \neq \emptyset)$.*

Figure 1 shows three bounded and responsive open nets. Net (a) and (b) are normal while (c) is not. Ports are visualized by dashed boxes. Final markings are those markings where no transition can fire in the inner of the respective net. Throughout this article, we study bounded and responsive open nets. For unbounded open nets, even simple controllability problems are undecidable [28]. The property of responsiveness rules out services which run into deadlocks or endless loops without communication. Both boundedness and responsiveness can be verified using standard state space verification techniques [8]. Translations from single WS-BPEL processes to open nets are responsive by construction. Deadlocks are avoided by a concept called *dead path elimination.* Using this concept, control flows through a disabled activity by explicitly skipping it. This way, there cannot be any permanently blocked point of control. Loops always be left as, on our level of abstraction, the exit condition is modeled as a nondeterministic choice.

Services are connected by simply connecting appropriate interface places. Informally, we may compose services which are connected via at most one pair of ports such that different nets use different ports.

Definition 3 (Composition of open nets). *Two open nets N_1 and N_2 are* composable *iff there exist a port $P_1 \in \mathcal{P}_1$ and a port $P_2 \in \mathcal{P}_2$ such that $\emptyset \neq S_1 \cap S_2 \subseteq P_1 \cup P_2$, $T_1 \cap T_2 = \emptyset$, $S_{i1} \cap S_{i2} = \emptyset$, and $S_{o1} \cap S_{o2} = \emptyset$. The* composition

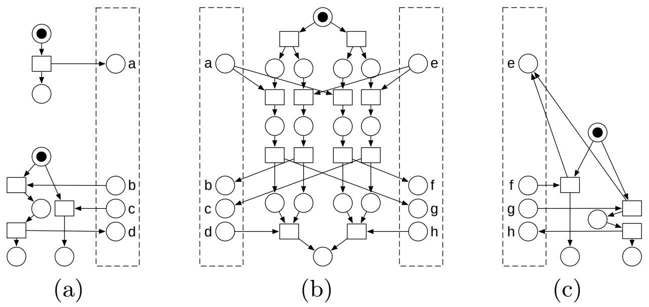


Fig. 1. Two single-port open nets (a,c) and a multi-port open net (b)

$N_1 \oplus N_2$ of two composable open nets N_1 and N_2 is the net N with the following constituents: $S = S_1 \cup S_2$, $T = T_1 \cup T_2$, $F = F_1 \cup F_2$, $m_0 = m_{01} \oplus m_{02}$, $S_i = (S_{i1} \cup S_{i2}) \setminus (P_1 \cup P_2)$, $S_o = (S_{o1} \cup S_{o2}) \setminus (P_1 \cup P_2)$, $M_F = \{m \oplus m' \mid m \in M_{F1}, m' \in M_{F2}\}$, and $\mathcal{P} = (\mathcal{P}_1 \cup \mathcal{P}_2) \setminus \{P_1, P_2\}$. Thereby, $m_1 \oplus m_2$ is the marking satisfying $(m_1 \oplus m_2)(s) = m_1(s)$ for $s \in S_1$, and $(m_1 \oplus m_2)(s) = m_2(s)$ for $s \in S_2$.

For the markings involved in this definition, the composition operation $\oplus$ is well defined, as none of them marks interface places. If the result of multiple composition does not depend on the order of application (up to isomorphism), we use the notation $N_1 \oplus N_2 \oplus \cdots \oplus N_k$ for the composition of k open nets. In Fig. 1, open nets (a) and (c) are composable to net (b). Composition of all three leads to a closed net.

Services are executed in composition with other services. Consequently, behavioral properties are only defined for closed nets, i.e. complete service choreographies.

Definition 4 (Behavior). *A closed net N is* deadlock-free *(DF) if, for every $m \in R_N(m_0) \setminus M_F$, there is a transition enabled in m. N is* livelock-free *(LF) if, for all $m \in R(m_0)$, $R_N(m) \cap M_F \neq \emptyset$. N is* quasi-live *(QL) if, for all $t \in T$, there is an $m \in R(m_0)$ such that $m \xrightarrow{t}_N$.*

The composition of the three nets in Fig. 1 forms a closed net with properties DF, LF, and QL. The well-known property of soundness of workflow nets [36] closely corresponds to the properties LF and QL. Note that the composition of responsive nets is not necessarily deadlock-free, livelock-free, or quasi-live.

Definition 5 (Controllability, Strategy). *Let $X \subseteq \{DF, LF, QL\}$ and $k \in \mathbf{N} \setminus \{0\}$. Let N be a normal, bounded, and responsive open net with $|\mathcal{P}| = j$, for some j. N is X, k-*controllable *if there exist normal, bounded, and responsive single-port services $N_1, \ldots, N_j$ such that $N^* = N \oplus N_1 \oplus \cdots \oplus N_j$ is a closed net holding all properties in X, and, for all markings m reachable from m_0^* in N^*, and all $s \in I_N$, $m(s) \leq k$. In this case, $[N_1, \ldots, N_j]$ is called an X, k-*strategy *of N. Denote $Strat_{X,k}(N)$ the set of all X, k-strategies for a given open net N.*

In Fig.1, all depicted nets are X, k-controllable for any $X \subseteq \{DF, LF, QL\}$ and all $k > 0$. For net (b), the ordered pair consisting of nets (a) and (c) forms an X, 1strategy, for net (a), the composition of nets (b) and (c) forms an X, 1-strategy, and for net (c), the composition of (a) and (b) forms an X, 1-strategy.

The introduction of the parameter k establishes an artificial limit on the number of pending messages in a single channel. Technically, it assures (for any value) boundedness of N^* which is a pre-requisite for the algorithms studied in the sequel. Pragmatically, it could either represent a reasonable buffer size in the middleware, be the result of a static analysis of the communication behavior of a service, or simply be chosen sufficiently large. Note that N and $N_1, \ldots, N_j$ are required to be responsive while N^* is intentionally not required to be responsive (N^* is not the model of a service but rather the representation of a transition system).

Open nets have been proposed in various contexts. For the modeling of interorganizational workflows or services, they have been proposed under various names and with different but mostly insignificant syntactic restrictions in [19,18,25,26,31,27].

3 Controllability of Single-Port Services

We decide controllability of a single-port service N through the attempt of constructing a particular strategy of N. We actually construct a transition system which forms the state space of the inner of the strategy. This transition system can, however, be transformed into a Petri net by either transforming it straightforward into a Petri net state machine, or using the sophisticated theory of regions [12,3,10]. As these translations are well understood, this paper shall only consider the construction of the transition system. For simplicity, a transition system that corresponds to a strategy is called strategy, too.

In the first part of this section, we describe the principles of constructing a strategy. Then we illustrate our approach with an example. In the last part, we discuss techniques for improving the performance of our construction.

3.1 Constructing a Strategy as Transition System

Throughout this section, let N be a normal, bounded, and responsive open net.

In a first step, we overapproximate the behavior of arbitrary N' which are composable to N by a partner transition system TS_0. Then, we iteratively remove states (and edges) which cause violations of the properties to be preserved. If the resulting transition system is empty, we shall conclude uncontrollability of N. Otherwise, the remaining transition system represents a strategy of N.

TS_0 is inductively defined. We use sets of markings of N and an additional item # for the definition of states of TS_0. The core idea is to identify a state q of TS_0 with the set of markings that N can be in while TS_0 is in q. Item # is used to represent final states. For final states, we immediately implement the restrictions required in Def. 1.

Definition 6 (Overapproximation of arbitrary partners). *For a set of markings M of N, define $closure(M) = \{m' \mid m \in M, m \xrightarrow{*}_N m'\}$. Let $TS_0 = [Q, E, q_0, Q_F]$ (consisting of a set Q of states, a set $E \subseteq Q \times (I_N \cup \{\tau\}) \times Q$ of edges, an initial state $q_0 \in Q$), and a set $Q_F \subseteq Q$ of final states be defined inductively as follows.*

- *$q_0 = closure(\{m_0\})$; $q_0 \in Q$;*
- *If $q \in Q$ then*
  - *if $\# \notin q$ then $q' = q \cup \{\#\} \in Q$, $[q, \tau, q'] \in E$, and $[q, \tau, q] \in E$;*
  - *if $x \in S_i$ and $\# \notin q$ then $q' = q + x := closure(\{m + [x] \mid m \in q\}) \in Q$ and $[q, x, q'] \in E$;*
  - *if $x \in S_o$ then $q' = q - x := \{m - [x] \mid m \in q, m(x) > 0\} \setminus \{\#\} \in Q$ and $[q, x, q'] \in E$.*

Thereby, $[x]$ is a marking with $[x](x) = 1$ and $[x](y) = 0$, for $y \neq x$. Operations $+$ and $-$ on markings are defined pointwise. Let $Q_F = \{q \in Q \mid \# \in q\}$.

TS_0 represents arbitrary N' composable to N in the sense that TS_0 can simulate the behavior of N'. The following lemma exhibits the corresponding simulation relation.

Lemma 1. *Let $TS_0 = [Q, E, q_0, Q_F]$ be constructed as defined above. Let N' composable with N and consider the relation $\rho \subseteq R_{Inner(N')}(m_{0Inner(N')}) \times Q$ inductively defined as follows:*

- *$m_{0Inner(N')} \rho q_0$*
- *If $m \rho q$ and $m \xrightarrow{t}_N m'$, then there is a q' such that $\{\#\} \notin q'$, $[q, l(t), q'] \in E$, and $m' \rho q'$;*
- *If $m \rho q$ and $m \in M_{FN'}$ then $m \rho (q \cup \{\#\})$.*

With this ρ, marking m^* is reachable in $N \oplus N'$ if and only if there exists a $q \in Q$ such that $m^* |_{Inner(N')} \rho q$ and $m^* |_N \in q$.

ρ is actually a simulation relation between N' and TS_0. Uniqueness of ρ is easily verified since, for every state q of TS_0 and every $x \in I \cup \{\tau\}$, there is exactly one edge with source q and label x (in Def. 1, q' id always uniquely determined from q and x). τ-steps in TS_0 from $q \setminus \{\#\}$ to $q \cup \{\#\}$ are executed exactly if the corresponding marking in N' is final.

For the main claim, observe that the edge relation of TS_0 just reflects the impact of producing or consuming tokens in the interface of N which in turn corresponds to firing transitions in N' which are adjacent to interface places.

In the subsequent steps, we modify TS_0 (by removing states and edges) such that the assertions of Lemma 1 are preserved at least for all strategies of N. For establishing the preservation result, we argue that the removed states cannot be related to any reachable state of a strategy. To this end, we consult the composition $TS \oplus N$ of a transition system TS (like TS_0) with N. This composition is straightforward: a state in the composed system consists of a state of TS and a marking of N, and enabled transitions of the two subsystems are arbitrarily interleaved. The only remarkable feature is that a transition $[q, x, q']$ of TS with $x \in S_i$ adds a token to x, a transition $[q, x, q']$ with $x \in S_o$ removes a token from x (and is only enabled if such a token is present), and a transition $[q, \tau, q']$ does not touch the marking of N at all.

Definition 7 (Transformation of TS_0 to DF, k-strategy). *Let TS_1 be the transition system that is obtained from TS_0 by removing all states q which contain a marking m where, for some $x \in I_N$, $m(x) > k$ (removing a state is always meant to include removal of all adjacent edges as well as removal of all states which become unreachable. Given a transition system TS_i ($i > 0$), transition system TS_{i+1} is obtained by removing, for all $q \in Q_i$, state q if*

- *$[q, \tau, q]$ is the only edge with source q in TS_i, or*
- *there exists an $m \in q$ where*
  - *no state $[q', m']$ is reachable in $TS_i \oplus N$ from $[q, m]$ where $m' \in M_F$, and*
  - *no transition t with $l(t) \neq \tau$ is reachable in $TS_i \oplus N$ from $[q, m]$.*

Let $Q_{FTS_{i+1}} = \{q \in Q_{TS_{i+1}} <| \# \in q\}$. Let TS^* be TS_j for the smallest j with $TS_j = TS_{j+1}$.

Theorem 1. *N is DF, k-controllable if and only if $Q_{TS^*} \neq \emptyset$.*

Proof (idea). The construction terminates as TS_1 is finite, due to boundedness of N and the restriction to k tokens on interface places (TS_1 can actually be constructed directly, i.e. without explicitly constructing the possibly infinite TS_0 first). Let N' be a DF, k-strategy of N and consider the simulation relation ρ of Lemma 1. No marking of $Inner(N')$ can be related to any removed state q as otherwise

- N' would not be a $\emptyset, k$-strategy (q not present in TS_1), or
- $N' \oplus N$ would contain a deadlock (as the responsive net N cannot perform τ-transitions forever, and neither a final marking nor a communicating transition are reachable).

Due to the removal of τ-edges in the first item, TS^* is responsive. Consequently, if N has any DF, k-strategy, simulation relation ρ to TS^* exists and must have states in its co-domain, so $Q \neq \emptyset$. On the other hand, a nonempty TS^* as such is a DF, k-strategy. q.e.d.

Note that final states only play a minor role for DF-controllability. For LF, k-controllability, we need to modify the transformation. In particular, final states of TS_i become more important.

Definition 8 (Transformation of TS_0 to LF, k-strategy). *Proceed as in Def. 7, but remove q if there exists an $m \in q$ where no state $[q', m']$ is reachable in $TS_i \oplus N$ from $[q, m]$ where $q \in Q_{FTS_i}$ and $m' \in M_F$.*

It is easy to verify that the removed states cannot be related to states of any strategy.

Corollary 1. *N is LF, k-controllable if and only if the modified construction yields $Q_{TS^*} \neq \emptyset$.*

As all transformations preserve the assertions of Lemma 1 for all strategies of N, we can use the constructed partner for verifying quasi-liveness.

Corollary 2. *Let TS^{**} be obtained from TS^* by removing state $\emptyset$. N is $\{DF, QL\}, k$-controllable if and only if it is DF, k-controllable and $TS^{**} \oplus N$ is quasi-live (where TS^* is constructed as in Def. 7). N is $\{LF, QL\}, k$-controllable if and only if it is LF, k-controllable and $TS^{**} \oplus N$ is quasi-live (where TS^* is constructed as in Def. 8).*

State $\emptyset$ may be an element of Q. It represents states which may be present in a strategy but are actually unreachable in composition with N. This happens if N' waits for the receipt of an x which is not being sent by N. For DF-controllability and LF-controllability, this state does not harm. For quasi-liveness it is, however, necessary to take care of $\emptyset \in Q$. Due to Lemma 1 and the subsequent constructions, all states $[q, m]$ with $q \in Q^*$ and $m \in q$ are actually reachable.

3.2 Example

Consider Fig. 2 with the depicted marking as initial marking and $M_F = \{[\gamma]\}$. The net is normal, bounded, and responsive. Consider first DF, 1-controllability. Figure 3 shows the relevant part of transition system TS_0. Boxes represent states (with adjacent numbers). The content of a state consists of the markings depicted within the box (one marking per line).

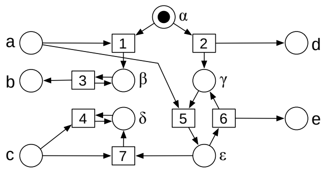


Fig. 2. Example net for studying single-port controllability

We depicted only one state ($2'$) containing #. For the remaining states, corresponding #-states are rather irrelevant as they do not contain final markings of N. All other absent states violate the bound of 1 for tokens on the interface. States 1 and 4 are shown as examples of such states (with infinite contents in state 1). These states are removed during the transformation from TS_0 to TS_1 (in practice, their generation would be stopped upon first appearance of a second token on an interface place). That is, TS_1 consists of states $0, 2, 2', 3, 5, 6, 7$.

During further transformation, state 7 is removed as presence of marking $[\delta]$ contradicts responsiveness. In consequence, state 6 and afterward state 5 are removed since γc (and then γcd) become states from which neither a final

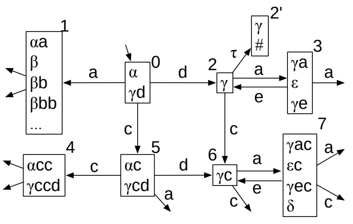


Fig. 3. Relevant part of TS_0 as computed for the net in Fig.2

marking nor a communicating transition is reachable. For states $0, 2, 2', 3$, there is no reason to remove them. So these states form a transition system that witnesses the existence of a DF, 1-strategy.

The same result is obtained if LF, 1-controllability is considered instead, since $[2', \gamma]$ is reachable from every marking in the system composed of N and TS^*.

If the set of final markings of N is set to $M_{FN} = \{[\delta]\}$ instead, results for DF, 1-controllability remain unchanged as the net can loop forever. The net would, however, become LF, 1-uncontrollable as, after removing states 7 and the corresponding state $7'$ (not depicted), the composed system would not have a remaining final state and our procedure would remove all remaining states of TS^*.

3.3 Implementation

The results of this section have been implemented in the tool Fiona [21]. A number of optimizations can be applied. First, states of Q can be represented by simple references to states in the composed system $TS_i \oplus N$. Second, the removal of states can be interleaved with the generation of TS_1. Third, the stubborn set method [34] and BDD-based symbolic representations [7] can be applied in order to reduce the size required for storing $TS_i \oplus N$, or TS^*. Fourth, the constructed strategies enjoy a number of regularities that can be exploited. For instance, subsequent send (receive, resp.) transitions in TS^* appear in all permutations (reflecting the asynchronous nature of communication). Receive transitions cannot be disabled by send transitions. Send transitions cannot be disabled by receive transitions. Fifth, we can investigate N for obtaining upper limits for the future number of consumable tokens on certain message channels.

Despite a devastating theoretical complexity somewhere in the double exponential area, it turns out that our approach is indeed applicable to interesting examples. The services in our benchmark set stem from WS-BPEL processes that we obtained from small and medium size companies (23), WS-BPEL processes from tutorials in WS-BPEL engines, the WS-BPEL specification, or text books on WS-BPEL (21), and from UML activity diagrams (40). The UML diagrams actually model workflows describing the claim management of a real insurance company. We cut these workflows into services according to the annotated role information.

Using some of the optimizations stated above, computing a DF, 1-strategy (as transition system) took less than a second for 66 of these 84 processes and between 1 and 10 seconds for another 17 processes. In the single remaining case, computation required 143 seconds. In this time, Fiona investigated an initial transition system (TS_1) of 108733 states. The final system (TS^*) consists of 11280 states. The harmful service had a rather exceptional interface consisting of 16 output places but 0 input places. Thus, Fiona had to investigate all possible orders of receiving 16 messages. Another source of complexity would be a non-trivial internal behavior of a service. Fortunately, our translation from WS-BPEL to open nets uses a number of Petri net reduction rules for simplifying this internal behavior.

As far as DF-controllability is concerned, results in this section stem from [32,25,37,21]. The approach to LF-controllability is original work.

4 Controllability of Multi-port Services

Let $N = [S, T, F, m_0, S_i, S_o, M_F, \mathcal{P}]$ be a multi-port service. Depending on assumptions on the coordination of partners of N, we distinguish three scenarios. In the first scenario, partners may communicate with each other arbitrarily. Controllability in this setting (run-time coordination) is equivalent to the single-port controllability of $N^{\cup} = [S, T, F, m_0, S_i, S_o, M_F, \bigcup \mathcal{P}]$ as one of the partners N_i may just execute a strategy of $N^{\cup}$ while the remaining partners just transmit messages from N_i to N and vice versa.

In the second scenario, partners of N communicate with N, but not with each other. They may, however, act in a coordinated fashion (build-time coordination), i.e. single partners cannot be exchanged. This setting is discussed in the first subsection.

In the third scenario, we deal with controllability under the assumption that there is neither run-time coordination nor build-time coordination between the partners of N.

The following example may illustrate the difference. Net (a) of Fig. 4 depicts a service. Assume that a and b belong to different ports. Then the two pairs of transition systems in the right part depict multi-port strategies of the net. The partners do not communicate with each other at run-time. However, by executing a, the left partner must somehow "rely" on the fact that the right partner does not execute b. By not executing a, this partner relies on the emission of b by the other partner. In this sense, the two partners act coordinatedly. If we rule out build-time coordination, we would like to qualify the net as uncontrollable. We believe that the absence of build-time coordination is typical in practical applications: consider a service of a travel agency which has customers and flight reservation systems among its partners. The customers should not talk directly to the flight reservation, and they should not even know the program that the flight reservation service is running for its communication with the travel agency.

4.1 Decentralized Control

The presence of build-time coordination is immediately reflected in Def. 5. This is demonstrated using Fig. 4. Net (a) is controllable as the strategies (c) and (d)

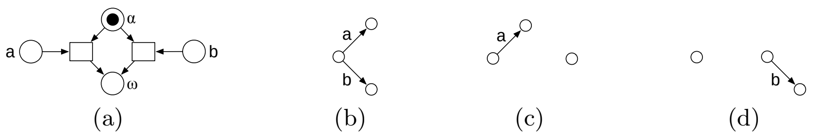


Fig. 4. An open net (a). In the single-port strategy (b), there is a choice between sending an a and sending a b. In the two multi-port strategies (c) and (d), one partner can send a message while the other one has to remain silent (consists of a single state which is both initial and final).

prove. In these strategies, one of the two partners is depicted as a single (initial and final) state which says that this partner remains silent. However, in being silent, the respective partner *relies* on the fact that the respective other partner does not remain silent. The other way round, a partner who performs an activity (sending a or b) relies on the silence of the respective other partner. Net (a) is only controllable by pairs of partner where each partner acts in a coordinated fashion with the other one. This coordination is a build-time coordination. At run-time, there is no direct communication between the partners.

Results for multi-port controllability exist only for acyclic services. For these services, properties DF and LF coincide.

For deciding multi-port controllability, we observe that, for every multi-port strategy $[N_1, \ldots, N_j]$ of N, its composition $N_1 \oplus \cdots \oplus N_j$ is in fact a single-port strategy of $N^{\cup}$. Consequently, it must be possible to embed $N_1 \oplus \cdots \oplus N_j$ into the transition system TS^* considered in the previous section. Additionally, transitions of different N_i do not interfere with each other, that is, they occur arbitrarily interleaved in $N_1 \oplus \cdots \oplus N_j$.

The idea for deciding multi-port controllability is thus to start from the single-port strategy TS^* of $N^{\cup}$ and to remove edges which violate the requirement of arbitrary interleaving with edges that belong to different ports. Arbitrary interleaving is violated if one of the following conditions hold in TS^* for some a and b from different ports:

- a enables b: $q \xrightarrow{ab}$ but not $q \xrightarrow{b}$;
- a disables b: $q \xrightarrow{a}$, $q \xrightarrow{b}$, but not $q \xrightarrow{ab}$;
- a and b are not independent: $q \xrightarrow{ab} q'$, $q \xrightarrow{ba} q''$, but q' and q'' are not independent.

The resulting algorithm is nondeterministic since independence can sometimes be enforced by removing different edges (for example, if a disables b, we can remove either the a-successor or the b-successor of q). Thus, the computed strategy is not necessarily unique. However, every multi-port strategy is at most as permissive as one of the strategies that can be computed by investigating *all* continuations of a non-deterministic choice.

For the formalization of the involved concepts in [32], it was necessary to consider an unrolling of TS^* to a *tree*. This way, the restriction to acyclic services (where the resulting tree is finite) comes into play.

As an acyclic net may not have livelocks (beyond deadlocks), the result covers LF, k-controllability as well. For extending the results to QL, k-controllability, we need to consider all possibilities for non-deterministic choices.

As an example, we study net (a) in Fig.4. Part (b) of this figure depicts the single-port strategy as computed in the previous section. In this strategy, actions a and b belong to different ports and should thus be independent. In the strategy (b), however, they disable each other. Dependency can be resolved by either removing the a-edge or the b-edge, leading to the strategies (c) and (d). The example shows that nondeterminism is almost unavoidable for multi-port control since otherwise, there would be no way to break the symmetry from the symmetric strategy (b) to the asymmetric strategy (c) (or the asymmetric strategy (d)).

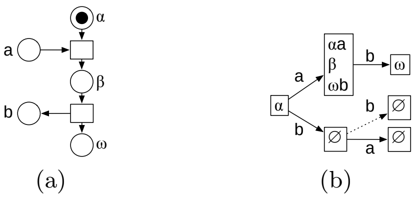


Fig. 5. An open net (a) with a single-port strategy (b) where all actions are independent

Figure 5 shows another example. The example illustrates that it is important to treat the node $\emptyset$ in TS_0 rather carefully. In fact, Lemma 1 states that such a state is unreachable in the composed system. Nevertheless, its presence (repeated due to the transformation into a tree) is important for reasoning about dependency or independence of activities. In fact, activities a and b are independent, so the single-port strategy (b) of net (a) can be transformed into a multi-port strategy where one partner executes a and the other one b. In the resulting composed systems, reachable states correspond to the reachable states (nonempty nodes) of strategy (b).

The results of this section summarize results in [32].

4.2 Autonomous Control

The concept of autonomous control is concerned with the absence of built-time coordination. The idea of autonomous control is to define general constraints on the behavior of each partner such that, if every partner obeys the constraints (regardless how), proper behavior (here: DF) of the overall system is guaranteed. The constraints are such that they can be verified by only considering the given multi-port service N and a particular partner. Autonomous controllability is then defined as the satisfiability of the given constraints at every port of N.

The constraints are summarized in the notion of *cooperative partners.* As the conditions are quite technical, we rephrase them in parenthesis as "rules of engagement" for the considered partner N' of N.

Definition 9 (Cooperative partner). *Let N be a multi-port service and N' a single-port service that is composable to N. Let $P \in \mathcal{P}_N$ be the unique port of N where $P \cap S_{N'} \neq \emptyset$. Let both services be normal, bounded, and responsive. For a set X of places, let $\underline{0}_X$ be the marking with 0 tokens on all places in X. N' is DF, k-*cooperative *w.r.t. N iff the following conditions hold for every $m \in R_{Inner(N \oplus N')}(m_0)$:*

(1) For all $s \in P \cup I_{N'}$, $m(s) \leq k$ (do not violate the message bound of k);
(2) If $m \notin M_{FInner(N \oplus N')}$ and $m \mid_{Inner(N)} \in M_{FInner(N)}$ then there is a $t \in T_N \cup T_{N'}$ where $m \oplus \underline{0}_{I_{N \oplus N'}} \xrightarrow{t}_{N \oplus N'}$ (If N is in a final state and no transition is executable, you must be in a final state, too, and no messages may be pending on your channels));
(3) If $m \mid_{Inner(N)} \notin M_{FInner(N)}$ then there is a $t \in T_N \cup T_{N'}$ where $m \oplus \underline{0}_{I_{N \oplus N'}} \xrightarrow{t}_{N \oplus N'}$, or, for all $t' \in T_N$, $m \mid_{Inner(N)} \xrightarrow{t'}_{Inner(N)}$ implies $l(t) \notin P$. (If N is blocking then you must be able to execute a transition unless all transitions out of the blocking marking of N are under the control of other ports than yours).

Observe that this definition only involves N and N' and is completely independent of other partners connected to N. Nevertheless, the following theorem holds.

Theorem 2. *If $|\mathcal{P}_N| = k$, $N_1, \ldots, N_k$ are composable to N and DF, k-cooperative w.r.t. N such that $N \oplus N_1 \oplus \cdots \oplus N_k$ is well-defined (i.e., all N_i use different ports of N) then $N \oplus N_1 \oplus \cdots \oplus N_k$ is deadlock-free and cannot reach a state with more than k tokens on an interface place of any involved net.*

Proof. It is easy to see that reachability in $Inner(N \oplus N_i)$ overapproximates reachability in $N \oplus N_1 \cdots \oplus N_k$. Thus, k-boundedness of interface places follows immediately from Item 1 of Def. 9.

Let $m \in R_{N \oplus N_1 \oplus \cdots \oplus N_k}$ be a deadlock. Thus, $m \notin M_{FN \oplus N_1 \oplus \cdots \oplus N_k}$. Consider first the case that $m \mid_{Inner(N)} \notin M_{FInner(N)}$. As N is assumed to be responsive, there is a $t \in T_N$ where $m \mid_{Inner(N)} \xrightarrow{t}_{Inner(N)}$. Since m is a deadlock in $N \oplus N_1 \oplus \cdots \oplus N_k$, there is an $s \in S_{iN}$ such that $m(s) = 0$ and $[s, t] \in F_N$. Let N_i be the partner that is connected to the port of N containing s. Then, Item 2 of Def. 9 asserts existence of a transition t' that can fire in $m_{N \oplus N_i} \oplus \underline{0}$ which implies fireability of t' in m in $N \oplus N_1 \oplus \cdots \oplus N_k$. Thus, m would not be a deadlock. Consider now the case that $m \mid_{Inner(N)} \in M_{FInner(N)}$. Then there must be an i where $m|_{Inner(N \oplus N_i)} \notin M_{FInner(N \oplus N_i)}$. Now, Item 3 of Def. 9 asserts existence of a transition t' that can fire in $m_{N \oplus N_i} \oplus \underline{0}$ which implies fireability of t' in m in $N \oplus N_1 \oplus \cdots \oplus N_k$. Again, m would not be a deadlock. q.e.d.

Consider open net (b) in Fig. 1. This net makes an internal decision and then opens two parallel threads. In each thread, one partner sends a message to the net after which the respective other partner gets a message from which it can deduce the outcome of the internal decision. Finally, depending on the initial decision, one of the partner must send an additional message.

Partner (a) is cooperative while partner (c) is not. The most important situation in this regard is the marking reached after firing one transition in (b). In this marking, a cooperative partner *must* send a message. If both partners are cooperative (behave like net (a)), the composed system is deadlock-free. If one partner is cooperative (like in the depicted situation), the system may or may not work correctly (in this case, it does). If, however, both partners behave like (c), the marking mentioned above is a deadlock in the composed system.

Net (a) in Fig. 4 does not have cooperative partners which accords to the intuition about this service. Thus, the following definition is useful.

Definition 10 (Autonomous controllability). *A multi-port service N is* autonomously controllable *iff, for every port of N, there is a cooperative partner.*

The conditions for cooperative behavior are such that a cooperative partner can be computed using basically the same idea as in Sect. 3. The result is (in the case of an autonomously controllable service) a most permissive cooperative partner. For single-port nets, single-port controllability, multi-port controllability, and autonomous controllability coincide.

The result in this subsection is a generalization of results in [32]. There, we restricted services to acyclic ones.

5 Controllability under Additional Constraints

In this section, we study the following problem: Is there an X, k-strategy N' for a given service N such that N' satisfies some additional specification? Additional specifications include, but are not restricted to the enforcement or exclusion of exchanged messages. Enforcing a message a means that every run (execution sequence) in the composed system contains a state where interface place a is marked. Excluding a means that no run of the composed system contains a state where a is marked.

There are several motivations for such additional specification. First, they allow the owner of N to validate certain features of N ("can the service be used with payment by credit card", "can the service terminate without using some abortion feature", etc.). Second, a specification can express a user's choice of particular features to be used or not to be used. In this case, controllability does not only decide whether these choices are consistent with N. The actual value of this approach is that the computed strategy can in fact be used as code to be run by the partner. In a third scenario, a service N can be projected to some subbehavior for fitting into the requirements of some service repository. For instance, if a service offers various goods, including books and toys, restricted versions of the same service could be registered with service repositories specialized to toy vendors, as well as specialized service repositories that collect book sellers.

Many additional specifications can be dealt with through product constructions involving N. We synchronize additional structures with N such that every strategy of the modified net is a strategy of N which satisfies the additional specification. This way, the new problem is reduced to standard controllability questions.

Let N^* be a closed net which is disjoint to N in all its constituents. Assume further that there is a labeling of transitions $\lambda : T_N \cup T_{N^*} \rightarrow L \cup \{\tau\}$ into some set L. Then the synchronous product $N \otimes N^*$ of N and N^* can be defined as follows. We take the disjoint union of places of N and N^*. As transitions, we use the set $\{[t, t^*] \mid t \in T_N, t^* \in T_{N^*}, \lambda(t) = \lambda(t^*) \neq \tau\} \cup \{t \mid t \in T_N \cup T_{N^*}, \lambda(t) = \tau\}$. Arcs and initial marking are added canonically, the interface is the one of N. Final markings are all those markings where both the projection to N and the projection to N^* is a final marking of the respective net.

With the construction, we aim at not altering the firing behavior of N. This can be achieved by requiring the following property for N^*.

Definition 11 (Monitor net). *Closed net N^* is a* monitor net *for N w.r.t. labeling λ iff, for all markings m reachable in $N \otimes N^*$, and every $l \in L$, if some transition t with label l is activated in the projection of m to N, then some sequence of τ-labeled transitions can be fired in the projection of m to N^* such that a transition t^* with label l in N^* becomes enabled.*

Due to the defined monitor property, every sequence of N can be transformed into a sequence of $N \otimes N^*$ which establishes the same behavior in the N-part of $N \otimes N^*$. The only difference is that reaching a final state in N does not necessarily correspond to reaching a final state in $N \otimes N^*$. This way, only a subset of successful runs of N corresponds to successful runs in $N \otimes N^*$.

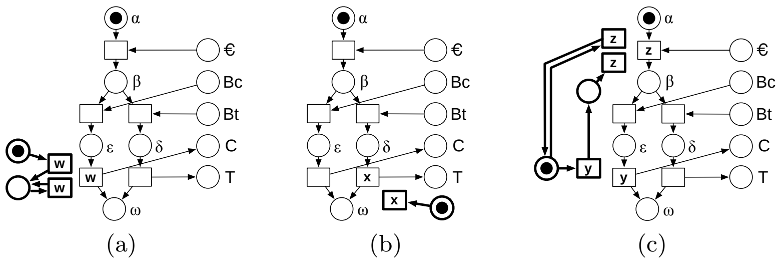


Fig. 6. A vending machine with various constraints attached

In Fig. 6, we added constraints to an open net that models a vending machine where, after inserting a coin and pressing a button, coffee (C) or tea (T) is released. The parts painted with bold lines are the respective monitor nets. Overlapping transitions are meant to carry the same label. If, in the constraint net in part (a), the marking with a token on the bottom place is final, the constraint models the property “enforce coffee”. A strategy of the composed net will inevitably use the service such that message C is emitted by the machine. Part (b), with the initial marking as final marking of the monitor net, models the “exclude coffee” property. A net where message C can be sent by the machine cannot be a strategy of the net. Part (c) shows a more complex monitor net. With the empty marking as final marking, it enforces a behavior where, after having received coffee, a coin must be inserted. As this constraint is not satisfiable in this particular vending machine, the composed net becomes uncontrollable.

In this section, results of [22] have been extended with the explicit definition of the monitor property.

6 Controllability for Alternative Communication Models

In this section, we study the impact of the communication model on the previous results. So far, we assumed asynchronous message passing and did not take into consideration any semantic dependency between message contents. For both assumptions, there may be cases where they are inappropriate.

6.1 Synchronous Communication

It is commonly agreed that services, in particular web services, communicate asynchronously. For service *models*, however, the shape of communication may depend on the chosen level of abstraction. It is thus desirable to have a formalism where the mode of communication (synchronous versus asynchronous) can be chosen individually for each channel.

The model of open nets can be extended with synchronous communication channels. One particular option is to represent each such channel by a label. The labels are attached to transitions in the considered open nets N and N'. If a transition carries a label other than τ, it can only fire if, at the same time, a transition with same label is fired in the respective other net. If a transition can carry more than one label, we fire a step U of transitions in $T_N \cup T_N'$ such that the sum of label occurrences in $U \cap T_N$ is equal to the sum of label occurrences in $U \cap T_{N'}$, for each individual label. Using this idea, composition of N and N' can be executed in a similar fashion as in Sect. 5.

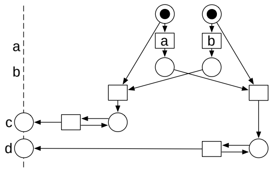


Fig. 7. An open net with synchronous channels a, b, and asynchronous channels c, d

In Sect. 2, we mentioned that every open net can be transformed into an equivalent normal open net (where every transition is connected to at most one interface place). Unfortunately, a corresponding result for synchronous channels does not hold. Consider Fig. 7 with a final marking where the post-places of the transitions labeled a and b are marked. If this service is in the initial marking, reachability of a malfunctioning state can be avoided only if the partner fires a transition which carries both a and b as label, i.e., takes care that the two transitions carrying these labels are fired simultaneously.

However, every open net with both asynchronous and synchronous channels can be transformed into one where every transition which is linked to an asynchronous channel does not carry labels for synchronous channels. Thus, a similar construction as in Def. 6 can be applied where, in addition to transitions for asynchronous messages, transitions are created which correspond to steps of synchronous channels. Then, the constructions of Defs. 7 and 8 can be applied as before.

The result in this subsection has been reported in [38]. The same study extends the notion of independent transitions to synchronous channels and thus generalizes multi-port controllability to open nets with synchronous channels.

Another approach to synchronous communication can be found in [16]. There, N and N' are connected via signal arcs between transitions. The source transition of a signal arc fires according to the usual transition rule. The sink transition, however, fires only if a step consisting of both source and sink transition is enabled. This means that a sent message is lost if the receiving net is not ready to process it. This asymmetric means of synchronization has interesting applications in technical systems where signal arcs represent pulsed electronic signals. In the area of web services, we have not yet identified a scenario where pulsed signals are an appropriate way of modeling communication.

6.2 Semantic Dependencies between Messages

So far, we treated exchanged messages as being completely independent of each other. As a consequence, every message can be created and sent at any point in time. In reality, however, certain dependencies between messages exist. For instance, it does not always make sense to send a message "filled form" before having received a message "empty form". Likewise, "credit approved" or "credit rejected" should never be sent before having received those data on which this decision depends.

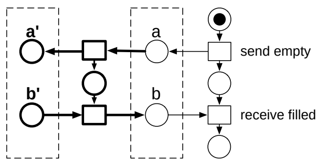


Fig. 8. Modeling a semantic dependency between messages: If N' is a strategy of N (depicted in normal ink) plus K (depicted in bold) then N' plus K is a strategy of N that does not send a filled form before having received the empty one

We found that many relevant semantic dependencies between messages can be implemented as a Petri net fragment K which are placed between the net N to be controlled, and the actual strategy N'. In Fig. 8, unit K corresponds to "b cannot be sent before having received a", i.e. a could model an empty form an b the filled version of that form.

In the example, it is easy to see that (plain) controllability of $N \oplus K$ implies that N is controllable such that the specified restrictions are met. $N' \oplus K$ is a strategy that witnesses this property. The other way round, a strategy N' of N that obeys the specified restrictions can be easily transformed into a strategy for $N \oplus K$.

The treatment of semantic dependencies has been investigated in [17].

7 Related Work

The synthesis of controllers in general has been extensively studied on the basis of the work by Wonham and Ramadge [30]. Our setting is, however, situated not in the center of their focus, so we could not find directly applicable solutions fitting to our settings.

In the area of workflows, the verification of soundness [36] is certainly a similar approach that aims at providing a necessary criterion for correct behavior. Our focus on deadlock freedom, livelock freedom, and quasi-liveness is directly inspired by the notion of soundness of workflow nets. There is a version of soundness for workflows, called *relaxed soundness* [9], that involves controller synthesis. In this case, however, the controller is sitting inside the workflow and may observe parts of the internal state (white box) while our setting sees the controller as sitting outside the service (black box).

Results on constraints are closely related to product constructions in classical automata theory. The study of autonomous controllability is related to the problem of *protocol separation* for which we are only aware of some singular ad-hoc solutions.

Controllability is closely linked to the concept of *realizability* of service choreographies. Here, the question is whether there are services which interact according to a pattern of interaction.

8 Conclusions and Future Work

The behavior of services can be modeled using open nets. This class of models is bidirectionally linked to the industrially relevant language WS-BPEL. Controllability for services is a criterion which is similar to the notion of soundness for workflow nets. We decide controllability by the construction of (the transition system of) a corresponding strategy. This strategy is quite useful as it can be transformed into code that represents the communication with the given service. With the proposed techniques, verification of controllability is feasible for realistic services.

Controllability can be studied in various settings. Among the parameters defining a setting are the behavioral property to be established in the composition (deadlock freedom, livelock freedom, quasi-liveness, or others), the shape of the interface (single-port, multi-port with or without build-time coordination between the partners), the nature of the message passing model (synchronous vs. asynchronous, with or without considering semantic dependencies), and the presence of additional constraints on the set of strategies (enforce or exclude activities).

We showed that several settings can be dealt with either by adapting the original procedure or by modifying the given open net. However, the space of studied settings is far from being exhaustive. Furthermore, some settings required restrictions like acyclicity of the given service. Thus, more research efforts are required for an exhaustive coverage of all relevant settings. A useful side effect of our approach is the synthesis of an actual strategy. There has already been a significant amount of work that uses the synthesized strategy for a characterization of *all* strategies. There seem to be a number of other useful applications of this synthesis, e.g. in adapter generation. Other potential applications for a synthesized partner need to be explored. Furthermore, we should investigate decision procedures for controllability which do not require the synthesis of a strategy. In this point, there are only few results linking the structure of an open net and its controllability.

References

1. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services: Concepts, Architectures and Applications. Springer, Heidelberg (2003)
2. Alves, A., et al.: Web Services Business Process Execution Language Version 2.0. OASIS Standard (April 2007)
3. Badouel, E., Darondeau, P.: Theory of Regions. In: Reisig, W., Rozenberg, G. (eds.) APN 1998. LNCS, vol. 1491, Springer, Heidelberg (1998)
4. Benatallah, B., Casati, F., Grigori, D., Motahari Nezhad, H.R., Toumani, F.: Developing Adapters for Web Services Integration. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 415–429. Springer, Heidelberg (2005)
5. Bracciali, A., Brogi, A., Canal, C.: A formal approach to component adaptation. J. Systems and Software 74(1), 45–54 (2005)
6. Brogi, A., Canal, C., Pimentel, E., Vallecillo, A.: Formalizing Web Service Choreographies. ENTCS 105, 73–94 (2004)
7. Bryant, R.: Graph-based algorithms for Boolean function manipulation. IEEE Trans. on Computers C-35(8), 677–691 (1986)
8. Clarke, G., Grumberg, O., Peled, D.: Model Checking. MIT Press, Cambridge (1999)
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. Desel, J., Reisig, W.: The synthesis problem of Petri nets. Acta Informatica 33, 297–315 (1996)
11. 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)
12. Ehrenfeucht, A., Rozenberg, G.: Partial 2-structures. Acta Informatica 27, 315–368 (1990)
13. Fahland, D., Reisig, W.: ASM-based semantics for BPEL: The negative Control Flow. In: Proc. ASM, pp. 131–151 (2005)
14. Gottschalk, K.: Web Services Architecture Overview. IBM Whitepaper, IBM DeveloperWorks (September 2000), `http://ibm.com/developerWorks/web/library/w-ovr`
15. Hull, R., Benedikt, M., Christophides, V., Su, J.: E-services: a look behind the curtain. In: Proc. PODS, pp. 1–14. ACM, New York (2003)
16. Juhas, G., Lorenz, R., Neumair, C.: Modelling and Control with Modules of Signal Nets. In: Desel, J., Reisig, W., Rozenberg, G. (eds.) Lectures on Concurrency and Petri Nets. LNCS, vol. 3098, pp. 585–625. Springer, Heidelberg (2004)
17. Kerlin, A.: Bedienbarkeit unter Kausalität. Diploma thesis, Humboldt-Universität zu Berlin (2007)

18. Kindler, E.: A compositional partial order semantics for Petri net components. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 235–252. Springer, Heidelberg (1997)
19. Kindler, E., Martens, A., Reisig, W.: Inter-operability of Workshop Applications – Local Criteria for Global Soundness. In: van der Aalst, W.M.P., Desel, J., Oberweis, A. (eds.) Business Process Management. LNCS, vol. 1806, pp. 235–253. Springer, Heidelberg (2000)
20. Lohmann, N., Kleine, J.: Fully-automatic Translation of Open Workflow Net Models into Human-readable Abstract BPEL Processes. In: Proc. Modellierung, vol. LNI P-127, pp. 57–72 (2008)
21. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing interacting WS-BPEL processes using flexible model generation. Data Knowl. Eng. 64(1), 38–54 (2008)
22. 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)
23. 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)
24. Lohmann, N., Verbeek, H.M.W., Ouyang, C., Stahl, C., van der Aalst, W.M.P.: Comparing and Evaluating Petri Net Semantics for BPEL. Computer Science Report 07/23, Eindhoven University of Technology (2007)
25. Martens, A.: Verteilte Geschäftsprozesse – Modellierung und Verifikation mit Hilfe von Web Services. Dissertation, Humboldt-Universität zu Berlin (2003)
26. Martens, A.: Analyzing Web Service based Business Processes. In: Cerioli, M. (ed.) FASE 2005. LNCS, vol. 3442, pp. 19–33. Springer, Heidelberg (2005)
27. Massuthe, P., Reisig, W., Schmidt, K.: An Operating Guideline Approach to the SOA. Annals of Mathematics, Computing & Teleinformatics 1(3), 35–43 (2005)
28. Massuthe, P., Serebrenik, A., Sidorova, N., Wolf, K.: Can I find a partner? (accepted for IPL)
29. Papazoglou, M.P.: Agent-oriented technology in support of e-business. Commun. ACM 44(4), 71–77 (2001)
30. Ramadge, P.J., Wonham, W.M.: Supervisory control of a class of discrete -event processes. SIAM J. Control and Optimization 25(1) (1987)
31. Reisig, W., Schmidt, K., Stahl, C.: Kommunizierende Workflow-Services modellieren und analysieren. Informatik - Forschung und Entwicklung, 90–101 (2005)
32. Schmidt, K.: Controllability of Open Workflow Nets. In: Enterprise Modelling and Information Systems Architectures, vol. LNI P-75, pp. 236–249 (2005)
33. Stahl, C.: A Petri Net Semantics for BPEL. Technical Report 188, Humboldt-Universität zu Berlin (2005)
34. Valmari, A.: A stubborn attack to state explosion. In: Clarke, E., Kurshan, R.P. (eds.) CAV 1990. LNCS, vol. 531, pp. 156–165. Springer, Heidelberg (1991)
35. van der Aalst, W., Lohmannn, N., Massuthe, P., Stahl, C., Wolf, K.: From public views to private views – correctness-by-design for services. In: Dumas, M., Heckel, R. (eds.) WS-FM 2007. LNCS, vol. 4937, pp. 139–153. Springer, Heidelberg (2008)
36. van der Aalst, W.M.P.: The application of petri nets in workflow management. J. Circuits, Systems, and Computers 8(1), 21–66 (1998)
37. Weinberg, D.: Analyse der Bedienbarkeit. Diplomarbeit, Humboldt-Universität zu Berlin (2004)
38. Wolf, M.: Synchrone und asynchrone Kommunikation in offenen Workflownetzen. Studienarbeit, Humboldt-Universität zu Berlin (2007)

Deciding Substitutability of Services with Operating Guidelines

Christian Stahl[1,2,⋆], Peter Massuthe[1,2], and Jan Bretschneider[1]

[1] Humboldt-Universität zu Berlin, Institut für Informatik,
Unter den Linden 6, 10099 Berlin, Germany
{stahl,massuthe,bretschn}@informatik.hu-berlin.de
[2] Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Abstract. Deciding whether a service S can be substituted by another service S' is an important problem in practice and one of the research challenges in service-oriented computing. In this paper, we define three substitutability notions for services. *Accordance* specifies that S' cooperates with at least the environments that S cooperates with. S and S' are *equivalent* if they cooperate with the same environments. To guarantee that S' cooperates with a fixed subset of environments that S cooperates with, the notion of *restriction* can be used. For each substitutability notion we present a decision algorithm. To this end we apply the concept of an *operating guideline* of a service as an abstract representation of all environments the service cooperates with.

Keywords: Open nets, Operating guidelines, Service substitutability.

1 Introduction

In the paradigm of service-oriented computing (SOC) [1], a service serves as a building block for designing flexible business processes by composing multiple services. Such a (composed) service is subject to changes. There may hardly ever be a total renewal or upgrade of the overall service. Instead, individual services will be replaced by better ones, because the service was too expensive or some new functionality has been added, for instance. *Service substitutability*, that is, deciding whether a service can be substituted by another service, is one of the most notable SOC research challenges.

Obviously, a service S can be substituted by another service S' if no environment can distinguish them, that is, they are equivalent. In practice, however, more flexible notions than equivalence are relevant as well. In general, substituting S by S' either should *gain* or *preserve* properties of the overall service.

In order to guarantee that substituting S by S' indeed gains and/or preserves specific properties, support of formal methods is needed. To this end we need

[⋆] Funded by the DFG project "Substitutability of Services" (RE 834/16-1).

to characterize different properties of substitutability, resulting in different substitutability notions. In the next step, we have to develop algorithms to decide substitutability for each notion.

In this paper, we restrict ourselves to the service protocol, that is, to the *behavior* of a service and abstract from other important aspects like quality of service and semantics. As our formal model we use *open nets*, a special class of Petri nets. An open net has an interface for communication with other open nets via asynchronous message passing. To meet different application scenarios that are relevant in practice we introduce three *substitutability notions*: accordance (S' cooperates with at least every environment S cooperates with), equivalence (S and S' cooperate with the same environments), and restriction (S' cooperates with at least a fixed subset of environments S cooperates with). Furthermore, a constraint-conforming substitutability notion is derived which is more fine-grained than restriction. For each such notion we present a decision algorithm based on the concept of an *operating guideline* as an abstract representation of all environments a given service can cooperate with. Operating guidelines have been suggested to support service discovery so far. In this paper, we show that operating guidelines are well-suited for deciding substitutability of services, too. To this end we use known results, extend some notions, and also provide new results on operating guidelines. An extended version of this article has been published as a technical report [2].

The remainder of this paper is structured as follows. Sections 2 and 3 present the preliminaries. There, we recall our service models, open nets and service automata, as well as operating guidelines. Then, in Sect. 4 we introduce the notion of accordance. Restriction is explained in Sect. 5. From accordance and restriction we derive in Sect. 6 two further substitutability notions. Related work is discussed in Sect. 7 and finally, conclusions are drawn in Sect. 8.

2 Service Models

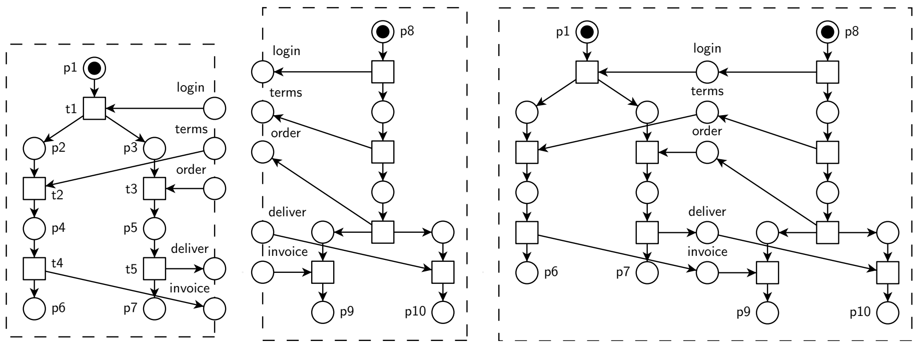
In this section, we introduce *open nets*, a special class of Petri nets, as a formal model for services and *service automata* as a technique to analyze the interaction behavior of open nets.

2.1 Open Nets

We assume the usual definition of a (place/transition) *Petri net* $N = [P, T, F]$ (see [3], for instance) and use the standard notation to denote the *preset* and *postset* of a place or a transition: ${}^\bullet x = \{y \mid (y, x) \in F\}$ and $x^\bullet = \{y \mid (x, y) \in F\}$.

A *marking* of a Petri net N is a mapping $m : P \to \mathbb{N}$. We use a *multiset* notation to denote markings and write $m = [p_1, p_1, p_2]$ for a marking m with $m(p_1) = 2$, $m(p_2) = 1$, and $m(p) = 0$ for all $p \in P \setminus \{p_1, p_2\}$. If $Q \supseteq P$, a marking $m : P \to \mathbb{N}$ extends canonically to $m : Q \to \mathbb{N}$ by $m(p) = 0$ for each $p \in Q \setminus P$.

Open nets (introduced in [4] using the term "open workflow nets") are a special class of Petri nets. An open net has an *interface* that consists of a set



(a) Open net N_{shop} (b) Open net N_{client} (c) Composition $N_{\text{shop}} \oplus N_{\text{client}}$

Fig. 1. (a) An open net N_{shop} modeling an online shop. In the initial marking [p1], it waits for a login from a client. After the client logged in, the shop concurrently waits for an order which it then will deliver and it waits for a confirmation of the terms of payment and sends an invoice afterwards. Finally, the shop reaches the single final marking [p6, p7]. (b)-(c) An open net N_{client} modeling a client of the shop with its final marking [p9, p10] and the composition of shop and client.

of *input places* and a set of *output places* for asynchronous communication with an environment. This idea is based on the module concept for Petri nets which was proposed by Kindler [5]. Suitability of open nets for modeling services has been proven through an implemented translation (see [4], for instance) from the industrial service description language WS-BPEL [6] into open nets.

As a global name space, we assume a set $\mathcal{MC}$ of *message channels* given. For technical reasons, we require that the special symbols τ (representing a non-communicating step) and *final* (used to denote final states) are not in $\mathcal{MC}$.

Definition 1 (Open net). *An* open net $N = [P, P_{in}, P_{out}, T, F, m_0, \Omega]$ *consists of a Petri net* $[P, T, F]$ *together with*

- *two disjoint sets* $P_{in} \subseteq (P \cap \mathcal{MC})$ *of* input places *such that* ${}^\bullet p_{in} = \emptyset$ *for all* $p_{in} \in P_{in}$ *and* $P_{out} \subseteq (P \cap \mathcal{MC})$ *of* output places *such that* $p_{out}{}^\bullet = \emptyset$ *for all* $p_{out} \in P_{out}$,
- *a distinguished* initial marking m_0, *and*
- *a set* Ω *of* final markings *s.t. no transition of* N *is enabled at any* $m \in \Omega$.

Let $P_{io} = P_{in} \cup P_{out}$ *denote the* interface *of* N. *We further require that neither the initial nor a final marking marks any interface place* $p \in P_{io}$.

The behavior of an open net (i.e. enabledness and firing of transitions) is defined using the standard (place/transition) Petri net semantics (see [3], for instance). In order to assign an intuitively consistent meaning to *final* markings, we restrict our approach to open nets where a marking in Ω does not enable any transition. As an example, Fig. 1(a) shows an open net model of an online shop.

Interaction of open nets is represented by their *composition*. Two open nets N_1 and N_2 are *composable* if they only share interface places and the input places of N_1 are exactly the output places of N_2 and vice versa (i.e. $P_{in1} = P_{out2}$ and $P_{in2} = P_{out1}$). For two markings m_1 of N_1 and m_2 of N_2, their *composition* $m_1 \oplus m_2$ is defined by $(m_1 \oplus m_2)(p) = m_1(p) + m_2(p)$. From now on, if two open nets N_1 and N_2 are composed, we implicitly assume they are composable.

Definition 2 (Composition of open nets). *The* composition *of two (composable) open nets N_1 and N_2 is the open net $N = N_1 \oplus N_2$ defined as follows: $P = P_1 \cup P_2$, $P_{in} = \emptyset$, $P_{out} = \emptyset$, $T = T_1 \cup T_2$, $F = F_1 \cup F_2$, $m_0 = m_{0_1} \oplus m_{0_2}$, and $\Omega = \{m_1 \oplus m_2 \mid m_1 \in \Omega_1, m_2 \in \Omega_2\}$.*

A marking m of an open net N is a *deadlock* in N iff m is no final marking of N and m does not enable any transition of N. This definition of a deadlock differs from the standard definition in literature as we discriminate between terminating (final) states and non-terminating states (i.e. deadlocks). Deadlock-freedom is a fundamental correctness criterion for cooperating services. In contrast, an open net representing a service in isolation usually has deadlocks. As an example, each of the open nets in Fig. 1(a) and Fig. 1(b) on its own has a deadlock, whereas the open net in Fig. 1(c) is deadlock-free.

Definition 3 (Strategy). *An open net M is a (open net)* strategy *for an open net N if their composition is deadlock-free. $Strat(N)$ denotes the set of all strategies for N.*

If $Strat(N) \neq \emptyset$, then N is called *controllable*, otherwise N is *uncontrollable*. Uncontrollable services are fundamentally ill-designed. Note that according to Definition 3, the strategy notion is symmetric, that is, M is a strategy for N iff N is a strategy for M. In Sect. 3 we will show how to decide controllability of a given service N by synthesizing a strategy M, thus fixing one side of this symmetry. If N is uncontrollable, then the synthesis produces an "empty strategy".

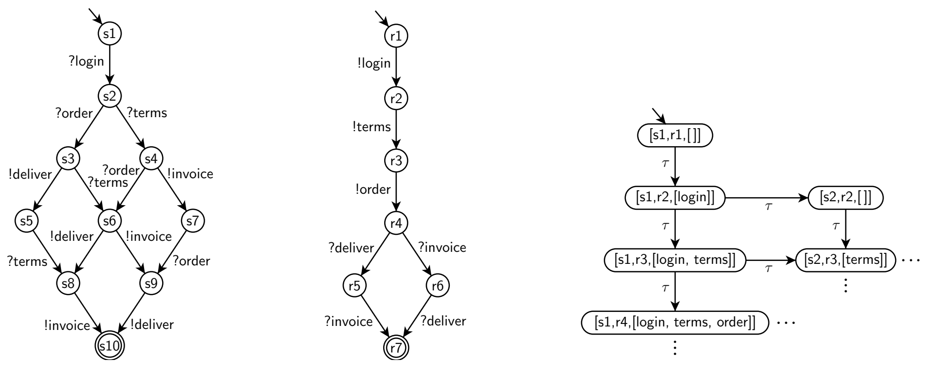
Obviously, the client N_{client} in Fig. 1(b) is a strategy for the shop N_{shop} in Fig. 1(a) (and vice versa). Hence, N_{shop} is controllable (and so is N_{client}).

The set $Strat(N)$ is of particular importance as it gives a semantics of an open net N in terms of N's deadlock-free interacting environments. In Sections 4 − 6, we introduce several substitutability notions which all are based on comparing the corresponding sets of strategies.

2.2 Service Automata

Service automata [7] form the basis of operating guidelines and are used for representing the behavior of open nets. We will firstly introduce service automata and then present a back and forth translation between open nets and service automata that uses the set $\mathcal{MC}$ as interconnection. A service automaton is closely related to the reachability graph of the inner of an open net, that is, the *inner* of an open net N is the open net $inner(N)$ where all interface places of N as well as all their adjacent arcs are removed.

Definition 4 (Service automaton). *A* service automaton *is an automaton $A = [Q, I_{in}, I_{out}, \delta, q_0, \Omega]$ that consists of*



(a) A_{shop} (b) A_{client} (c) Composition $A_{\text{shop}} \oplus A_{\text{client}}$

Fig. 2. Three service automata of the online shop, its client, and their composition

- *a set Q of* states*,*
- *two disjoint sets $I_{in} \subseteq \mathcal{MC}$ of* input channels *and $I_{out} \subseteq \mathcal{MC}$ of* output channels*, with $I_{io} = I_{in} \cup I_{out}$ is the* interface *of A,*
- *a nondeterministic* transition relation *$\delta \subseteq Q \times (I_{io} \cup \{\tau\}) \times Q$,*
- *a distinguished* initial state *$q_0 \in Q$, and*
- *a set $\Omega \subseteq Q$ of* final states*, s.t. $q \in \Omega$ and $(q, x, q') \in \delta$ implies $x \in I_{in}$.*

For a transition $(q, x, q') \in \delta$, x is called the *label* of (q, x, q'). An x-labeled transition is a *sending transition* if $x \in I_{out}$, a *receiving transition* if $x \in I_{in}$, and an *internal transition* if $x = \tau$. To emphasize the direction of an interface channel $x \in I_{io}$ in the graphical representation of a service automaton, we add an exclamation mark, $!x$, if $x \in I_{out}$, or a question mark, $?x$, if $x \in I_{in}$.

Figure 2 shows three service automata which correspond to the three open nets of Fig. 1.

In the following, we lift notions defined for open nets to service automata.

Two service automata are *composable* if they have disjoint sets of states and the input channels of one automaton are the output channels of the other automaton and vice versa. From now on, we assume all composed service automata are composable. The *composition* $A \oplus B$ of composable service automata A and B introduces an *internal message bag* (i.e. a multiset) of currently pending messages that were sent by one automaton, but not yet received by the other one. That way, a prior x-labeled sending transition of A is represented in $A \oplus B$ by an internal (i.e. τ-labeled) transition that adds one x element to the message bag M. Correspondingly, a prior transition receiving an x is represented by a now internal transition removing an x from the message bag. Prior internal transitions remain as internal transitions in $A \oplus B$. This is formalized in the following definition. Therefore, let $\mathit{bags}(\mathcal{MC})$ denote the set of all multisets over $\mathcal{MC}$.

Definition 5 (Composition of service automata). *For two (composable) service automata A and B, their* composition *is the service automaton*

$A \oplus B = [Q, I_{in}, I_{out}, \delta, q_0, \Omega]$ *defined as follows:* $Q = Q_A \times Q_B \times bags(\mathcal{MC})$, $I_{in} = I_{out} = \emptyset$, $\delta \subseteq Q \times \{\tau\} \times Q$, $q_0 = [q_{0_A}, q_{0_B}, \emptyset]$, $\Omega = \Omega_A \times \Omega_B \times \{\emptyset\}$, *such that the transition relation* δ *contains the elements*

- $([q_A, q_B, M], \tau, [q'_A, q_B, M])$ *iff* $(q_A, \tau, q'_A) \in \delta_A$,
- $([q_A, q_B, M], \tau, [q_A, q'_B, M])$ *iff* $(q_B, \tau, q'_B) \in \delta_B$,
- $([q_A, q_B, M], \tau, [q'_A, q_B, M - [x]])$ *iff* $(q_A, x, q'_A) \in \delta_A$, $x \in I_{in\,A}$, $M(x) > 0$,
- $([q_A, q_B, M], \tau, [q'_A, q_B, M + [x]])$ *iff* $(q_A, x, q'_A) \in \delta_A$, $x \in I_{out\,A}$,
- $([q_A, q_B, M], \tau, [q_A, q'_B, M - [x]])$ *iff* $(q_B, x, q'_B) \in \delta_B$, $x \in I_{in\,B}$, $M(x) > 0$,
- $([q_A, q_B, M], \tau, [q_A, q'_B, M + [x]])$ *iff* $(q_B, x, q'_B) \in \delta_B$, $x \in I_{out\,B}$.

In the rest of this paper, we will only consider the connected part of the service automaton $A \oplus B$ which contains the initial state.

A state q is a *deadlock* in A if $q \notin \Omega$ and at most receiving transitions leave q. Hence, a service automaton cannot leave a deadlock by its own.

Definition 6 (Strategy). *A service automaton* A *is a* strategy *(service automaton) for a service automaton* B *if their composition is free of deadlocks.*

In analogy to open nets, let $Strat(A)$ denote the set of all strategies for a service automaton A. A is *controllable* iff $Strat(A) \neq \emptyset$.

2.3 Translating Open Nets into Service Automata and Back

In [4] we have shown that it is possible to translate each open net N into a service automaton A_N which is basically the reachability graph of $inner(N)$. The set $\mathcal{MC}$ is used as a common name space of N and A_N, as both the interface places of N and the interface of A_N are subsets of $\mathcal{MC}$. For example, the service automata of Fig. 2 correspond to the open nets of Fig. 1 under this translation.

Additionally, it is easily possible to translate a service automaton A into an open net N_A by constructing a Petri net state machine out of A or by applying the theory of regions [8], for instance.

Open nets are well-suited to *model* services as they have a more implicit and compact model and are thus more understandable for service designers than service automata. Service automata, in contrast, adequately model service behavior and thus are well-suited to *analyze* services. The value of the back and forth translation is that we can change arbitrarily between these two formalisms without loosing information w.r.t. *Strat*. Hence, it is sufficient to develop our analysis techniques for service automata, but to model on the Petri net level.

3 Operating Guidelines

Operating guidelines were first introduced in [7] and generalized in [4]. Basically, an *operating guideline* OG_A of a service automaton A is a special service automaton B where each state q of B is annotated with a Boolean formula $\phi(q)$.

Such a *Boolean annotated service automaton* (*BSA*) B^ϕ can be used to characterize a set of service automata [7,4]. Therefore, we define a matching relation between a service automaton C and a *BSA* B^ϕ. B^ϕ characterizes C iff C *matches* with B^ϕ. An *operating guideline* OG_A of a service automaton A is a special *BSA* where C matches with OG_A iff C is a strategy for A.

A *literal* of our Boolean formulae is a channel in $\mathcal{MC}$ or one of the special literals τ and *final* (representing an internal transition and a final state, respectively). Let, for the rest of this paper, $\mathcal{MC}^+$ denote the set $\mathcal{MC} \cup \{\mathit{final}, \tau\}$. As Boolean connectors, we only use $\vee$ (Boolean *or*) and $\wedge$ (Boolean *and*). Let $\mathcal{BF}$ be the set of all such Boolean formulae over $\mathcal{MC}^+$. As usual, we fix the truth values *true* and *false*. A Boolean *assignment* is a mapping $\beta : \mathcal{MC}^+ \rightarrow \{\mathit{true}, \mathit{false}\}$ assigning to each literal a truth value. Furthermore, an assignment β *satisfies* a formula $\phi \in \mathcal{BF}$, $\beta \models \phi$, if ϕ evaluates to *true* using standard propositional logic semantics.

The restriction of *BSA*s to deterministic structures and negation-free formulae eases the decision procedures in the upcoming sections while providing all sufficient information needed for operating guidelines.

Definition 7 (Boolean annotated service automaton, *BSA*). *A* Boolean annotated service automaton *(BSA)* $B^\phi = [B, \phi]$ *consists of a* deterministic *service automaton* $B = [Q, I_{in}, I_{out}, \delta, q_0, \Omega]$ *and a Boolean* annotation *function* $\phi : Q \rightarrow \mathcal{BF}$. *Thereby, a service automaton is deterministic if it has no internal transitions and each state has at most one x-labeled outgoing transition.*

For matching a service automaton C with a *BSA* B^ϕ, the present outgoing transitions of a state q of C constitute an assignment for $\phi(q)$:

Definition 8 (Assignment). *An* assignment *of a service automaton C assigns to each state q of C a Boolean assignment $\beta_C(q) : \mathcal{MC}^+ \rightarrow \{\mathit{true}, \mathit{false}\}$ defined by*

$$\beta_C(q)(x) = \begin{cases} \mathit{true}, & \text{if } x \neq \mathit{final} \text{ and there is a state } q' \text{ with } (q, x, q') \in \delta_C, \\ \mathit{true}, & \text{if } x = \mathit{final} \text{ and } q \in \Omega_C, \\ \mathit{false}, & \text{otherwise.} \end{cases}$$

A *BSA* is used to characterize a *set* of service automata. Let therefore be the matching of a service automaton with a *BSA* defined as follows:

Definition 9 (Matching). *Let C be a service automaton and B^ϕ be a BSA. C* matches *with B^ϕ if there is a weak simulation relation $\varrho \subseteq Q_C \times Q_B$ such that for each $(q_C, q_B) \in \varrho$: $\beta_C(q_C) \models \phi(q_B)$. Let $\mathit{Match}(B^\phi)$ denote the set of all service automata that match with B^ϕ.*

The weak simulation relation [9] ϱ in Definition 9 together with possible τ literals in ϕ allow the deterministic B^ϕ for characterizing deterministic as well as nondeterministic service automata [4]. Figure 3(a) shows an example *BSA*. Figures 3(b)−3(d) demonstrate the matching.

Definition 10 (Operating guideline, *OG*). *A BSA $OG_A = B^\phi$ is an* operating guideline *(OG) of a service automaton A iff $\mathit{Match}(OG_A) = \mathit{Strat}(A)$.*

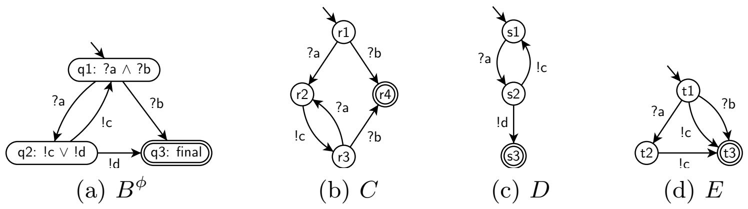


Fig. 3. (a) A *BSA* B^ϕ. The annotation $\phi(q)$ is depicted inside the state q. (b) – (d) Three service automata C, D, and E. C matches with B^ϕ: for instance, the assignment $\beta_C(\mathtt{r1})$ assigns *true* to the literals ?a and ?b (because both transitions leave the state r1), satisfying the annotation ?a $\wedge$?b. However, D and E do not match with B^ϕ: state s1 does not satisfy the annotation of state q1; and the !c-labeled transition leaving state t1 causes B not simulating E.

According to this definition, an operating guideline is a special *BSA*. For uncontrollable service automata A (i.e. $\mathit{Strat}(A) = \emptyset$) we fix an *OG* that consists of a single state that is annotated with *false*, assuring that *no* service automaton matches with this *OG*.

Figure 4 depicts an *OG* of the online shop of Fig. 2(a). Applying Definition 9 we conclude that the client of Fig. 2(b) matches with this *OG*.

In [4] we have presented an algorithm to compute an operating guideline of a service where the inner of the service (cp. Sect. 2.2) has finitely many reachable states. For services without this restriction, we were able to show that controllability is undecidable [10]. The *OG* construction algorithm computes a special strategy. Therefore it starts with an overapproximation of compatible behavior of any strategy and then removes deadlock-causing states iteratively. Finally, the annotations are derived from information collected during the computation. If the service is uncontrollable, the algorithm eventually removes all states. The algorithm is implemented in our tool Fiona [11].[1]

4 Accordance

In this section, we define our first substitutability notion, *accordance*. A service A' accords with a service A if A' cooperates with at least the environments that A cooperates with. In other words, if the composition of A and an environment B is deadlock-free, then deadlock-freedom is preserved if A is substituted by A'.

4.1 A Notion of Accordance

Given a service automaton A, it might be necessary to change or add some functionality of A by substituting it by a new version A'. With accordance, we demand that this substitution must not affect any client of A: every current client of A has to be supported by A' as well. Because we assume that A does

[1] Fiona is available at `http://www.service-technology.org/fiona`

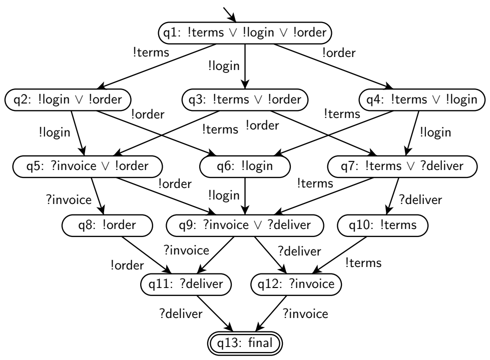


Fig. 4. An operating guideline of the online shop of Fig. 2(a). To characterize also nondeterministic strategies, each Boolean annotation $\phi(q)$ is implicitly extended to $\phi(q) \vee \tau$ and thus evaluated to *true* if the matched service automaton has an outgoing τ-transition in the corresponding state (cp. Definition 8).

not know each client that uses A, A' must support each *potential* client of A, i.e. all elements in $\mathit{Strat}(A)$. An application for accordance is the upgrade of a web shop which should not affect any client. This motivates the following notion of accordance between service automata A and A'. To this end A and A' must be *interface equivalent* (i.e. $I_{in\,A} = I_{in\,A'}$ and $I_{out\,A} = I_{out\,A'}$).

Definition 11 (Accordance). *Let A and A' be interface equivalent service automata. A'* substitutes A under accordance *(short: A' accords with A) iff $\mathit{Strat}(A) \subseteq \mathit{Strat}(A')$.*

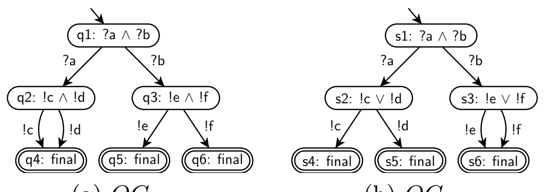
Accordance guarantees that every strategy for A is a strategy for A' as well. In other words, if A' accords with A, then every client of A is also a client of A'. In addition, accordance allows for new clients of A'. Thus, accordance seems to be the right notion to achieve the goal mentioned above.

The notion of accordance has been first introduced in [12]. However, the decision procedure for accordance was limited to acyclic finite state services there. In this paper, we extend this procedure to cyclic finite state services.

4.2 Deciding Accordance

In order to decide accordance of A and A', we need to compare $\mathit{Strat}(A)$ and $\mathit{Strat}(A')$. The problem is that the set Strat may correspond to a large (in fact infinite) set of service automata. With the operating guidelines of A and A' we have, however, a finite representation of $\mathit{Strat}(A)$ and $\mathit{Strat}(A')$. In the following, we show how accordance can be decided by using operating guidelines. To this end we define a relation $\sqsubseteq$ for operating guidelines. Informally, $OG_A \sqsubseteq OG_{A'}$ iff there is a simulation relation between the states of OG_A and $OG_{A'}$ such that the annotations in OG_A imply the annotations in $OG_{A'}$.

Definition 12 (Relation $\sqsubseteq$ of OGs). *Let A and A' be interface equivalent service automata and let $OG_A = [Q, I_{in}, I_{out}, \delta, q_0, \Omega, \phi]$ and $OG_{A'} = [Q', I'_{in}, I'_{out},$*



(a) OG_{small} (b) OG_{big}

Fig. 5. Two operating guidelines with $OG_{\text{small}} \sqsubseteq OG_{\text{big}}$. For instance, $(\mathsf{q2}, \mathsf{s2}) \in \xi$ with $\phi(\mathsf{q2}) \Rightarrow \phi(\mathsf{s2})$, and $(\mathsf{q4}, \mathsf{s4}) \in \xi$ and $(\mathsf{q4}, \mathsf{s5}) \in \xi$ with $\phi(\mathsf{q4}) \Rightarrow \phi(\mathsf{s4})$ and $\phi(\mathsf{q4}) \Rightarrow \phi(\mathsf{s5})$.

$\delta', q'_0, \Omega', \phi'$] be the corresponding operating guidelines. Then, $OG_A \sqsubseteq OG_{A'}$ iff there is a simulation relation $\xi \subseteq Q \times Q'$ such that for all $(q, q') \in \xi$, the formula $\phi(q) \Rightarrow \phi'(q')$ is a tautology.

The relation $\sqsubseteq$ is a preorder, that is, it is reflexive and transitive. By help of the next theorem we show that $OG_A \sqsubseteq OG_{A'}$ iff A' accords with A and thus it can be used to decide accordance of A and A'. An example is depicted in Fig. 5.

Theorem 1 (Checking accordance). *Let A and A' be two service automata and let OG_A and $OG_{A'}$ be the corresponding operating guidelines. Then, $OG_A \sqsubseteq OG_{A'}$ iff $Strat(A) \subseteq Strat(A')$.*

For the proof of this theorem we rely on a fact about operating guidelines as constructed in [4]. As we cannot repeat the whole approach of [4], we only include the following proposition and prove Theorem 1.

Proposition 1. *For every operating guideline $OG_A = [Q, I_{in}, I_{out}, \delta, q_0, \Omega, \phi]$ (of some controllable service automaton A) and all $q \in Q$, the formula $\phi(q)$*

(1) uses only literals x where there is some $q' \in Q$ with $(q, x, q') \in \delta$, and

(2) is satisfied for the assignment assigning true to all literals in $\phi(q)$.

Proof (of Theorem 1). Let $OG_A = [Q, I_{in}, I_{out}, \delta, q_0, \Omega, \phi]$ and $OG_{A'} = [Q', I'_{in}, I'_{out}, \delta', q'_0, \Omega', \phi']$ be the OGs of service automata A and A', respectively.

($\Rightarrow$): Let $OG_A \sqsubseteq OG_{A'}$ and let B be an arbitrary strategy service automaton for A. We show that B is a strategy for A', too.

Because of $B \in Strat(A)$ there is by Definition 9 a simulation relation $\varrho \subseteq Q_B \times Q$ between the states of B and OG_A and, because of $OG_A \sqsubseteq OG_{A'}$ there is by Definition 12 a simulation relation $\xi \subseteq Q \times Q'$ between the states of OG_A and $OG_{A'}$. Define a relation $\varrho' \subseteq Q_B \times Q'$ between the states of B and $OG_{A'}$ such that $(q_B, q') \in \varrho'$ iff there is a state q of OG_A such that $(q_B, q) \in \varrho$ and $(q, q') \in \xi$. We show that ϱ' is a simulation relation.

Let $q_B \in Q_B$ and suppose there is a transition $(q_B, x, q'_B) \in \delta_B$. From ϱ being a simulation relation follows that there is a $q \in Q$ with $(q_B, q) \in \varrho$, $(q, x, q_1) \in \delta$, and $(q'_B, q_1) \in \varrho$. By ξ being a simulation relation there is a q' such that $(q, q') \in \xi$, $(q', x, q'_1) \in \delta'$, and $(q_1, q'_1) \in \xi$. According to the definition of ϱ' we have (q_B, q') and (q'_B, q'_1) in ϱ', and hence we conclude that ϱ' is a simulation relation between the states of B and $OG_{A'}$.

Next we show for all $(q_B, q') \in \varrho'$ that q_B satisfies $\phi'(q')$. As B matches with OG_A, for all states q_B with $(q_B, q) \in \varrho$, q_B satisfies $\phi(q)$ (for the assignment described in Definition 9). By $OG_A \sqsubseteq OG_{A'}$ we know $\phi(q) \Rightarrow \phi'(q')$ for all $(q, q') \in \xi$. Hence, q_B satisfies $\phi(q')$ for all $(q_B, q') \in \varrho'$, too.

Thus, B is a strategy for A' and, hence, we conclude $\mathit{Strat}(A) \subseteq \mathit{Strat}(A')$.

($\Leftarrow$): Let $\mathit{Strat}(A) \subseteq \mathit{Strat}(A')$. We show that $OG_A \sqsubseteq OG_{A'}$.

Consider the underlying service automaton $B = [Q, I_{in}, I_{out}, \delta, q_0, \Omega]$ of OG_A. By construction, the transition systems of B and OG_A are isomorphic and hence there is a weak simulation relation ϱ between the states of B and OG_A. From Proposition 1 we conclude that B satisfies the annotations of OG_A. Hence, B is a strategy for A and thus, by $\mathit{Strat}(A) \subseteq \mathit{Strat}(A')$, a strategy for A'. Being a strategy for A', there is a simulation relation $\varrho' \subseteq Q_B \times Q'$ between the states of B and $OG_{A'}$. As B and OG_A are isomorphic, ϱ' constitutes a simulation relation $\xi \subseteq Q \times Q'$ between the states of OG_A and $OG_{A'}$, too.

Next we show for all $(q, q') \in \xi$, $\phi(q) \Rightarrow \phi'(q')$. Let $q \in Q$ and let β be an arbitrary assignment to literals occurring in $\phi(q)$ where $\phi(q)$ is *true*. Remove from B all transitions (q_1, x, q_2) where $\beta(x)$ is *false*. By Definition 9, the resulting service automaton is still a strategy for A and thus a strategy for A'. Using Definition 9 again, we can see that β satisfies $\phi'(q')$ as well. Thus, $\phi(q) \Rightarrow \phi'(q')$ is a tautology for all $(q, q') \in \xi$. Hence we conclude $OG_A \sqsubseteq OG_{A'}$. □

The value of this theorem is that accordance can be checked independently of the environments that A cooperates with and only A and A' have to be known to decide accordance. In order to design a service automaton A' that accords with A, a designer can either try and check the resulting service or he derives A' from A by applying accordance-preserving transformation rules [12].

For an implementation of the criteria in Theorem 1, finding the relation ξ is the crucial task. As both OG_A and $OG_{A'}$ are deterministic, this task actually amounts to a depth-first search through OG_A which is mimicked in $OG_{A'}$. The time and space required for finding ξ is thus linear in the number of states and edges of OG_A. This size, in turn, is equal to the number of states and edges of a particular strategy for A. The accordance check based on Theorem 1 has been implemented in our tool Fiona.

5 Restriction

In this section, we introduce another substitutability notion, *restriction*. Restriction is — as accordance — used to compare the sets of environments of two service automata A and A'. The goal of restriction is to preserve at least a *fixed subset* of the environments of A by A' (instead of *all* environments of A as in the accordance setting).

5.1 A Notion of Restriction

Given a service automaton A, we may want to preserve at least a fixed subset $\mathcal{S} \subseteq \mathit{Strat}(A)$ of its strategies when substituting A by a service automaton A'.

This means, every service automaton $S \in \mathcal{S}$ is a strategy for both A and A'. In contrast to the notion of accordance, here we assume that A *has* knowledge of its environments. To motivate the need of such a substitutability notion, consider again an upgrade of a web shop. Applications for restriction include: the upgraded shop only supports behavior which is used by major clients and all other clients have to adjust their services; the shop restricts itself to its core competencies and rejects all unprofitable strategies; the shop restricts its behavior to certain scenarios such as payment by credit card, for instance. These considerations lead to the following definition of restriction.

Definition 13 (Restriction). *Let A and A' be interface equivalent service automata and let $\mathcal{S} = \{S_1, \ldots, S_n\} \subseteq \mathit{Strat}(A)$. Then, A'* substitutes A under restriction to $\mathcal{S}$ *(short: A' preserves $\mathcal{S}$) iff $\mathcal{S} \subseteq \mathit{Strat}(A')$.*

According to this definition, at least every service automaton in $\mathcal{S}$ is a strategy for A', meaning, the substitution preserves at least strategies $\mathcal{S}$. Hence, restriction seems to be the right notion to achieve the above mentioned goal.

5.2 Deciding Restriction

The aim of this section is to introduce a decision procedure whether substituting a service automaton A by a service automaton A' preserves a set $\mathcal{S} \subseteq \mathit{Strat}(A)$ of strategies. Therefore, we have to check that every service automaton $S \in \mathcal{S}$ is a strategy for A'. This decision procedure becomes particularly complex if the set $\mathcal{S}$ contains many service automata and we want to check several A'. Therefore, we consider the following alternative: since the notion of a strategy is symmetric, it is equivalent to check whether A' is a strategy for all $S \in \mathcal{S}$. In other words, $A' \in \bigcap_{S \in \mathcal{S}} \mathit{Strat}(S)$ must hold.

We will show that the intersection $\bigcap_{S \in \mathcal{S}} \mathit{Strat}(S)$ of sets of strategies can be represented by the *product of the operating guidelines* of all service automata $S \in \mathcal{S}$. We start by defining the product $OG_A \otimes OG_B$ of two operating guidelines OG_A and OG_B of service automata A and B as an operating guideline which characterizes exactly the intersection $\mathit{Strat}(A) \cap \mathit{Strat}(B)$. To this end OG_A and OG_B must be interface equivalent, that is, their underlying automata must be interface equivalent.

Definition 14 (Product of OGs). *Let $OG_A = C_1^{\phi_1}$ and $OG_B = C_2^{\phi_2}$ be two (interface equivalent) operating guidelines with $C_i = [Q_i, I_{in\,i}, I_{out\,i}, \delta_i, q_{0i}, \Omega_i]$, $i \in \{1, 2\}$. Define $\varrho \subseteq Q_1 \times Q_2$ inductively as follows: let $(q_{01}, q_{02}) \in \varrho$. If $(q_1, q_2) \in \varrho$, $(q_1, x, q_1') \in \delta_1$, and $(q_2, x, q_2') \in \delta_2$, then $(q_1', q_2') \in \varrho$.*

Then, the product *$OG_A \otimes OG_B = [Q, I_{in}, I_{out}, \delta, q_0, \Omega, \phi]$ of OG_A and OG_B is defined by*

- $Q = \{(q_1, q_2) \mid (q_1, q_2) \in \varrho\}$,
- $I_{in} = I_{in\,1} = I_{in\,2}$, *and* $I_{out} = I_{out\,1} = I_{out\,2}$,
- $((q_1, q_2), x, (q_1', q_2')) \in \delta$ *iff* $(q_1, x, q_1') \in \delta_1$ *and* $(q_2, x, q_2') \in \delta_2$,
- $q_0 = (q_{01}, q_{02})$,

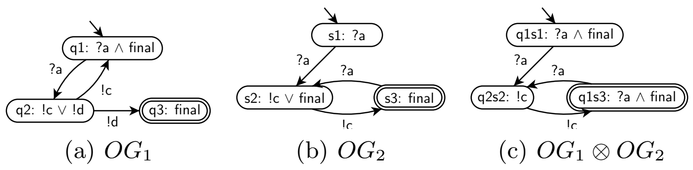


Fig. 6. Two operating guidelines and their product

- $\Omega = \{(q_1, q_2) \in Q \mid q_1 \in \Omega_1, q_2 \in \Omega_2\}$, *and*
- $\phi((q_1, q_2)) = \phi_1(q_1) \wedge \phi_2(q_2)$, *for all* $(q_1, q_2) \in Q$.

In a way, the product of operating guidelines is defined analogously to the product of finite automata [13]. Figure 6 shows two interface equivalent operating guidelines and their product (with $\varrho = \{(\mathsf{q1}, \mathsf{s1}), (\mathsf{q2}, \mathsf{s2}), (\mathsf{q1}, \mathsf{s3})\}$). We assume !d is an output channel of OG_2, even though there is no !d-labeled transition in OG_2. Note that the annotations of $OG_1 \otimes OG_2$ are equivalently minimized: the annotation of state $\phi(\mathsf{q1s1}) = (\mathsf{?a} \wedge \mathsf{final}) \wedge \mathsf{?a}$ is equivalent to $\mathsf{?a} \wedge \mathsf{final}$. Furthermore, the annotation $\phi(\mathsf{q2s2}) = (\mathsf{!c} \vee \mathsf{!d}) \wedge (\mathsf{!c} \vee \mathsf{final})$ can be minimized to $(\mathsf{!c}) \wedge (\mathsf{!c} \vee \mathsf{final})$ (which is equivalent to !c) because there is no outgoing !d-labeled transition at state q2s2.

Next, we prove that the product of two operating guidelines characterizes indeed the intersection of the strategies represented by these operating guidelines.

Theorem 2 (Product OG characterizes intersection). *Let* OG_A *and* OG_B *be operating guidelines and* $OG_\otimes = OG_A \otimes OG_B$ *be their product. Then,* $Match(OG_\otimes) = Match(OG_A) \cap Match(OG_B)$.

Proof. Let $OG_A = [Q_A, I_{in\,A}, I_{out\,A}, \delta_A, q_{0A}, \Omega_A, \phi_A]$, $OG_B = [Q_B, I_{in\,B}, I_{out\,B}, \delta_B, q_{0B}, \Omega_B, \phi_B]$ and $OG_\otimes = OG_A \otimes OG_B = [Q, I_{in}, I_{out}, \delta, (q_{0A}, q_{0B}), \Omega, \phi]$.

($\Rightarrow$) Let $C \in Match(OG_\otimes)$ and let $\varrho_\otimes$ be a simulation relation between C and $OG_\otimes$. We will show that $C \in Match(OG_A)$ and $C \in Match(OG_B)$, too.

Let $(q_C, (q_A, q_B)) \in \varrho_\otimes$ be arbitrary. As $\varrho_\otimes$ is simulation relation there is a sequence σ such that q_C is reached from q_{0C} via σ in C and (q_A, q_B) is reached from (q_{0A}, q_{0B}) via σ in $OG_\otimes$. By construction of $OG_\otimes$, q_{0A} and q_{0B} are reached via σ in OG_A and OG_B, too. By Definition 9 again, we have $(q_C, q_A) \in \varrho_A$ and $(q_C, q_B) \in \varrho_B$. Let there be an x-transition leaving q_C. From $C \in Match(OG_\otimes)$ and Definition 9 (i.e. the weak simulation relation), we can conclude that there is an x-transition leaving (q_A, q_B), too. By the construction of δ in Definition 14, there is an x-transition leaving q_A and one leaving q_B. Hence, q_A of OG_A and q_B of OG_B simulate q_C, too.

Furthermore, we conclude from $C \in Match(OG_\otimes)$ and Definition 9 that the assignment $\beta_C(q_C)$ satisfies $\phi((q_A, q_B))$. Hence, by the construction of ϕ in Definition 14, $\beta_C(q_C)$ also satisfies $\phi_A(q_A)$ and $\phi_B(q_B)$. Consequently, C matches with OG_A and OG_B and therefore $C \in Match(OG_A) \cap Match(OG_B)$.

($\Leftarrow$) Let $C \in Match(OG_A)$ and $C \in Match(OG_B)$. We show that $C \in Match(OG_\otimes)$.

By $C \in \mathit{Match}(OG_A)$ and Definition 9 there is a simulation relation ϱ_A between C and OG_A. Let $(q_C, q_A) \in \varrho_A$ be arbitrary. As ϱ_A is simulation relation there is a sequence σ such that q_C is reached from q_{0_C} via σ in C and q_A is reached via σ in OG_A. By $C \in \mathit{Match}(OG_B)$ and Definition 9, there is a simulation relation ϱ_B and a state q_B such that $(q_C, q_B) \in \varrho_B$ and q_B is reached via σ in OG_B. By the construction of δ in Definition 14, (q_A, q_B) is reachable in $OG_\otimes$ via σ, too. The rest of the proof follows the same argumentation as above. □

The product $\otimes$ of operating guidelines is commutative and associative, that is, for operating guidelines OG_A, OG_B, OG_C holds $OG_A \otimes OG_B = OG_B \otimes OG_A$ and $(OG_A \otimes OG_B) \otimes OG_C = OG_A \otimes (OG_B \otimes OG_C)$. So we conclude that the product operating guideline represents exactly the intersection of all strategy sets for service automata in $\mathcal{S}$:

Corollary 1. *Let $\mathcal{S} = \{S_1, \dots, S_n\}$ be a set of interface equivalent service automata and let OG_{S_i} be the operating guideline of S_i, for all $1 \leq i \leq n$. Let $OG_\otimes$ denote the product of all OG_{S_i}. Then, $\mathit{Match}(OG_\otimes) = \bigcap_{S \in \mathcal{S}} \mathit{Strat}(S)$.*

With the help of the above corollary we can prove a theorem which shows that substituting A by A' preserves $\mathcal{S}$ iff A' is a strategy represented by the product operating guideline $OG_\otimes$.

Theorem 3 (Restriction check with product OGs). *Let A and A' be service automata and let $\mathcal{S} = \{S_1, \dots, S_n\} \subseteq \mathit{Strat}(A)$. Let OG_{S_i}, $1 \leq i \leq n$, be the operating guideline of S_i and let $OG_\otimes$ denote the product of all OG_{S_i}. Then, A' preserves $\mathcal{S}$ iff $A' \in \mathit{Match}(OG_\otimes)$.*

Proof. We will show that $\mathit{Match}(OG_\otimes)$ characterizes all service automata A' that can substitute A while preserving $\mathcal{S}$.

We have: $\mathit{Match}(OG_\otimes) = \bigcap_{S \in \mathcal{S}} \mathit{Strat}(S)$ (by Corollary 1) $= \{A' \mid$ for all $S \in \mathcal{S} : A' \in \mathit{Strat}(S)\} = \{A' \mid$ for all $S \in \mathcal{S} : S \in \mathit{Strat}(A')\}$ (because strategy is symmetric) $= \{A' \mid A'$ preserves $\mathcal{S}\}$ (by Definition 13). □

In order to decide whether substituting A by A' preserves $\mathcal{S} \subseteq \mathit{Strat}(A)$, we have to construct the operating guideline for each $S \in \mathcal{S}$ and then calculate the product of these operating guidelines. Time and space complexity for calculating the product of two operating guidelines is proportional to the product of their states. Therefore, this complexity effort only pays off if we check several A'. The restriction check based on Theorem 3 has been implemented in our tool Fiona.

Intuitively, the fewer strategies shall be preserved by the substitution (i.e. the smaller $\mathcal{S}$ is), the more service automata A' exist that may substitute A (i.e. the bigger is $\mathit{Match}(OG_\otimes)$). Because accordance requires all strategies for A to be preserved by A', but restriction requires only a subset of A's strategies to be preserved by A', there are less services A' that accord with A than services A' that satisfy restriction. For $\mathcal{S} = \mathit{Strat}(A)$, restriction coincides with accordance.

As an advantage, the notion of restriction provides with $OG_\otimes$ an abstract representation of all substituting service automata A'. So, with the help of $OG_\otimes$, a service automaton A' can be *derived* from $OG_\otimes$, such that A' substitutes A

while preserving $\mathcal{S}$. For instance, the underlying service automaton of $OG_{\otimes}$ itself serves as a valid A'. Theorem 3 states that every element in $\mathit{Match}(OG_{\otimes})$ can be used to substitute A. Together with the transformation of a service automaton A' into a corresponding open net $N_{A'}$, we are immediately able to derive substituting open nets as well.

In case of accordance, in contrast, we can only *check* an already given A' whether it accords to A or not. To support the construction of according services, we developed accordance-preserving transformation rules to derive from A a service automaton A' that accords with A by construction [12].

6 Derived Substitutability Notions

In this section, we introduce two more substitutability notions. Both notions can be derived from the notions of accordance and restriction.

6.1 Equivalence

The first substitutability notion we derive is a notion of *equivalence* for service automata. This can be achieved easily by restricting the notion of accordance. Two service automata are equivalent iff they have the same set of strategies.

Definition 15 (Equivalence). *Let A and A' be interface equivalent service automata. Then, A'* equivalently substitutes *A (short: A' and A are equivalent) iff $\mathit{Strat}(A) = \mathit{Strat}(A')$.*

Obviously, in order to check equivalence of two service automata, we can check equivalence of their respective operating guidelines. Since equivalence means accordance in both directions, we apply Theorem 1 in both directions.

Corollary 2 (Checking equivalence with OGs). *Two operating guidelines OG_A and OG_B are* equivalent *iff $OG_A \sqsubseteq OG_B$ and $OG_A \sqsupseteq OG_B$.*

6.2 Constraints

For many substitutability scenarios the three notions of substitutability we have introduced so far are well-suited. However, there are other scenarios in practice that require less restrictive notions. Accordance demands to preserve all strategies for a given service, even those which are practically infeasible: consider that a service A has to interact with *two* other services, B and C. Assume that A sends a request to either service B or C and concurrently expects an acknowledgement from the respective service. There is a strategy S for A such that S receives the request which A has sent to B and acknowledges on behalf of C. This is, in fact, a valid strategy, but practically impossible if B and C do not communicate with each other. This problem arises in the decentralized setting [14]. Such strategies need not to be preserved when substituting A by A'.

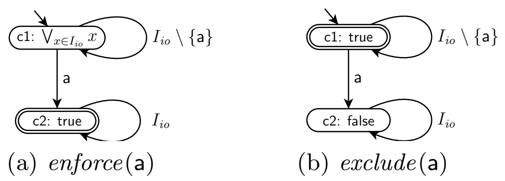


Fig. 7. Generic constraint automata to enforce or exclude a communication action a

As another example, if we want to restrict the set of strategies to profitable strategies or to enforce or exclude certain scenarios (e.g. payment by credit card), then restriction is too inflexible, because we would have to identify all infeasible strategies.

These examples motivate the introduction of a notion of a *constraint.* Such a constraint can been seen as a behavioral pattern or communication scenario. We will show how to restrict a set of strategies to those strategies that *enforce* or *exclude* certain behavioral patterns. In [15] such constraints have been introduced to characterize all strategies for a service that conform to a constraint. This approach is used to filter *service registries* for services that fit respective strategies and for validating services by checking whether there exist strategies that access certain features. In contrast to [15], we are interested in services that preserve all strategies that conform to a constraint.

In the following, we define the notion of a constraint *BSA* C^ψ. Intuitively, C^ψ is a *BSA* that constrains send and receive actions of an operating guideline OG_A. Here, to constrain means to enforce or to exclude the respective actions of OG_A.

Definition 16 (Constraint *BSA*). *Let A and C be two interface equivalent service automata. Let OG_A be an operating guideline of A and let ψ be an annotation to C. Then, C^ψ is a* constraint *BSA for OG_A.*

Intuitively, OG_A represents the set of strategies for A and the constraint *BSA* C^ψ describes the behavior we want to allow or disallow in the restricted subset of strategies. Therefore, their product characterizes all strategies for A that conform to C^ψ. Figure 7 depicts generic constraint automata for enforcing or excluding a communication action a.

Given a product $OG_A \otimes C^\psi$, each service automaton A' where $OG_{A'}$ characterizes exactly these strategies is a well-suited candidate for substituting A. This yields a more fine-grained notion of substitutability under restriction which is covered by the following definition.

Definition 17 (Constraint-conforming substitution). *Let A, A' be service automata and OG_A, $OG_{A'}$ be the corresponding operating guidelines. Let C^ψ be a constraint BSA for OG_A. Then,* the substitution of A by A' conforms to C^ψ *iff $Match(OG_{A'}) = Match(OG_A \otimes C^\psi)$.*

In analogy to our accordance check based on Theorem 1, this definition provides a *verification* method only. That is, a *given* A' can be checked whether it is a valid constraint-conforming substitution of A or not. The notion of constraints and the substitutability check based on Definition 17 has been implemented in our tool Fiona.

7 Related Work

Compatibility (e.g. [16]) is the ability of two services N and M to interact properly. The literature distinguishes between *structural* and *behavioral* compatibility. Structural compatibility demands that the interfaces of both services match whereas behavioral compatibility demands the control flow of the composition $N \oplus M$ to fulfill some property such as deadlock-freedom. Thus, the notion of a strategy we used in this paper is a synonym for compatibility: M and N are (structural and behavioral) compatible if and only if they are strategies.

A service implementation N' is *consistent* (e.g. [16]) with a service specification N if every compatible service M for N is compatible to N'. Thereby, consistency is a synonym for *conformance*, a notion used in process theory (e.g. [17,18,19]). Conformance is a refinement relation between services (i.e. a preorder). In this paper, we have introduced two consistency notions: accordance which guarantees (the compatibility notion) deadlock-freedom and restriction for which we have extended compatibility to support specific behavior besides deadlock-freedom. Remember that restriction required the preservation of a certain set of strategies.

Substitutability is a collection of consistency notions (together with their corresponding compatibility notions) and each consistency notion corresponds to a specific substitutability scenario. The problem of deciding substitutability is strongly related to the *compositionality problem* (e.g. [20]) which refers to constructing a system from components while preserving certain properties of the whole system such as the absence of deadlocks and livelocks.

Various substitutability notions can be found in literature. However, most of them lack of an asynchronous communication model as it is necessary in the context of SOC or efficient decision algorithms; or they are too restrictive.

Vogler presents in [20] a livelock and deadlock preserving equivalence between Petri nets with interfaces. However, there is no direct implication in either direction between the equivalence of Vogler and accordance.

For workflow nets (WFNs) [21] the notion of *inheritance* [22] is used two relate two WFNs that can be substituted. Inheritance bases on branching bisimulation. As a difference, the inheritance approach assumes a synchronous communication model (i.e. transition fusion). In [12] accordance has been proven to be strictly coarser than projection inheritance.

Similarly, accordance is also strictly coarser than *saturated bisimulation* as introduced in [23], because the latter does not allow to reorder sending messages.

In [24,25,26], automata models are used to decide substitutability. All these approaches use only synchronous communication whereas we consider asynchronous message passing. Benatallah et al. [26] present four notions of substitutability. In this paper, we cover all of them.

Bravetti and Zavattaro [17] present a conformance relation which is stricter than accordance because besides deadlocks it also excludes livelocks and infinite runs. As the main difference, they present only a sufficient criterion to decide conformance whereas our algorithm to decide accordance is sufficient and necessary and it is implemented in our analysis tool Fiona.

Castagna et al. [18] introduce a conformance notion that formalizes the absence of deadlocks and livelocks in finite-state systems. In contrast to accordance and other conformance notions, their conformance notion only demands the termination of the environment but not the termination of the process itself.

In [2] we have proven that accordance is coarser than stuck-free conformance in [19]. Stuck-freedom formalizes like accordance the absence of deadlocks in the system. As two uncontrollable processes are not necessarily structurally related, accordance seems to be different from most preorders in literature. Thus, there seems to be no obvious relationship between accordance and a known preorder. A detailed comparison of accordance and other preorders is, however, outside the scope of this paper.

Pathak et al. [27] focus on a substitutability notion that preserves certain properties of a service S to be substituted. The properties are expressed by a μ-calculus formula ϕ. Then, a μ-calculus formula ψ is calculated such that all services S' that satisfy ψ can substitute S. Due to the expressiveness of the μ-calculus, this approach generalizes our constraint-conforming substitution, but it assumes a synchronous communication model.

Finally, the idea of using annotated automata as a representation of a set of automata has been first published in [28].

8 Conclusion

We have investigated the problem whether a service S can be substituted by another service S'. Based on our formal models of open nets and service automata, we have defined different substitutability notions for services: accordance, restriction (in two variants), and equivalence. That way we can formally support various substitutability scenarios which may occur in practice.

As our substitutability notions compare the infinite sets of all deadlock-free interacting services for S and S', the presented decision algorithms apply the concept of an operating guideline as a finite representation of these sets of services. That way we can decide accordance and equivalence for S and S'. In addition, we defined the notion of a product operating guideline to specify the intersection of the services represented by several operating guidelines. Product operating guidelines are well-suited to characterize all deadlock-free interacting services for a fixed set of services and can therefore be used for deciding restriction and its more fine-grained variant of constraint-conforming substitutability.

We implemented all decision algorithms presented in this paper in our analysis tool Fiona. The main functionality of Fiona is to calculate an operating guideline of a service modeled as an open net. With the help of the compiler BPEL2oWFN [11] we can translate WS-BPEL processes into our formal model open nets. Using Fiona we can decide on the open net model whether these WS-BPEL processes can be substituted according to one of the presented substitutability notions. That way we can apply our results to practical applications.

In ongoing research, we will work on other termination criteria than deadlock-freedom. This includes the absence of livelocks and the absence of infinite runs. We will also continue working out a precise relationship of our accordance notion with preorders known from process algebra.

References

1. Papazoglou, M.P.: Web Services: Principles and Technology. Pearson - Prentice Hall, Essex (2007)
2. Stahl, C., Massuthe, P., Bretschneider, J.: Deciding Substitutability of Services with Operating Guidelines. Informatik-Berichte 222, Humboldt-Universität zu Berlin (2008)
3. Reisig, W.: Petri Nets. In: EATCS Monographs on Theoretical Computer Science edn., Springer, Heidelberg (1985)
4. 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)
5. Kindler, E.: A compositional partial order semantics for Petri net components. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 235–252. Springer, Heidelberg (1997)
6. Alves, A., et al.: Web Services Business Process Execution Language Version 2.0. Committee Specification, OASIS (2007)
7. Massuthe, P., Schmidt, K.: Operating Guidelines - An Automata-Theoretic Foundation for the Service-Oriented Architecture. In: Cai, K.Y., Ohnishi, A., Lau, E.M.F. (eds.) QSIC 2005, pp. 452–457. IEEE Computer Society, Los Alamitos (2005)
8. Badouel, E., Darondeau, P.: Theory of Regions. In: Reisig, W., Rozenberg, G. (eds.) APN 1998. LNCS, vol. 1491, pp. 529–586. Springer, Heidelberg (1998)
9. Milner, R.: Communication and Concurrency. Prentice-Hall, Inc., Englewood Cliffs (1989)
10. Massuthe, P., Serebrenik, A., Sidorova, N., Wolf, K.: Can I find a Partner? Undecidablity of Partner Existence for Open Nets. Inf. Process. Lett. 108(6), 374–378 (2008)
11. 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)
12. van der Aalst, W.M.P., Lohmann, N., Massuthe, P., Stahl, C., Wolf, K.: From Public Views to Private Views – Correctness-by-Design for Services. In: Dumas, M., Heckel, R. (eds.) WS-FM 2007. LNCS, vol. 4937, pp. 139–153. Springer, Heidelberg (2008)
13. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Reading (1979)
14. Schmidt, K.: Controllability of Open Workflow Nets. In: Desel, J., Frank, U. (eds.) EMISA 2005. LNI, vol. P-75, pp. 236–249. Bonner Köllen Verlag (2005)
15. 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)

16. Decker, G., Weske, M.: Behavioral Consistency for B2B Process Integration. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 81–95. Springer, Heidelberg (2007)
17. Bravetti, M., Zavattaro, G.: Contract Based Multi-party Service Composition. In: Arbab, F., Sirjani, M. (eds.) FSEN 2007. LNCS, vol. 4767, pp. 207–222. Springer, Heidelberg (2007)
18. Castagna, G., Gesbert, N., Padovani, L.: A Theory of Contracts for Web Services. SIGPLAN Not. 43(1), 261–272 (2008)
19. Fournet, C., Hoare, C.A.R., Rajamani, S.K., Rehof, J.: Stuck-Free Conformance. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 242–254. Springer, Heidelberg (2004)
20. Vogler, W.: Modular Construction and Partial Order Semantics of Petri Nets. LNCS, vol. 625. Springer, Heidelberg (1992)
21. 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)
22. van der Aalst, W.M.P., Basten, T.: Inheritance of Workflows: An Approach to Tackling Problems Related to Change. Theor. Comput. Sci. 270(1-2), 125–203 (2002)
23. Bonchi, F., Brogi, A., Corfini, S., Gadducci, F.: A Behavioural Congruence for Web Services. In: Arbab, F., Sirjani, M. (eds.) FSEN 2007. LNCS, vol. 4767, pp. 240–256. Springer, Heidelberg (2007)
24. Bordeaux, L., Salaün, G., Berardi, D., Mecella, M.: When are Two Web Services Compatible? In: Shan, M.-C., Dayal, U., Hsu, M. (eds.) TES 2004. LNCS, vol. 3324, pp. 15–28. Springer, Heidelberg (2005)
25. Beyer, D., Chakrabarti, A., Henzinger, T.: Web service interfaces. In: Ellis, A., Hagino, T. (eds.) WWW 2005, pp. 148–159. ACM, New York (2005)
26. Benatallah, B., Casati, F., Toumani, F.: Representing, Analysing and Managing Web Service Protocols. Data Knowl. Eng. 58(3), 327–357 (2006)
27. Pathak, J., Basu, S., Honavar, V.: On Context-Specific Substitutability of Web Services. In: ICWS 2007, pp. 192–199. IEEE Computer Society Press, Los Alamitos (2007)
28. Wombacher, A., Fankhauser, P., Mahleko, B., Neuhold, E.J.: Matchmaking for business processes based on choreographies. Int. J. Web Service Res. 1(4), 14–32 (2004)

A Framework for Linking and Pricing No-Cure-No-Pay Services

K.M. van Hee[1], H.M.W. Verbeek[1], C. Stahl[2,1,⋆], and N. Sidorova[1]

[1] Department of Mathematics and Computer Science,
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{k.m.v.hee,h.m.w.verbeek,n.sidorova}@tue.nl

[2] Humboldt-Universität zu Berlin, Institut für Informatik
Unter den Linden 6, 10099 Berlin, Germany
stahl@informatik.hu-berlin.de

Abstract. In this paper, we present a framework that allows us to orchestrate web services such that the web services involved in this orchestration interact properly. To achieve this, we predefine service interfaces and certain routing constructs.

Furthermore, we define a number of rules to incrementally compute the price of such a properly interacting orchestration (i.e. a web service) from the *price* of its web services. The fact that a web service gets only payed after its service is delivered (no-cure-no-pay) is reflected by considering a probability of success. To determine a safe price that includes the *risk* a web service takes, we consider the variance of costs.

1 Introduction

In the future, the world of web services will most likely behave in a way similar to business units in a real economy. Web services will sell services to their clients, that may be web services themselves. In order to produce these services, they may buy services from other web services. So each web service has two (inter)faces: a *sell-side* and a *buy-side*. In order to deliver one service, a tree of other web services forming a *supply chain* may be needed. Two services in the chain are linked if the buy-side of the one is connected to the sell-side of the other. When designing web services, one should take into account that the execution of a partner web service may be successful or it may fail. Examples of web services that might be used by other web services are public web services providing the stock market prices, weather forecasts, and news items. Other examples of web services that could be used in supply chains are computations, such as actuarial computations and risk analysis computations. Also services providing movies, television, and telephone services could be part of such a web service supply chain.

Web services will be paid for by a system of micro payments [1]. We assume a no-cure-no-pay system, which implies that a service is paid for if and only if

⋆ Funded by the DFG project "Substitutability of Services" (RE 834/16-1).

the service is successfully delivered. There are at least two important questions for such world of services: Firstly, how do we guarantee that two linked services work properly together? Secondly, what should be the price of service in order to be profitable?

For the first question we have to define what it means for two services to work properly together. Informally, this means that a service request is always followed by a response: Either the requested service is delivered, or a message is sent saying that the requested service cannot be delivered. We provide in this paper a framework where this property is guaranteed. In our framework, each web service contains a *sell-side process* that represents the protocol for selling a service. Internally, there is a process for the *orchestration* of the services. A service is built up from tasks that are either elementary or compound. An elementary task is either executed by the web service itself or it is outsourced to another web service. In the latter case the task contains a *buy-side process* that takes care of the purchase of a service. The sell-side of the service and buy-side processes of all the tasks of the orchestration together form the *choreography* of the world of web services. They are designed in such a way that two arbitrary services fit together properly.

To answer the second question, we devote the second part of the paper to the *pricing mechanism* for services. For the price of a service we have to take into consideration that a service may fail. Since we assume a no-cure-no-pay system, a web service may have to pay all or some of its service providers while it fails to deliver its service to its client. This can be due to stochastic phenomena, such as the availability of servers. That way, a web service takes *risks*. Therefore, it is a nontrivial problem to determine the price of a service. We develop a method to determine the expectation μ and the variance σ^2 of the cost of the service. Assuming a normal distribution for service cost C, we may say that the cost of service is less or equal to $\mu + 1.65 \cdot \sigma$ with probability 95%. This assumption may be justified by the fact that a service is built up by several independent services and by the famous central limit theorem of statistics, the convolution of independent identical distributions can be approximated by a normal distribution. We also provide an exact probabilistic bound for the cost of a service.

The paper is structured as follows. Section 2 introduces Petri nets and some probability notions. The framework for linking web services is presented in Sect. 3. Here, web services are considered as components and the choreography and orchestration processes are modeled as Petri nets. Then, in Sect. 4, we present our pricing model for web services. We show how to compute the expected cost and the cost variance of a web service in an inductive way. Related work is presented in Sect. 5 and finally, Sect. 6 concludes the paper.

2 Preliminaries

Petri nets and workflow nets. We assume the usual definition of a (place/transition) Petri net $N = (P, T, F)$ (see [2,3], for instance) and use the standard notation to denote the preset and postset of a place or a transition: ${}^\bullet x = \{y \mid F(y,x) > 0\}$ and $x^\bullet = \{y \mid F(x,y) > 0\}$.

The state of a net is a marking m which is a distribution of (black) *tokens* over the set of places. A transition t is *enabled* if each place p of its preset holds at least $F(p,t)$ tokens. An enabled transition t can *fire* in a marking m by consuming $F(p,t)$ tokens from the every preset place p and producing $F(t,q)$ tokens for every postset place q, yielding a marking m'. The firing of t is denoted by $m \xrightarrow{t} m'$, reachability of a marking m' from a marking m by a firing of a (possibly empty) sequence of transitions is denoted by $m \xrightarrow{*} m'$. The set of all reachable markings is denoted $N[m\rangle$.

A *workflow net* (WF-net) [4] is a Petri net N that is specifically tailored towards modelling workflow processes. A workflow net $N = (P,T,F)$ has exactly one input place i, i.e. ${}^\bullet i = \emptyset$, and one output place o, i.e. $o^\bullet = \emptyset$, and every place and transition belongs to some path from i to o.

An important correctness property of workflows is *soundness* [4], which comprises the requirements that for every transition sequence that fires starting from the initial marking a marking can be reached where only o is marked, and no transition is dead in N. Soundness can be automatically checked by a number of Petri net tools, like the tool Woflan [5,6].

Some notions of probability. A service consists of tasks. Each task t has a random variable X_t with a Bernoulli distribution: $X_t = 1$ indicates success whereas $X_t = 0$ indicates failure. Furthermore, the probability of success for a task t is $\mathbb{P}[X_t = 1] = s_t$.

Each task t (either elementary or compound) has a random variable C_t denoting the cost of the task. In case t is a compound task, C_t includes the costs of the tasks it contains. This random variable depends in general on X_t and has the following characteristics:

- $\mathbb{E}[C_t] = \mu_t$, cost expectation,
- $\sigma^2(C_t) = \sigma_t^2$, cost variance.

In Sect. 4 we will see that, to compute the cost variance, we need to compute the following auxiliary variable:

- $\mathbb{E}[C_t | X_t = 1] = \nu_t$, cost expectation in case of success.

We use the well known inequality of Chebyshev [7] for a non-negative random variable Y

$$\mathbb{P}[Y \geq c] \leq \frac{\mathbb{E}[\varphi(Y)]}{\varphi(c)}$$

for each non-decreasing function φ and any positive number c. In particular,

$$\mathbb{P}[|\frac{C_t - \mu_t}{\sigma_t}| \geq c] \leq \frac{\mathbb{E}[(\frac{C_t - \mu_t}{\sigma_t})^2]}{c^2} = \frac{1}{c^2}$$

and therefore

$$\mathbb{P}[C_t \geq \mu_t + c \cdot \sigma_t] \leq \mathbb{P}[|\frac{C_t - \mu_t}{\sigma_t}| \geq c] \leq \frac{1}{c^2}.$$

Running example. We will restrict ourselves to orchestration processes that are constructed using the four routing constructs sequence, parallel composition, choice, and iteration. Therefore, we can present an orchestration process in a tree-like fashion, where the leaf nodes in the tree correspond to elementary tasks and the non-leaf nodes correspond to compound tasks, where every compound task uses one of the four routing constructs.

This allows us to represent the orchestration processes in a process algebraic notation where the sequence operator is denoted by $\cdot$, parallel composition by $||$, choice by $+$, and iteration by *. We will assume that * binds strongest, then $\cdot$, then $+$, and last $||$. This notation will be very useful for the derivation of the cost formulas in Sect. 4.

As running example, consider an image editing process p. The customer uploads an image (task u). Then, the following procedure is applied: The image is finished (task f) and concurrently a thumbnail is created (task t). Afterwards the results are evaluated (task e). In case the evaluation is negative, the procedure is repeated. Finally, if the image is too big, it is stored temporarily and only a link is sent to the customer (task l); otherwise the image is sent by e-mail (task m). The process can be expressed in our process-algebraic notation as follows (note that we have to unfold the iteration $(f||t) \cdot e$ once as the iteration is executed at least once): $p = u \cdot (f||t) \cdot e \cdot ((f||t) \cdot e)^* \cdot (l + m)$.

For the example, we assume that the following characteristics are given:

- In 90% of the cases, the upload succeeds.
- In 95% (80%) of the cases, finishing (creating the thumbnail) succeeds which costs the customer €20 (€10).
- In 99% of the cases the evaluation succeeds which costs another €5; in only 20% of the cases the loop has to be repeated.
- In 30% of the cases, the image is too big; the storing is charged with €10 whereas the sending of the images costs €1.

Note that we have two kinds of probabilities: the probability of a choice in the orchestration, here used for the iteration, and the probability of failure of a service. If some task of a service fails, the whole service fails. The question now is, what should the service provider charge the consumers to make any profit, and what are the risks involved? In the following, we present a mathematical framework to answer this question.

3 Web Services Framework

In the web services domain, Petri nets provide a good formalism to model web services that communicate with each other through an interface, which is typically modeled by *interface places* (see [8], for example). For this reason, we distinguish interface places from other places. An interface place can either be an input place or an output place. An input place has no (internal) input arcs, whereas an output place has no (internal) output arcs. As a result, a web service is not allowed to communicate with itself.

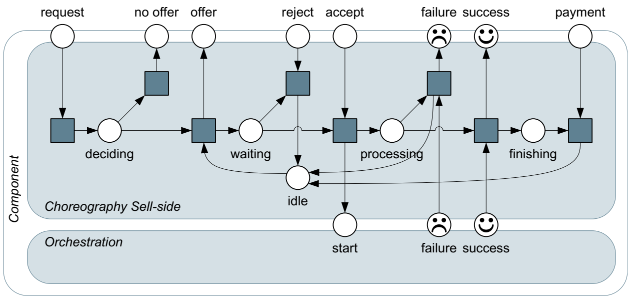


Fig. 1. The sell-side of a web service

A web service has a *sell-side process* and an *orchestration process*. The orchestration process has to be executed in order to deliver the service. The orchestration process consists of tasks. We distinguish *elementary* tasks, *outsourced* tasks, and *compound* tasks. An elementary task is modeled as a Petri net transition. An outsourced task is in fact a subnet that contains a *buy-side process*. A compound task is a subnet containing elementary or outscourced tasks. In fact, the whole orchestration process can be considered as one (compound) task. As mentioned in the introduction, we can distinguish between a sell-side and a buy-side of a web service. Typically, when some service wants to use another service, that is, if the former consumes a service that is being provided by the latter, the former first requests a quote from the latter. Based on the offer from the provider (which is optional, as the provider may decide not to offer the service), the client decides to either accept or reject the offer. If the offer is accepted, the provider actually provides the service, which might either succeed or fail. This result is communicated to the client, which pays the provider if the result was a success (no-cure-no-pay).

To provide the service, the provider might have to consume third-party services in some order. Clearly, the provider needs to *orchestrate* these third party services on-the-fly to achieve its goal. In contrast, the negotiation between the provider and the customer belongs to the *choreography process*.

Choreography. The choreography in the framework consists of the sell-side process of the web service and the buy-side processes of the outsourced tasks. Figure 1 visualizes the sell-side of a web service, whereas Fig. 2 visualizes the buy side of a task. As usual, circles represent places and squares represent transitions. For the ease of reference, sad smileys have been used for the failure places and happy smileys for the success places.

Underlying assumption for the sell-side is that it can handle a predefined number of requests simultaneously. This number corresponds to the number of tokens which are initially put into the place idle. Thus, if the maximum number of requests is being handled, then no offer can be made for the next request.

The buy-side of an outsourced task contains a buy-side process in which first a request for a proposal is sent to a potential supplier. The process is the "mirror" of the sell-side process except that the buy-side process handles requests in a sequential order.

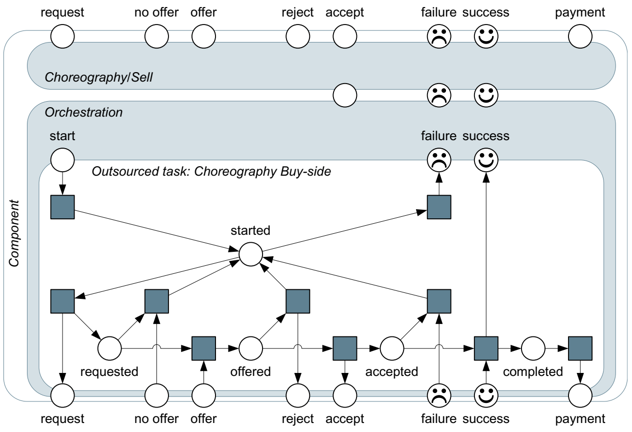


Fig. 2. A task in the buy-side of a web service

Orchestration. The web service can perform an *elementary task* like a computation, an *outsourced* task that requests a service from another web service like retrieving some information from an underlying database, or a *compound task* that needs to orchestrate a number of (sub)tasks. In principle, such an orchestration can be arbitrarily complex, but in this paper we consider four operations to construct compound tasks:

1. *sequence*, i.e. performing a number of tasks in a given order,
2. *parallel composition*, i.e. performing a number of tasks simultaneously,
3. *choice*, i.e. performing one task chosen from several tasks, based on some decision, and
4. *iteration*, i.e. performing a task as long as some condition holds.

Nevertheless, we would like to stress that the framework can be extended with additional operations if needed (as long as soundness is guaranteed, see further down in this section). Reason for restricting to this set of operations in this paper is that these four types are sufficient to explain the matters at hand, whereas additional ones might only distract the reader.

Figure 3 visualizes these four basic orchestration types. Again, we use the sad smileys for the failure places and the happy smileys for the success places, and we use grey boxes to visualize the orchestrated tasks. For the sake of simplicity, we used only two tasks for the sequence, parallel composition, and choice, but it is not hard to see that this scheme can be extended to any number.

Figure 4 depicts the example service using the web service framework, containing one choreography component (the sell-side process of the service), nine orchestration components (which are compound tasks), and nine elementary tasks. Six of the elementary tasks are outsourced tasks, which form the buy-side processes of the service.

Soundness. Clearly, any task should lead to either a success or a failure, i.e. if a task starts with one token in its start place, then for any reachable marking

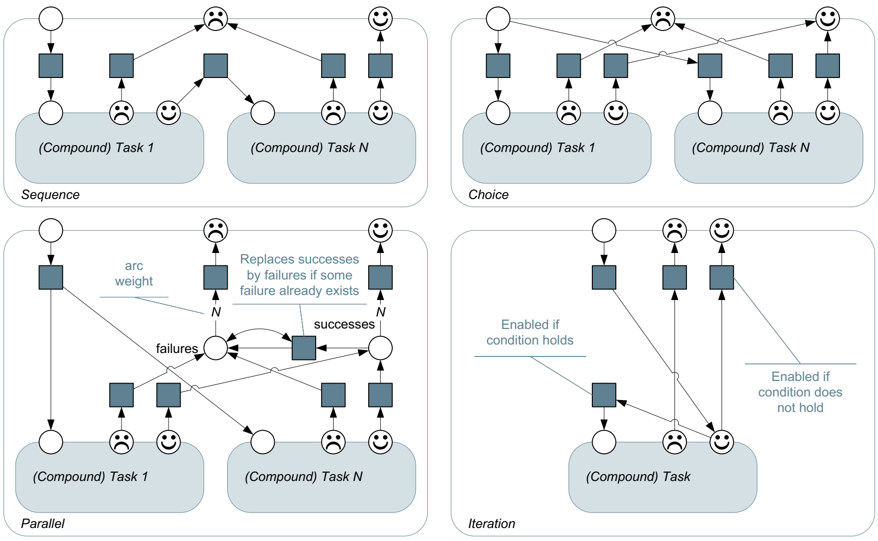

(Compound) Task 1
(Compound) Task N
Sequence
Choice
arc weight
Replaces successes by failures if some failure already exists
N
successes
failures
Parallel
Enabled if condition holds
Enabled if condition does not hold
(Compound) Task
Iteration

Fig. 3. The basic orchestration operations

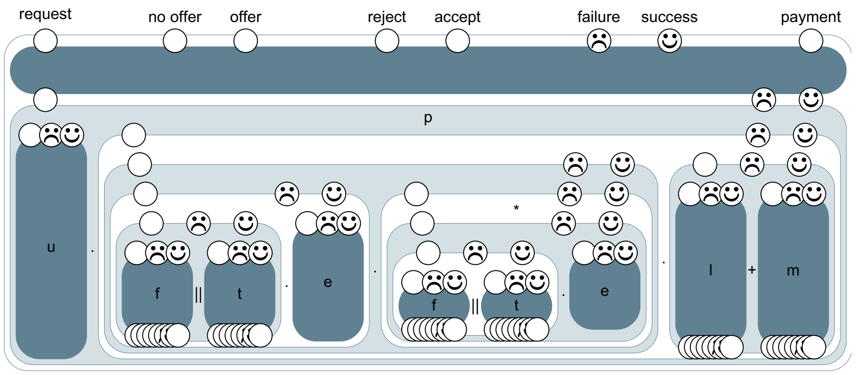


Fig. 4. The running service using the framework

it should be possible to reach the marking with only one token either in place success or in place failure. We call this the *soundness of a task*. This soundness concept differs a little from the one given in Sect. 2, namely we have two final places (success and failure) instead of one. By the construction of Fig. 5 these notions can be unified. In order to analyze soundness of a task, we distinguish two parts: soundness of the choreography part and of the orchestration part.

For the choreography part we connect the buy-side of one task to the sell-side of another service as displayed in Fig. 6. We first neglect the three places start, failure and success at the bottom. Then, soundness is not difficult to verify, e.g. by brute force model checking since the set or reachable markings is finite. It is also easy to verify this property by observing that the upper (the buy-side) and

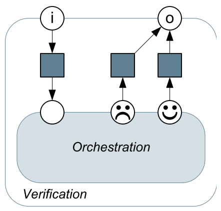


Fig. 5. Checking orchestration soundness

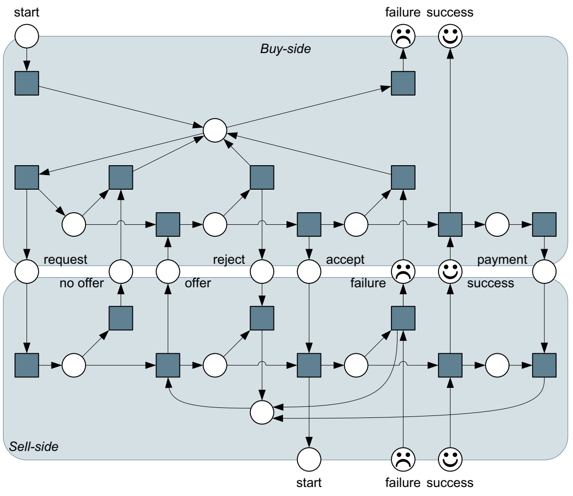


Fig. 6. Buy-side and sell-side combined

lower part (sell-side) of the model are almost symmetrical state machine nets where each choice is made by only one of them. When considering places start, failure and success at the bottom, we need the assumption that if a token is put into start, eventually there will be a token in either failure or success. Under this assumption the system is sound. So we introduce here the concept of *conditional soundness*: the choreography part is sound if the invoked tasks are sound.

For the orchestration part we use the same conditional soundness. Any orchestration is sound if and only if its tasks are sound. This is straightforward to check using only Fig. 3. So we only have to check the soundness of elementary tasks, which are just transitions. In case the framework is extended by some new orchestration operations, we have to make sure that the orchestration is still sound if and only if the tasks are sound.

4 Pricing Model

The probability of success of a task t is s_t. The cost C_t of task t is a random variable, with expectation μ_t, conditional expectation ν_t, given the service does not fail and variance σ_t^2. With these four characteristics we will determine a safe price and the risk in case of failure. The price should compensate the loss in case the service fails. So a lower bound for the price is μ_t/s_t which gives an expected reward of $(\mu_t/s_t) \cdot s_t$ which equals to the expected cost. However, the service must have an earning capacity to cover losses, so we choose a price $\mu_t/s_t + b$ where b is a parameter to be determined. The expected profit will be $b \cdot s_t$ which is the difference between the expected reward and the expected cost. In order to determine b, we have to calculate the risk. There are several ways to define the risk.

An obvious definition of *risk* is the probability that the real cost exceed the expected reward. So $\mathbb{P}[C_t \geq \mu_t + b \cdot s_t]$ should be small enough. If we accept a 5% risk we have to choose b such that

$$\mathbb{P}[C_t \geq \mu_t + b \cdot s_t] = \mathbb{P}[\frac{C_t - \mu_t}{\sigma_t} \geq \frac{b \cdot s_t}{\sigma_t}] \leq 0.05.$$

Assuming a normal distribution for C_t, we derive from the standard normal distribution, that $b \cdot s_t = 1.65$ and so $b = 1.65 \cdot \sigma_t/s_t$ and therefore the price of the service is $(\mu_t + 1.65 \cdot \sigma_t)/s_t$. The assumption of normality is justified by the fact that a service is built up by several independent services and so the central limit theorem [7] justifies a normal approximation. We also have an exact probabilistic bound based on the Chebyshev inequality (see Sect. 2)

$$\mathbb{P}[\frac{C_t - \mu_t}{\sigma_t} \geq \frac{b \cdot s_t}{\sigma_t}] \leq \mathbb{E}[(\frac{C_t - \mu_t}{\sigma_t})^2 \geq (\frac{b \cdot s_t}{\sigma_t})^2] = \frac{\sigma_t^2}{b^2 \cdot s_t^2} = 0.05$$

since $\mathbb{E}[((C_t - \mu_t)/\sigma_t)^2] = 1$. This gives $b = (\sigma_t/s_t)/\sqrt{0.05} = 4.47 \cdot \sigma_t/s_t$ which is almost three times as large as the former bound.

A second way to define the risk is the expected cost, under the condition that the task fails. Since we will calculate ν_t this can be expressed as $\mu_t - \nu_t \cdot s_t$. Note that this quantity is independent of the price. A third way is to define the risk is $\mathbb{E}[max(0, R_t - C_t)]$ where R_t is a random variable that represents the reward for a task t, so $R_t = 0$ in case of a failure and $R_t = \mu_t/s_t + b$ in case of success. This measure for risk, however, requires knowledge of the complete distribution of C_t. Therefore we will use the first interpretation of risk.

In the rest of this section we will calculate the four characteristics (success probability, cost expectation, cost variance and conditional cost expectation) in an inductive way: For an elementary task t we assume they are given. This is reasonable, because such values can be estimated from log files, for instance. For a compound task we will derive the characteristics for the four orchestration constructs sequence, parallel, choice, and iteration. We use X_t as the random variable that indicates if the task t is a success ($X_t = 1$) or a failure ($X_t = 0$). So this random variable has a Bernoulli distribution. Note that C_t and X_t are dependent. The reason for this is in the sequence construct: the cost in case of failure might be less than the cost of success. In the rest of the paper we assume that each invocation of a task instance is (stochastically) independent of all other instances, in particular for $a \neq b$ we have X_a and C_a are mutually independent of X_b and C_b. For a complete set of the proofs we refer to a technical report [9]. For the convenience of the reviewers, all proofs are listed in an appendix of this article which will be removed in a final version. To illustrate the techniques, we give the proofs for the sequence and the iteration.

Sequence. Assume that we have a sequence construct $a \cdot b$ that contains (in the given order) tasks a and b. Thus, $a \cdot b$ prescribes that first a has to be completed, after which b can be started. First, we compute the success probability ($s_{a\cdot b}$) for $a \cdot b$; second, we compute its cost expectation ($\mu_{a\cdot b}$); third, we compute the cost variance ($\sigma^2_{a\cdot b}$); and fourth, we compute the conditional cost expectation of $a \cdot b$ in case of success ($\nu_{a\cdot b}$). In all cases, we assume that these characteristics for a and b are known (by induction, if you will).

For the following proofs two properties are important: $X_{a\cdot b} = X_a \cdot X_b$ and $C_{a\cdot b} = C_a + X_a \cdot C_b$.

Theorem 1 (Success probability of $a \cdot b$)
$s_{a\cdot b} = s_a \cdot s_b$.

Proof

$$\begin{aligned} s_{a\cdot b} &= \mathbb{P}[X_{a\cdot b} = 1] = \mathbb{P}[X_a \cdot X_b = 1] = \mathbb{P}[X_a = 1 \wedge X_b = 1] \\ &= \mathbb{P}[X_a = 1] \cdot \mathbb{P}[X_b = 1] = s_a \cdot s_b \end{aligned}$$

Theorem 2 (Cost expectation of $a \cdot b$)
$\mu_{a\cdot b} = \mu_a + s_a \cdot \mu_b$

Proof

$$\mu_{a\cdot b} = \mathbb{E}(C_{a\cdot b}) = \mathbb{E}(C_a + X_a \cdot C_b) = \mathbb{E}(C_a) + \mathbb{E}(X_a) \cdot \mathbb{E}(C_b) = \mu_a + s_a \cdot \mu_b$$

For the proof of the next theorem, we need a lemma (for a proof see [9]).

Lemma 1
$\mathbb{E}(C^2_{a\cdot b}) = \sigma^2_a + \mu^2_a + s_a \cdot (\sigma^2_b + \mu^2_b) + 2 \cdot \mu_b \cdot \nu_a \cdot s_a$

Theorem 3 (Cost variance of $a \cdot b$)
$\sigma^2_{a\cdot b} = \sigma^2_a + s_a \cdot \sigma^2_b + (s_a - s^2_a) \cdot \mu^2_b + 2 \cdot (\nu_a - \mu_a) \cdot s_a \cdot \mu_b$

Proof

$$\begin{aligned} \sigma^2_{a\cdot b} &= \mathbb{E}(C^2_{a\cdot b}) - (\mathbb{E}(C_{a\cdot b}))^2 \\ &= \sigma^2_a + \mu^2_a + s_a \cdot (\sigma^2_b + \mu^2_b) + 2 \cdot \mu_b \cdot \nu_a \cdot s_a - (\mu_a + s_a \cdot \mu_b)^2 \\ &= \sigma^2_a + \mu^2_a + s_a \cdot (\sigma^2_b + \mu^2_b) + 2 \cdot \mu_b \cdot \nu_a \cdot s_a - \mu^2_a - s^2_a \cdot \mu^2_b - 2 \cdot \mu_a \cdot s_a \cdot \mu_b \\ &= \sigma^2_a + s_a \cdot \sigma^2_b + (s_a - s^2_a) \cdot \mu^2_b + 2 \cdot (\nu_a - \mu_a) \cdot s_a \cdot \mu_b \end{aligned}$$

Theorem 4 (Conditional cost expectation of $a \cdot b$ if success)
$\nu_{a\cdot b} = \nu_a + \nu_b$

Proof

$$\begin{aligned} \nu_{a\cdot b} &= \mathbb{E}[C_{a\cdot b} | X_{a\cdot b} = 1] = \mathbb{E}[C_a + X_a \cdot C_b | X_a = X_b = 1] \\ &= \mathbb{E}[C_a | X_a = X_b = 1] + \mathbb{E}[X_a \cdot C_b | X_a = X_b = 1] \\ &= \mathbb{E}[C_a | X_a = 1] + \mathbb{E}[C_b | X_a = X_b = 1] = \mathbb{E}[C_a | X_a = 1] + \mathbb{E}[C_b | X_b = 1] \\ &= \nu_a + \nu_b \end{aligned}$$

Parallel composition. Assume that we have a parallel construct $a||b$ that contains tasks a and b. Thus, $a||b$ prescribes that both a and b have to be executed, and that they can be executed in parallel. Like with the sequence construct, the parallel construct $a||b$ is successful if both a and b are successful. The following theorems use the properties: $X_{a||b} = X_a \cdot X_b$ and $C_{a\|b} = C_a + C_b$.

Theorem 5 (Success probability of $a||b$)
$s_{a||b} = s_a \cdot s_b$.

Theorem 6 (Cost expectation of $a||b$)
$\mu_{a||b} = \mu_a + \mu_b$

Theorem 7 (Cost variance of $a||b$)
$\sigma^2_{a||b} = \sigma^2_a + \sigma^2_b$

Theorem 8 (Conditional cost expectation of $a||b$ if success)
$\nu_{a||b} = \nu_a + \nu_b$

Choice. Assume that we have a choice construct $a + b$ that contains tasks a and b, and that the alternatives (a and b) are chosen with an independent random variable A, with $\mathbb{P}[A = 1] = \alpha = 1 - \mathbb{P}[A = 0]$. If $A = 1$, then a will be chosen, else if $A = 0$, then b will be chosen. The following theorems use: $X_{a+b} = A \cdot X_a + (1 - A) \cdot X_b$ and $C_{a+b} = A \cdot C_a + (1 - A) \cdot C_b$.

Theorem 9 (Success probability of $a + b$)
$s_{a+b} = \alpha \cdot s_a + (1 - \alpha) \cdot s_b$.

Theorem 10 (Cost expectation of $a + b$)
$\mu_{a+b} = \alpha \cdot \mu_a + (1 - \alpha) \cdot \mu_b$

Theorem 11 (Cost variance of $a + b$)
$\sigma^2_{a+b} = \alpha \cdot \sigma^2_a + (1 - \alpha) \cdot \sigma^2_b + \alpha \cdot (1 - \alpha) \cdot (\mu_a - \mu_b)^2$

Theorem 12 (Conditional cost expectation of $a + b$ if success)
$\nu_{a+b} = \frac{\nu_a \cdot \alpha \cdot s_a + \nu_b \cdot (1-\alpha) \cdot s_b}{\alpha \cdot s_a + (1-\alpha) \cdot s_b}$

Iteration. Assume that we have an iteration construct a^*, which contains the task a, and that the i-th iteration is chosen with an independent random variable Y_i, with $\mathbb{P}[Y_i = 1] = \alpha = 1 - \mathbb{P}[Y_i = 0]$. To compute iteration characteristics we can simply unfold the iteration once using an alternative, a sequence, and a skip action τ, which has the following characteristics: $s_\tau = 1$, $\mu_\tau = 0$, $\sigma^2_\tau = 0$, and $\nu_\tau = 0$. Thus, $a^* = \tau + a \cdot a^*$, where τ has probability $1 - \alpha$ and $a \cdot a^*$ has probability α. In terms of random variables we have the recursive equation: $P = Y \cdot T + (1 - Y) \cdot A \cdot P$ where P stands for the process a^*, T stands for task τ and A for task a. We get the following characteristics:

Theorem 13 (Success probability of a^*)
$s_{a^*} = \frac{1-\alpha}{1-\alpha \cdot s_a}$.

Proof

$$\begin{aligned} s_{a^*} &= s_{\tau+a\cdot a^*} = (1-\alpha)\cdot s_\tau + \alpha\cdot s_{a\cdot a^*} = 1-\alpha+\alpha\cdot s_{a\cdot a^*}\\ &= 1-\alpha+\alpha\cdot s_a\cdot s_{a^*} = \tfrac{1-\alpha}{1-\alpha\cdot s_a} \end{aligned}$$

Theorem 14 (Cost expectation of a^*)

$\mu_{a^*} = \frac{\alpha\cdot\mu_a}{1-\alpha\cdot s_a}$

Proof

$$\begin{aligned} \mu_{a^*} &= \mu_{\tau+a\cdot a^*} = (1-\alpha)\cdot\mu_\tau + \alpha\cdot\mu_{a\cdot a^*} = \alpha\cdot\mu_{a\cdot a^*} = \alpha\cdot(\mu_a + s_a\cdot\mu_{a^*})\\ &= \tfrac{\alpha\cdot\mu_a}{1-\alpha\cdot s_a} \end{aligned}$$

Theorem 15 (Cost variance of a^*)

$\sigma^2_{a^*} = \frac{\alpha\cdot(\sigma^2_a+(s_a-s^2_a)\cdot\mu^2_{a^*}+2\cdot(\nu_a-\mu_a)\cdot s_a\cdot\mu_{a^*})+(1-\alpha)\cdot\alpha\cdot\mu^2_{a\cdot a^*}}{1-\alpha\cdot s_a}$

Proof

$$\begin{aligned} \sigma^2_{a^*} &= \sigma^2_{\tau+a\cdot a^*} = (1-\alpha)\cdot\sigma^2_\tau + \alpha\cdot\sigma^2_{a\cdot a^*} + (1-\alpha)\cdot\alpha\cdot(\mu_\tau-\mu_{a\cdot a^*})^2\\ &= \alpha\cdot\sigma^2_{a\cdot a^*} + (1-\alpha)\cdot\alpha\cdot\mu^2_{a\cdot a^*}\\ &= \alpha\cdot(\sigma^2_a + s_a\cdot\sigma^2_{a^*} + (s_a-s^2_a)\cdot\mu^2_{a^*} + 2\cdot(\nu_a-\mu_a)\cdot s_a\cdot\mu_{a^*}) +\\ &\qquad (1-\alpha)\cdot\alpha\cdot\mu^2_{a\cdot a^*}\\ &= \alpha\cdot s_a\cdot\sigma^2_{a^*} + \alpha\cdot(\sigma^2_a + (s_a-s^2_a)\cdot\mu^2_{a^*} + 2\cdot(\nu_a-\mu_a)\cdot s_a\cdot\mu_{a^*}) +\\ &\qquad (1-\alpha)\cdot\alpha\cdot\mu^2_{a\cdot a^*}\\ &= \tfrac{\alpha\cdot(\sigma^2_a+(s_a-s^2_a)\cdot\mu^2_{a^*}+2\cdot(\nu_a-\mu_a)\cdot s_a\cdot\mu_{a^*})+(1-\alpha)\cdot\alpha\cdot\mu^2_{a\cdot a^*}}{1-\alpha\cdot s_a} \end{aligned}$$

Theorem 16 (Conditional cost expectation of a^* if success)

$\nu_{a^*} = \frac{\nu_a\cdot\alpha\cdot s_a}{1-\alpha\cdot s_a}$

Proof

$$\begin{aligned} \nu_{a^*} &= \nu_{\tau+a\cdot a^*} = \tfrac{\nu_\tau\cdot(1-\alpha)\cdot s_\tau+\nu_{a\cdot a^*}\cdot\alpha\cdot s_{a\cdot a^*}}{(1-\alpha)\cdot s_\tau+\alpha\cdot s_{a\cdot a^*}} = \tfrac{\nu_{a\cdot a^*}\cdot\alpha\cdot s_{a\cdot a^*}}{(1-\alpha)+\alpha\cdot s_{a\cdot a^*}}\\ &= \tfrac{(\nu_a+\nu_{a^*})\cdot\alpha\cdot s_{a\cdot a^*}}{(1-\alpha)+\alpha\cdot s_{a\cdot a^*}} = \tfrac{\nu_a\cdot\alpha\cdot s_{a\cdot a^*}+\nu_{a^*}\cdot\alpha\cdot s_{a\cdot a^*}}{(1-\alpha)+\alpha\cdot s_{a\cdot a^*}} = \tfrac{\nu_a\cdot\alpha\cdot s_{a\cdot a^*}+\nu_{a^*}\cdot\alpha\cdot s_{a\cdot a^*}}{(1-\alpha)+\alpha\cdot s_{a\cdot a^*}}\\ &= \tfrac{\nu_a\cdot\alpha\cdot s_{a\cdot a^*}}{(1-\alpha)+\alpha\cdot s_{a\cdot a^*}} + \tfrac{\nu_{a^*}\cdot\alpha\cdot s_{a\cdot a^*}}{(1-\alpha)+\alpha\cdot s_{a\cdot a^*}} = \tfrac{\nu_a\cdot\alpha\cdot s_{a\cdot a^*}}{(1-\alpha)+\alpha\cdot s_{a\cdot a^*}} + \nu_{a^*}\cdot\tfrac{\alpha\cdot s_{a\cdot a^*}}{(1-\alpha)+\alpha\cdot s_{a\cdot a^*}}\\ &= \tfrac{\nu_a\cdot\alpha\cdot s_{a\cdot a^*}}{(1-\alpha)+\alpha\cdot s_{a\cdot a^*}}/(1-\tfrac{\alpha\cdot s_{a\cdot a^*}}{(1-\alpha)+\alpha\cdot s_{a\cdot a^*}}) = \tfrac{\nu_a\cdot\alpha\cdot s_{a\cdot a^*}}{1-\alpha} = \tfrac{\nu_a\cdot\alpha\cdot s_a\cdot\frac{1-\alpha}{1-\alpha\cdot s_a}}{1-\alpha}\\ &= \tfrac{\nu_a\cdot\alpha\cdot s_a}{1-\alpha\cdot s_a} \end{aligned}$$

Applying our proposed cost model for the example process p we can (incrementally) compute the following values for the four characteristics (see Fig. 7):

- $s_p = 0.64$; i.e. the service succeeds in about 64% of the cases.
- $\mu_p = 34.95$, i.e. the provider has expected costs of about €34.95.
- $\sigma^2_p = 427.81$ is the cost variance
- $\nu_p = 44.90$ are the conditional cost in case of success

Hence $\sigma_p/s_p = 32.31$ and $\mu_p/s_p = 54.60$. We assume the provider wants to accept a risk of 5%. If we apply the normal approximation, we obtain $b = 53.31$ and so the price of the service $\mu_p/s_p + b$ should be €107.91. The expected profit $b\cdot s_p$ equals €34.11. If we apply the exact Chebyshev inequality we obtain $b = 142.16$ and then the price becomes €199.02. Then the expected profit equals €90.98 which is obviously larger. In both cases is the expected cost, given the service fails, $\mu_p - \nu_p\cdot s_p$, equals €6.21. It is not difficult to repeat this exercise for other choices of the risk.

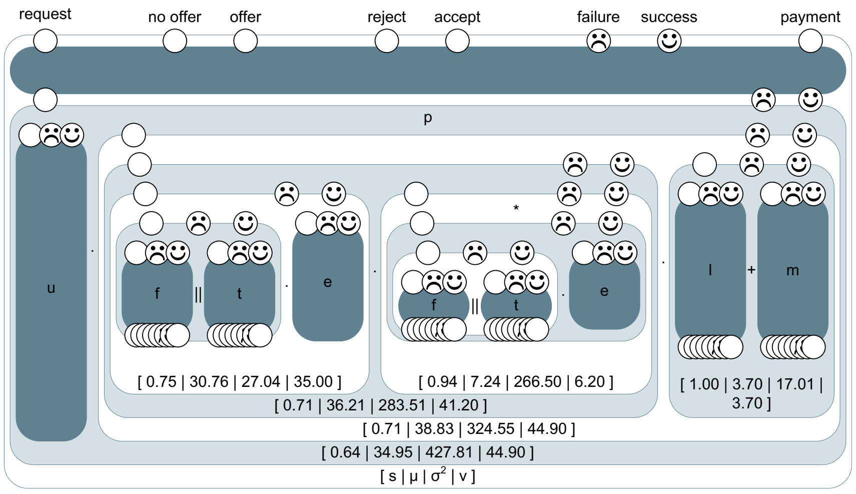


Fig. 7. The characteristics of the running service

5 Related Work

In our previous work [10], we have developed an SOA-based architecture framework which is similar to the service component architecture (SCA) [11]. In this article, we extended this work by a web service framework which allows to check the soundness property compositionally. The goals of this part of our work are close to the ones of Milanovic [12], where the author seeks for correctness proofs for compositions of web services. The framework there is based of abstract state machines, the same four basic operations are considered (the iteration is however left out of consideration in the correctness part), and proof obligations are generated to show the correctness of the composition.

In [13], Diaz et al. represent a translation from WS-CDL to timed automata and from timed automata to WS-BPEL to generate web services. The correctness is checked by model checking properties of the obtained timed automata.

Furthermore, we have presented a method to compute the cost of a web service. There is a long list of publications dealing with cost of services. However, to the best of our knowledge none of them takes the risk (i.e. the variance of cost) into consideration.

Magnani and Montesi [14] present rules to calculate the cost of BPMN models. These rules cover operations like sequence, parallel, choice, and loop. However, there are no rules given for calculating the probability of success of these operations.

Cardoso et al. [15] present a QoS model for time, reliability, and cost of workflows. Each task has a QoS attribute. Based on these attributes, the cost of the overall workflow can be computed using the METEOR workflow system. In [16], Zeng et al. present a framework for QoS-aware service selection. Price is one of the nonfunctional properties which are taken into account. Paoli et al. [17] address the problem of designing a composed system that has to guarantee certain quality criteria such as security, completion time, and also cost. It is shown how these criteria can be computed on the structure of a service. To this end, quality evaluation rules (similar those in [15]) for sequence, parallel, switch, and loop are proposed. To summarize, all three approaches cover a broader spectrum of QoS criteria than cost. However, probabilities for successful termination and for the price calculation of activities and services are not considered.

In [18], Brocke and Lindner present a general framework for the evaluation of the financial consequences of outsourcing, while in [19] Günter et al. investigate pricing mechanisms appropriate for web services. This is, however, far beyond the scope of this paper. In [20], Ding proposes a method of value-based pricing where the price is a function reflecting the expected value of the product. This method can be used, for example, to derive the price of each elementary task.

6 Conclusion

In this paper, we have presented a web services framework to design service orchestrations that are sound by design. To this end, service interfaces and routing constructs are designed in such a way that two arbitrary services always interact properly.

In order to provide a certain functionality, web services often have to buy some functionality from other web services. In such a setting calculating cost of a web service is an important issue, because a web service can only survive on the market if it is profitable. To this end, we have provided an approach to compute the expected cost of a sound service orchestration (i.e. web service). Since we consider no-cure-no-pay services, that is, a customer only pays for a delivered service, the proposed approach takes probabilities for successful execution of a web service into account. We also consider the risk involved by calculating the cost variance. The approach is compositional in the sense that the cost of the whole web service is computed from the cost of all tasks. Therefore we developed rules to compute the cost, the probability of success, and the cost variance for the constructs sequence, parallel, choice, and iteration.

So far the proposed approach is subject to some restrictions. With sequence, parallel, choice, and iteration we only consider a restricted set of operations to construct compound tasks. Concepts such as cancelation and compensation of tasks are also not considered. Furthermore, it might be interesting to see to what extent processes other than no-cure-no-pay processes can be supported in a similar way with our approach.

In ongoing research we plan to extend our web service framework with more types of choreography protocols, in particular protocols that will ease cancelation of (parts of) services and allow for compensation mechanisms. An other extension we consider is to relax the assumption that all tasks in an iteration are independent of each other. This is more realistic, however, it requires more knowledge (data) about the performance of tasks.

References

1. Micali, S., Rivest, R.L.: Micropayments revisited. In: Preneel, B. (ed.) CT-RSA 2002. LNCS, vol. 2271, pp. 149–163. Springer, Heidelberg (2002)
2. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77(4), 541–580 (1989)
3. Reisig, W.: Petri Nets: An Introduction. EATCS Monographs on Theoretical Computer Science, vol. 4. Springer, Berlin (1985)
4. 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)
5. Verbeek, H.M.W., Basten, T., van der Aalst, W.M.P.: Diagnozing workflow processes using Woflan. The Computer Journal 44(4), 246–279 (2001)
6. Verbeek, H.M.W., van der Aalst, W.M.P.: Woflan 2.0: A Petri-net-based workflow diagnosis tool. In: Nielsen, M., Simpson, D. (eds.) ICATPN 2000. LNCS, vol. 1825, pp. 475–484. Springer, Heidelberg (2000)
7. Ross, S.: Introduction to probability models. Academic Press, London (2007)
8. Massuthe, P., Reisig, W., Schmidt, K.: An Operating Guideline Approach to the SOA. Annals of Mathematics, Computing & Teleinformatics 1(3), 35–43 (2005)
9. van Hee, K.M., Verbeek, H.M.W., Stahl, C., Sidorova, N.: A Framework for Linking and Pricing No-Cure-No-Pay Services. Computer Science Report 08/19, Technische Universiteit Eindhoven, The Netherlands (2008)
10. van der Aalst, W.M.P., Beisiegel, M., van Hee, K.M., König, D., Stahl, C.: An SOA-based architecture framework. International Journal of Business Process Integration and Management (IJBPIM) 2(2), 91–101 (2007)
11. Beisiegel, M., et al.: Service Component Architecture – Assembly Model Specification. SCA Version 1.00, March 15 2007, IBM, SAP et al (2007)
12. Milanovic, N.: Contract-based web service composition framework with correctness guarantees. In: Malek, M., Nett, E., Suri, N. (eds.) ISAS 2005. LNCS, vol. 3694, pp. 52–67. Springer, Heidelberg (2005)
13. Díaz, G., Cambronero, M.-E., Pardo, J.J., Valero, V., Cuartero, F.: Automatic generation of correct web services choreographies and orchestrations with model checking techniques. In: AICT/ICIW (2006)
14. Magnani, M., Montesi, D.: BPMN: How Much Does It Cost? An Incremental Approach. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 80–87. Springer, Heidelberg (2007)
15. Cardoso, J., Sheth, A.P., Miller, J.A., Arnold, J., Kochut, K.: Quality of service for workflows and web service processes. J. Web Sem. 1(3), 281–308 (2004)
16. Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H.: Qos-aware middleware for web services composition. IEEE Trans. Software Eng. 30(5), 311–327 (2004)
17. De Paoli, F., Lulli, G., Maurino, A.: Design of quality-based composite web services. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 153–164. Springer, Heidelberg (2006)

18. vom Brocke, J., Lindner, M.A.: Service portfolio measurement: a framework for evaluating the financial consequences of out-tasking decisions. In: Aiello, M., Aoyama, M., Curbera, F., Papazoglou, M.P. (eds.) ICSOC 2004, pp. 203–211. ACM, New York (2004)
19. Günther, O., Tamm, G., Leymann, F.: Pricing web services. International Journal of Business Process Integration and Management (IJBPIM) 2(2), 132–140 (2007)
20. Ding, W.: Services pricing through business value modeling and analysis. In: 2007 IEEE International Conference on Services Computing (SCC 2007), July 9-13, 2007, pp. 380–386. IEEE Computer Society, Salt Lake City (2007)

Empirical Studies in Process Model Verification

Jan Mendling

Humboldt-Universität zu Berlin
Spandauer Straße 1, 10178 Berlin, Germany
jan.mendling@wiwi.hu-berlin.de

Abstract. Despite the large body of knowledge on formal analysis techniques for process models, in particular Petri nets, there has been a notable gap of empirical research into verification. In this paper we compare the few studies that report results from applying verification techniques to real-world process model collections. For this comparison we are particularly interested in the different approaches, their computational performance, and the number of errors found. Our comparison reveals that most of the samples have error rates of 10% to 20%. Some of the studies have established a connection between error probability and process model metrics, as well as between model understanding and both metrics and modeling competence of the model reader. Based on these results, we discuss implications and directions for future research.

1 Introduction

Even though workflow and process modeling have been used extensively over the past 30 years, we know surprisingly little about the factors that contribute to the quality of a process model and how respective quality assurance can be facilitated in real-world projects. This observation contrasts the large body of knowledge that is available on the formal analysis and verification of desirable properties, in particular for Petri nets. Furthermore, there is a notable disconnection between these mostly formal contributions and high-level conceptual work on guidelines and quality frameworks, e.g. [1,2,3]. Clearly, an empirical research agenda is needed to acquire new insights on quality and usage aspects of process modeling [4,5] and its relationship to validation and verification (V&V).

It is a fundamental insight of software engineering that design errors should be detected as early as possible [6,7,4]. The later errors are detected, the more rework has to be done, and the more design effort has been at least partially useless. This also holds for the consecutive steps of analysis, design and implementation in the business process management life cycle [8,9]. Yet, there are only a few papers that discuss validation and verification (V&V) as part of process design, e.g. [10], and the support for V&V activities is rather poor in current process modeling tools [11]. On the other hand, there are clearly quality issues with real-world process models. Recent studies report a significant rate of models with control flow errors in industry process model collections [11,12,13,14,15]. The mentioned lack of V&V features in tools as well as the lack of modeling competence in large-scale modeling projects [8] contribute to such high error rates. These rates are particularly problematic when design models with undetected errors are forwarded to the implementation phase, to the frustration of system engineers. Therefore, the so-called gap between business process design and implementation phase, i.e. the limited reuse of conceptual process models in later design stages, might be partially caused by a lack of quality assurance in the early design phase.

Against this background, this paper aims to compare five recent studies on process model verification for real-world model collections, namely [11,12,13,14,15]. Such a comparison is beneficiary for the process verification community. It offers conclusions on the performance of verification approaches, though limited, and on the value of verification as a means of quality assurance. For instance, the decomposition approach used in [14] can be combined with reachability analysis in future research. Furthermore, the comparison is important for the process modeling community. The studies highlight issues with process modeling practice that have implications for the design activity and its tool support. Error patterns and their relative frequency allow us to derive guidelines that contribute to a less error-prone modeling style. The Seven Guidelines of Process Modeling (7PMG) [16] demonstrate that empirical insights can be directly fueled back into the design process.

The remainder of the paper is organized as follows. In Section 2 we introduce validation and verification as two complementary activities for quality assurance of process models. We discuss how both relate to each other and which techniques have been applied in verification studies. Section 3 presents findings from these verification studies and compares them in terms of error rates and performance. Building on these results, Section 4 discusses explanations of error rates and implications for process modeling. Finally, Section 5 concludes the paper.

2 The Scope of Verification

Validation and verification (V&V) are integral parts for establishing confidence in the quality of a process or a workflow model. Section 2.1 defines the scope of both validation and verification. Our focus will be the control flow of the process model. Section 2.2 introduces Event-driven Process Chains (EPCs), a modeling language that is considered in several verification studies, and define the notion of error. We use EPCs also because they include a superset of routing elements that are found in languages like BPMN, UML and YAWL. Furthermore, we illustrate typical verification issues by the help of an example. Finally, Section 2.3 presents verification techniques that have been used in recent empirical studies.

2.1 Validation Versus Verification

The importance of V&V was recognized in software engineering from the start. As programming is in essence a problem-solving task, it implies that the validity of the solution must be established [17]. In this context, the IEEE Standard Glossary [18] defines Validation and Verification (V&V) as

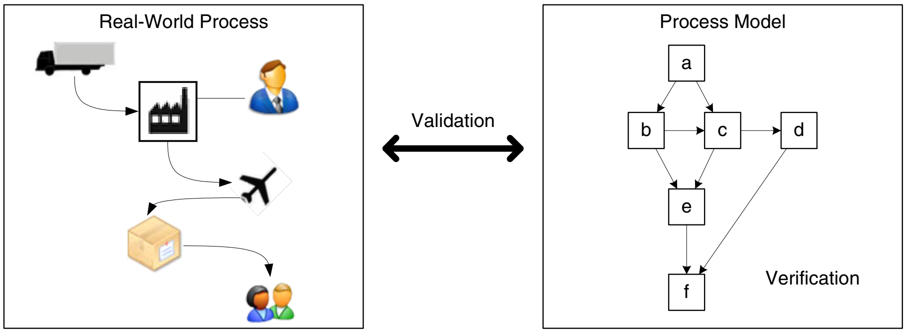


Fig. 1. Real-world process and process model

> "the process of determining whether the requirements of a system or component are complete and correct, the products of each development phase fulfill the requirement or conditions imposed by the previous phase, and the final system or component complies with specified requirements."

Different authors distinguish validation and verification as two complementary quality assurance steps [19,20,21,22].

Verification. In essence, verification addresses both the general properties of a model and the satisfaction of a given formula by a model (see Figure 1). Related to the first aspect, formal correctness criteria play an important role in process modeling. Several criteria have been proposed including soundness for Workflow nets [23], relaxed soundness [24], or well-structuredness (see [25,26] for comparisons). The second aspect is the subject of model checking and involves issues like separation of duty constraints, which can be verified, for example, by using linear temporal logic (LTL) [27]. The key characteristic of verification is that it relates to the internal correctness of a process model. Since it operates on the formal structure of the process model, it can be conducted without considering the real-world process.

Validation. In contrast to that, validation addresses the consistency of the model with the universe of discourse, i.e. the real-world process (see Figure 1). As it is an external correctness criterion, it is more difficult and more ambiguous to decide. While verification typically relies on an algorithmic analysis of the process model, validation requires the consultation of the specification and discussion with process stakeholders. Although validation requires human judgement as a key characteristic, it should be noted that formal methods are useful to support it. For instance, simulation, animation or derivation of natural-language statements facilitate the validation of a process model by users.

The relationship between validation and verification in process modeling has been discussed in [10]. Most notably, the authors propose to perform first verification, then validation. In this way, validation is only conducted if verification succeeds. People involved with the real-world process have to be consulted if the verification problem cannot be resolved based on the existing documentation. If one of the steps points to problems, the model has to be modified appropriately followed by a new iteration of verification and validation. A consequence of this procedure is that validation will be conducted less often than verification. This speeds up the design time and reduces cost: since verification is performed automatically by the help of tools, it is cheaper and faster than validation.

2.2 Formal Errors of Process Models

Four of the recent verification studies [11,12,13,15] use Event-driven Process Chain (EPC) business process models as input. Therefore, we briefly introduce EPCs and some potential correctness issues of their control flow. For formalization of EPC syntax and semantics refer to [28,29].

The Event-driven Process Chain (EPC) is a business process modeling language for the represention of temporal and logical dependencies of activities in a business process [31]. EPCs offer *function type* elements to capture the activities of a process and *event type* elements describing pre- and post-conditions of functions. Furthermore, there are three kinds of *connector types* (i.e. AND, OR, and XOR) for the definition of complex routing rules. Connectors have either multiple incoming and one outgoing arc (join) or one incoming and multiple outgoing arcs (split). *Control flow arcs* are used to link elements.

The informal (or intended) semantics of an EPC can be described as follows. The AND-split activates all subsequent branches in a concurrent fashion. The XOR-split represents a choice between exclusive alternative branches. The OR-split triggers one, two or up to all of multiple branches based on conditions. In both cases of the XOR- and OR-split, the activation conditions are given in events subsequent to the connector. Accordingly, splits from events to functions are forbidden with XOR and OR since the activation conditions do not become clear in the model. The AND-join waits for all incoming branches to complete, then it propagates control to the subsequent EPC element. The XOR-join merges alternative branches. The OR-join synchronizes all active incoming branches, i.e., it needs to know whether the incoming branches may receive tokens in the future. This feature is called *non-locality* since the state of all (transitive) predecessor nodes has to be considered.

Figure 2 shows an EPC of the SAP Reference Model that was analyzed in a recent verification study [13]. This figure illustrates nine errors that were found. In essence, there are two types of errors that may occur in an incorrect process model: deadlocks and lack of synchronization. In the simplest case, a *deadlock* results from a combination of an XOR-split with an AND-join: while the split only activates one control path, the AND is waiting for both to be completed. A deadlock can be therefore described as a situation where synchronizing joins do not receive control from enough incoming branches to proceed. In the figure, the errors 4 and 9 are potential deadlocks because the AND can receive control from one input branch but, in that case, it is not guaranteed that it also gets control from the other. Error 8 is a deadlock according to the semantics of [29] because

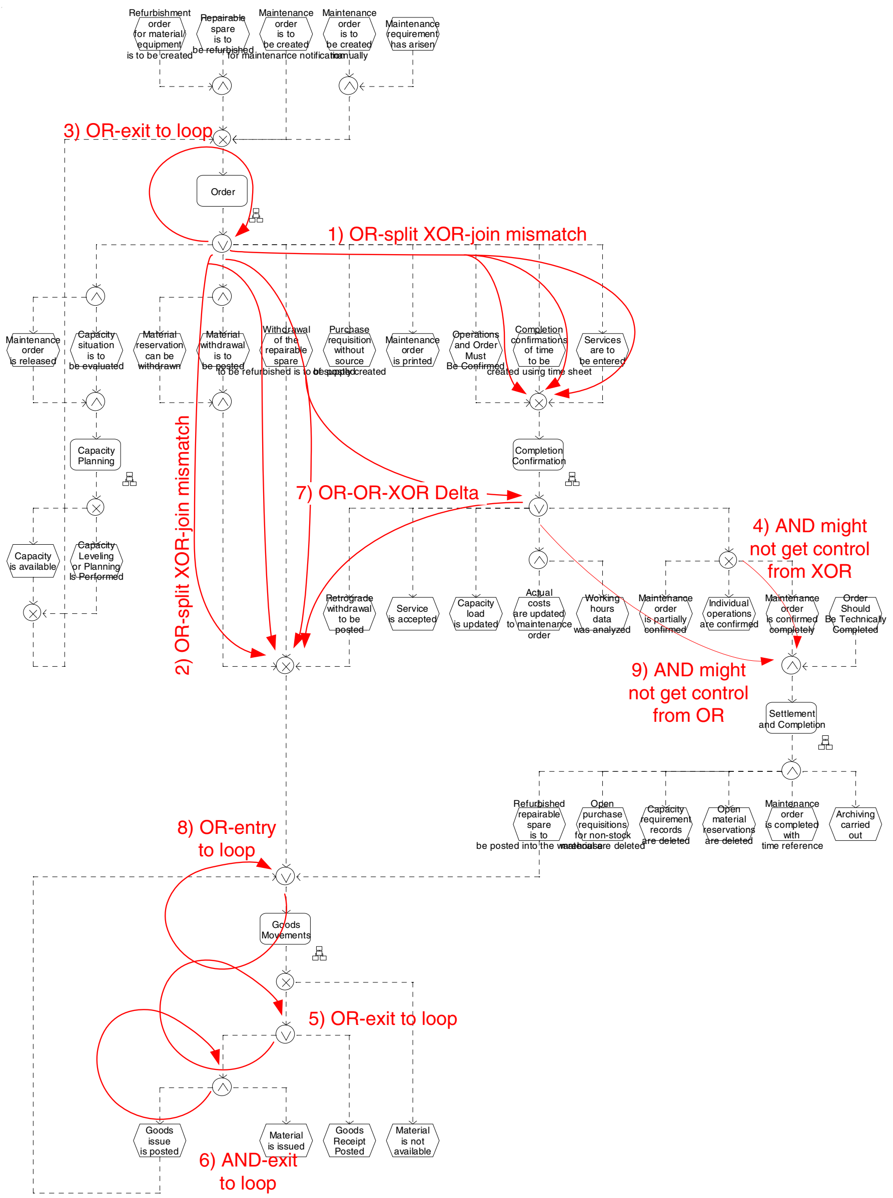

Refurbishment order for material equipment is to be created
Repairable spare is to be refurbished
Maintenance order is to be created for maintenance notification
Maintenance order is to be created manually
Maintenance requirement has arisen
3) OR-exit to loop
Order
1) OR-split XOR-join mismatch
Maintenance order is released
Capacity situation is to be evaluated
Material reservation can be withdrawn
Material withdrawal is to be posted
Withdrawal of the repairable spare to be refurbished is to be posted
Purchase requisition without source of supply created
Maintenance order is printed
Operations and Order Must Be Confirmed
Completion confirmations of time to be created using time sheet
Services are to be entered
Capacity Planning
Completion Confirmation
2) OR-split XOR-join mismatch
7) OR-OR-XOR Delta
Capacity is available
Capacity Leveling or Planning Is Performed
4) AND might not get control from XOR
Retrograde withdrawal to be posted
Service is accepted
Capacity load is updated
Actual costs are updated to maintenance order
Working hours data was analyzed
Maintenance order is partially confirmed
Individual operations are confirmed
Maintenance order is confirmed completely
Order Should Be Technically Completed
9) AND might not get control from OR
Settlement and Completion
Refurbished repairable spare is to be posted into the warehouse
Open purchase requisitions for non-stock materials are deleted
Capacity requirement records are deleted
Open material reservations are deleted
Maintenance order is completed with time reference
Archiving carried out
8) OR-entry to loop
Goods Movements
5) OR-exit to loop
Goods issue is posted
Material is issued
Goods Receipt Posted
Material is not available
6) AND-exit to loop

Fig. 2. The Refurbishment Processing in Plant Maintenance EPC from the SAP Reference Model [30, p.153] with errors being highlighted

the OR-join waits for itself via the branch from within the loop. A process model suffers from a *lack of synchronization* if multiple branches are activated, but not synchronized later at XOR-joins. Instances of this error type are errors 1, 2, 3, 5, 6, and 7 where multiple incoming branches can be active at the same time.

We define an *error* of a process model as a structural pattern that results in a deadlock or in a lack of synchronization. In order to identify errors, the formal behavior of the process has to be analyzed. This behavior is often defined based on *states and transitions*. Different *correctness criteria* can be verified using the graph of reachable states (*reachability graph*). They differ in computational complexity and in the precision of pinpointing errors. The following section presents the verification approaches of the five studies considered in this paper.

2.3 Verification Approaches

Verification is of particular importance to process modeling. Careless design can easily lead to process models with deadlocks or lack of synchronization which can have a serious business consequences if these models are used as workflows in operations. There are different aspects of a process model (e.g. control flow, data flow or resource allocation), and we focus on control flow here. The need for verification techniques stems from the fact that the complexity of the behavior specified in a process model is difficult to analyze for a human modeler [13]. This complexity is often referred to as the state explosion problem which is inherent to the specification of concurrency [19]. This problem imposes constraints on the applicability of state space analysis since the reachability graph can become exponentially large or even infinite. Therefore, reduction and decomposition techniques are used in order to improve the performance of verification. Below we discuss the correctness criteria that have been used in recent studies and the way they are implemented to verify industry-scale process models. First, we introduce soundness as the classical correctness criterion for process models as a reference. Then, relaxed soundness, an interactive verification approach, EPC soundness, single-entry-single-exit decomposition, and structuredness are discussed. These have been used in the five studies.

Soundness. Soundness is an important and prominent correctness criterion for business process models and was first introduced in [23]. The original soundness property is defined for a Workflow net, a Petri net with one source and one sink, and requires that (i) for every state reachable from the source, there exists a firing sequence to the sink (option to complete); (ii) the state with a token in the sink is the only state reachable from the initial state with at least one token in it (proper completion); and (iii) there are no dead transitions [23]. It has been shown that soundness of a Workflow net is equivalent to liveness and boundedness of the corresponding short-circuited Petri net [23]. Therefore, several liveness and boundedness analysis techniques [32,33] are directly applicable to the verification of soundness. Tools like Woflan facilitate the verification of soundness in practice [34,35]. *Soundness identifies all deadlocks and lack of synchronization for process models with one start and one end node.*

Relaxed Soundness. The soundness property of workflow models has stimulated the specification of several soundness derivatives, mainly because some soundness aspects proved to be too restrictive in certain application domains. In [24] the authors argue that business processes are often conceptually modeled in such a way that only the desired behavior results in a proper completion. Since such conceptual models are not used for workflow execution, deadlocks are resolved by the people working in the process in a cooperative and ad-hoc fashion. The authors define a process to be relaxed sound if every transition in a Petri net representation of the process model is included in at least one proper execution sequence [24]. The relaxed soundness property is used in a study on the verification of the SAP Reference Model [36,12]. For this study the authors automatically transform the 604 EPCs of the reference model to YAWL nets which are then analyzed using the WofYAWL tool. WofYAWL uses Petri nets analysis techniques including reduction rules and transition invariants to avoid calculating the whole state space [37]. *Relaxed soundness does not necessary identify all deadlocks and lack of synchronization.*

Interactive Verification. In [38,11] the authors introduce an interactive approach for the verification of EPCs. In a first step, reduction rules are automatically applied on the original EPC. Then, the user has to specify the meaningful combinations of start events that can initiate the process. After that, the EPC is translated to a Workflow net and the state space is generated. Then, the user decides for each final state whether it is intended or not. Given these pieces of information, the EPC is classified as correct if the Workflow net is sound, as maybe correct if the net is relaxed sound, and incorrect otherwise. The approach is applied to the procurement module of the SAP Reference Model. The verification is facilitated by an analysis plug-in for ProM [11]. *The interactive approach can identify all those deadlocks and lack of synchronization that occur for all start combinations.*

EPC Soundness. Since EPCs may have multiple start and end events, the original soundness definition for workflow nets cannot be directly used for them. Therefore, the property of EPC soundness is proposed in [29]. It builds on the identification of a set of initial markings that covers all start events. The EPC soundness definition demands that (i) there exists such a non-empty set of initial markings, and (ii) that for each initial marking in this set proper completion is guaranteed. Furthermore, (iii) there must be a set of final markings reachable from some of these initial markings such that there exists at least one final marking in which a particular end arc holds a token. If that is fulfilled, proper completion is guaranteed for a set of initial markings that cover all start arcs. The EPC soundness property is stricter than the relaxed soundness criterion since it requires a guarantee of proper completion. Therefore, using it might reveal more errors. EPC soundness is used in another verification study of the SAP Reference Model [13]. The authors use a two-step approach using first reduction rules and then, if necessary, state space calculation. *EPC soundness can identify all those deadlocks* *and lack of synchronization that occur for all start combinations in which a particular start event participates.*

Decomposition. The approach reported in [14] builds on the decomposition of a workflow graph (essentially a free-choice Workflow net) into single-entry-single-exit (SESE) components. Such components can be calculated in linear time using depth-first search techniques [39]. The authors identify some heuristics for sound and for unsound components. In essence, these heuristics match EPC reduction rules as described in [30]. For their study, the authors use an implementation that reads process models in the format of the IBM WebSphere Business Modeler. The verification is conducted for a sample of 340 workflow graphs. *The decomposition approach identifies all those deadlocks and lack of synchronization that are defined in the heuristics.*

Reduction. Another heuristic for checking the correctness of process models is used in [15]. The authors identify reduction rules for structured EPC models and heuristics to correct simple connector mismatch errors. The verification is performed for 285 EPCs from student projects, theses, text books, and scientific papers. *This approach identifies all those deadlocks and lack of synchronization that can be traced back to variants of unstructuredness.*

The different approaches have in common that they use reduction and decomposition techniques to address the state explosion problem. The performance of these approaches partially comes at the cost of precision, for instance, fast decomposition does not reveal all errors. While there is a complete set of reduction rules for free-choice Petri nets [40], none of the techniques used in the studies is complete. Therefore, reduction rules and state space analysis are often combined to balance performance and completeness [13,11]. In practice, reduction is at least that powerful that the reduced process models can be analyzed with state space techniques [13].

3 Verification Results

In this section we discuss the results from the five verification studies. The samples used in these studies include:

1. *SAP Reference Model:* The development of the SAP reference model started in 1992 [41, p.VII] and continued until version 4.6 of SAP R/3, released in 2000 [42]. It includes 604 EPCs for documentation of the system.
2. *Service Model:* This collection of EPCs stems from a German process reengineering project in the service sector, conducted with academic supervision. The models that were defined in this project include 381 EPCs.
3. *Finance Model:* This collection contains the EPCs of a process documentation project in the Austrian financial industry. It includes 935 EPCs.
4. *Consulting Model:* This collection covers a total of 83 EPCs from three different consulting companies. The models are mainly used as reference models to support consulting activities of the companies.

Table 1. Verification Studies

Sample	Ref.	Criterion	Models	Error Models	Error Rate	Average Arcs	Average Time (sec)
SAP RM	[12]	Relaxed Sound	604	34	5.6%	21	46.6
SAP Procurement	[11]	Interactive	40	4	10.0%		
SAP RM	[13]	EPC Sound	604	126	20.9%	21	1.8
Services	[13]	EPC Sound	381	37	9.7%		
Finance	[13]	EPC Sound	935	31	3.3%		
Consulting	[13]	EPC Sound	83	21	25.3%		
BPM Books	[13]	EPC Sound	113	25	21.4%		
Workflow 1	[14]	Decomposition	140	0	0.0%	67	0.06
Workflow 2	[14]	Decomposition	200	24	12.0%	126	0.08
Academia	[15]	Reduction	285	107	37.5%		

5. *BPM Books:* This sample is built from four German textbooks on EPC process modeling [43,44,45,46] and includes 113 EPCs.
6. *Workflow 1 and Workflow 2:* These samples include IBM workflow models from industry projects [14]. The first includes 140 workflows, the second 200.
7. *Academia:* This collection includes documentation EPCs from student and industry projects from the University of Leipzig [15].

The different background of the samples highlights limitations of doing a comparison. Most notably, samples have been created for different purposes and using different languages. We would assume that workflow models were more rigorously defined and that restrictions in the IBM workflow language (one start, one end node) might provide less opportunities for errors. There are indeed low error rates with the workflow samples as Table 1 reveals. This table shows the number of models, the number of models that had errors, and the error rate for all samples. For some samples we have also indications of size (average number of arcs) and average processing time. We discuss the table below. In particular, we consider three aspects. Section 3.1 compares figures on the performance of the verification algorithm for the studies that report them. Section 3.2 analyzes the error rates of the different samples. Finally, Section 3.3 investigates explanations for the variation in error rates across the samples.

3.1 Verification Performance

Information on the performance of verification is only reported in a few studies. Table 1 summarizes the average number of arcs and the average processing time per model in seconds. The analyses reported in [12] and [13] operate on the same sample, therefore they can be easily compared. The EPC soundness verification using reduction rules is about 20 times faster than the relaxed soundness analysis using WofYAWL. Still, it needs to be mentioned that the WofYAWL processing time includes the conversion of an EPC to a YAWL model by a command-line tool and the internal conversion of a YAWL net into a Petri net in

WofYAWL. Without further information it is difficult to estimate which share the conversion contributes to overall processing. Still, it is reasonable that the simple element-wise conversion would not be too expensive. The slow processing might be explained by the inherent complexity of invariant calculation. WofYAWL uses invariant detection algorithms [47] that are exponential in space in the worst case. In contrast to that, the approach based on reduction rules benefits from reducing the models quite quickly. As reported in [30] that approach eliminates on average more than 91% of the model elements. Furthermore, it has a polynomial complexity since it uses reduction rules. In contrast to that, the approach of [14], that builds on calculating the program structure tree, is linear in complexity, see [39]. This fact is clearly visible in the relative performance: while the SAP reference model is analyzed by reduction in $1.8/21 = 0.09$ seconds per edge, the larger library 2 sample of [14] only requires $0.08/126 = 0.0006$ seconds per edge. This is about 150 times faster. Such performance figures clearly demonstrate the feasibility of improving performance of process model verification in practice.

3.2 Average Error Rates

The average error rates of the different samples in Table 1 give a good indication of how many errors can be expected in process model collections in practice. It must be mentioned that none of the verification approaches is complete. Accordingly, all figures have to be interpreted as lower bounds as non of the applied techniques guarantees all errors to be detected. Furthermore, the different approaches differ in precision: for example, the EPC soundness verification points to 126 errors in the SAP reference model while the relaxed soundness analysis only finds 34 problematic processes. Most of the approaches find a considerable amount of errors in the different samples: Library 1 has the lowest rate of 0%, followed by the Finance sample (3.3%) up to 37.5% in the Academia sample used in [15]. Altogether, from the 2741 distinct models of the studies 371 are not correct. This yields an error rate of 13.53%. The large variation in error rates raises the question why different samples are less error-prone. In the next section we discuss some of the factors.

3.3 Variation in Error Rates

Based on cognitive considerations like bounded rationality [48] and limited information processing capabilities of humans [49], it has been hypothesized that the understanding of a process model has a significant impact on error probability [30]. In the process of constructing a model, understanding relates to two factors: the structure of the model and the modeling capabilities of the modeler. These two factors have been investigated in prior research, namely [12,13,50,51,52]. In the following we summarize the findings.

Different sets of structural metrics of process models have been analyzed for their potential to explain and predict error probability [12,13,52]. The authors use logistic regression as a statistical tool. This way, it can be estimated how the value of a particular metric influences error probability. The relaxed soundness study [12] uses a set of simple count metrics (number of functions, events, and connectors of different type) and the SAP reference model as a sample for the estimation. It is shown that a substantial part of the variation can be explained as measured by the Nagelkerke R^2 value of about 30% as a coefficient of determination. In [13] the authors extend this work by using a larger sample of about 2000 models (SAP reference model plus the Service, Finance, and Consulting sample) and a set of more sophisticated ratio metrics including structuredness, connector heterogeneity, and depth. The resulting model explains more than 90% of the variation as measured by the Nagelkerke R^2 value. Accordingly, the fact whether a process model contains errors seems to be largely influenced by its size, its degree of structuredness and similar metrics.

The experiments reported in [50] and [51] investigate both model structure and competence of the model reader as determinants for model understanding. The questionnaires of these experiments use different process models and ask the participants questions that reveal whether the content of the model is interpreted correctly. The first experiment is conducted on paper with a sample of 73 graduate students from Eindhoven University of Technology, the University of Madeira, and the Vienna University of Economics and Business Administration [50]. It shows that the average degree of connectors is negatively correlated to understanding and that those students being trained in Petri nets concepts perform better. The second experiment was conducted as an online questionnaire. It was filled out by 46 respondents [51]. The result show a strong correlation between theoretical process modeling knowledge and understanding of the different models. From the structural metrics the degree of separability,[1] i.e. how easy can the model be separated in two disconnected parts, had a strong correlation with understanding. Apparently, the simpler models were understood better. It is interesting to note that in both experiments participants tended to overestimate their ability to correctly read the models. This confirms earlier observations [49].

These findings suggest that errors do not occur arbitrarily, but in specific constellations. This has several implications as the following section discusses.

4 Implications

In this section we discuss implications of the verification studies. In particular, Section 4.1 identifies implications for the way business process modeling is conducted in practice. Then, Section 4.2 highlights some facts that tool vendors should consider. Finally, Section 4.3 points to some directions of future research.

4.1 Implications for Process Modeling

The results of the different studies confirm that process models in practice suffer from quality problems. Furthermore, the studies suggest that industry process

[1] This is the ratio of cut vertices to all nodes of the process model.

Table 2. Seven Process Modeling Guidelines [16]

G1	Use as few elements in the model as possible
G2	Minimize the routing paths per element
G3	Use one start and one end event
G4	Model as structured as possible
G5	Avoid OR routing elements
G6	Use verb-object activity labels
G7	Decompose a model with more than 50 elements

model collections are likely to have error rates of 10% to 20%. Clearly, there are differences in error rates, and there are different structural metrics that are closely connected with error probability. Based on these connections and on work on activity labeling [53], Mendling, Reijers, and Van der Aalst propose a set of seven process modeling guidelines (7PMG) that are supposed to direct the modeler to creating understandable models that are less prone to errors [16].

Table 2 summarizes the 7PMG guidelines. Each of them is supported by empirical insight into the connection of structural metrics and errors or understanding. The size of the model has undesirable effects on understandability and likelihood of errors [50,12,13]. Therefore, **G1** recommends to use as few elements as possible. **G2** suggests to minimize the routing paths per element. The higher the degree of elements in the process model the harder it becomes to understand the model [50,13]. **G3** demands to use one start and one end event, since the number of start and end events is positively connected with an increase in error probability [13]. Following **G4**, models should be structured as much as possible. Unstructured models tend to have more errors and are understood less well [13,15,52,50]. **G5** suggests to avoid OR routing elements, since models that have only AND and XOR connectors are less error-prone [13]. **G6** recommends using the verb-object labeling style because it is less ambiguous compared to other styles [53] Finally, according to **G7** models should be decomposed if they have more than 50 elements.

These guidelines have to be considered as directions for achieving a good verification result. They are founded on the insight that there are alternative ways of expressing the same behavior in a process model. Clearly, a rework of a process model should not affect the validation goal. Creating small models might help to achieve several guidelines, for instance **G1**, but it reduces the precision of the model. Therefore, the guidelines should be followed if the validity is not reduced.

4.2 Implications for Tool Vendors

The amount of errors in the model samples from practice emphasizes the importance of verification. Most of the samples have error rates between 10% and 20%. While verification has been discussed for some time, the studies demonstrate that different approaches can handle large sets of several hundred business process models on a common desktop computer. In particular, the performance of the decomposition approach is that good (linear time) that a continuous verification would be possible every time the designer modifies parts of a model.

While the verification techniques are apparently mature enough to deal with large models from practice, there seems to be too little attention paid to verification issues by tool vendors. Indeed, tool vendors should be interested in these techniques since the lack of respective features has a negative impact on the productivity of the business process modeling exercise: models cannot be reused for system development, business users cannot interpret the models properly, and conclusions can hardly be drawn from the models regarding process performance. Building on validation and verification features, the tool vendors can easily provide a greater benefit to their customers and help to improve the process of designing business process models.

4.3 Implications for Future Research

The results of the five studies have several implications of future research. In this section we focus on some of them, in particular, error explanation and auto-repair, derivation of behavior-equivalent models, novel verification techniques, and concepts for process model quality assurance.

Explanation. The Petri net analyzer Woflan already emphasizes the need for redesign assistance in case the verification of a model fails [35]. The approaches of the studies we considered address this issue as well. The reduction rule approach for EPC soundness generates SVG graphics of the process model with errors being highlighted (cf. Figure 2). The work on structuredness proposes an auto-repair for certain error patterns like connector mismatch in structured blocks [15]. The problem with such an approach is that there are usually at least two options to repair a model: changing the split behavior or changing the join behavior. Which of them is applicable must be decided by the domain expert who knows the real-world process. In a more sophisticated subgraph structure errors can be caused by the combination of multiple splits and multiple joins. Recommendations for repairing such structures would be a valuable contribution of future research.

Behavior-Equivalent Models. Some work has been done on the creation of behavior-equivalent process models. Most notable is the work on Petri net synthesis from a labeled transition system [54,55]. This work has been used to eliminate OR-joins from process models [56]. Further research discusses the untangling of unstructured loops and the derivation of structured BPEL from arbitrary Petri nets [57,58]. A desirable contribution would be to change a process model in such a way that the behavior is kept, but the structure optimized, e.g. along the lines of the seven process modeling guidelines. Designers would highly appreciate such a feature that would contribute to more understandable models.

Verification Techniques. In the comparison we have seen the power of the program structure tree decomposition. To our best knowledge this technique has not yet been applied to the analysis of Petri nets. The EPC soundness study highlights the benefit of combining reduction and reachability graph analysis. A similar approach can be taken to increase the precision of the program structure tree verification.

Quality Assurance. Validation and verification techniques are discussed in different neighboring disciplined of process modeling, namely requirement engineering [59], knowledge-based systems [60], simulation [61], or conceptual modeling [62]. Some validation and verification techniques for process models are discussed in [10,9]. Yet, an overarching framework and a systematic analysis of how suitable of such techniques from other areas are for process modeling is missing.

Despite these technical challenges, the analysis of process models is a field where commercial tool support is lagging behind academic concepts. There is a need for a closer collaboration between academia and industry in order to make verification techniques available in a broader range of tools.

5 Conclusions

In this paper we discussed verification of process models and findings from empirical studies on it. Most of the approaches use some kind of decomposition technique, or a combination of reduction and reachability analysis. The best performance shows the linear time decomposition approach based on the program structure tree. The precision of the different approaches is difficult to compare since different model collections are used and a benchmark verification sample is not available. The comparison of the studies reveals that most of the samples have error rates of 10% to 20%. Some of the studies have established a connection between error probability and process model metrics, as well as between model understanding and both metrics and modeling competence of the model reader. These findings have implications for the way process models are constructed. In this context, we discussed the seven process modeling guidelines. The high error rates are a clear signal to tool vendors to add verification features to their tools. Finally, there are still several challenges for future work. In particular, we identified a need for further research into error repair, derivation of behavior-equivalent models, combined verification techniques, and quality assurance for process models.

References

1. Lindland, O., Sindre, G., Sølvberg, A.: Understanding quality in conceptual modeling. IEEE Software 11, 42–49 (1994)
2. Becker, J., Rosemann, M., Uthmann, C.: Guidelines of Business Process Modeling. In: van der Aalst, W., Desel, J., Oberweis, A. (eds.) Business Process Management. Models, Techniques, and Empirical Studies, pp. 30–49. Springer, Berlin (2000)
3. Krogstie, J., Sindre, G., Jørgensen, H.: Process models representing knowledge for action: a revised quality framework. Europ. J. of Inf. Systems 15, 91–102 (2006)

4. Moody, D.: Theoretical and practical issues in evaluating the quality of conceptual models: current state and future directions. Data & Knowl. Eng. 55, 243–276 (2005)
5. Davies, I., Green, P., Rosemann, M., Indulska, M., Gallo, S.: How do practitioners use conceptual modeling in practice? Data & Knowl. Eng. 58, 358–380 (2006)
6. Boehm, B.: Software Engineering Economics. Prentice-Hall, Englewood Cliffs (1981)
7. Wand, Y., Weber, R.: Research Commentary: Information Systems and Conceptual Modeling - A Research Agenda. Information Systems Research 13, 363–376 (2002)
8. Rosemann, M.: Potential pitfalls of process modeling: part a. Business Process Management Journal 12, 249–254 (2006)
9. Philippi, S., Hill, H.: Communication support for systems engineering - process modelling and animation with april. Journal of Sys. & Softw. 80, 1305–1316 (2007)
10. van Hee, K., Sidorova, N., Somers, L., Voorhoeve, M.: Consistency in model integration. Data & Knowledge Engineering 56 (2006)
11. van Dongen, B., Vullers-Jansen, M., Verbeek, H., van der Aalst, W.: Verification of the sap reference models using epc reduction, state-space analysis, and invariants. Computers in Industry 58, 578–601 (2007)
12. Mendling, J., Verbeek, H., van Dongen, B., van der Aalst, W., Neumann, G.: Detection and Prediction of Errors in EPCs of the SAP Reference Model. Data & Knowl. Eng. 64, 312–329 (2008)
13. Mendling, J., Neumann, G., van der Aalst, W.: Understanding the occurrence of errors in process models based on metrics. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 113–130. Springer, Heidelberg (2007)
14. 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)
15. Gruhn, V., Laue, R.: What business process modelers can learn from programmers. Science of Computer Programming 65, 4–13 (2007)
16. Mendling, J., Reijers, H., van der Aalst, W.: Seven Process Modeling Guidelines (7PMG). In: Qut eprint (2008)
17. Adrion, W., Branstad, M., Cherniavsky, J.: Validation, verification, and testing of computer software. ACM Computing Surveys 14, 159–192 (1982)
18. IEEE: IEEE Std 610.12-1990 IEEE Standard Glossary of Software Engineering Terminology. IEEE Computer Society Press (1990)
19. Valmari, A.: The state explosion problem. In: Reisig, W., Rozenberg, G. (eds.) APN 1998. LNCS, vol. 1491, pp. 429–528. Springer, Heidelberg (1998)
20. Hoppenbrouwers, S., Proper, H., van der Weide, T.: A fundamental view on the process of conceptual modeling. In: Delcambre, L.M.L., Kop, C., Mayr, H.C., Mylopoulos, J., Pastor, Ó. (eds.) ER 2005. LNCS, vol. 3716, pp. 128–143. Springer, Heidelberg (2005)
21. Boehm, B.W.: Software engineering; R & D trends and defense needs. In: Research Directions in Software Technology. MIT Press, Cambridge (1979)
22. Sommerville, I.: Software Engineering, 6th edn. Addison-Wesley, Reading (2001)
23. van der Aalst, W.: Verification of Workflow Nets. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 407–426. Springer, Heidelberg (1997)
24. 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)

25. Dehnert, J., Zimmermann, A.: On the suitability of correctness criteria for business process models. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 386–391. Springer, Heidelberg (2005)
26. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer, Heidelberg (2007)
27. Pnueli, A.: The Temporal Logic of Programs. In: Proceedings of the 18th IEEE Annual Symposium on the Foundations of Computer Science, pp. 46–57 (1977)
28. Kindler, E.: On the semantics of EPCs: Resolving the vicious circle. Data & Knowledge Engineering 56, 23–40 (2006)
29. Mendling, J., van der Aalst, W.: 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)
30. Mendling, J.: Detection and Prediction of Errors in EPC Business Process Models. PhD thesis, Vienna University of Economics and Business Administration (2007)
31. Keller, G., Nüttgens, M., Scheer, A.W.: Semantische Prozessmodellierung auf der Grundlage "Ereignisgesteuerter Prozessketten (EPK)". Heft 89, Institut für Wirtschaftsinformatik, Saarbrücken, Germany (1992)
32. Murata, T.: Petri Nets: Properties, Analysis and Applications. Proceedings of the IEEE 77, 541–580 (1989)
33. Desel, J., Esparza, J.: Free Choice Petri Nets. Cambridge Tracts in Theoretical Computer Science, vol. 40. Cambridge Univ. Press, Cambridge (1995)
34. Verbeek, H., van der Aalst, W.: Woflan 2.0: A Petri-net-based Workflow Diagnosis Tool. In: Nielsen, M., Simpson, D. (eds.) ICATPN 2000. LNCS, vol. 1825, pp. 475–484. Springer, Heidelberg (2000)
35. Verbeek, H., Basten, T., van der Aalst, W.: Diagnosing Workflow Processes using Woflan. The Computer Journal 44, 246–279 (2001)
36. Mendling, J., Moser, M., Neumann, G., Verbeek, H., van Dongen, B., van der Aalst, W.: 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)
37. Verbeek, H., van der Aalst, W., ter Hofstede, A.: Verifying workflows with cancellation regions and or-joins: An approach based on relaxed soundness and invariants. The Computer Journal 50, 294–314 (2007)
38. van Dongen, B., Jansen-Vullers, M.H.: Verification of SAP reference models. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 464–469. Springer, Heidelberg (2005)
39. Johnson, R., Pearson, D., Pingali, K.: The program structure tree: Computing control regions in linear time. In: Proceedings of the ACM SIGPLAN'94 Conference on Programming Language Design and Implementation (PLDI). SIGPLAN Notices, vol. 29(6), pp. 171–185 (1994)
40. Esparza, J.: Reduction and synthesis of live and bounded free choice petri nets. Information and Computation 114, 50–87 (1994)
41. Keller, G., Teufel, T.: SAP(R) R/3 Process Oriented Implementation: Iterative Process Prototyping. Addison-Wesley, Reading (1998)
42. Mendling, J., van der Aalst, W., van Dongen, B., Verbeek, H.: Referenzmodell: Sand im Getriebe - Webfehler. iX - Magazin für Professionelle Informationstechnik (in German), 131–133 (2006)
43. Becker, J., Schütte, R.: Handelsinformationssysteme. 2nd edn. Moderne Industrie, Landsberg/Lech (2004)

44. Scheer, A.W.: Wirtschaftsinformatik: Referenzmodelle für industrielle Geschäftsprozesse, 2nd edn. Springer, Heidelberg (1998)
45. Seidlmeier, H.: Prozessmodellierung mit ARIS. Vieweg Verlag (2002)
46. Staud, J.: Geschäftsprozessanalyse: Ereignisgesteuerte Prozessketten und Objektorientierte Geschäftsprozessmodellierung für Betriebswirtschaftliche Standardsoftware, 3rd edn. Springer, Heidelberg (2006)
47. Colom, J., Silva, M.: Convex geometry and semiflows in P/T nets, A comparative study of algorithms for computation of minimal P-semiflows. In: Rozenberg, G. (ed.) APN 1990. LNCS, vol. 483, pp. 79–112. Springer, Heidelberg (1991)
48. Simon, H.: Sciences of the Artificial, 3rd edn. MIT Press, Cambridge (1996)
49. Burton-Jones, A., Meso, P.: How Good are these UML Diagrams? An Empirical Test of the Wand and Weber Good Decomposition Model. In: Applegate, L., Galliers, R., DeGross, J. (eds.) Proceedings of ICIS, pp. 101–114 (2002)
50. Mendling, J., Reijers, H., Cardoso, J.: What makes process models understandable? In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 48–63. Springer, Heidelberg (2007)
51. Mendling, J., Strembeck, M.: Influence factors of understanding business process models. In: Abramowicz, W., Fensel, D. (eds.) Proc. of the 11th International Conference on Business Information Systems (BIS 2008). LNBIP, vol. 7, pp. 142–153 (2008)
52. Laue, R., Mendling, J.: The impact of structuredness on error probability of process models. In: Kaschek, R., Kop, C., Steinberger, C., Fliedl, G. (eds.) UNISCON 2008. LNBIP, vol. 5, pp. 585–590. Springer, Heidelberg (2008)
53. Mendling, J., Reijers, H.: How to define activity labels for business process models? In: Oberweis, A., Hesse, W. (eds.) Proc. of the Third AIS SIGSAND Europe 2008. LNI (2008)
54. Ehrenfeucht, A., Rozenberg, G.: Partial (Set) 2-Structures - Part 1 and Part 2. Acta Informatica 27, 315–368 (1989)
55. Cortadella, J., Kishinevsky, M., Lavagno, L., Yakovlev, A.: Deriving petri nets from finite transition systems. IEEE Transactions on Computers 47, 859–882 (1998)
56. Mendling, J., van Dongen, B., van der Aalst, W.: Getting Rid of the OR-Join in Business Process Models. In: Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007), pp. 3–14 (2007)
57. Zhao, W., Hauser, R., Bhattacharya, K., Bryant, B., Cao, F.: Compiling business processes: untangling unstructured loops in irreducible flow graphs. Int. Journal of Web and Grid Services 2, 68–91 (2006)
58. van der Aalst, W., Lassen, K.: Translating unstructured workflow processes to readable BPEL: Theory and implementation. Inf.& Softw. T. 50, 131–159 (2008)
59. Nuseibeh, B., Easterbrook, S.: Requirements engineering: a roadmap, pp. 35–46 (2000)
60. Tsai, W., Vishnuvajjala, R.: Verification and Validation of Knowledge-Based Systems. IEEE Transactions on Knowledge and Data Engineering 11, 202–212 (1999)
61. Sargent, R.: Verification and validation of simulation models, pp. 130–143 (2005)
62. Frederiks, P., van der Weide, T.: Information modeling: The process and the required competencies of its participants. Data & Knowl. Eng. 58, 4–20 (2006)

Process Mining: Overview and Outlook of Petri Net Discovery Algorithms

B.F. van Dongen[1], A.K. Alves de Medeiros[1], and L. Wen[1,2]

[1] Eindhoven University of Technology,
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
{b.f.v.dongen,a.k.medeiros}@tue.nl
[2] Tsinghua University, Beijing 100084, P.R. China
wenlj00@mails.tsinghua.edu.cn

Abstract. Within the research domain of process mining, process discovery aims at constructing a process model as an abstract representation of an event log. The goal is to build a model (e.g., a Petri net) that provides insight into the behavior captured in the log. The theory of regions can be used to transform a state-based model or a set of words into a Petri net that exactly mimics the behavior given as input. Recently several papers appeared on the application of the theory of regions for process discovery.

This paper provides an overview of different Petri net based discovery algorithms from both the area of process mining and the theory of regions. The overview encompasses five categories of algorithms, for which common assumptions and problems are indicated. Furthermore, based on the shortcomings of the algorithms in each category, a set of directions for future research in the process discovery area is discussed.

1 Introduction

Many researchers have investigated the process discovery problem, i.e. the problem of how to synthesize a process model from event logs. In this paper, we provide an overview of the existing algorithms in the field of process discovery, where we focus our attention on algorithms tailored towards constructing Petri nets instead of other process models such as EPCs, YAWL models or even Markov Chains. We refer to [2,4] and the www.processmining.org website for a more complete overview of the whole research domain.

The research area of process mining focusses on extracting information about processes by examining event logs. Practical experience has shown that typical information recorded in these logs includes information about which activities are performed, at what time, by whom and in the context of which case (i.e., process instance). By explicitly using the case context, process discovery algorithms are capable of constructing process models (i.e. Petri nets in our case) that accurately describe the process, as it takes place in real life.

As an illustration, consider the log in Table 1. This event log contains six instances (or cases) of a process for attending to a one-day conference. The events are the "tasks" that have been performed by the participants of this conference.

Table 1. An event log

Id	Process Instance
1	Start, Get Ready, Travel by Train, Conference Starts, Ask Question, Join Guided Tour, Join Dinner, Go Home, Travel by Train, End
2	Start, Get Ready, Travel by Car, Conference Starts, Give a Talk, Ask Question, Ask Question, Ask Question, Ask Question, Join Guided Tour, Join Dinner, Go Home, Pay for Parking, Travel by Car, End
3	Start, Get Ready, Travel by Train, Conference Starts, Give a Talk, Ask Question, Join Guided Tour, Join Dinner, Go Home, Travel by Train, End
4	Start, Get Ready, Travel by Car, Conference Starts, Give a Talk, Join Guided Tour, Join Dinner, Go Home, Pay for Parking, Travel by Car, End
5	Start, Get Ready, Travel by Train, Conference Starts, Give a Talk, Join Guided Tour, Join Dinner, Go Home, Travel by Train, End
6	Start, Get Ready, Travel by Car, Conference Starts, Join Guided Tour, Join Dinner, Go Home, Pay for Parking, Travel by Car, End

By looking at these instances, one could remark that (i) participants use the same means of transportation to go to the conference and to come back home, (ii) not all participants give talks, and (iii) some participants have not asked questions, although some of them have asked multiple questions, etc. Based on this log and these observations, process mining tools could be used to retrieve a Petri net like the one in Figure 1. Such a Petri net can then be used for further analysis, for example by focussing on the deviations between the recorded and the intended behavior.

A problem similar to process discovery arises in areas such as hardware design and control of manufacturing systems. There, the so called *theory of regions* is used to construct a Petri net from a behavioral specification (e.g., a language or a state space), such that the behavior of this net corresponds with the specified behavior (if such a net exists). The general question answered by the theory of regions is: *Given the specified behavior of a system, what is the Petri net that represents this behavior?*.

Mainly, two types of region theory can be distinguished, namely state-based region theory [8, 9, 11, 12, 18] and language-based region theory [13, 7, 23]. The state-based theory of regions focusses on the synthesis of Petri nets from state-based models, where the state space of the Petri net is bisimilar to the given state-based model. The language-based region theory, considers a language over a finite alphabet as a behavioral specification. Using the notion of regions, a Petri net is constructed, such that all words in the language are firing sequences in that Petri net.

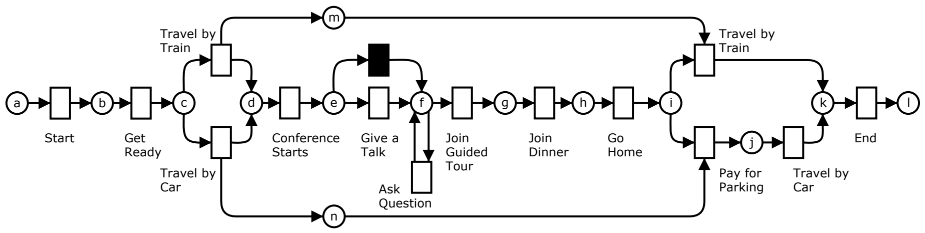


Fig. 1. Example of a Petri net that could be mined for the event log in Table 1

In this paper, we discuss the wide variety of currently available process discovery algorithms in Section 3 and we show their differences based on the several perspectives explained in Section 2. Additionally, we discuss the existing tool support for process discovery techniques in Section 4 and future directions for developments in this area in Section 5. Our conclusions are presented in Section 6.

2 Comparison

One of the aims of this paper is to present an overview of the current process discovery techniques that can mine Petri nets. Since this overview also contains a comparison of these techniques, this section explains which factors have been taken into account during this comparison. Understanding these factors helps in comprehending the analysis of the current status (cf. Section 3) and the ideas for future directions (cf. Section 5). The factors to be considered during the evaluation are *supported control-flow constructs*, *assumptions about log completeness*, *supported levels of abstraction* and *underfitting/overfitting of discovered models.*[1] These factors are respectively explained in Subsections 2.1 to 2.4.

2.1 Supported Control-Flow Constructs

Process discovery algorithms mine a process model describing the relations between tasks in an event log. From event logs, one can find out information about which tasks belong to which process instances, the time at which tasks are executed, the originator of tasks, etc. Therefore, the mined process model is an *objective picture* that depicts possible flows that were followed by the cases in the log (assuming that the events were correctly logged). Because the flow of tasks is to be portrayed, process discovery techniques need to support the correct mining of the common control-flow constructs that appear in process models. These constructs are *sequences (seq), parallelism (par), choices (cho), loops (lo), non-free-choice (nfc), invisible tasks (it) and duplicate tasks (dt)* [4]. Figure 1 shows an example of a Petri net with all these constructs.

2.2 Completeness of Information in Logs

Process discovery algorithms mine process models based on data contained in event logs. Therefore, the notion of *completeness* is very important. Note that, like in any data mining or machine learning context, one cannot assume to have

[1] Runtime is not included in our comparison for several reasons. First, we feel that it cannot be established fairly for all algorithms. Some have a runtime linear or exponential in the size of the log, others exponential in the number of activities to which the log refers and so on. Second, process discovery is a technique typically applied to single logs in case-study settings. In these settings, fast lead times are rarely essential, hence it does not matter how much time algorithms take. Finally, we feel that algorithms should be selected for other properties than their runtime.

seen all possibilities in the "training material" (i.e., the event log at hand). Therefore, all algorithms presented in this overview make some assumptions on completeness. For some, the information about which tasks directly succeed or precede others is sufficient. For others, the frequency of tasks (or events) is also taken into account. Furthermore, some assume the log to contain all the possible behavior (what is rather unrealistic for real-life settings). In the following we elaborate on the notions of log completeness that are used by the algorithms in Section 3.

The most stringent notion of completeness is called *global completeness* (GC). If an algorithm assumes global completeness, then it assumes that the log shows all possible behavior of the process. However, often information is not complete, or cannot even be complete, which is the case in the model of Figure 1, where the transition "Ask Question" can be executed an infinite number of times. Global completeness is a typical theoretical notion. In practice, the probability of a log being globally complete can be assumed to be 0, e.g. the probability of two patients following the same path through a hospital is 0.

Most algorithms discussed in this paper use different notions of completeness. An notion that is used often is called "completeness of direct succession" (DS) where it is assumed that "if two transitions can follow each other directly, then this has occurred at least once in the log". This can be extended with a similar notion for loops of length 2, where it is assumed that "if a transition can follow itself with one transition in between, then this has occurred at least once" (DS+), or by considering long-term successions as well (DS++).

A less restrictive variant of this is the completeness of causal dependencies, which states that "if two transitions are causally dependent, then they succeed each other directly at least once" (CD).

Finally, the algorithms that use heuristic approaches typically have completeness assumptions relating to the frequencies of succession. For one, if there is a causal dependency between two transitions, then the direct successions of these transitions should reflect that, which is called "event significance" (ES). Another notion is "trace significance" (TS) which states that the most relevant behavior in terms of cases appears most frequently in the log.

2.3 Abstraction Levels

All techniques presented in Section 3 assume the events within the log to be at the same level of abstraction. Mostly this is expressed by the fact that each transition in the log corresponds to exactly one event class, e.g. the start of activity A, or the completion of activity B. Nonetheless, there are some exceptions which is why we feel it is important to consider this perspective.

Most algorithms assume a direct correspondence between the events-classes in the log and the transitions in the model, i.e. each transition refers to a single event class `(1:1)`.[2] Some algorithms assume knowledge about the start and end

[2] We denote the abstraction level by `(x:y)` to indicate that (i) each transition corresponds to `y` event classes in a log and (ii) each event class corresponds to `x` transitions.

of activities and therefore relate each transition to two event classes `(1:2)`, i.e. the assumption is made that transition executions are not atomic.

Sometimes, assuming that each event class is represented by a single transition is not a good idea. Consider again Figure 1. If you travel by train on the way to the conference, then you travel by train back. These two transitions labelled "Travel by Train" would be logged as the same event class, hence the resulting Petri net should reflect this by a `(1..*:1)` correspondence (each event corresponds to one or more transitions). If an algorithm sets a global maximum on the number of transitions per event, we use the `(1..n:1)` notation.

Transitions, such as the one skipping "Give a Talk" in Figure 1, do not refer to any event in the log and some algorithms introduce those as well, indicated by `(1:0..1)`.

2.4 Underfitting vs. Overfitting

Finally, we investigate the balance between underfitting and overfitting [16, 3] for all algorithms. Let L be a log and M be a model.

- M *is overfitting* L if M does not generalize and is sensitive to particularities in L. In an extreme case, M could merely be a representation of the log without any inference, e.g. a tree-like Petri net where each transition corresponds to 1 event (and not 1 event-class). A mining algorithm is producing overfitting models if the removal or addition of a small percentage of the process instances in L would lead to a remarkably different model.
- M *is underfitting* L if M allows for "too much behavior" that is not supported by L. Two simple examples of underfitting models are (i) the model without places where each transition corresponds to 1 event class and (ii) the "flower-model", where each transition both consumes a token from and produces a token in a single, initially marked, place.

To illustrate the problem between overfitting and underfitting, consider some process in a hospital. When observing such a process over a period of years it is very likely that many patients follow a "unique process", i.e., seen from the viewpoint of a particular patient it is very unlikely that there is another patient that has exactly the same sequence of events. Therefore, it does not make sense to assume that the event log contains all possible paths a particular case can take. In fact, it is very likely that the next patient will have a sequence of events different from all earlier patients. Therefore, one cannot assume that an event log is "complete" and one is forced to generalize to avoid overfitting. However, at the same time underfitting ("anything is possible") should be avoided.

In our discussion of Petri net discovery algorithms, we will not quantify the notion of under/overfitting. Instead, we provide a qualitative insight into the position of each algorithm.

3 Current Status

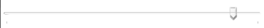
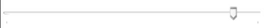
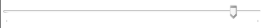
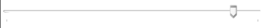
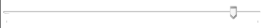
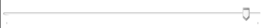
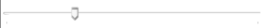
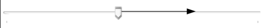
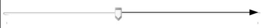
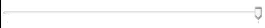
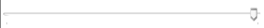
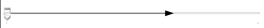
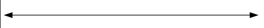
In this section, we present several categories of mining algorithms. The categories encompass the algorithms listed in Table 2. For each of these categories, we explain the assumptions on the log, with respect to completeness and we show which control flow constructs can be discovered. Furthermore, Table 2 shows the abstraction level of each algorithm and the balance between under- and overfitting. This balance is expressed using a slider representation. The position of the slider shows where the discovered model of the algorithm should be put when the default parameter values are used. If the algorithm uses search methods to improve its results, but can be stopped during the search, the arrow shows to which direction the mined model would generally move. Note that the state discovery algorithm can discover a Petri net at any point in the spectrum, depending on the parameters, but does not define default parameters. Therefore, we did not include the slider for that algorithm.

The following sections contain details about how current techniques to mine Petri nets from event logs work. The algorithms discussed in Subsections 3.1 and 3.2 construct Petri nets in three stages, namely *abstraction*, *induction* and *construction*. The algorithms in Subsection 3.3 take a more global approach and all the traces in the event log are considered when constructing a process model (i.e., these algorithms do not abstract from the log first). Subsection 3.4 discusses algorithms that are based on the language-based theory of regions and Subsection 3.5 presents an algorithm that uses state-based region theory.

3.1 Abstraction-Based Algorithms

This section will summarize the abstraction-based mining algorithms. These algorithms construct a net based on an abstraction of the log and most of them are derived from the α-algorithm [5], so we call them α-series algorithms. As a result, the mining procedures of these algorithms are very similar.

Table 2. Comparison of Petri net discovery algorithms

Algorithm	Complete-	Constructs							Abstraction	Fitness
	ness	seq	par	cho	lo	nfc	it	dt		underfitting overfitting
α	DS	+	+	+	+/-	-	-	-	1:1	
$\alpha+$	DS+	+	+	+	+	-	-	-	1:1	
tsinghua-α	CD	+	+	+	+	-	-	-	1:2	
$\alpha++$	DS++	+	+	+	+	+	-	-	1:1	
$\alpha\#$	DS+	+	+	+	+	-	+	-	1:0..1	
$\alpha*$	DS	+	+	+	+/-	-	-	+/-	1..n:1	
Heuristic Miner	ES	+	+	+	+	+/-	+	-	1:0..1	
Genetic Alg.	TS	+	+	+	+	+	+	-	1:0..1	
Dupl. GA	TS	+	+	+	+	+	+	+	1..n:0..1	
LangReg Basis	GC	+	+	+	+	+	-	-	1:1	
LangReg Sep	GC	+	+	+	+	+	-	+	1:1	
LangReg ILP	none	+	+	+	+	+	-	-	1:1	
State Discovery	none	+	+	+	+	+	+	+	1..*:0..*	

This mining procedure consists of three phases: the *abstraction phase*, the *induction phase* and the *construction phase*. In the abstraction phase, any event in each event trace in the event log is scanned and basic ordering relations between tasks are recorded. These relations contain information about the number of direct successions between tasks. For example, the basic ordering relations based on Table 1 show that activity "Start" is directly followed by "Get Ready", but that "Get Ready" is never directly followed by "Start".

In the induction phase, advanced ordering relations are induced from the basic ones and new tasks (e.g., invisible tasks and duplicate tasks) may be created from scratch. Advanced ordering relations for example show whether tasks are causally dependent, or in parallel. From our log in Table 1, the advanced ordering relations record that "Start" is causally followed by "Get Ready", but that "Travel by Car" and "Travel by Train" are in a choice relation (not in parallel, and not causally dependent).

In the construction phase, the final model is constructed from the advanced ordering relations according to some heuristic rules. The principle for adding places with arcs between tasks is that if there is a causal relation between any two tasks, there should be at least one place connecting them, e.g. place "b" in Figure 1 expresses the causal dependency between "Start" and "Get Ready".

All α-series algorithms make some assumptions about the given log. There are totally three most important assumptions. The first one is that there should be no noisy data in the log. In other words, no event is missing or redundant or in a wrong place. The second one is that the given log should be complete according to the basic ordering relations (like the *follows* relation). The last one is that the process model generating the log (we call it the potential model) should be expressed in Petri nets properly and must not contain some special constructs (i.e., short loops, non-free-choice constructs, invisible tasks, duplicate tasks and OR-split/join constructs). For different algorithms, the contents of the latter two assumptions may be a little different. Once the above assumptions are satisfied, the correctness of the construction phase can be proved theoretically.

The **α-algorithm** assumes that the potential model is a sound Structured Workflow Petri net (SWF-net) without short loops. SWF-nets are a subclass of workflow nets (WF-nets) in which the net structure explicitly shows its behavior. Consequently, in SWF-nets (i) choice and synchronization are not mixed, and (ii) if there is a synchronization, all of its preceding transitions will have fired [5]. Short loops are loops of length 1 or 2 in the Petri net. The model in Figure 1 is not an SWF-net, due to places "m" and "n" (non-free-choice), the black transition (invisible task), and the duplication of the "Travel by Train" transition. The "Ask Question" transition is an example of a short loop.

In [5], the authors show that given a log from an SWF-net that is complete with respect to the follows relation, the model used to generate the log can be discovered. In the abstraction phase, direct successions of events in the log are considered. Then, in the induction phase, causal dependencies and parallel dependencies between activities are induced, which are used in the construction phase to construct the Petri net.

The α-algorithm was the first of the abstraction-based algorithms and all other abstraction based algorithms are extensions thereof. These extensions aim at extending the class of nets to which the potential model can belong, while maintaining the property that: "if the log is a complete log generated by a model belonging to the given class, then that model can be rediscovered". Typically however, extending the class of potential models also puts extra demands on the completeness of information in the log.

The **$\boldsymbol{\alpha}^{+}$-algorithm** is the first extension of the α-algorithm. It adds support for short loops in sound SWF-nets [6]. With this extension, the restriction of not having short loops is lifted (e.g. in Figure 1 transition "ask question" is a short loop), however changes are made in all three phases.

In the abstraction phase, an extra demand is put on the information contained in the log. Specifically, the log should not only be complete with respect to the follows relation, but it should also explicitly show the occurrence of each length-two loop (i.e., if transitions t_1 and t_2 are in a length two loop, then both the subsequences "t_1, t_2, t_1" and "t_2, t_1, t_2" should appear at least once in the log). The induction phase is changed in such a way that activities involved in short loops are not considered to be in parallel, but to be causally dependent. In the construction phase, the process model without the transitions appearing in length-one loops is first constructed in a way very similar to the original α-algorithm. In a post-processing phase, length-one-loop tasks are inserted to the right position of the model.

The **tsinghua-$\boldsymbol{\alpha}$-algorithm** concentrates on event logs containing non-atomic events [28]. Each execution of a task t is associated with a pair of events referring to the start and completion of that task. In the constructed Petri net, transitions belong to such non-atomic tasks, but they can be decomposed into transitions representing the start and completion of a task.

The potential model is a sound SWF-nets not containing short loops on the atomic level. However, as tasks are not atomic, short-loops can exist in the constructed model. The abstraction phase is similar to the abstraction phase of the α-algorithm. However, in the induction phase, causal and parallel dependencies between tasks are inferred based on the overlap of tasks in time, i.e. if one task started before another completed then they are assumed to be in parallel. The construction phase is similar to that of the α^{+}-algorithm.

The **$\boldsymbol{\alpha}^{++}$-algorithm** aims at finding non-free-choice constructs [27]. As a result, the potential model is beyond a sound SWF-net, but not exactly defined (i.e., in contrast to the other abstraction-based algorithms, there is no given class of model that can always be rediscovered from a complete log). In Figure 1, places "m" and "n" control the choice after "Go Home". These are typical examples of places which can be discovered by this algorithm.

In the abstraction phase, relations are established between events that follow each other not directly, but on a larger distance. This distance is then used in the induction phase to derive indirect causal dependencies between activities. In the construction phase, specific non-free-choice constructs are introduced based on these indirect dependencies.

The **$\alpha^{\#}$-algorithm** is an extension of the α^{+} algorithm, such that some Petri nets not in the class of sound SWF-nets can be discovered on a complete log [29]. However, the exact class of potential models is not defined, as is the case for the α^{++}-algorithm.

The idea of the $\alpha^{\#}$-algorithm is that based on the causal and parallel dependencies derived on the activities in the induction phase, invisible transitions are introduced, such as the black transition in Figure 1, which skips the "Give a Talk" transition. These invisible transitions change the state of a Petri net, without appearing in the log. This way, a larger collection of logs still result in a sound Petri net, although this Petri net is no longer an SWF-net. Non-free-choice constructs are not introduced by this algorithm.

Similar to the $\alpha^{\#}$-algorithm, the **α^{*}-algorithm** extends the α algorithm to allow for the discovery of Petri nets not in the class of SWF-nets [22]. The difference with the $\alpha^{\#}$-algorithm is that instead of invisible transitions, duplicate transitions are used (i.e., multiple transitions carrying the same label such as the "Travel by Train" transition in Figure 1).

In the abstraction phase, information about the locations where events occur is recorded, together with the direct predecessors and successors (so-called P/S-tables). This information is then used in the induction phase to identify which events correspond to multiple transitions. However, the heuristic rules used to do this identification are not effective enough to cover all logs (i.e., the resulting model might be unsound).

By now, we have reviewed all abstraction-based algorithms. However, there is still not a single abstraction-based algorithm that can handle all the special constructs in a sound SWF-net. The proper use of invisible tasks, non-free-choice constructs and duplicate tasks needs further investigation.

3.2 Heuristic-Based Algorithms

Although the different algorithms presented in Subsection 3.1 can handle many of the control-flow constructs outlined in Subsection 2.1, they are all unable to handle a common factor in real-life event logs: the presence of *noise*. Noise can appear in two situations: event traces were somehow incorrectly logged (for instance, due to temporary system misconfiguration) or event traces reflect exceptional situations. In short, noise is any low-frequent behavior in a log. For instance, in our example of Figure 1, one conference attendee that pays for parking and still travels by train would result in noise in the log.

The algorithms presented in Section 3.1 do not take the *frequency* of ordering relations into account when inferring the advanced ordering relations to build the process model. Therefore, in this section we present an algorithm that considers the frequency of tasks when building process models: the **Heuristics Miner** (HM) [25, 26].

The Heuristics Miner can deal with noise and can be used to express the main behavior (i.e., not all the details and exceptions) registered in an event log. It supports the mining of all common constructs in process models (i.e., sequence, choice, parallelism, loops, invisible tasks and some kinds of non-free-choice), except for duplicate tasks. The HM algorithm has two main steps. In the first step, a *dependency graph* is built. This dependency graph contains the causal dependencies that are going to be kept when building the Petri net model. In contrast to the abstraction-based algorithms, the Heuristics Miner takes the frequencies of the basic ordering relations into account during the computation of the strength of the causal relations. By default, the algorithm creates one dependency to the best causal successor and predecessor of a given task.

In a second step, the *semantics of the split/join points* in the dependency graph are set. Using the frequencies of the dependencies the type of the split/join is decided. Consider Figure 1 again. If there is no noise in the log and "Go Home" occurs 100 times in the log, then the succeeding "Travel by Train" and "Pay for Parking" together also occur 100 times. However, if noise is present, these may occur more than 100 times. If they both occur 100 times, this indicates a parallel relation ("Go Home" is always followed by both "Travel by Train" and "Pay for Parking"). Using threshold values, a decision is made whether there should be a choice or a parallel construct in the Petri net.

3.3 Search-Based Algorithms

Although the use of heuristics has led to the development of algorithms that are more robust to noise, these algorithms are still very much based on *local* information in the log. In other words, the way of inferring the ordering relations is pretty much based on which tasks *directly* precede or follow each other. Furthermore, *none of the algorithms in Subsections 3.1 and 3.2 is able to handle all common constructs and be robust to noise at once.* Apparently, it is difficult to develop heuristics to address all these issues at the same time. In this sense, process mining techniques using *genetic algorithms* have been developed to benefit from the *global search* that these algorithms provide. The algorithms are the **Genetic Algorithm Miner** (GA) and the **Duplicates Genetic Algorithm Miner** (DGA) [15,16]. In the following we elaborate on the main characteristics of these algorithms. However, before doing so, let us introduce the main concepts behind genetic algorithms.

Genetic algorithms are adaptive search methods that try to mimic the process of evolution [19]. These algorithms start with an initial *population* of *individuals*. Every individual is assigned a *fitness* measure to indicate its quality. In the case of the GA and DGA, an individual is a possible process model and the fitness is a function that evaluates how well an individual is able to reproduce the behavior in the log. Populations evolve by selecting the fittest individuals and generating new individuals using *genetic operators* such as crossover (combining parts of two or more individuals) and mutation (random modification of an individual). Additionally, it is common practice to directly copy a number of the best individuals in a current population (the *elite*) to the next population. This way, one ensures that the best found individuals are kept in future populations. Every individual in a population has an *internal representation*, which defines the search space of a genetic algorithm. Together, the *internal representation*, the *fitness measure* and the *genetic operators* constitute the *three main issues* to be addressed when developing genetic algorithms. The remainder of this subsection explains how the GA and the DGA address these three issues.

The internal representation of the GA supports all constructs but duplicate tasks. Thus, the process models returned by this algorithm will contain *at most* one task for each unique task label in the log. Furthermore, the fitness measure used by the GA makes sure that individuals with a maximal fitness can correctly parse *most of* the process instances (traces) in the log and, ideally, not more than the behavior that can be derived from those traces. The reason for this is that these algorithms aim at mining a model that reflects as close as possible the behavior expressed in an event log. If the mined model allows for lots of extra behaviors that cannot be derived from the log, it does not give a precise description of what is actually happening. In other words, this model underfits the log. If the model captures all the behavior, it may overfit the log. The genetic operators work over the input and output sets of tasks. The input set of a task contains all the input places of this task and all the input tasks of these input places. In a similar way, the output set of a task has all of its output places plus the output tasks of these places. Thus, the genetic operators work by removing or adding places and tasks from/to these sets.

The DGA algorithm is an extension of the GA one to also support the discovery of process models containing duplicate tasks. In this sense, it has extended the internal representation used by the GA so that every unique task label in a log can have up to a maximum number of duplicates in the mined process model. Note that a maximum number is set to guarantee a finite search space. The assumption used to set the maximum number of duplicates per task is that no duplicate tasks should share input or output elements. Furthermore, since the inclusion of duplicates bring into the search space more individuals that can overfit the log, a dimension to punish for unnecessary unfolding (or duplication) of tasks has been added to the fitness used by the GA algorithm. Finally, the genetic operators were refined to take into account the existence of duplicates when recombining or mutating individuals.

Since these algorithms are geared to benefit discovery of process models that capture the most frequent behavior in a log, they work better when the traces in a log are grouped based on some similarity criteria (for instance, the order in which tasks appear in these traces). Additionally, note that the computational time required to run these algorithms is as least exponential to (i) the maximum number of tasks in individuals (because it impacts the size of the search space) and (ii) the number of unique traces in a log (since the fitness measure is based on how well individuals can replay these traces).

3.4 Language-Based Region Algorithms

A problem similar to process discovery arises in areas such as hardware design and control of manufacturing systems. There, the so called *theory of regions* is used to construct a Petri net from a behavioral specification (e.g., a state space or a language), such that the behavior of this net corresponds with the specified behavior, if such a net exists. If such a net does not exist, the best upper approximation is constructed.

In this section, we consider *language-based region theory* and its application in process mining. In [23], the authors show how for different classes of languages (step languages, regular languages and partial languages) a Petri net can be derived such that the resulting net is the Petri net with minimal additional behavior in which the words in the language are possible firing sequences. The classes of Petri nets described in [23] are general Petri nets, elementary nets, and inhibitor nets however the theory for elementary and inhibitor nets is not backed by an implementation. Duplicate transitions and invisible transitions are not supported.

The application of language-based region theory seems natural, i.e. each activity in the log corresponds to a letter in an alphabet and each trace to a word in the language defined by the log. By also including all prefixes of all traces, a prefix-closed language is defined, which we introduced as a pre-requisite for language-based region theory.

The application of the theory of regions to process mining has recently received more attention. In [10] for example, language-based regions are proposed to synthesize process models from event logs. In [10], the authors provide two methods to derive Petri nets from event logs, namely by using a *basis representation* or a *separating representation*, which are also mentioned in [23]. In [30], an approach based on Integer Linear Programming is introduced, which uses ideas from abstraction based process discovery, i.e. causal dependencies to guide the search for places.

Regions are defined by the solution space of a linear inequation system and most of the time, the number of different regions of a language is infinite, as the sum of two regions is a region again. Therefore, three finite representations are typically considered, all of which follow a similar search procedure. They start with a Petri net containing only transitions (i.e. a maximal underfitted net). Then, places are added to restrict the behavior of the Petri net. How many places are added depends on the representation.

In the **Basis Representation**, the set of places is chosen such that their corresponding regions form a basis for the non-negative integer solution space of the linear inequation system. Although such a basis always exists for homogeneous inequation systems, it is worst-case exponential in the number of equations [23], which in process mining is equal to the number of events in the log, and thus, the basis representation is worst-case exponential in the number of events in the log. Although [10] provides some ideas on how to remove redundant places from the basis, these procedures still require the basis to be fully constructed. Therefore, the result is a completely overfitted model, which is only valid under the assumption that the log is globally complete.

To reduce the size of the resulting Petri net, the authors of [23, 10] propose a **Separating Representation**. In this representation, places are only added to the Petri net that separates the allowed behavior (as recorded in the log) from behavior of which is known that it is not allowed. This representation requires the user to specify undesired behavior (i.e., behavior that is not recorded in the log and should not be supported by any model).

In [23, 10] the authors propose the separating representation for regular and step languages. They propose a method to automatically construct unwanted behavior, using the behavior not in the log, hence the log is assumed to be globally complete. Again, this leads to a model that is overfitting the log.

In [30] the authors present an approach for control flow discovery that uses **Integer Linear Programming** techniques to find places. However, in contrast to the basis and separating representations, the Petri net constructed is not exact (i.e., it is a Petri net that can reproduce the log under consideration), but it might allow for much more behavior than observed.

In contrast to the two other representations, the ILP representation does not try to generate a completely overfitted Petri net. Instead, it stops the search for places as soon as all causal dependencies between transitions are expressed. The way these causal dependencies are computed is comparable to the procedure we discussed when presenting abstraction based algorithms. For each dependency, the authors then search for a place, such that (i) this place expresses that causal dependency, and (ii) this place is as expressive as possible (i.e., it has a minimum number of incoming arcs and a maximum number of outgoing arcs). Hence, the number of places in the Petri net is at most quadratic in the number of activities in the log. Note that no assumptions are made about the completeness of the log (i.e., the causal dependencies only guide the search for places), but if no causal dependencies are found, then a completely underfitting model is the result.

3.5 State Discovery Algorithms

So far, all algorithms presented in this paper focus on the ordering of events in the log and use that to derive a Petri net. A **state discovery algorithm** takes a different approach towards mining. They use the log to translate each case into a sequence of states and the transitions between these states. All cases are then combined into a transition system describing the log and the transition system is converted into a Petri net.

In [21], an extensive discussion is presented on how to define states based on the information in the log. One way of defining a state, is by considering the prefix of each event to be the state of the model prior to executing the event. However, more advanced techniques are presented, considering the past and future behavior of each event in the log, as well as arbitrary data attributes. In this phase, unlabelled (or invisible) transitions are introduced in the transition system.

To construct a Petri net from a transition system, state-based region theory is used [8, 9, 11, 12, 18]. This theory focusses on the synthesis of Petri nets from state-based models, where the state space of the Petri net is bisimilar to the given state-based model. Initially the theory could be applied only to a restricted set of transition systems. However, over time the approach has been extended to allow for the synthesis from any finite transition system. The resulting Petri net might contain duplicate transitions (referred to as label-splitting in region theory).

4 Tool Support

Ever since the first work on process mining appeared, good tool support has proven essential [2]. Over the years, many implementations have been presented by different authors and although most of these implementations served their purpose as a testbed for the author's algorithms on simple examples, they were not so suitable in real-life situations, where logs easily contain thousands of cases referring to dozens of events each.

Around 2003, a first attempt was made to develop a single tooling platform that was (i) easy to extend with new algorithms, (ii) applicable to real-life examples and (iii) easy to use. This attempt led to the development of the ProM framework (prom.sourceforge.net), of which currently version 5.0 is the latest release, containing over 250 implementations of algorithms, categorized in import/export plugins, conversion plugins, analysis plugins and mining plugins.

As intended, ProM can handle large, industrial sized, log files as well as many different modelling paradigms, such as Petri nets, EPCs, Heuristic nets and so on. Furthermore, the ideas implemented in ProM have been adopted by industry and have become part of industrial tools, such as the social network analysis in Aris PPM and the process mining components of Protos. Furthermore, a Dutch company called Futura Technology (www.futuratech.nl) uses ideas and concrete implementations of ProM to provide custom process mining services to industry.

Although ProM, together with its tutorials, satisfies the requirements above, we feel that it has reached its limits. More and more we see that real life logs contain information that was never considered to be present in logs, such as data modifications, event triggers, organizational structures, and so on. As the logs required by ProM do not allow for semantic information to be added directly onto the data in the log, the interpretation of each data attribute is different for each plugin thus reducing the ease of use for end-users of the tool.

Furthermore, the tight coupling between the implementations of algorithms and the user interfaces to control the settings of these algorithms is becoming a bottleneck when ProM is applied in an industrial setting. Companies like Philips, for example use ProM to monitor their processes, using a custom-built user interface which took 9 months to develop.

5 Future Directions

This section presents the three directions we believe future discovery algorithms should focus on. The underlying assumption is that tool support for these future directions should also be provided.

5.1 Support Different Abstraction Levels

In the previous section, we presented many different algorithms to construct Petri nets from logs. Most of these algorithms make a similar assumption on the relation between the Petri net and the underlying log, namely that each transition not used solely for routing purposes represents one event of the log (an exception is the tsinghua-α algorithm which assumes that each transition represents 2 events). From a theoretical point of view, this is a reasonable assumption as the firing sequences of the resulting Petri net relate directly to traces in the log. Nonetheless, *the assumption that all events in the log are logged on the same abstraction level has proven to be invalid in practice.*

In practice, event logs usually contain many events on very different levels of abstraction. For example, a user logging on to an information system is on a very different abstraction level than a data attribute in a database being changed by an application. Many attribute changes for example might correspond to one activity which is performed by the user. An example of a process mining technique which takes into account these different abstraction levels is the work presented in [20]. There, events are clustered using heuristics to generate a model representing the process on different, hierarchical, abstraction levels. However, the so-called "fuzzy models" used in that work lack executable semantics and hence are not very useful for analysis and simulation purposes. Nonetheless, as shown by the authors of [24], companies like Philips appreciate the models and use them in an industrial setting.

Therefore, *future process discovery algorithms should target the mining of process models with different abstraction levels.* In fact, it would be nice if the algorithm could identify the nature (i.e., more structured or unstructured) of the behavior registered in the event log and provide the discover model at the best insightful abstraction level.

5.2 Consider Event Properties

In Section 3.5, we discussed how to derive state information from a log. In [3] the authors do not only use the events in the log to derive state information, but they also use so-called "properties" of events. These properties give information which describe the events in more detail. For example, when an insurance claim is rejected, the event entitles "send rejection letter" might also contain information about the amount of the claim and the reason why the claim was rejected. Obviously, not all of these properties are directly useful for process discovery. However, *many properties provide great insights into the context of an event, which can help in discovery.*

Consider, for example, an activity entitled "archive claim", which seems to occur at several points in a log from an insurance company. Mining such a log might prove to be very difficult due to the seemingly random occurrences of that activity. However, when looking at the properties of the events, it might become clear that the "archive claim" activity is always performed as the last activity before handing the claim over to another department, hence the Petri net discovered from the log should contain multiple transitions referring to the claim being archived by different departments. Furthermore, recent work on cycle time prediction [17] has shown that considering data attributes recorded in a log indeed provides great insights into the underlying process.

Using properties of events, we feel that it should be possible to determine equivalence classes of events using classical data mining techniques. This allows for existing process discovery algorithms such as the ones presented in this paper to be applied in a broader context.

5.3 Semantic Information

Currently, all techniques see the events in the log as syntactic labels. However, it would be nice if the semantics behind the labels could be taken into account. For instance, consider the example of the claim archiving again. Previously, we stated that properties might be logged directly which indicate the department in which the "archive claim" activity is executed. However, sometimes this information is not available directly, but indirectly using semantic information. Suppose for example that the log only shows who executed the activity "archive claim". Then the organizational model of the insurance company provides the department to which the person belongs, i.e. the department is derived from the person using semantic information.

The idea of using semantic information in process discovery is currently being investigated in the context of the European project SUPER [1]. Actually, the work in [14] provides an outlook on how process mining techniques can benefit from semantic information. Furthermore, as the experiences in [24] indicate, companies can typically provide the semantic information relating to different properties.

6 Conclusion

In this paper, we have provided a critical overview of the 13 Petri net discovery algorithms currently available in the process discovery area. The overview includes a comparison of these algorithms based on four different perspectives: (i) assumptions about the *completeness* of event logs, (ii) supported control-flow *constructs*, (iii) provided *abstraction* levels for correspondence between transitions in Petri nets and event classes in logs, and (iv) bias to mine *under-/over-fitting* models. Based on this analysis and the current tool support to process discovery techniques, we have proposed an outlook with the following three directions: (i) there is a need to *support different levels of abstraction*, especially while mining less structured behavior, (ii) more *contextual information (like event properties) should be considered* when performing process discovery, and (iii) the *meaning (or semantic) of labels* in event logs should be incorporated when mining process models. When addressed, these three future directions will advance the feedback provided by techniques in the process discovery research area towards more insightful and robust models.

Acknowledgments. This research is partly supported by SUPER [1].

References

1. European Project SUPER - Semantics Utilised for Process Management within and between Enterprises, `http://www.ip-super.org/`
2. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow Mining: A Survey of Issues and Approaches. Data and Knowledge Engineering 47(2), 237–267 (2003)
3. van der Aalst, W.M.P., Rubin, V., van Dongen, B.F., Kindler, E., Günther, C.W.: Process Mining: A Two-Step Approach using Transition Systems and Regions. BPM Center Report BPM-06-30, BPMcenter.org (2006)
4. van der Aalst, W.M.P., Weijters, A.J.M.M. (eds.): Process Mining, Special Issue of Computers in Industry, vol. 53(3). Elsevier Science Publishers, Amsterdam (2004)
5. van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow Mining: Discovering Process Models from Event Logs. IEEE Transactions on Knowledge and Data Engineering 16(9), 1128–1142 (2004)
6. de Medeiros, A.K.A., van Dongen, B.F., van der Aalst, W.M.P., Weijters, A.J.M.M.: Process Mining for Ubiquitous Mobile Systems: An Overview and a Concrete Algorithm. In: Baresi, L., Dustdar, S., Gall, H.C., Matera, M. (eds.) UMICS 2004. LNCS, vol. 3272, pp. 151–165. Springer, Heidelberg (2004)
7. Badouel, E., Bernardinello, L., Darondeau, P.: Polynomial algorithms for the synthesis of bounded nets. In: Mosses, P.D., Schwartzbach, M.I., Nielsen, M. (eds.) CAAP 1995, FASE 1995, and TAPSOFT 1995. LNCS, vol. 915, pp. 364–378. Springer, Heidelberg (1995)
8. Badouel, E., Bernardinello, L., Darondeau, P.: The Synthesis Problem for Elementary Net Systems is NP-complete. Theoretical Computer Science 186(1-2), 107–134 (1997)
9. Badouel, E., Darondeau, P.: Theory of regions. In: Reisig, W., Rozenberg, G. (eds.) APN 1998. LNCS, vol. 1491, pp. 529–586. Springer, Heidelberg (1998)
10. 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)
11. Cortadella, J., Kishinevsky, M., Lavagno, L., Yakovlev, A.: Synthesizing Petri Nets from State-Based Models. In: Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design (ICCAD 1995), pp. 164–171. IEEE Computer Society, Los Alamitos (1995)
12. Cortadella, J., Kishinevsky, M., Lavagno, L., Yakovlev, A.: Deriving Petri Nets from Finite Transition Systems. IEEE Transactions on Computers 47(8), 859–882 (1998)
13. Darondeau, P.: Deriving unbounded petri nets from formal languages. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 533–548. Springer, Heidelberg (1998)
14. de Medeiros, A.K.A., Pedrinaci, C., van der Aalst, W.M.P., Domingue, J., Song, M., Rozinat, A., Norton, B., Cabral, L.: An Outlook on Semantic Business Process Mining and Monitoring. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2007, Part II. LNCS, vol. 4806, pp. 1244–1255. Springer, Heidelberg (2007)
15. de Medeiros, A.K.A.: Genetic Process Mining. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands (2006)
16. de Medeiros, A.K.A., Weijters, A.J.M.M., van der Aalst, W.M.P.: Genetic Process Mining: an Experimental Evaluation. Data Mining and Knowledge Discovery 14(2), 245–304 (2007)

17. van Dongen, B.F., Crooy, R.A., van der Aalst, W.M.P.: Cycle Time Prediction: When Will This Case Finally Be Finished. In: Meersman, R., Tari, Z. (eds.) OTM 2008. LNCS, vol. 5331, pp. 319–336. Springer, Heidelberg (2008)
18. Ehrenfeucht, A., Rozenberg, G.: Partial (Set) 2-Structures - Part 1 and Part 2. Acta Informatica 27(4), 315–368 (1989)
19. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. In: Natural Computing. Springer, Berlin (2003)
20. Günther, C.W., van der Aalst, W.M.P.: Fuzzy Mining - Adaptive Process Simplification Based on Multi-perspective Metrics. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 328–343. Springer, Heidelberg (2007)
21. Kindler, E., Rubin, V., Schäfer, W.: Process Mining and Petri Net Synthesis. In: Eder, J., Dustdar, S. (eds.) BPM Workshops 2006. LNCS, vol. 4103, pp. 105–116. Springer, Heidelberg (2006)
22. Li, J., Liu, D., Yang, B.: Process mining: Extending α-algorithm to mine duplicate tasks in process logs. In: Chang, K.C.-C., Wang, W., Chen, L., Ellis, C.A., Hsu, C.-H., Tsoi, A.C., Wang, H. (eds.) APWeb/WAIM 2007. LNCS, vol. 4537, pp. 396–407. Springer, Heidelberg (2007)
23. Lorenz, R., Juhás, R.: How to Synthesize Nets from Languages - a Survey. In: Proceedings of the Wintersimulation Conference, WSC 2007 (2007)
24. van Uden, K.: Extracting User Profiles with Process Mining at Philips Medical Systems. Master's thesis, Eindhoven University of Technology, Eindhoven, The Netherlands (2008)
25. Weijters, A.J.M.M., van der Aalst, W.M.P.: Rediscovering Workflow Models from Event-Based Data using Little Thumb. Integrated Computer-Aided Engineering 10(2), 151–162 (2003)
26. Weijters, A.J.M.M., van der Aalst, W.M.P., Alves de Medeiros, A.K.: Process Mining with HeuristicsMiner Algorithm. BETA Working Paper Series, WP 166, Eindhoven University of Technology, Eindhoven (2006)
27. Wen, L., van der Aalst, W.M.P., Wang, J., Sun, J.: Mining process models with non-free-choice constructs. Data Mining and Knowledge Discovery 15(2), 145–180 (2007)
28. Wen, L., Wang, J., van der Aalst, W.M.P., Wang, Z., Sun, J.: A Novel Approach for Process Mining Based on Event Types. BETA Working Paper Series, WP 118, Eindhoven University of Technology, Eindhoven (2004)
29. Wen, L., Wang, J., Sun, J.: Mining invisible tasks from event logs. In: Dong, G., Lin, X., Wang, W., Yang, Y., Yu, J.X. (eds.) APWeb/WAIM 2007. LNCS, vol. 4505, pp. 358–365. Springer, Heidelberg (2007)
30. van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process Discovery using Integer Linear Programming. In: Application and Theory of Petri Nets 2008 (2008)

Construction of Process Models from Example Runs

Robin Bergenthum[1], Jörg Desel[1], Sebastian Mauser[1], and Robert Lorenz[2,⋆]

[1] Department of Applied Computer Science, Catholic University of Eichstätt-Ingolstadt, Ostenstr. 14, 85072 Eichstätt, Germany
{robin.bergenthum,joerg.desel,sebastian.mauser}@ku-eichstaett.de
[2] Department of Computer Science, University of Augsburg, Eichleitnerstrasse 30, 86159 Augsburg, Germany
robert.lorenz@informatik.uni-augsburg.de

Abstract. This contribution suggests a novel approach for a systematic and automatic generation of process models from example runs. The language used for process models is place/transition Petri nets, the language used for example runs is labelled partial orders. The approach adopts techniques from Petri net synthesis and from process mining. In addition to a formal treatment of the approach, a case study is presented and implementation issues are discussed.

1 Introduction

Business process modelling and management has attracted increasing attention in recent years [1,2,3]. However, little attention has been paid to the first phases of business process modelling, i.e., to the question of how to derive a valid process model in an informal setting.

The usual approach to process model construction and validation is shown on the left hand side in Figure 1. A domain expert edits a formal process model. Simulation tools generate single runs of that process model which can also be viewed as formal objects, such as occurrence sequences representing possible sequential occurrences of activities or occurrence nets representing occurrences of activities and their causal ordering. Then the expert checks whether these runs correspond to possible executions of the intended process. In the negative case, he changes the process model and iteratively repeats the simulation.

In this paper, we consider Petri net process models. There are many simulation tools that are able to generate sequential runs. Our VipTool generates and visualizes causally ordered occurrence nets [4,5].

The aim of this paper is to suggest a proceeding in the opposite direction. We call causally ordered executions of the process to be modelled scenarios. We assume that the domain experts know some or all scenarios of the process to be modelled better than the process itself. Actually, experts might also know parts of the process model including parts of its branching structure, but in this case scenarios can be derived from this partially known process model. Experience shows that in various application areas processes are specified in terms of example scenarios (an evidence are the commonly used sequence diagrams in UML to specify scenarios).

⋆ Supported by the project "SYNOPS" of the German Research Council.

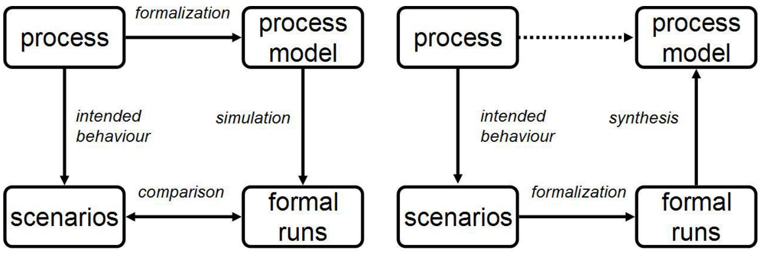


Fig. 1. Old and new approach

In a first step, an expert formalizes the scenarios, yielding formal runs. In other words, he provides formal models of the scenarios. In our setting, the scenarios are formalized in terms of labelled partial orders representing occurrences of process activities and their mutual order relation. In a second step, a process model is automatically generated from these formal runs. For this step, we apply algorithms developed for Petri net synthesis [6,7,8] and for process mining [9,10]. This procedure is shown on the right hand side of Figure 1.

The synthesized process model has at least the specified runs, but it might have additional runs. In a third step, additional runs are generated and presented to the expert. Runs that also represent legal scenarios are added to the set of runs specifying the process. If a run represents a behaviour which was not intended then the process model is changed accordingly.

In the following section we provide the necessary formalism to specify runs. The partial order approach taken in this paper avoids to consider all possible orderings of concurrent occurrences of activities as it was necessary in a sequential approach. However, in case of branching due to alternatives, the number of necessary example runs can be quite large (or even infinite). Therefore, we develop a term based specification of runs in section 3, where the atoms of the terms are comparably small labelled partial orders. Section 4 deals with the synthesis of a p/t-net process model from such term specification. In section 5 we tackle the problem of hierarchical process definitions. Section 6 provides a case study supported by our tool VipTool. In section 7 we discuss related work.

2 Specifying Runs

The core idea of our approach is to specify behaviour of a process in terms of single runs, playing the role of example runs of the process. Therefore, the process model to be generated should at least have the behaviour given by these runs.

We claim that modelling a single run is an easy and intuitive task, also for domain experts unexperienced in modelling. There are many possibilities to describe scenarios (also textual descriptions are adequate) and formalization of scenarios in terms of runs is relatively easy (on this level of single instances). Using scenarios in requirements engineering has received significant attention in the last years in the field of software modelling (see section 7). In this paper we focus on scenarios to model business processes. In this area, scenarios do not necessarily have to be designed from scratch. In many cases it is possible to exploit already existing descriptions of scenarios supported by the business process. In an enterprise, typical sources of scenario descriptions are log files recorded by information systems (process mining focuses on this source of information), process instructions for employees or textual and formal process descriptions from some requirements analysis.

In the first step of the design approach, single runs of the process are identified. This leads to a preferably complete description of the behaviour of the process. In this paper, we consider *labelled partial orders (LPOs)* to specify single runs formally. LPOs are a very general formalism and most languages used in practice can be mapped to LPOs.

Definition 1 (labelled partial order). *A labelled partial order (LPO) is a triple* $\mathrm{lpo} = (V, <, l)$*, where* V *is a set of events,* $<$ *is an irreflexive and transitive binary relation on* V*, and* $l : V \to T$ *is a labelling function with set of labels* T*.*

The behaviour specified by an LPO includes its so called prefixes and sequentializations. An LPO $(V', <', l')$ *is called a prefix of another LPO* $(V, <, l)$ *if* $V' \subseteq V$*,* $(v' \in V' \wedge v < v') \Longrightarrow (v \in V')$*,* $<' = < \cap (V' \times V')$ *and* $l' = l|_{V'}$*. An LPO* $(V, <', l)$ *is called a sequentialization of another LPO* $(V, <, l)$ *if* $< \subseteq <'$*.*

Two LPOs $(V, <, l)$ *and* $(V', <', l')$ *are called isomorphic if there is a bijective mapping* $\psi : V \to V'$ *such that* $l(v) = l'(\psi(v))$ *for* $v \in V$*, and* $v < w \Longleftrightarrow \psi(v) <' \psi(w)$ *for* $v, w \in V$*. Isomorphic LPOs model the same behaviour. Therefore, we consider LPOs only up to isomorphism, i.e., isomorphic LPOs are not distinguished.*

An LPO models a single run by specifying "earlier than"-dependencies between events, where an event represents an occurrence of the process activity given by its label. LPOs offer the following advantages in process modelling compared to sequential approaches where behaviour is given in terms of occurrence sequences [4]:

• *A natural and intuitive representation of the behaviour of processes:* Since concurrency plays an important role in process models, it is appropriate to model concurrency also in single runs of a process. In particular, instead of considering sequential runs and detecting the concurrency relation from a set of runs, it is easier and more intuitive to work with partially ordered runs.
• *An efficient representation of the behaviour of processes:* A single LPO represents a set of sequential runs, which can be quite large (exponential in the number of transition occurrences) in the presence of concurrency.
• *A high degree of expressiveness:* First, considering sequential runs, concurrency cannot be distinguished from non-deterministic resource sharing. Second, LPOs explicitly model causal dependencies between transition occurrences, which allows the explicit modelling of the flow of objects and of information in processes (this is not even implicitly possible with sequential runs).
• *Efficient analysis algorithms for business process models:* In many cases, analysis techniques applied to LPOs are more efficient than those working on sequential runs [5,11].

We start our process generation procedure with a collection of LPOs (runs) representing scenarios of the process. An example of such a set of LPOs is shown in Figure 2 for the workflow triggered by a damage report in an insurance company. In all shown runs, a received claim is negatively evaluated and a refusal letter is sent. In the first run, the refusal letter is sent after the damage and the insurance of the client was checked. In

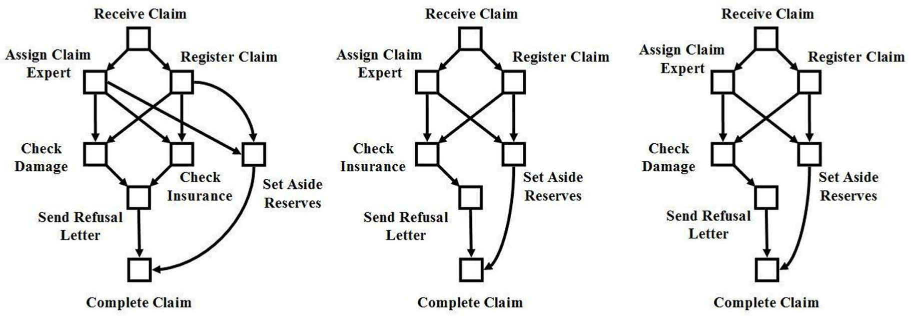


Fig. 2. Example for a collection of LPOs representing scenarios of a process

runs 2 and 3 only one check is performed before the negative evaluation and the sending of the refusal letter. In all runs, after the assignment and the registration of the claim, reserves are set aside.

Before formalizing scenarios, the domain expert should specify initial and final conditions for the considered process (see [12]) or some other kind of reference to the environment of the process in order to embed the process into its context. Moreover, he should identify the possible activities because they appear as labels in the runs. This is of particular importance when more than one expert provides example runs, because all occurrences of the same activity have to be labelled by the same activity name.

3 Composed Runs

In case of large processes the procedure of considering complete runs as described in the previous section may still be difficult for domain experts. In particular, it suffers from the fact that the number of runs that have to be specified might grow exponentially with the number of alternatives and can be even infinite if the process contains loops. Also, single example runs may become very large and difficult to handle. The problem can be solved by partly incorporating the process structure in the specification. The idea is that it is possible for a domain expert to only specify parts of runs, called run segments, which can be as small as desired and also can be used to locally specify alternative or iterative parts of runs (avoiding the explicit specification of all alternative runs). Complete runs are then given by specifying appropriate compositions of run segments. This possibility to modularly develop runs by means of run segments makes the specification of runs easier, faster and more intuitive for domain experts. Sometimes such approach is even necessary, e.g. for complete specifications in the case of iterations or if some domain expert actually only knows parts of runs.

Figure 3 depicts a run segment showing the registration process, several run segments describing possible evaluation procedures of the claim, a run segment modelling the payment of the insurance company and the three singleton runs for building reserves, gathering information for the payment by asking queries and the completion of the workflow. In this section, we specify a set of runs by means of such run segments.

Run segments can be related in four ways:

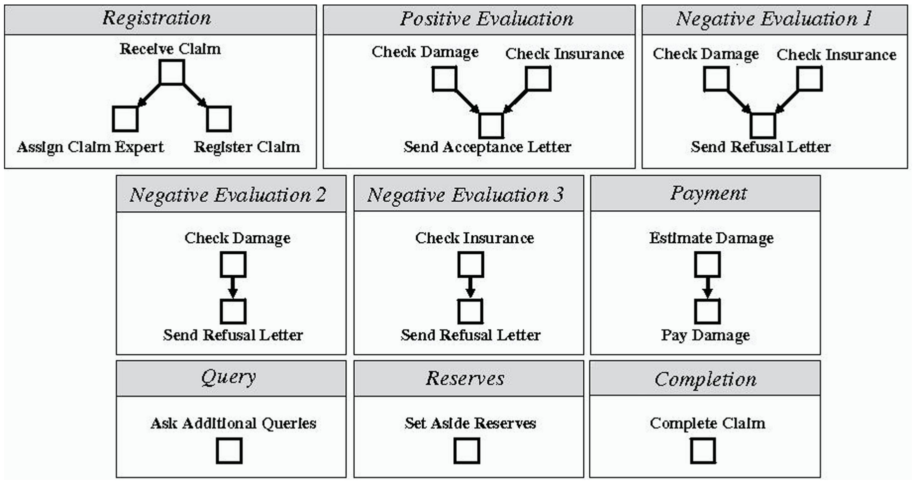
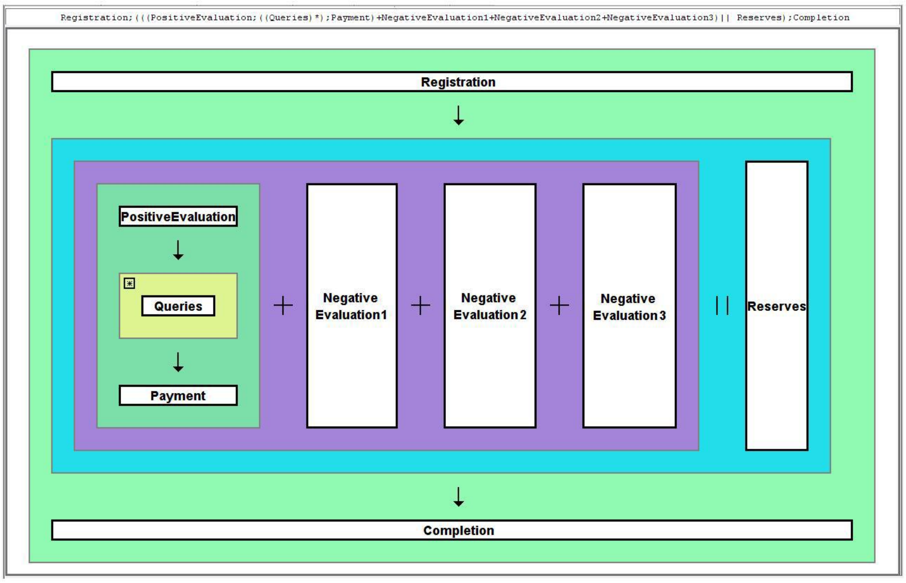


Fig. 3. Example for a collection of run segments in terms of LPOs

- A run segment LPO_2 occurs after another segment LPO_1 (sequence).
- Either LPO_1 occurs or LPO_2 but not both (alternative).
- A run segment LPO_1 can recur arbitrarily often (iteration).
- Two segments LPO_1 and LPO_2 occur concurrently (concurrency).

Similar to the approach for scenario integration based on statecharts in [13], higher level structures of runs are built by concatenating and nesting blocks according to the relationships sequence (;), alternative ($+$), iteration ($*$), and concurrency ($||$). For the run segments of Figure 3, we assume that they are related as depicted in Figure 4 to faithfully model the underlying workflow. That means, the workflow starts with the "Registration" of the claim. Then one of the evaluation runs is performed concurrently to the subprocess of setting aside reserves for the claim. The four possibilities of evaluation are alternatives. The run modelling the positive case is a sequential composition of the three single run segments "Positive Evaluation", "Queries" and "Payment". Asking additional queries can be iterated arbitrarily often until a sufficient degree of information is reached. Finally, after the execution of all other run segments, the process is finished by the run segment "Completion". Figure 4 uses a graphical representation of the four composition templates for runs by means of a block structure (; -composition is depicted by arcs, $+$, $*$ and $||$ by respective symbols). Since the binary composition operators are associative, the readability of the graphical representation is improved by composing more than two blocks in figures (e.g. the $+$-composition of the four alternative evaluation possibilities). We call such a behavioural specification generated by the composition of single run segments a *composed run*. Composed runs nicely integrate modular scenario specifications (run segments) which may be given by different experts and support the specification of infinite behaviour (by the iteration operator).

Since single run segments are given by LPOs, the semantics of a composed run $\mathbf{R}$ is defined as a set of LPOs $\mathfrak{Lpo}(\mathbf{R})$ modelling possible behaviour each, called the *set of runs defined by a composed run*. The set of runs defined by the composed run depicted



```
Registration;(((PositiveEvaluation;((Queries)*);Payment)+NegativeEvaluation1+NegativeEvaluation2+NegativeEvaluation3)|| Reserves);Completion
```

Fig. 4. A composed run over the set of run segments depicted in Figure 3 (screenshot of VipTool)

in Figure 4 comprises the runs shown in Figure 2 and the infinite set of runs illustrated in Figure 5 (activity "Ask Additional Queries" can be iterated).

Definition 2 (composed run). *Given a finite set of single run segments $\mathcal{A}$, a composed run over $\mathcal{A}$ is inductively defined as follows:*
Each single run segment lpo *of $\mathcal{A}$ and the empty LPO $\lambda = (\emptyset, \emptyset, \emptyset)$ are composed runs. Let $\mathbf{R}_1$ and $\mathbf{R}_2$ be composed runs. Then $\mathbf{R}_1; \mathbf{R}_2$ (sequential composition), $\mathbf{R}_1 + \mathbf{R}_2$ (alternative composition), $(\mathbf{R}_1)^*$ (iteration) and $\mathbf{R}_1 \parallel \mathbf{R}_2$ (concurrent composition) are composed runs.*

Assume two LPOs $\mathrm{lpo}_1 = (V_1, <_1, l_1), \mathrm{lpo}_2 = (V_2, <_2, l_2)$ with disjoint sets of events. We define:

- $\mathrm{lpo}_1; \mathrm{lpo}_2 := (V_1 \cup V_2, <_1 \cup <_2 \cup (V_1 \times V_2), l_1 \cup l_2)$,
- $\mathrm{lpo}_1 \parallel \mathrm{lpo}_2 := (V_1 \cup V_2, <_1 \cup <_2, l_1 \cup l_2)$,
- $\mathrm{lpo}_1^0 := \lambda$ *and* $\mathrm{lpo}_1^n := \mathrm{lpo}_1^{n-1}; \mathrm{lpo}_1$ *for* $n > 0$.

The set of runs $\mathfrak{Lpo}(\mathbf{R})$ of a composed run $\mathbf{R}$ over $\mathcal{A}$ is a possibly infinite set of LPOs. Given a composed run $\mathbf{R}$, we first inductively define a set of LPOs $K(\mathbf{R})$ represented by $\mathbf{R}$. The set $\mathfrak{Lpo}(\mathbf{R})$ is the prefix and sequentialization closure of $K(\mathbf{R})$. We set $K(\lambda) = \{\lambda\}$ and $K(\mathrm{lpo}) = \{\mathrm{lpo}\}$ for $\mathrm{lpo} \in \mathcal{A}$. For composed runs $\mathbf{R}_1$ and $\mathbf{R}_2$,

- $K(\mathbf{R}_1 + \mathbf{R}_2) = K(\mathbf{R}_1) \cup K(\mathbf{R}_2)$,
- $K(\mathbf{R}_1; \mathbf{R}_2) = \{\mathrm{lpo}_1; \mathrm{lpo}_2 \mid \mathrm{lpo}_1 \in K(\mathbf{R}_1),\ \mathrm{lpo}_2 \in K(\mathbf{R}_2)\}$,
- $K((\mathbf{R}_1)^*) = \{\mathrm{lpo}_1; \dots; \mathrm{lpo}_n \mid \mathrm{lpo}_1, \dots, \mathrm{lpo}_n \in K(\mathbf{R}_1),\ n \in \mathbb{N}^+\} \cup \{\lambda\}$,
- $K(\mathbf{R}_1 \parallel \mathbf{R}_2) = \{\mathrm{lpo}_1 \parallel \mathrm{lpo}_2 \mid \mathrm{lpo}_1 \in K(\mathbf{R}_1),\ \mathrm{lpo}_2 \in K(\mathbf{R}_2)\}$.

A problem excluded so far is that some runs may overlap. That means, the knowledge about one run segment may be distributed on several experts, each knowing only a part

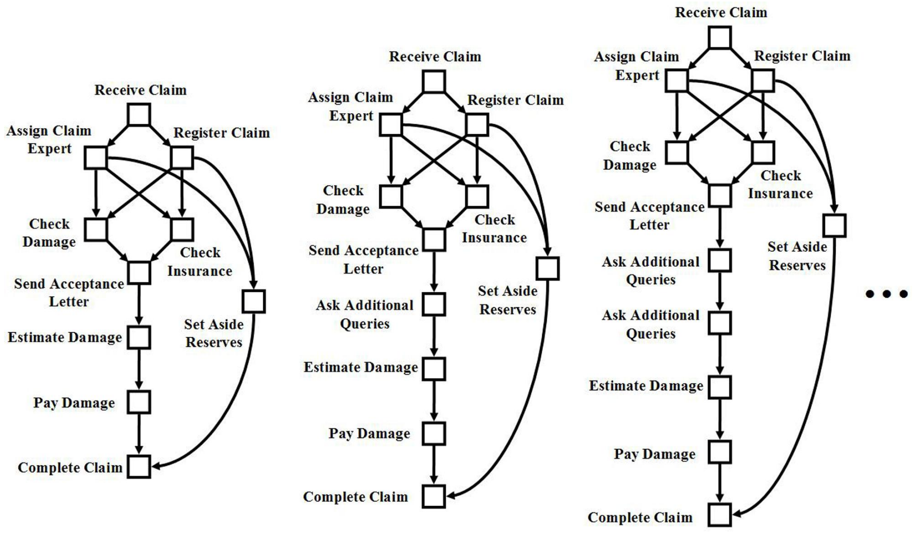


Fig. 5. Infinite set of runs of the composed run of Figure 4, where the claim is positively evaluated

of the segment. In the simplest case these parts can be treated as single run segments that can be composed as shown above. But this is not possible if the parts contain common events. Then they have to be fused to one single run segment. The situation of runs having common events occurs if several experts have different views to one process execution, i.e. they observe different subsets of all events of the respective run, whereas respective other parts of the run are hidden.

We propose the following concept to *fuse run segments*. Given several parts of one segment, first the involved people have to determine which events observed by one expert coincide with which events observed by another expert. This problem has to be solved by an appropriate communication between the experts and is part of the specification process. The experts have to agree on a *fusion equivalence relation*, defined on the set of events of all parts of the considered run segment, such that different observations of one event are equivalent. Obviously, only events having the same label (referring to the same activity) can be equivalent. Also, the orderings given by different observations must not contradict each other. This has to be ensured in an adjustment phase by the modelers. The fusion of the parts is then given by a new LPO, which has an event for each equivalence class. If two events are ordered in some part of the run segment, then their respective classes are ordered in this LPO. Thus, each dependency observed (respectively modelled) by some expert is regarded in the fused run segment.

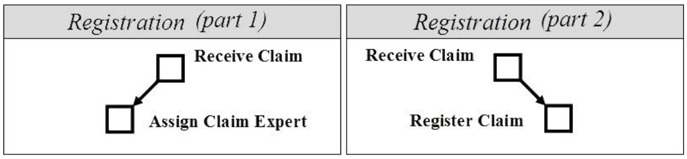


Fig. 6. Two possible parts of the run segment ”Registration” shown in Figure 3

No further dependencies are introduced. Conversely, we assume concurrency between events if no expert detected any dependency. This is motivated by the idea that parts of one run observed by different experts tend to be concurrent.

A simple example is shown in Figure 6. Assume that the run segment ”Registration” of Figure 3 is not given directly, but rather by two parts of the run segment, as shown in Figure 6. If the two events labelled by ”Receive Claim” coincide, the described fusion approach generates the fused run ”Registration” of Figure 3.

Definition 3 (fusion). *Assume LPOs* $\mathrm{lpo}_i = (V_i, <_i, l_i)$, $i \in \{1, \ldots, n\}$ *with pairwise disjoint sets of events, modelling different parts of one run segment.*

An equivalence relation $\sim$ *on* $\bigcup_{i=1}^{n} V_i$ *fulfills the fusion requirement if*

$$v \sim v', v \neq v', v \in V_i, v' \in V_j \Longrightarrow$$

$$i \neq j \wedge l_i(v) = l_j(v') \wedge \forall v'' \in V_i, v''' \in V_j, v'' \sim v''' : (v <_i v'' \Longrightarrow v''' \not<_j v').$$

In this case, the fused LPO of $\mathrm{lpo}_i = (V_i, <_i, l_i)$, $i \in \{1, \ldots, n\}$ *w.r.t.* $\sim$ *is defined by* $\mathrm{lpo} = (V, <, l)$, *where*

- $V = \{[v]_\sim \mid v \in \bigcup_{i=1}^{n} V_i\}$,
- $[v]_\sim < [v']_\sim \Longleftrightarrow (\exists v'' \in [v]_\sim, v''' \in [v']_\sim, i \in \{1, \ldots, n\} : v'', v''' \in V_i, v'' <_i v''')$,
- $l([v]_\sim) = l_i(v)$ *(for* $v \in V_i$*).*

The fused LPO is well defined because of the fusion requirement.

4 Synthesizing a Process Model

The next step in the design approach starts with a specification of a process by means of a composed run. The aim is to automatically create a Petri net model from the composed run. Petri net based models are the standard to compactly represent processes in the area of workflow design and Petri nets offer a huge repertoire of analysis methods. Formally, the composed run is defined as a term over the alphabet of single run segments employing the composition operators ; , $+$, $*$ and $||$. In [6], we show how to synthesize a *place/transition net (p/t-net)* from such a term.

The activities of the process are modelled by the transitions of the synthesized Petri net. The places together with their connections to the transitions and their markings define dependencies between the activities. As usual, places are drawn as circles, tokens in places represent the initial marking, transitions are drawn as rectangles and the flow relation as arcs annotated with values of the weight function (arcs with weight 0 are not drawn, the weight 1 is not shown). Note that events of a composed run having the same label model different occurrences of the same transition. Therefore, it is not possible to convert a composed run into a Petri net the naive way by adding places in between run segments and within run segments putting places in between ordered events.

A *run of a p/t-net* N is given by an LPO with event labels referring to transitions, such that the events can occur, respecting the concurrency and dependency relations of the LPO. Thus, a run describes executable behaviour of the net in the sense that the transition occurrences given by the events are possible in the net, only using the dependencies specified by the run for the flow of tokens. The set of all runs of N is denoted by $\mathfrak{Lpo}(N)$.

Definition 4 (p/t-net). *A (marked) p/t-net is a quadruple $N = (P, T, W, m_0)$, where P is a finite set of places, T is a finite set of transitions satisfying $P \cap T = \emptyset$, $W : (P \times T) \cup (T \times P) \to \mathbb{N}$ is a weight function defining the flow relation, and $m_0 : P \to \mathbb{N}$ ($\mathbb{N}$ denotes the non-negative integers) is an initial marking.*

A multi-set of transitions $\tau : T \to \mathbb{N}$ is called a step *(of transitions). A step τ is enabled to occur (concurrently) in a marking $m : P \to \mathbb{N}$ of a p/t-net if and only if $m(p) \geq \sum_{t \in \tau} \tau(t) W(p, t)$ for each place $p \in P$. In this case, its occurrence leads to the marking $m'(p) = m(p) + \sum_{t \in \tau} \tau(t)(W(t, p) - W(p, t))$, abbreviated by $m \xrightarrow{\tau} m'$. A finite sequence of steps $\sigma = \tau_1 \ldots \tau_n$, $n \in \mathbb{N}$, is called a step occurrence sequence enabled at m and leading to m_n, denoted by $m \xrightarrow{\sigma} m_n$, if there exists a sequence of markings $m_1, \ldots, m_n$ such that $m \xrightarrow{\tau_1} m_1 \xrightarrow{\tau_2} \ldots \xrightarrow{\tau_n} m_n$.*

Given an LPO $\mathrm{lpo} = (V, <, l)$, two events $v, v' \in V$ are called independent if $v \not< v'$ and $v' \not< v$, denoted by $v \,\mathrm{co}\, v'$. A co-set is a subset $C \subseteq V$ fulfilling: $\forall v, v' \in C : v \,\mathrm{co}\, v'$. A cut is a maximal co-set. For a co-set C and an event $v \in V \setminus C$ we write $v < (>)\, C$, if $v < (>)\, v'$ for an element $v' \in C$ and $v \,\mathrm{co}\, C$, if $v \,\mathrm{co}\, v'$ for all elements $v' \in C$. Given a marked p/t-net N, an LPO $\mathrm{lpo} = (V, <, l)$ with $l : V \to T$ is called a run of N if $m_0(p) + \sum_{v \in V \wedge v < C} (W(l(v), p) - W(p, l(v))) \geq \sum_{v \in C} W(p, l(v))$ for every cut C of lpo and every place p. The set of runs of a p/t-net N is defined by $\mathfrak{Lpo}(N) = \{(V, <, l) \mid (V, <, l)$ is a run of $N\}$.

A p/t-net N synthesized from a specified composed run $\mathbf{R}$ by the algorithm presented in [6] is a best upper approximation to $\mathbf{R}$ in the sense that

- $\mathfrak{Lpo}(\mathbf{R}) \subseteq \mathfrak{Lpo}(N)$ and
- $\forall (N') : (\mathfrak{Lpo}(\mathbf{R}) \subseteq \mathfrak{Lpo}(N')) \Longrightarrow (\mathfrak{Lpo}(N) \subseteq \mathfrak{Lpo}(N'))$.

Synthesizing an upper approximation is useful, because the behaviour explicitly specified by $\mathbf{R}$ should definitely be included in the behaviour of the synthesized model. The best upper approximation property ensures that only necessary additional behaviour is added to the synthesized net. Thus, computing a best upper approximation may be seen as a natural completion of the specified behaviour $\mathbf{R}$ by a Petri net.

After the generation of a process model in this way, in a follow-up validation step runs of the synthesized net which have not been specified are visualized. An expert is interactively asked whether these runs are legal or not. In the positive case, they are added to the specification. In the negative case, changes of the net to prohibit such runs are proposed to the expert. These changes always have the problem that the changed net does not anymore allow all specified runs. But it is possible to automatically compute such changes which prohibit a minimal number of specified runs. Additionally, since the specified behaviour is often incomplete and also the synthesized net tends to be incomplete, reasonable continuations of runs of the net are generated following certain heuristics and presented to the expert. The applied technique is deduced from the concept of wrong continuations [14,8,11]. Runs of the net are extended by appropriate events, and it is asked whether such additional runs model intended behaviour or not. To only propose a reasonable choice of such possible additional runs, different heuristic criteria such as considering runs occurring only once as a wrong continuation of the specified behaviour or runs prohibited by certain places can be applied. If such runs are desired, the net is changed accordingly and the runs are added to the set of runs specifying the process.

Finally, further heuristics to improve the readability of the net, as e.g. known from process mining [9], as well as partial order based validation techniques, as e.g. supported by VipTool [4], and verification techniques can be applied to further improve the process model.

5 Hierarchy

For large processes, knowledge about the process and its behaviour is often distributed in several involved people's minds. Some domain experts might have knowledge about the general process where single activities are on a high level of abstraction and have to be refined. Providing runs of this process leads to a corresponding model. Other people might know the behaviour of some details of the process, i.e. about the refinement of an activity of the main process, which defines a subprocess.

The paper [12] deals with synthesis of process models from this kind of distributed knowledge on process behaviour on different abstraction levels. Its results are mainly based on the observation that for partial order behaviour (in contrast to sequential order behaviour) subprocesses and the main process can first be synthesized independently and then be integrated. This section provides the core idea of this approach.

In the underlying design procedure, a special class of Petri nets is considered to model processes: connected p/t-nets with two distinguished sets of input and output transitions. In Figure 7 such nets are shown, where the input (output) transitions are depicted with two ingoing (outgoing) arcs.

Actually, in [12] no arc weights are considered. Moreover, the nets are required to have a certain 1-boundedness property (no reachable marking assigns more than one token to a place). The aim is that input and output transitions strictly alternate. This property ensures that the refinement step of the design procedure is correct. In [14], we show how 1-boundedness can be guaranteed by the considered synthesis algorithm. Nevertheless, in the present paper we consider the definition of processes based on general p/t-nets, because the synthesis algorithm is a lot more efficient in this case [15,10,8] (it is possible to apply fast standard linear programming techniques, while in the case of 1-boundedness integer linear programming methods are needed). To still allow the *refinement* procedure for transitions from [12], it has to be ensured that never a second instance of some subprocess is started (by an initial transition) before a prior instance of the same subprocess is finished (by a final transition). This can be achieved by adding to a transition t, before it is refined, a self-loop place p_t with $W(p_t, t) = W(t, p_t) = m_0(p_t) = 1$. Refining t by a subprocess, the place p_t has an outgoing arc with weight one to every input transition of the subprocess and an ingoing arc with weight one from every output transition of the subprocess (see Figure 7). Now we can guarantee the desired property by requiring that in the subprocess extended by p_t, input and output transitions can only occur strictly alternatingly, i.e. it is not possible that two input transitions occur without an intermediate output transition and vice versa. To avoid deadlocks, it additionally has to be required that from each reachable marking of the subprocess extended by p_t, there is a firing sequence including an output transition.

Formally, the refinement steps in our setting are defined as follows: We consider one main process. A subprocess refines a transition (to which a self-loop place was added)

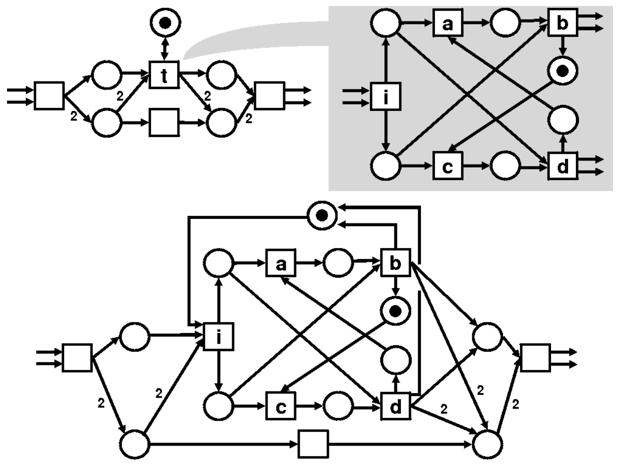


Fig. 7. Top: Two abstract processes. Bottom: The refinement of the left process w.r.t. t and the right process.

which either belongs to the main process or to another subprocess. It replaces the transition, the transition's input places are connected to the input transitions of the subprocess with arcs having the same weights, whereas the output transitions are connected to the output places of the transition by arcs having the same weights (see Figure 7). If we require the above behavioural restrictions for the subprocess, the external behaviour of the subprocess resembles the behaviour of the transition, with the difference that first the input tokens are consumed and later the output tokens are produced. That means, the behaviour of the main process is preserved when refinement is applied. The order of refinements does not matter.

Definition 5 (refinement). *A process net is a connected p/t-net* (P, T, W, m_0) *with two distinguished sets of input and output transitions* $T_i, T_o \subseteq T$.

Let $N = (P, T, W, m_0)$ *be a process net with a transition t and let* $N^t = (P^t, T^t, W^t, m_0^t)$ *be a process net with input transitions* T_i^t *and output transitions* T_o^t*. Assume w.l.o.g. that the elements of P and of* P^t *as well as of T and of* T^t *are disjoint. The refinement of N w.r.t. transition t and process* N^t *is defined as* $(P \cup P^t, T \cup T^t \setminus \{t\}, W_{N,N^t}, m_0 \cup m_0^t)$*, where* W_{N,N^t} *is defined by*

$$W_{N,N^t}(x,y) = \begin{cases} W(x,y) & \text{if } x,y \in P \cup T \setminus \{t\}, \\ W^t(x,y) & \text{if } x,y \in P^t \cup T^t, \\ W(x,t) & \text{if } x \in \bullet t \wedge y \in T_i^t, \\ W(t,y) & \text{if } x \in T_0^t \wedge y \in t\bullet, \\ 0 & \text{otherwise} . \end{cases}$$

It only remains to tune the considered synthesis algorithm to the definition of process nets. The sets of input and output transitions have to be regarded in the synthesis algorithm to create reasonable process models.

Therefore, first the sets of input transitions T_i and output transitions T_o have to be defined. To ensure that a process starts with an input transition, finishes with an output transition and in between no input and output transitions occur, we require for the specification given by the composed run $\mathbf{R}$ that

• every non-empty LPO $(V, <, l)$ in the set of runs $\mathfrak{Lpo}(\mathbf{R})$ of $\mathbf{R}$ contains exactly one event $v_0 \in V$ (called initial event) labelled by an input transition of the process $(l(v_0) \in T_i)$,

• the initial event v_0 of every non-empty LPO $(V, <, l) \in \mathfrak{Lpo}(\mathbf{R})$ is a unique minimal event, i.e. it fulfills $v_0 < v$ for every $v \in V \setminus \{v_0\}$,

• every LPO $(V, <, l)$ in the set of runs $\mathfrak{Lpo}(\mathbf{R})$ of $\mathbf{R}$ which is not prefix of another LPO in $\mathfrak{Lpo}(\mathbf{R})$, called maximal LPO of $\mathfrak{Lpo}(\mathbf{R})$, contains exactly one event $v_{max} \in V$ (called final event) labelled by an output transition of the process $(l(v_{max}) \in T_o)$,

• the final event v_{max} of every maximal LPO $(V, <, l) \in \mathfrak{Lpo}(\mathbf{R})$ is a unique maximal event, i.e. it fulfills $v < v_{max}$ for every $v \in V \setminus \{v_{max}\}$.

With these requirements, it is ensured that a net synthesized from $\mathbf{R}^*$ ($*$ is considered because a subprocess can be invoked arbitrarily often) fulfills the above behavioural requirements for subprocesses.

In the running example of the business process of an insurance company, the only input transition is "Receive Claim" and the only output transition is "Complete Claim". Therefore, the formulated requirement is fulfilled by our example composed run shown in Figure 4.

Figure 7 shows an example of a subprocess net refining a transition of a main process. Notice that this subprocess is equipped with a memory feature: In the first invocation of the subprocess, only the sequence *cd* is executable, in a second invocation only *ab* is possible, in a third one again *cd*, and so on. This memory feature was not possible if we would expect the subprocess to start with the empty marking, as it is the case for workflow nets [1].

6 Case Study and Tool Support

In this section we briefly illustrate the proposed synthesis procedure by the small case study which was already used for examples. We recently implemented appropriate synthesis features into our Petri net toolset VipTool, which offers a flexible xml-based open plug-in architecture. New plug-ins of VipTool allow to graphically specify a composed run and then to automatically synthesize a net from such specification. The refinement aspects of the business process design procedure of [12] are not yet supported by VipTool. Thus we consider the generation of a main process to show the applicability of the implemented synthesis algorithm.

The new editing functionalities for composed runs allow to compose LPOs drawn with other plug-ins of VipTool by respective composition operators. The composition of LPOs is graphically supported by a visualization of composed runs by block structures as introduced in this paper (compare Figure 4). The new editor also offers an alternative visualization in the form of UML activity diagrams (see Figure 8). The new algorithm to synthesize a net from a composed run is described in [16]. It combines the idea of wrong continuations of LPOs [14,11], which proved to yield nice synthesis results, with the notion of regions introduced in [6]. A synthesized net can be loaded, visualized, layouted, edited and analyzed by other plug-ins of VipTool.

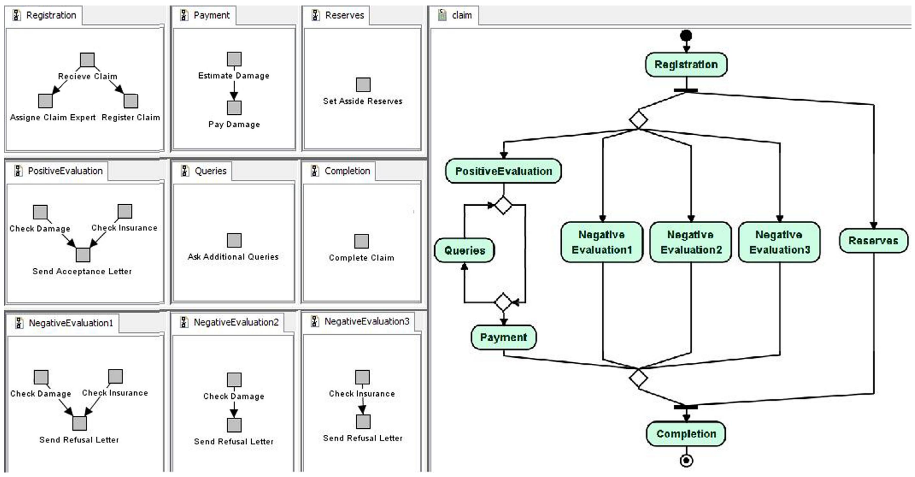
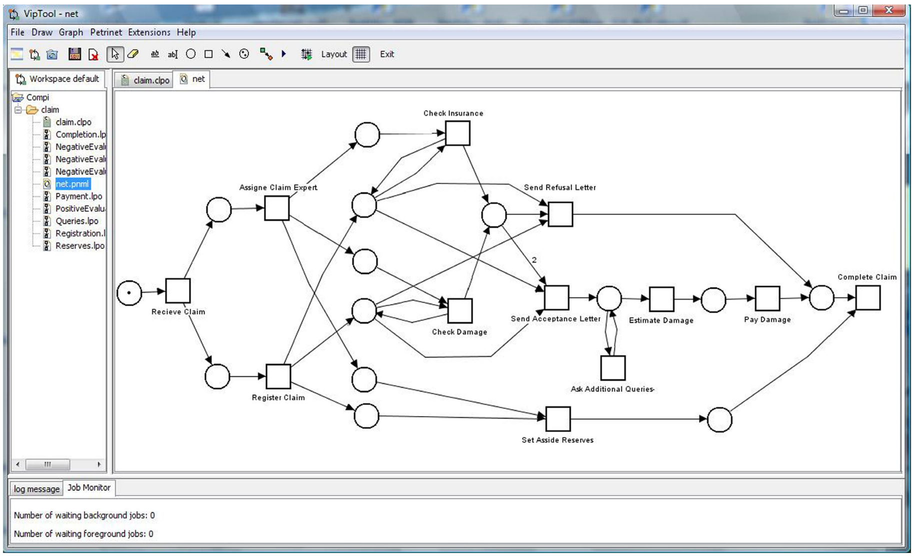


Fig. 8. Screenshot of VipTool showing the composed run from Figure 4 represented as an activity diagram and the corresponding run segments drawn in VipTool

Figure 8 shows the specification of the running example depicted as an activity diagram in VipTool. The possible runs of this workflow have been depicted in Figures 2 and 5. The workflow starts by receiving a claim submitted by a client, followed by two concurrent activities ”Assign Claim Expert” and ”Register Claim” (segment ”Registration”). The first one models the assignment of a claim expert in charge for this claim, the latter is concerned with the registration of the client and the loss form. Then, concurrently reserves for the claim are established (segment ”Reserves”) and the evaluation of the claim is started. The evaluation comprises four alternative runs. Each begins with two concurrent activities ”Check Damage” and ”Check Insurance”. ”Check Insurance” represents checking validity of the clients insurance, ”Check Damage” models checking of the damage itself. The segment ”Positive Evaluation” models the situation that both checks are evaluated positively, meaning that an acceptance letter is sent after the two checks. If one evaluation is negative, the company sends a refusal letter. Thus, the activity ”Send Refusal Letter” is performed after the two ”Check” activities if one is evaluated negatively (segment ”Negative Evaluation 1”). If a negative evaluation of one ”Check” activity already causes sending a refusal letter, while the other ”Check” activity has not been performed yet, this second ”Check” activity has to be disabled (i.e. it does not occur in a respective run), since it is no more necessary (segments ”Negative Evaluation 2” and ”Negative Evaluation 3”). In the case of a positive evaluation, either the damage is immediately estimated and payed (segment ”Payment”), or before the damage is estimated additional queries to improve estimation of the loss (segment ”Queries”) are repeatedly asked until sufficient information is collected. If the evaluation of the claim (including possibly paying the damage) and the segment ”Reserves” are finished, the process can be completed by the segment ”Completion”.

Figure 9 shows the net automatically created with the synthesis algorithm of VipTool from the specification depicted in Figure 8 (actually the synthesis algorithm generated



```
Number of waiting background jobs: 0
Number of waiting foreground jobs: 0
```

Fig. 9. Screenshot of VipTool showing the user interface of the editor for Petri nets

two more places which have been deleted by a plug-in of VipTool searching for implicit places). Although the net seems to be complex on the first glance, it represents a very appealing model of the described business process. In fact, the net exactly has the specified behaviour (no additional, not specified runs) and there is no more compact possibility to describe the complex control flow of this business process by a Petri net.

The example illustrates that directly designing a Petri net model of a business process is often challenging, while modelling runs and synthesizing a net is more easy. Manually developing a complex Petri net such as our example net for the described workflow is an error-prone task.

7 Discussion

In the field of software engineering, the main approaches to system modelling have been structured analysis and structured design, developed in the late 1970's, as well as object-oriented analysis and design, starting in the late 1980's [17]. In the 1990's it was recognized on a broad front that requirements engineering – the elicitation, documentation and validation of requirements - is a fundamental aspect of software development and requirements engineering emerged as a field of study in its own right. Scenarios, firstly introduced by Jacobson's use cases [18], proved to be a key concept for writing system requirements specifications. Important advantages of using scenarios in requirements engineering include the view of the system from the viewpoint of users, the possibility to write partial specifications, the ease of understanding, short feedback cycles and the possibilities to directly derive test cases [19,20]. Modelling software systems by means of scenarios received much attention over the past years. The dozens of popular scenario notations including e.g. the ITU standard of Message Sequence

Charts, UML Sequence Diagrams, UML Communication Diagrams, UML Activity Diagrams and UML Interaction Overview Diagrams as well as Live Sequence Charts, are an evidence for this development. Several methodologies to bridge the gap between the scenario view of a system and state-based system models, which are closer to design and implementation, have been proposed [21,22]. There are analytic, fully-automated synthetic and interactive synthetic software engineering methods to construct design models from scenarios. In a more radical approach [17] it is even suggested that a scenario specification may be considered not just the system's requirements but actually its final implementation.

In this paper we suggested focusing on scenarios to design business processes. Looking at scenarios to specify the requirements of a business process has similar advantages as in the software engineering domain. We developed a comprehensive methodology to model business processes which on the one hand regards the specifics of business processes and on the other hand exploits the benefits of scenario modelling. The approach starts by specifying example scenarios of the process in terms of (composed) runs and establishing respective necessary preconditions and relationships. Then a fully-automated synthesis algorithm generates a Petri net model of the process from the example scenarios which is afterwards interactively adjusted. In the domain of business processes modelling such construction methodology of process models focusing on example scenarios is an innovative approach. Most of the methods known from software engineering are not suited for business process design, because there are several differences which have to be regarded. E.g., in software modelling [17,22,21] the focus is on components or objects, communication (dependencies) between components and the distinction between inter- and intra-object behaviour, while in business process modelling [1,2,3] the emphasis is on global activities, dependencies through pre- and post-conditions of activities, and resources for activities. Modularity comes into play by appropriate refinement and composition concepts. In business process modelling so far (partially ordered) scenarios have only rarely been used and their application was restricted to analysis of process models, e.g. [4,23]. An exception, where scenarios are directly used in the design phase, is process mining [9]. But usually process mining is very much adjusted to scenarios of event logs and restricted to sequential scenarios. Nevertheless, there is one approach, called multi-phase mining [24,25,26], where the final phase [25] algorithmically generates a process model from a finite collection of arbitrary partially ordered scenarios given e.g. by instance EPCs [25] or message sequence charts [26] (by directly translating dependencies from the scenarios to the process model). In contrast to this approach, we build our modelling methodology on formal methods known from Petri net synthesis [8,6]. The advantage compared to [24,25] is that our approach generates reliable results for all kinds of specifications (also in the presence of very complex routing structures). However, our techniques may be inferior w.r.t. performance issues.

8 Future Research

Results from practical application and evaluation will be important for further development of our methodology. The suggested approach to design business processes is based on several assumptions, but we believe that it could be helpful in many cases. However, this research is still in an initial phase and we do not have experiences from real applications. Future work includes defining of success criteria and empirical research. In particular, it would be interesting to identify and characterize settings in which our approach is superior to other approaches.

References

1. Aalst, W., Hee, K.: Workflow Management: Models, Methods, and Systems. MIT Press, Cambridge (2002)
2. Aalst, W., Dumas, M., Hofstede, A.: Process-Aware Information Systems – Bridging People and Software. Wiley, Chichester (2005)
3. Weske, M.: Business Process Management – Concepts, Languages and Architectures. Springer, Heidelberg (2007)
4. Desel, J., Juhás, G., Lorenz, R., Neumair, C.: Modelling and Validation with Viptool. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 380–389. Springer, Heidelberg (2003)
5. Bergenthum, R., Desel, J., Juhás, G., Lorenz, R.: Can I Execute My Scenario in Your Net? Viptool Tells You! In: Donatelli, S., Thiagarajan, P.S. (eds.) ICATPN 2006. LNCS, vol. 4024, pp. 381–390. Springer, Heidelberg (2006)
6. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Synthesis of Petri Nets from Infinite Partial Languages. In: ACSD 2008, pp. 170–179. IEEE, Los Alamitos (2008)
7. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Synthesis of Petri Nets from Finite Partial Languages. Fundam. Inform (to appear)
8. Badouel, E., Darondeau, P.: Theory of Regions. In: APN 1998. LNCS, vol. 1491, pp. 529–586. Springer, Heidelberg (1996)
9. Aalst, W.: Finding Structure in Unstructured Processes: The Case for Process Mining. In: ACSD 2007, pp. 3–12. IEEE, Los Alamitos (2007)
10. 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)
11. Bergenthum, R., Mauser, S.: Comparison of Different Algorithms to Synthesize a Petri Net from a Partial Language. In: Workshop CHINA, Petri Nets 2008, X'ian (2008)
12. Desel, J.: From Human Knowledge to Process Models. In: Kaschek, R., Kop, C., Steinberger, C., Flirdl, G. (eds.) UNISCON 2008. LNBIP, vol. 5, pp. 84–95. Springer, Heidelberg (2008)
13. Glinz, M.: An Integrated Formal Model of Scenarios Based on Statecharts. In: Botella, P., Schäfer, W. (eds.) ESEC 1995. LNCS, vol. 989, pp. 254–271. Springer, Heidelberg (1995)
14. Lorenz, R., Juhás, G., Mauser, S.: How to Synthesize Nets from Languages - a Survey. In: Wintersimulation Conference 2007, Washington, pp. 638–647 (2007)
15. Lorenz, R., Bergenthum, R., Desel, J., Mauser, S.: Synthesis of Petri Nets from Finite Partial Languages. In: ACSD 2007, pp. 157–166. IEEE, Los Alamitos (2007)
16. Bergenthum, R., Mauser, S.: Synthesis of Petri Nets from Infinite Partial Languages with VipTool. In: AWPN 2008, Rostock, pp. 81–86 (2008)
17. Harel, D., Marelly, R.: Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine. Springer, Heidelberg (2003)
18. Jacobson, I.: Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley, Reading (1992)
19. Glinz, M., Seybold, C., Meier, S.: Simulation-Driven Creation, Validation and Evolution of Behavioral Requirements Models. In: MBEES 2007, Dagstuhl, pp. 103–112 (2007)

20. Glinz, M.: Improving the Quality of Requirements with Scenarios. In: Second World Congress on Software Quality, Yokohama, pp. 55–60 (2000)
21. Amyot, D., Eberlein, A.: An Evaluation of Scenario Notations and Construction Approaches for Telecommunication Systems Development. Telecommunication Systems 24(1), 61–94 (2003)
22. Liang, H., Dingel, J., Diskin, Z.: A Comparative Survey of Scenario-Based to State-Based Model Synthesis Approaches. In: SCESM 2006, pp. 5–12. ACM, New York (2006)
23. Scheer: IDS Scheer: ARIS Process Performance Manager, `http://www.ids-scheer.com`
24. Dongen, B., Aalst, W.: Multi-Phase Process Mining: Building Instance Graphs. In: Atzeni, P., Chu, W., Lu, H., Zhou, S., Ling, T.-W. (eds.) ER 2004. LNCS, vol. 3288, pp. 362–376. Springer, Heidelberg (2004)
25. Dongen, B., Aalst, W.: Multi-Phase Process Mining: Aggregating Instance Graphs into EPC's and Petri Nets. In: 2nd Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management, Petri Nets, Miami, pp. 35–58 (2005)
26. Lassen, K., Dongen, B., Aalst, W.: Translating Message Sequence Charts to Other Process Languages Using Process Mining. In: PNSE 2007, Petri Nets 2007, Siedlce, pp. 82–97 (2007)

Online Interaction Analysis Framework for Ad-Hoc Collaborative Processes in SOA-Based Environments

Hong-Linh Truong and Schahram Dustdar

Distributed Systems Group, Vienna University of Technology
{truong,dustdar}@infosys.tuwien.ac.at

Abstract. Today's collaboration in e-science and business environments is no longer limited to the boundary of a single organization. People belonging to different organizations collaborate together to achieve a common goal by establishing virtual teams. The current trend is to rely on SOA-based tools and services for virtual teams and their collaborative processes because SOA offers many technologies to simplify the integration and interoperability of services belonging to different organizations and to allow the user to easily access existing services and tools. In complex environments comprising distributed software services and people, we need to understand how people and software services interact in order to adapt activities and services to the change of their operating environments as well as to allow them to self-manage their behaviors during the collaboration. We observed that there is a lack of tools supporting the analysis of interactions in such ad-hoc collaborative processes at runtime. In this paper we present our VOIA(Vienna Online Interaction Analysis) framework which aims at analyzing interactions within collaborative processes in SOA-based environments. We present a comprehensive list of interaction metrics and patterns associated with ad-hoc collaborations and techniques used to determine these metrics and patterns. We discuss how metrics and patterns can be used in process adaptation and illustrate VOIA's capabilities with several experiments.

1 Introduction

Today's collaboration in e-science and business environments is no longer limited to the boundary of a single organization. People belonging to different organizations collaboratively work together to achieve a common goal. In this context, they establish a *team*, often a *virtual team* [1], and conduct a *collaborative process* implementing their common goal. The current trend is to rely on SOA (Service-Oriented Architecture) to implement tools and services for virtual teams and their collaborative processes because SOA offers many technologies to simplify the integration and interoperability of services belonging to different organizations and to allow the user to easily access existing services and tools. Examples of such SOA-based collaboration tools and services are the inContext [2], ECOSPACE [3] and COIN [4] systems. Given these systems, one important aspect is to understand how *people and software services interact* in order to adapt activities of people and software services to the change of their operating environments as well as to allow them to self-manage their behaviors during their collaboration. Hence, metrics and patterns associated with interactions are a valuable source of information. An interaction metric is a quantitative measure that can be used to characterize and evaluate how an individual service or human is involved in interactions with another service or human[1]. We consider an interaction pattern as a reoccurring structure of interactions that is analyzed from a set of interactions, for example, the interactions among a set of services might follow a one-to-many model [5].

While existing research has been focused on defining and detecting patterns in workflows [5,6,7,8,9,10,11], most of them concentrate on rigid, well-defined processes and workflows in businesses and aim at supporting offline workflow mining. Support for runtime analysis of patterns in dynamic collaboration environments has got little attention. In complex, dynamic collaboration environments, ad-hoc collaboration processes are not pre-defined; their dynamic, ad-hoc activities are defined on-demand. These activities may be combined with well-defined workflows and composition patterns, but not necessarily. In such environments, metrics and patterns characterizing the collaboration and its activities are relevant because they can provide valuable insights into the collaborative process to support runtime adaptation. However, existing offline mining techniques are not suitable because they are not designed (and cannot be tested) with evolving collaborative processes. Existing mining techniques typically require complete log data and do not deal with runtime aspects, such as runtime processing data from various services and runtime provisioning of interaction metrics and patterns. Furthermore, most existing work focus on either human-to-human interactions (e.g., social networks analysis) or service-to-service interactions (e.g., performance analysis or service interaction pattern analysis), whereas SOA-based collaboration environments include diverse types of interactions among humans and services that should be considered together.

Runtime analysis of interactions in such environments poses many research challenges. First, data is obtained and analyzed while the collaborative process just continues to evolve as new activities emerge. This requires us to deal with different types of events collected at different levels, such as activities, interactions and service-specific events. Secondly, metrics and patterns have to be determined for both humans and software services according to different needs of clients, such as determining metrics and patterns based on collaboration contexts, e.g., in individual, group or the whole collaboration levels, and on user-specific conditions, e.g., in particular time period and with a specific threshold. All these challenges imply that runtime analysis frameworks must be flexible and customized: new metrics and patterns analyses can be easily added and analysis requests are user-customized. Currently, there is a lack of such frameworks supporting tools for detecting metrics and patterns in collaborative work. Providing such metrics and patterns to clients at runtime is what motivates our work. In this paper, we contribute to techniques to support online interaction analysis for collaborative processes in SOA-based environments, namely (1) a classification of interaction metrics and patterns covering human-to-human, human-to-service, and service-to-service interactions at different levels, and (2) the design and implementation of a flexible and customizable software framework supporting runtime interaction analysis.

The rest of this paper is organized as follows: Section 2 presents related work. We describe a motivating scenario in Section 3. Preliminaries on terminologies and models are presented in Section 4. Section 5 describes a holistic view of interaction metrics and patterns. The architecture and implementation of VOIA is presented in Section 6.

[1] For metric definition, see http://en.wikipedia.org/wiki/Metrics, last access: 26 August 2008.

Interaction analysis techniques are detailed in Section 7. Section 8 presents experiments to demonstrate VOIA. Section 9 summarizes the paper, presents remaining issues, and gives an outlook to the future work.

2 Related Work

In our work, we develop techniques to determine metrics and patterns in collaborative processes carried out by teams. A collaborative process includes various activities which are associated with humans, artifacts, and software services. Our approach is to study the interaction at multiple levels of abstraction, from the way how humans use services, to how services interact with each other, and how a human interacts with another human through the utilization of services.

Some existing works analyze and define patterns for workflows and services [8,5] and (performance) metrics for workflows [12]. Understanding workflows and business processes has attracted a lot of research efforts. This includes, for example, research on performance monitoring and analysis of workflows [13,12] and on mining of workflow logs [6]. Tools and techniques for monitoring and analyzing performance of Web services focus on extracting logs comprising of service invocations and analyzing the logs to provide performance metrics. However, they focus on metrics associated with individual services, rather than with interaction patterns. When analyzing performance metrics associated services, we apply well-defined metrics from existing works, such as in [12], and focus on the analysis of patterns.

Recent work on mining workflow logs, especially work done by van der Aalst et al., has introduced several novel techniques to determine the relationship between humans and services. ProM [6] introduces many process mining features. Information can be shown, e.g., using social network and clustering. Several issues such as analyzing the conformance between a process model and its execution logs [14] are addressed. [15] discusses how to apply process mining to less structured processes in CSCW (Computer Supported Cooperative Work) systems. These works, however, do not support online analysis of dynamic collaborations. We note that although activity-based collaboration systems exist [16], they do not support the analysis of the collaboration.

In our previous works [17,11,10], we focus on the analysis of patterns in well-defined workflows and email activities. In [17], we have discussed the importance of mining patterns at multiple levels of abstraction. These papers addressed Web service logging, workflow structure discovery and session reconstruction. Patterns, like proxy and broker, and social network were presented and patterns, like proxy and broker, were detected from social networks. We have reused previous patterns. However, neither the analysis of patterns in collaborative work in SOA-based environments nor the online interaction analysis has been performed. In particular, the analysis of patterns in this paper differs from these works. Patterns are not detected from social networks but from streams of events during runtime. Furthermore, these works focus on the analysis of particular patterns, rather than provide a classification of patterns and metrics and a generic software framework which is flexible and customizable.

Recently, the concept of complex event processing (CPE) [18] has been utilized to analyze complex events in SOA environments. However, CPE frameworks, such as Esper [19], just provide fundamental tools for handling complex events and existing work focuses on detecting service behaviors through events. We utilize Esper for filtering and matching events and implementing primitive metrics and patterns analysis.

In this paper, we focus on analyzing patterns for collaborative work, rather than on the definition of patterns which are well addressed elsewhere [8]. One of the major differences between our work and existing work is that existing work focuses on offline analysis with the assumption that logs are produced by existing workflow systems. Our work differs as we focus on online analysis. Our assumption is that in modern collaboration, collaborative work is continuously being performed. Thus, interaction analysis tools are required during runtime.

3 Collaboration Scenario and Research Approach

In our work, we consider dynamic collaboration environments in which teams utilize different collaboration services for their collaborative work. Team members define and perform activities which require the involvement of many other services and humans. The upper part of Figure 1 illustrates the dynamic collaboration environment, for example, in the inContext project [20]. In such an environment, both humans and software services exist. Humans use services to perform their activities, while services can interact with each other to fulfil requests from humans. Services will follow the SOA model, thus they can be easily integrated together, providing seamless access virtually from anywhere. The SOA-based service model also simplifies the monitoring and maintenance of services, making the acquisition of multiple sources of log information at different levels in the widely distributed system possible and easier.

Using services, people perform their collaboration, of which processes are typically not modeled beforehand. Furthermore, the execution of collaborative processes is continuous and evolving, thus in many cases we will not know in advance when a process finishes. Interactions between humans and services are highly concurrent and distributed as multiple activities are executed in parallel. These activities involve services and people spanning different geographical locations and organizations. Therefore, offline analysis, which either requires complete, centralized information or the completion of the process in order to analyze the process, is not suitable. In addition, the need for adapting activities of and resources for collaboration is required at runtime, due to highly dynamics of modern collaborations (e.g., team member is often on the move). Thus, online analysis techniques are more suitable. The middle part of Figure 1 shows that the *Online Interaction Analysis* analyzes log events obtained from services and provides metrics and patterns back to the services in the dynamic collaboration environment.

However, online interaction analysis for such an environment is a very challenging task due to highly concurrent and distributed actions, such as concurrent executions of distributed services, concurrent interactions among services and humans, and concurrent requests from various clients. Which metrics and patterns associated with interactions are useful for optimizing collaborative work and resources used for the work at runtime? How can we correlate metrics and patterns from different levels of collaboration context, such as individual, group and the whole collaboration? How do we support the customization of metrics and patterns analysis so that different clients can

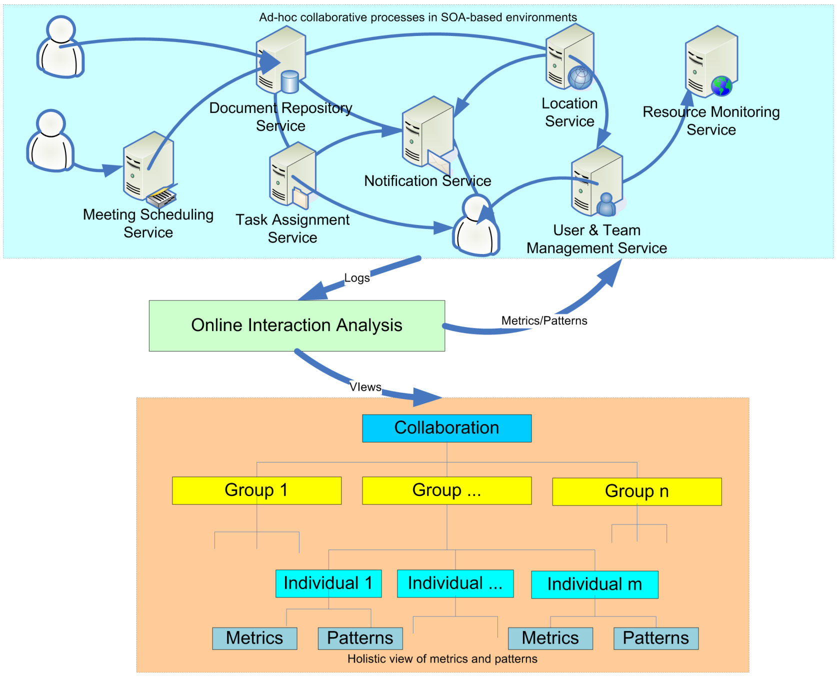

Ad-hoc collaborative processes in SOA-based environments
Document Repository Service
Location Service
Resource Monitoring Service
Notification Service
Meeting Scheduling Service
Task Assignment Service
User & Team Management Service
Logs
Online Interaction Analysis
Metrics/Patterns
Views
Collaboration
Group 1
Group ...
Group n
Individual 1
Individual ...
Individual m
Metrics
Patterns
Metrics
Patterns
Holistic view of metrics and patterns

Fig. 1. Online interaction analysis environment and metrics/patterns view

utilize the metrics and patterns differently? How to handle diverse events from various services? How to manage and provide these metrics and patterns to concurrent clients at runtime? We propose to provide a holistic view of patterns and metrics associated with services and interactions at multiple levels, ranging from the *individual* (services or human), to the *group* (of humans or services or the mix of them), and to the *whole collaboration*. The bottom part of Figure 1 shows the holistic view of interaction metrics and patterns that are associated with interactions. Then, we determine metrics and patterns based on different levels, client requests and periods of time, and provide and manage XML-based metrics and patterns at runtime.

4 Models

Before describing metrics and patterns and the analysis framework, in this section, we present our models of activities and services and humans in collaborations and data required for the interaction analysis.

4.1 Activity, Service, and People in Collaboration

In our model, *collaboration* is defined as a joint work between different people. Collaboration can be simple, e.g., including only a single task like `review of a document`, or complex, e.g., a real project. An *activity* describes a task of a collaboration, for example, `review a document`. An activity may consist of sub activities, but we do not distinguish between atomic activity and composite activity. An activity specifies various related information required for the execution of the activity. An *activity instance* represents all information associated with a particular execution of an activity; information includes, for example, start and end times, initiator, and associated service instances. An activity might be associated with a set of activity instances. Given an activity, we capture all changes in activities and activity instances in *activity events*.

A *collaboration service* is a service that is used in the collaboration; collaboration services are involved in the execution of activities. Collaboration services can be used to communicate between two team members, such as `Notification Service` and `Instant Messaging`, to host files such as `Document Repository Service`, and to store and manage activities such as `Activity Store Service`. Furthermore, there are middleware services which provide facilities for the operation of the collaboration such as `Service Registry` and `Logging Service`. In our model, we assume that services are well-defined based on the SOA principle. Most services are SOAP-based or RESTful (however, in our implementation, we tested only with SOAP-based services). *Service instance* is a particular deployment and running of a service in a particular hosting environment. A service invocation is a particular invocation of a service operation of a service instance. Information about service invocations is captured in *interaction events*. An interaction event consists of information related to the invocation, such as request and response message, service endpoint reference, consumer endpoint reference, etc., which are captured at the level of the hosting environment without the knowledge of service instances (e.g., by using SOAP intercepting mechanism). Furthermore, a service instance can also provide application-specific events about its operation. We call such events *service events*.

During a collaboration, people can initiate an activity, perform an activity or receive a message and handle the message sent by other people. In our model, we assume that a person defines a flow of activities that he/she has to perform. In our work, activities within a collaboration might be modeled in advance and executed in a pre-defined order or defined on-demand. Therefore, we do not assume that the structure of the collaboration and its execution order are known beforehand. Rather, we consider a collaboration to be a set of activities. The goal of this paper is to use online analysis techniques to detect metrics and patterns associated with interactions among humans and services in collaborations, but not on the detection of the process structure of the collaboration (like the work on discovering workflow structure from logs [21]). How to execute, manage and change the activities and the flow of activities are beyond of the scope of this paper.

4.2 Data for Interaction Analysis

To analyze collaboration processes, we rely on three sources of events: activity events, interaction events, and service events. In the following, we discuss the structure of data used in our online interaction analysis.

Each activity in our model is identified by a unique `activityURI`. When an actor performs an action to change the status of an activity, such as executing or delegating the activity, an event will be fired. In principle, we can use any activity model in collaboration, such as IBM UAM (Unified Activity Model) [16], Caramba [22] or the inContext Activity Model[2] to define activities. In our implementation, we use the inContext Activity Model which can be used to describe artifacts, involved people, and resources in detail. Given an activity event describing actions related to activities and activity instances, we assume that the activity event includes $activityURI$.

As mentioned before, an *activity event* describes changes in an activity and activity instance. This kind of event is captured by the `Activity Store Service` which maintains existing activities and by `Activity Execution Service` which executes the activity. Listing 1.1 presents a sample of an activity event which indicates that an activity has been created.

```
<activityEvent xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
actionURI="http://www.in-context.eu/ns/action/tCoordinationAction#6575"
activityURI="http://www.in-context.eu/Activity/Activity#226"
timestamp="1207840402595">
    <ExecutedByFoafAgent>
     http://www.vitalab.tuwien.ac.at/projects/incontext/TEST_LINH1#Martin
    </ExecutedByFoafAgent>
    <CoordinationType>
        <ActivityChangeType>Create</ActivityChangeType>
    </CoordinationType>
</activityEvent>
```

Listing 1.1. Example of an activity event

The second source of data used in the analysis is the *interaction event* which is captured in the Web services hosting environment. By using Web services handlers and SOAP interceptors, we can collect low level data of Web services invocations, such as SOAP messages. From this data, interaction events are generated and provided as an input for the analysis process. Activity events can be correlated to interaction events by using `activityURI` (for example, in the inContext project, the client modified the SOAP message header to include activity-related information). Listing 1.2 presents an example of an interaction event.

```
<InteractionEvent>
 <clientEndpoint>85.18.48.34</clientEndpoint>
 <messageCorrelationID>000a1460-25ba-4fa8-b766-9c3b50aa8c2b</messageCorrelationID>
 <messageType>Response</messageType>
 <serviceEndpoint>http://srvweb02.softeco.it/cgi-bin/SOAP.cgi/Eadt/Tasks/DocService
 </serviceEndpoint>
 <eventSourceID>AL-invoke@128.131.172.208</eventSourceID>
 <timeStamp>1207212091812</timeStamp>
</InteractionEvent>
```

Listing 1.2. Example of interaction event in which not all entries are available

The third source of events is *service event* which is provided by specific services. This type of events is optional and dependent on specific services. Since our goal is to provide a client-customized mining system, VOIA is designed to accept specific service events. However, the analysis of the patterns and metrics related to service-specific events are left to the client.

[2] http://www.in-context.eu/uploads/files/20070530D4.2v1.0Design20and20implementation20of20Context20Tunnelling.pdf

5 A Holistic View of Interaction Metrics and Patterns

Understanding interactions among humans and services in collaborative work highlights characteristics of not only individual humans and services but also of groups of them as well as the whole collaboration. For providing useful information in understanding interactions and in adapting the collaboration based on interactions, we focus on defining and providing quantitative information associated with interactions.

We classify three kinds of interactions associated with humans and services within the dynamic collaboration environments:

– *Service-to-service interaction*: is the interaction between two services, e.g., a service s_i calls another service s_j.
– *Human-to-service interaction*: is the interaction between a human and a service, e.g., how services are selected and used by a human. By saying human-to-service interaction, we mean a person needs and uses a service for his/her activities.

Table 1. Examples of interaction metrics and patterns for service-to-service

Level	**Metric/Pattern Name**	**Description**
Individual	ExecutionTime	The average execution time of a service.
	NumServiceCalls	The number of invocations of a service.
	NumUnavailableCalls	The number of times unavailability.
	NumFailureCalls	The number of failures.
	NumConsumers	The number of consumers that call a service.
Group	ServiceInteraction	The interaction between two services, $ServiceInteraction(s_i, s_j)$, represents the number of times that service s_i calls service s_j.
	UsageDistribution	The percent of usages distributed among services.
	UsageIsolatedPattern	Reflect whether a service is typically used in an isolated manner. Let S_j be a set of services invoked by a service s. Service s is in an isolated manner when $count(S_j) < \tau$ where $ServiceInteraction(s, s_j) > 0, \forall s_j \in S_j$. τ is a user-defined threshold and $count(S_j)$ is the number of services in the set S_j .
	UsageCompositePattern	Reflect whether a service is typically used together with other services in activities. Let S_j be a set of services invoked by a service s. Service s is in a composite manner when $count(S_j) > \tau$ where $ServiceInteraction(s, s_j) > 0, \forall s_j \in S_j$, τ is a user-defined threshold.
	OneToManyPattern	Related to one-to-many invocation, also called as one-to-many send, multicast, or scatter [5]. Let S_j be a set of services invoked by a service s. Service s and S_j are in this pattern when $ServiceInteraction(s, s_j) > \tau$, $\forall s_j \in S_j$, within a period of time t. t and τ are user-defined values.
Collaboration	as in the Group level	Similar to those in the group level but they are determined based on all information available in the collaboration.

Table 2. Examples of human-to-service interaction metrics and patterns

Level	Metric/Pattern Name	Description
Individual	UsageTime	Describe how much time that a human uses a service.
	NumHumanServiceCalls	Number of service invocations initiated by a human.
	TypicalServiceUsageTime	Identify the typical usage time that a human uses a service by eliminating outliers and calculating the mean value of a data set of UsageTime for the service.
	UsageCompositePattern	Reflect whether a set of services is used together by a human. Services $\{s_1, \cdots, s_n\}$ are considered in this pattern when they are called by a human h in a predefined period of time.
Group	HumanServiceInteraction	The interaction between a human and a service, $HumanServiceInteraction(h, s)$, represents the number of times that human h calls service s.
	UsageDistribution	Given a human, it reflects the usage distribution among services he/she calls.
	DurationUsage	Determine the typical usage time that a human uses a service.
Collaboration	as in the Group level	Similar to those in the group level but they are determined based on all information available in the collaboration.

– *Human-to-human interaction*: is the interaction between human and human, e.g., how a team member interacts with another one in order to perform activities. We determine interactions between two persons only by means of analyzing services they use in their communication and collaboration, e.g., `Notification Service`.

This classification differs from other works which focus on either human-to-human interactions (social networks) or service-to-service interactions by considering all types of interactions among humans and services. This consideration is necessary as these types of interactions are inherent in collaboration processes. Metrics and patterns are associated with these types of interactions. However, unlike other works which typically do not consider global versus local views on metrics and patterns, in our work, for each type of interaction, metrics and patterns are determined for specific time period at three levels: *individual* (for individual human or service), *group* (a team or a set of services), and *collaboration* (all available services and humans within a collaboration). This way takes into account the context of the collaboration in determining the metrics and patterns. The main reason is that interactions of a particular human or service are dependent on the context of the collaboration, such as a human might typically act as a proxy in a group but not in another group or within a set of services, a service might be well utilized, but not in the global view. Therefore, metrics and patterns characterizing the interactions should be determined differently. The metrics and patterns associated with three types of interactions at three levels provide a holistic view of interaction metrics and patterns in VOIA. This view allows us to utilize metrics and patterns differently, dependent on specific purposes and contexts.

Table 3. Examples of interaction metrics and patterns for human-to-human interaction

Level	**Metric/Pattern Name**	**Description**
Individual	NumCallers	Number of callers.
	NumCallees	Number of callees.
	NumInteractions	Number of interactions.
	NumAssignedActivities	Number of assigned activities.
	NumDelegatedActivities	Number of delegated activities.
	BrokerIndicator	Determines the broker role of an individual.
	ProxyIndicator	Determines the proxy role of an individual.
	MasterIndicator	Determines the master/slave role of an individual.
Group	TotalInteractions	Total number of interactions.
	AvgNumCallers	Average number of callers in a group.
	AgvNumCallees	Average number of callees in a group.
	AvgNumHumanInActivity	Average number of human involved in an activity.
	BrokerPattern	Related to broker pattern [10], including number of broker patterns, the structure of the broker pattern.
	ProxyPattern	Related to proxy patterns [10], including number of proxy patterns,the structure of the proxy pattern.
	MasterSlavePattern	Related to master/slave pattern [10], including number of master/slave patterns, the structure of the master/slave pattern.
	CoauthoringPattern	Related to co-authoring pattern, including the structure of co-authoring pattern. Two members h_i and h_j are in a coauthoring pattern when they both work on the same n activities, n is a user-defined value.
Collaboration	as in the Group level	Similar to those in the group level but they are determined based on all information available in the collaboration.

Note that the list of patterns and metrics is non exhaustive and many of them are common, well-understood and presented in literature [17,11,10,6,8,5,12]. Depending on specific needs, these patterns and metrics are utilized differently. However, VOIA aims at providing a rich catalog of metrics and patterns suitable for different clients. We would like to stress that the main objective of VOIA is not to define patterns and metrics but provide a framework which the determination of new patterns and metrics can be easily plugged in.

Table 1 presents metrics and patterns associated with service-to-service interactions. Many performance metrics at individual level are well-defined, e.g. in [12]. At the group level, we determine many novel patterns.

Table 2 describes metrics and patterns related to human-to-service interactions. Metrics and patterns are determined by analyzing humans who initiated or invoked services in their activities.

Table 3 describes metrics and patterns related to human-to-human interactions. In particular, we focus on metrics and patterns associated with broker, proxy, master/slave interactions.

6 VOIA Architecture and Implementation

Figure 2 depicts the architecture of VOIA which includes many components and provides Web services interfaces. In our architecture, the *Event Preprocessing* of VOIA accepts different types of events, such as `Activity Event`, `Service Event`, and `Interaction Event`, and pre-processes these events suitable for VOIA's *Metric/-Pattern Detection*. These events are collected at different services, such as `Activity Service`, `Logging Service`, `Communication Service`, and `Access Layer`. As mentioned before, events consist of data collected at different levels, ranging from Web middleware/container level (by using Web Handlers and SOAP interceptors, such as `Interaction Event`) to application level (by capturing application logs, such as `Service Event`).

Events processed by *Event Preprocessing* are passed to *Metric/Pattern Detection* which determines primitive metrics and patterns. The *Metric/Pattern Detection* analyzes events based on a set of pattern specifications and templates stored in *Pattern Specification and Template* to identify which metrics or patterns occurred. Such determined metrics and patterns are passed to the *Interaction Analysis* which provides high level analysis of patterns and metrics for interactions. During runtime, clients of VOIA can specify pattern specifications and submit the specifications to the VOIA service which informs the clients when patterns/metrics met the specifications exist. Thus, the clients can utilize the resulting metrics and patterns for online adaptation. The end-user can use the VOIA portal to manage VOIA and explore results produced by VOIA.

We have implemented the core of VOIA and provide it as a Web service based on JAX-WS[23]. The Web service provides fundamental interfaces for other clients to send events and pattern specifications and templates. VOIA can subscribe other services, such as `Logging Service`, to be informed with events or any services can send events to VOIA via a pre-defined interface. The *Metric/Pattern Detection* employs the Esper engine [19] to process events based on pattern expressions.

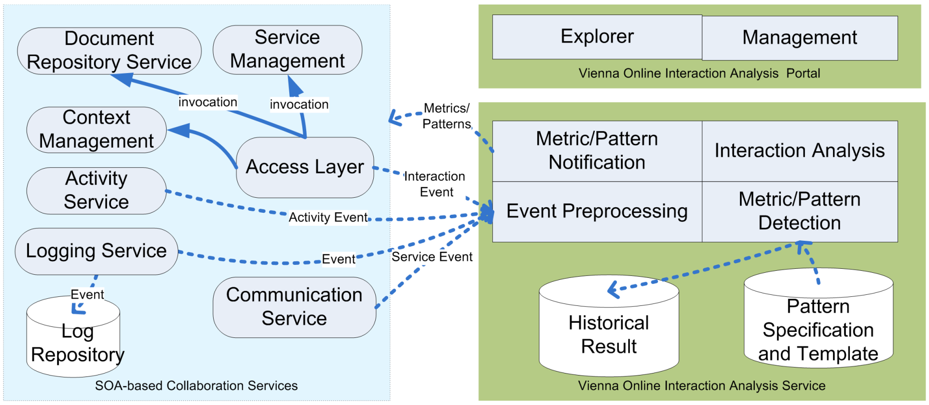


Fig. 2. Vienna Online Interaction Analysis (VOIA)

7 Analysis Techniques

The fundamental difference between our approach and existing ones is that our interaction analysis is conducted online. The metrics and patterns discussed in Section 5 are provided by either *Metric/Pattern Detection* or *Interaction Analysis* components, depending on the complexity of the analysis. The first component is used to determine primitive metrics and patterns which can be identified by applying pattern specification and template directly. The latter implements complex analyses which require different algorithms besides the primitive metrics and patterns.

As the data used for the analysis concurrently arrives in a stream of events, primitive patterns and metrics will be determined on the fly. For any analysis, a time window - specifying a period of time - or space window - specifying a number of events - or a combination of time and space windows will be specified. Together with the level (individual, group, or the collaboration), they determine the context of the analysis.

7.1 Primitive Metric and Pattern Detection

The *Metric/Pattern Detection* utilizes a set of predefined pattern specification and pattern templates in order to determine relevant metrics and patterns. We have defined several pattern specifications and templates based on that well-known metrics and patterns can be determined. Each pattern specification or template is described by

– *Pattern name*: is used to identify a pattern specification or template
– *Result handler*: is used to handle the result of a pattern specification or template.
– *Pattern expression*: is used to filter events and determine primitive metrics and patterns. It is described in Event Processing Language (EPL) [24].

All pre-defined pattern specifications and templates are stored in *Pattern Specification and Template* in XML form. A new pattern template can be defined by specifying the above-mentioned information. Unless we need specific treatment for pattern result handler, generic handler can be used. A result handler actually implements the algorithm to determine the corresponding pattern/metric. Result handlers adhere a pre-defined interface defined by VOIA so that a new handler can be plugged into VOIA. Note that VOIA provides mechanism for writing handlers.

Given a pattern specification/template, VOIA will create an EPL statement and a result handler object using a reflection mechanism. When events arrive to VOIA, in the *Event Preprocessing* component, a CEP engine which is developed atop Esper will use pattern expressions to process as well as to filter relevant events. Then, the engine passes processed events to corresponding result handlers. Depending on pattern expressions, some primitive patterns can be detected in the *Event Preprocessing* whereas the result handler will process patterns and metrics and provide results. The result is described in XML. For primitive metrics and patterns, we provide a generic result handler that provides the result of a specification or template. Note that by *"primitive metrics and patterns"*, we mean metrics and patterns which can be directly determined by issuing EPL-based pattern specification and template to the *Event Preprocessing*. There are metrics and patterns as well as other high level information associated with interactions which need particular analysis besides EPL-based pattern specification and template.

To explain how primitive patterns/metrics can be detected, let us consider that a client would like to determine a `NumHumanServiceCalls` (how many time a human calls a service). Listings 1.3 and 1.4 describe one example of the corresponding pattern expression and the resulting metric. In Listing 1.3, an EPL-based request is used to select all humans (determined by `userID`) and services they use (determined by `serviceEndpoint`) from *interaction event* in the last 30 minutes and then to return only humans who call a service more than 10 times. The corresponding result is given in Listing 1.4. Although it is a simple example, various parameters can be customized to detect the metric for different purposes.

```
<pattern name="NumHumanServiceCalls" resulthandler="voim.NumberServiceCallHandler">
  select userID, count(serviceEndpoint) as NumHumanServiceCall, serviceEndpoint from
        InteractionEvent.win:time(30 min) group by serviceEndpoint, userID
        having count(serviceEndpoint) > 10
</pattern>
```

Listing 1.3. Example of query for determining `NumHumanServiceCall` when a human uses a service more than 10 times in last 30 minutes

```
     <userID>
         http://www.vitalab.tuwien.ac.at/projects/incontext/TEST_LINH1#Linh
     </userID>
     <serviceEndpoint>http://madrid.vitalab.tuwien.ac.at:8080/axis2/
             services/activityservice</serviceEndpoint>
     <NumHumanServiceCall>13</NumHumanServiceCall>
</NumHumanServiceCalls>
```

Listing 1.4. Example of a $NumHumanServiceCalls$ result

7.2 Complex Interaction Analysis

The use of *Metric/Pattern Detection* helps to identify several patterns and metrics at runtime. However, many high-level information cannot be obtained from this component, for example, social network of people or a network of service interactions. The *Interaction Analysis* is used to analyze interaction information which cannot be determined by using pattern specifications or templates directly. The *Interaction Analysis* utilizes information provided by *Metric/Pattern Detection* and provides high-level information. A result handler in *Interaction Analysis* component will receive events which are the output of *Metric/Pattern Detection* handlers and will analyze these events to provide interaction metrics and patterns. Table 4 presents some high level analyses provided.

7.3 Extensibility and Customization of Interaction Analysis

Extensibility and customization are two major requirements for VOIA as different analyses are required for different purposes and because in dynamic collaboration environments various types of events can be used for detecting metrics and patterns. There are two ways to provide new interaction mining analysis: by providing new pattern specification and template and by developing a new plugin. In the first case, the client of VOIA can utilize existing result handlers and focus on writing the specification and template that detect their interesting metrics and patterns. This way is particular useful when dealing with service-specific events. In this case, a new pattern specification and

Table 4. Example of high level interaction analyses

Name	Description
ServiceInteractionNetwork	Describe interaction network between services. We use $ServiceInteraction(s_i, s_j)$ to build the network of service interactions in which the node is a service, the edge is the interaction between two services.
HumanInteractionNetwork	Describe the social network between human. We use $NumCallees$, $NumCallers$ and $NumInteractions$ in human-to-human metrics to build the network. We can also further identify which is a typical service used for the interaction between two persons.
HumanInAllActivity	Describe a network mapping activities to humans. This can be used to detect different types of activities typically performed by a human. Detailed information can be, e.g., the type of activities that a person typically performs.

template can be submitted to VOIA service during runtime. VOIA is acting like a metrics and patterns processing engine. In the latter case, a new plugin can be developed and straightforwardly integrated into VOIA, by (1) providing a pattern specification or template, and (2) a result handler which implements a generic handler interface provided by VOIA. Based on that, a new entry for *Pattern Specification and Template* can be created and VOIA will execute the new plugin.

7.4 Managing and Providing Resulting Metrics and Patterns

To access analysis results during runtime, any client can query or subscribe the results hold in a result handler based on pattern name by invoking Web services operations. Results can be also sent to the client based on notification mechanism. Thus, clients can easily obtain mining results from our framework to perform runtime adaptation.

Given a pattern or metric detected at a specific time, typically such a metric and pattern will be delivered to the corresponding client who initiates the pattern specification/template. However, we also manage such pattern and metric for later use. The information is stored in *Historical Results*. As in online analysis, a result handler will have a new result when events meet pattern specification or template. First, each result handler will keep only n latest results; n is pre-defined. Instead of providing a big XML document including analysis results, we provide a collection of small documents which are also used as events to notify interested clients. Each result is associated with a timestamp. When a result is determined based on space (e.g., number of events in a window) or time (time period associated with a window), we associate the result with the time window to identify the valid period of the result. We do not merge results provided by a handler into a big document. Instead, the list of results will be stored into an XML database. Management features can be used to remove unneeded results. As many results are produced at runtime, aggregating them into a more meaningful information is challenging. Currently, we consider them only historical data and let the client analyze the historical results for its own purpose. For accessing historical data, clients just specify a time period together with an XQuery-based request and a pattern name.

8 Experiments

In this section, we present some experiments to illustrate how VOIA can provide useful metrics and patterns for understanding and adapting collaboration environments. Our experiments are based on the inContext testbed including various collaboration services such as `Document Repository Service`, `Notification Service`, and `Activity Service`. All of them are SOAP-based services. We gathered events from collaborations executed in the inContext testbed and analyzed the collected events. To test VOIA's functionalities and performance, we performed two experiments.

In the first experiment, we analyzed events produced by the inContext testbed. These events were generated from the usage of different services during the integration and testing of the inContext system. Thus, patterns might or might not reflect some use cases[3]. Most events in this experiment are *interaction events*. In the second experiment, we randomly generated a collaboration based on a tree of activities. The tree of activities is generated based on (1) a list of 11 users, a list of 3 roles, 5 levels of a tree, average number of activities per level is 5, a list of more than 20 real services, a list of possible *human activity handling strategies* including *delegate, split, perform, reject, assign*.

Figure 3 presents a snapshot of the service interaction network from the first experiment(for brevity, we removed the hosting URI from the graph). This network evolves during the runtime analysis. The information is provided in XML and exported into a dot format visualized by GraphViz[4]. From that information, we can determine how services were used and then devised service selection strategies. We then examined how humans use services. Listing 1.5 presents the result of human-to-service calls for $UserTest$ which is used to access data from two services. Of more than 20 Web services in the testbed, only `TeamService` and `DocumentService` are interacted with `UserTest`. It is due to the fact that these services were used by the user to acquire the information about other people and to store and search relevant documents. Such activities still involve heavily humans. The information obtained from this type of analysis can be useful, e.g., for determining typical services used by team members to improve runtime service provisioning strategies.

In the second experiment, we examined the proxy pattern. Listing 1.6 presents the result of proxy indicator pattern. Out of 971 activity events, 38 proxy cases were found. For each person involved, we can determine who typically acts as a proxy. This kind of analysis can be useful, for example, for selecting team members and determining trust in collaborative networks of enterprises.

To test the performance of the engine, we wrote a test client that reads events from local file systems and invokes the VOIA service. The test was performed in an Intel Centrino Duo Core 1.83 GHz, 2GB RAM, Windows XP notebook with 10 patterns. Overall, VOIA can handle a high volume of events in a time-responsive manner: it took 9035 ms to process 6775 interaction events (in the first experiment) and 2218 ms for 971 activity events (in the second experiment).

[3] The inContext system includes real services and a research system for the EU project inContext
[4] http://www.graphviz.org/

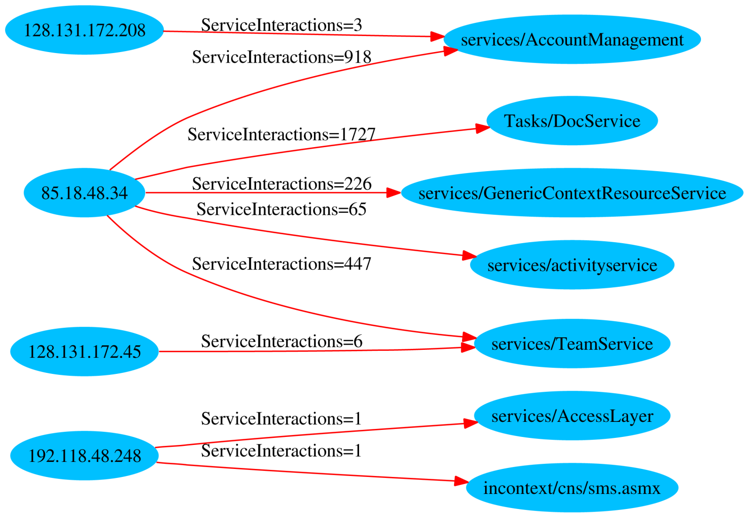


Fig. 3. A snapshot of service interaction network detected

```
<NumHumanServiceCalls>
  <entry>
    <userID>UserTest</userID>
      <serviceEndpoint>
        http://oslo.vitalab.tuwien.ac.at:8080/axis2/services/TeamService
      </serviceEndpoint>
      <NumHumanServiceCall>57</NumHumanServiceCall>
  </entry>
  <entry>
     <userID>UserTest</userID>
     <serviceEndpoint>
         http://srvweb02.softeco.it/cgi-bin/SOAP.cgi/Eadt/Tasks/DocService
      </serviceEndpoint>
     <NumHumanServiceCall>256</NumHumanServiceCall>
   </entry>
</NumHumanServiceCalls>
```

Listing 1.5. Example of human-to-service $NumHumanServiceCalls$ metrics

```
<ProxyIndicator total="38" at="13.04.2008␣15:00:21">
 <ProxyIndicator name="http://.../incontext/TEST_LINH1#Martin" value="7">
 <ProxyIndicator name="http://.../incontext/TEST_LINH1#Schahram" value="11">
 <ProxyIndicator name="http://.../incontext/TEST_LINH1#Vasko" value="2">
 <ProxyIndicator name="http://.../incontext/TEST_LINH1#Florian" value="2">
 <ProxyIndicator name="http://.../incontext/TEST_LINH1#Atif" value="3">
 <ProxyIndicator name="http://.../incontext/TEST_LINH1#Shariq" value="3">
 <ProxyIndicator name="http://.../incontext/TEST_LINH1#Lukasz" value="3">
 <ProxyIndicator name="http://.../incontext/TEST_LINH1#Christoph" value="3">
 <ProxyIndicator name="http://.../incontext/TEST_LINH1#Linh" value="2">
 <ProxyIndicator name="http://.../incontext/TEST_LINH1#Kamran" value="2">
</ProxyIndicator>
```

Listing 1.6. Example (simplified) of proxy-related metrics for people

9 Conclusions

Motivated by the lack of online interaction analysis tools in collaborative work and the need of acquiring insightful information for runtime adaptation of collaborative working environments, we presented VOIA (Vienna Online Interaction Analysis) framework which is capable of detecting and providing metrics and patterns associated with human and service in dynamic collaborations. The main contribution of our work is that we provide a Web service-based system that is capable of performing runtime analysis of interaction patterns and metrics. We have provided a rich view of metrics and patterns associated with interactions that spans different levels, including individual, group, and collaboration views, and that characterizes different types of interactions, including service-to-service, human-to-service, and human-to-human. Our system is flexible and customizable, allowing for the inclusion of new analysis and supports client-customized mining. If clients of VOIA understand the structure of their events, they can also define pattern specifications and templates for determining patterns and metrics at runtime. By supporting online analysis, VOIA can provide useful information for runtime adaptation in collaborative working environments.

Various future steps have to be done to fully support online interaction analysis of collaborative processes in SOA-based environments. We are working on determining trust in collaborations in networks of enterprises based on our proposed metrics and patterns. While online analysis allows the client to freely define the analysis they want, the challenge is how to manage the result of different analyses. Currently, we have just stored the result, thus advanced techniques for managing mining results will be studied. Our future work foresees to provide further testing of the systems and support more types of events. We will enhance the pattern specification and template catalog by incorporating new patterns. Another major effort is to perform runtime adaptation based on patterns that motivates this work, but has not been addressed in this paper.

Acknowledgements. We thank Christoph Dorn, Robert Gombotz and Daniel Schall for fruitful discussion on the interaction mining and their assistance in the implementation of VOIA framework. We thank anonymous reviewers for their fruitful and extensive reviews. This research is partially supported by the European Union through the FP6 project inContext and the FP7 project COIN.

References

1. Lipnack, J., Stamps, J.: Virtual teams: reaching across space, time, and organizations with technology. John Wiley & Sons, Inc., New York (1997)
2. Truong, H.L., Dustdar, S., Baggio, D., Corlosquet, S., Dorn, C., Giuliani, G., Gombotz, R., Hong, Y., Kendal, P., Melchiorre, C., Moretzky, S., Peray, S., Polleres, A., Reiff-Marganiec, S., Schall, D., Stringa, S., Tilly, M., Yu, H.: Incontext: A pervasive and collaborative working environment for emerging team forms. In: SAINT, pp. 118–125. IEEE Computer Society, Los Alamitos (2008)
3. ECOSPACE: eProfessionals Collaboration Space (last access April 14, 2008), `http://www.ip-ecospace.org/`

4. COIN: Enterprise Collaboration and Interoperability (last access Novomber 28, 2008), `http://www.coin-ip.eu/`
5. Barros, A.P., Dumas, M., ter Hofstede, A.H.M.: Service interaction patterns. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 302–318. Springer, Heidelberg (2005)
6. van der Aalst, W.M.P., van Dongen, B.F., Günther, C.W., Mans, R.S., de Medeiros, A.K.A., Rozinat, A., Rubin, V., Song, M., Verbeek, H.M.W.E., Weijters, A.J.M.M.: ProM 4.0: Comprehensive support for *real* process analysis. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 484–494. Springer, Heidelberg (2007)
7. Zdun, U., Hentrich, C., van der Aalst, W.M.P.: A survey of patterns for service-oriented architectures. IJIPT 1(3), 132–143 (2006)
8. 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)
9. Gombotz, R., Dustdar, S.: On web services workflow mining. In: Bussler, C.J., Haller, A. (eds.) BPM 2005. LNCS, vol. 3812, pp. 216–228. Springer, Heidelberg (2006)
10. Dustdar, S., Hoffmann, T.: Interaction pattern detection in process oriented information systems. Data Knowl. Eng. 62(1), 138–155 (2007)
11. Dustdar, S., Hoffmann, T., van der Aalst, W.M.P.: Mining of ad-hoc business processes with teamlog. Data Knowl. Eng. 55(2), 129–158 (2005)
12. Truong, H.L., Dustdar, S., Fahringer, T.: Performance metrics and ontologies for grid workflows. Future Generation Comp. Syst. 23(6), 760–772 (2007)
13. Zhang, P., Serban, N.: Discovery, visualization and performance analysis of enterprise workflow. Comput. Stat. Data Anal. 51(5), 2670–2687 (2007)
14. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. 33(1), 64–95 (2008)
15. van der Aalst, W.M.P.: Exploring the cscw spectrum using process mining. Advanced Engineering Informatics 21(2), 191–199 (2007)
16. Cozzi, A., Farrell, S., Lau, T., Smith, B.A., Drews, C., Lin, J., Stachel, B., Moran, T.P.: Activity management as a web service. IBM Syst. J. 45(4), 695–712 (2006)
17. Dustdar, S., Gombotz, R.: Discovering web service workflows using web services interaction mining. International Journal of Business Process Integration and Management 1(4), 256–266 (2006)
18. Luckham, D.C.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Longman Publishing Co., Inc., Boston (2001)
19. EsperTech - Esper: Event Stream and Complex Event Processing (last access April 14, 2008), `http://esper.codehaus.org`
20. The inContext project (last access April 12, 2008), `http://www.in-context.eu`
21. 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)
22. Dustdar, S.: Caramba - a process-aware collaboration system supporting ad hoc and collaborative processes in virtual teams. Distributed and Parallel Databases 15(1), 45–66 (2004)
23. Java Specification Request 224: Java API for XML-Based Web Services (JAX-WS) (last access April 14, 2008), `http://www.jcp.org/en/jsr/detail?id=224`
24. EPL Reference: Clauses (last access April 15, 2008), `http://esper.codehaus.org/esper-2.0.0/doc/reference/en/html/epl_clauses.html`

Exploiting Inductive Logic Programming Techniques for Declarative Process Mining

Federico Chesani[1], Evelina Lamma[2], Paola Mello[1], Marco Montali[1], Fabrizio Riguzzi[2], and Sergio Storari[2]

[1] DEIS – Università di Bologna
viale Risorgimento, 2 – 40136 – Bologna, Italy
{federico.chesani,paola.mello,marco.montali}@unibo.it
[2] ENDIF – Università di Ferrara
Via Saragat, 1 – 44100 – Ferrara, Italy
{evelina.lamma,fabrizio.riguzzi,sergio.storari}@unife.it

Abstract. In the last few years, there has been a growing interest in the adoption of declarative paradigms for modeling and verifying process models. These paradigms provide an abstract and human understandable way of specifying constraints that must hold among activities executions rather than focusing on a specific procedural solution. Mining such declarative descriptions is still an open challenge. In this paper, we present a logic-based approach for tackling this problem. It relies on Inductive Logic Programming techniques and, in particular, on a modified version of the Inductive Constraint Logic algorithm. We investigate how, by properly tuning the learning algorithm, the approach can be adopted to mine models expressed in the ConDec notation, a graphical language for the declarative specification of business processes. Then, we sketch how such a mining framework has been concretely implemented as a ProM plug-in called DecMiner. We finally discuss the effectiveness of the approach by means of an example which shows the ability of the language to model concurrent activities and of DecMiner to learn such a model.

1 Introduction

When facing the problem of defining and developing a Business Process (BP), we can mainly identify two different and complementary roles: the *business analyst*, a domain expert aiming at improving the performances of her company, and the *IT-expert*, who has the responsibility of bringing business-level models to an effective underlying implementation. The complementarity of these roles leads to different perspectives about the process to be developed: while the IT-expert typically adopts a procedural style of modeling, dealing with implementation aspects and trying to obtain an executable process, the business analyst follows a more declarative approach (see Figure 1). Indeed, at a business level it is very important to represent in an intuitive and concise way the domain and problem under study, rather than focusing on a specific solution. In this respect, the

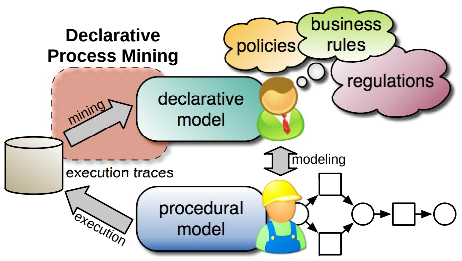


Fig. 1. Declarative and procedural perspectives when modeling Business Processes

model will typically involve business rules, covering best practices and internal constraints as well as internal/external regulations and compliance requirements.

The importance of adopting a declarative style of modeling has been recently pointed out by van der Aalst and Pesic [18]: we agree with their claim that declarative languages fit better complex, unpredictable processes, where a good balance between support and flexibility is of key importance. To this end, in [18] they propose a new graphical language for specifying process flows in a declarative manner. The language, called ConDec, does not completely fix the control flow among activities, but rather envisages a set of constraints expressing policies/business rules for specifying either what is forbidden as well as mandatory in the process. Therefore, the approach is inherently open and flexible, because workers can perform actions if they are not explicitly forbidden. ConDec adopts an underlying semantics by means of Linear Temporal Logics (LTL), and can also be mapped onto a logic programming-based framework called $\mathcal{S}$CIFF (Social Constrained IFF) [2,4], which was originally developed for the specification and verification of global interaction protocols in open Multi-Agent Systems but has recently been applied in the context of BPs and SOA (Service-Oriented Architecture) Choreographies. $\mathcal{S}$CIFF provides a declarative language based on Computational Logic, where constraints are imposed on activities in terms of reactive rules (namely Integrity Constraints). Such reactive rules mention in their body occurring activities, i.e., *events*, and additional constraints on their variables in the style of Constraint Logic Programming (CLP) [12]. $\mathcal{S}$CIFF rules contain in their head expectations over the course of events. Such expectations can be positive, when a certain activity is required to happen, or negative, when a certain activity is forbidden to happen.

An important topic related to declarative process specification, which is still an open challenge, concerns their discovery starting from execution traces, i.e., *declarative process mining*. Indeed, up to now, the goal of process mining has been the discovery of procedural process models (such as Petri Nets or Event-driven Process Chains [21,24]). We claim the necessity of mining also declarative models, to enable the possibility of inferring essential process constraints, easily understandable by business analysts and not affected by procedural details.

In this paper, we present a logic-based approach to address this issue. It relies on Inductive Logic Programming (ILP) techniques and, in particular, on a modified version of the Inductive Constraint Logic (ICL) algorithm [15]. The algorithm takes as input a set of process execution traces, previously labeled as compliant or not, and produces a set of $\mathcal{S}$CIFF rules which correctly classify them. This algorithm has been further modified, by properly tuning it and relying on the mapping presented in [4], for learning ConDec models. Then, we describe how the whole approach has been implemented as a plug-in of the ProM [23] process mining framework. The plug-in, called DecMiner, is capable of mining ConDec models starting from a set of process execution traces. The plug-in envisages different phases, ranging from the classification of traces into compliant and non-compliant subsets to the choice of which ConDec constraints have to be considered and finally to the presentation of the mined model. The effectiveness of the approach is illustrated by considering an example inspired by the one presented in [17] that involves the management of a hotel and spa.

Our previous papers on process mining [14,13] focused on the algorithm for learning $\mathcal{S}$CIFF rules and presented only a sketch of the technique for the translation into ConDec. In this work we describe how we automated this process and implemented it into the DecMiner ProM plug-in.

The paper is organized as follows. Section 2 describes the declarative languages we consider, namely $\mathcal{S}$CIFF and ConDec, and the mapping between ConDec and a subset of SCIFF rules. Section 3 presents the learning process and the DecMiner plug-in. Section 4 discusses the experiments performed for validating the approach. Section 5 presents related works and, finally, Section 6 concludes the paper and discusses future work.

2 Declarative Specification of Business Processes

In this section, we first briefly introduce the $\mathcal{S}$CIFF language, a logic-based language originally developed for specifying and verifying interaction protocols in open Multi-Agent Systems [2]. We then briefly describe ConDec [18], a graphical language supporting the intuitive modeling of declarative constraints on the flow of activities. Finally, we sketch how $\mathcal{S}$CIFF can be exploited to formalize ConDec models as well as to extend its expressiveness, relying on the results presented in [4].

2.1 An Overview of the $\mathcal{S}$CIFF Framework

The $\mathcal{S}$CIFF framework [2] is based on abduction, a reasoning paradigm which allows to formulate hypotheses (called *abducibles*) accounting for observations. In most abductive frameworks, *integrity constraints* are imposed over possible hypotheses in order to prevent inconsistent explanations. $\mathcal{S}$CIFF considers a set of interacting peers as an open society, formalizing interaction protocols by means of a set of global rules (constraints) which constrain the external and observable behavior of participants.

To represent that an event ev happened (i.e., an atomic activity has been executed) at a certain time T, $\mathcal{S}$CIFF uses the symbol $\mathbf{H}(ev, T)$, where ev is a term and T is a variable or a number indicating the time. Hence, an execution trace is modeled as a set of executed (happened) events. For example, we could formalize that *bob* has performed activity a at time 5 as follows: $\mathbf{H}(a(bob), 5)$. Furthermore, $\mathcal{S}$CIFF introduces the concept of expectation, which plays a key role when defining global interaction protocols, choreographies, and more in general event-driven processes. It is quite natural, in fact, to think of a process in terms of rules of the form: "if ev_1 happened, then ev_2 is expected to happen." Positive expectations are denoted by $\mathbf{E}(ev, T)$ meaning that ev is expected to happen at time T. To satisfy a positive expectation, an execution trace must contain a matching happened event. Negative expectations are denoted by $\mathbf{EN}(ev, T)$ meaning that ev is expected not to happen at time T. To satisfy a negative expectation an execution trace must not contain a matching happened event.

$\mathcal{S}$CIFF Integrity Constraints (ICs for short) are forward rules of the form *body* $\rightarrow$ *head*, where *body* can contain literals (i.e. a logical atom or its negation) and happened events, and *head* contains a disjunction of conjunctions of expectations and literals. In this paper, we consider a syntax of ICs that is a subset of the one in [2]. In this simplified syntax, an IC C is a logical formula of the form

$$Body \rightarrow DisjE_1 \vee \ldots \vee DisjE_n \vee DisjEN_1 \vee \ldots \vee DisjEN_m \qquad (1)$$

We will use $Body(C)$ to indicate $Body$ and $Head(C)$ to indicate $DisjE_1 \vee \ldots \vee DisjE_n \vee DisjEN_1 \vee \ldots \vee DisjEN_m$ of a rule C. $Body$ is of the form $b_1 \wedge \ldots \wedge b_l$ where the b_is are literals. Some of the literals may be of the form $\mathbf{H}(ev, T)$ meaning that event ev has happened at time T. $DisjE_j$ is a formula of the form $\mathbf{E}(ev, T) \wedge d_1 \wedge \ldots \wedge d_k$ where ev is an event and the d_is are literals. All the formulas $DisjE_j$ in $Head(C)$ will be called *positive disjuncts*. $DisjEN_j$ is a formula of the form $\mathbf{EN}(ev, T) \wedge d_1 \wedge \ldots \wedge d_k$ where ev is an event and the d_is are literals. All the formulas $DisjEN_j$ in $Head(C)$ will be called *negative disjuncts*.

The event ev can be a term. The literals b_is and d_is refer to predicates defined in a $\mathcal{S}$CIFF knowledge base. Variables in common to $Body(C)$ and $Head(C)$ are universally quantified ($\forall$) with scope the whole IC. Variables occurring only in positive disjuncts are existentially quantified ($\exists$) with scope the disjunct itself. Variables occurring only in negative disjuncts are universally quantified ($\forall$) with scope the disjunct itself. An example of an IC is

$$\begin{aligned}(IC.1)\quad & H(a(bob), T) \wedge T < 10\\ & \rightarrow E(b(alice), T1) \wedge T < T1 \vee\\ & EN(c(mary), T2) \wedge T < T2 \wedge T2 < T + 10\end{aligned}$$

The meaning of the $IC.1$ is the following: if *bob* has executed action a at a time $T < 10$, then we expect *alice* to execute action b at some time $T1$ later than T ($\exists T1$) or we expect that *mary* does not execute action c at any time $T2$ ($\forall T2$) within 9 time units after T.

The interpretation of an IC is the following: if there exists a substitution of variables such that the body is true in an interpretation representing a trace, then one of the disjuncts in the head must be true. A positive disjunct means that we expect event ev to happen with T and its variables satisfying $d_1 \wedge \ldots \wedge d_k$. Therefore the disjunct is true if there exist a substitution of variables occurring in it such that *ev* is present in the trace and the d_is are satisfied. A negative disjunct means that we expect event *ev* not to happen with T and its variables satisfying $d_1 \wedge \ldots \wedge d_k$. Therefore the disjunct is true if for all substitutions of variables occurring in it and not appearing in *Body* either *ev* does not happen or, if it happens, its properties violate $d_1 \wedge \ldots \wedge d_k$.

The main and original application of the $\mathcal{S}$CIFF framework and its proof procedure is to verify whether an execution of the process concretely adheres to the specification, i.e., to perform *compliance checking.* $\mathcal{S}$CIFF is seamlessly able to check compliance both at run-time, by dynamically collecting and reasoning upon occurring events, or a posteriori, by analyzing the log of an observed execution trace.

Roughly speaking, $\mathcal{S}$CIFF combines occurred events with the specified rules, to suitably generate the corresponding expectations; then expectations are verified against the execution trace: a positive expectation must have a corresponding matching event, whereas a negative expectation forbids the presence of a matching event. If such conditions are not met (i.e., a positive/negative expectation is not/is matched by a corresponding event), then the expectations are violated, and the execution trace is evaluated as non-compliant.

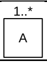
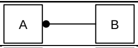
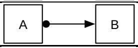
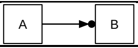
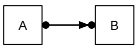
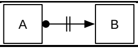
A posteriori compliance checking has been wrapped into a ProM plug-in called $\mathcal{S}$CIFFChecker [3], which can be exploited to classify MXML execution traces as compliant or non-compliant w.r.t. a high-level declarative criterion. Such a criterion is specified by configuring reactive business rules expressed in a natural language-like manner and by automatically mapping them onto the underlying formalism.

2.2 ConDec and Its $\mathcal{S}$CIFF Mapping

ConDec [18,16] is a graphical language suitable for the declarative specification of flexible Business Processes. Flexibility is provided since ConDec does not fix a completely specified process flow, but rather imposes only the (minimal) set of constraints that must be satisfied when executing the process activities. Constraints are policies/business rules which can be exploited to describe both what is mandatory and what is forbidden in the process. They are mainly organized into three basic groups: (i) *existence constraints*, unary relationships constraining the cardinality of activity executions; (ii) *relation constraints*, positive relationships between two activities used to specify what should be executed when a given situation holds; (iii) *negation constraints*, the negated version of relation ones, imposed to forbid the execution of a certain activity when a given situation holds.

We have provided a complete mapping of ConDec relationships to $\mathcal{S}$CIFF [4]. Table 1 shows some basic ConDec constraints, together with their corresponding formalization. For example, the existence constraint specifies that the involved activity must be executed at least once; this can be expressed in $\mathcal{S}$CIFF by simply stating that the activity is *expected to happen.* The responded existence between A and B imposes the existence of B only if activity A is executed, without putting any temporal condition between the two executions. Temporizing such

Table 1. Mapping of some ConDec formulas onto $\mathcal{S}$CIFF

	Name	$\mathcal{S}$CIFF Mapping
	existence	true $\rightarrow \mathbf{E}(A,T)$
	responded existence	$\mathbf{H}(A,T_A) \rightarrow \mathbf{E}(B,T_B)$
	response	$\mathbf{H}(A,T_A) \rightarrow \mathbf{E}(B,T_B) \wedge T_B > T_A$
	precedence	$\mathbf{H}(B,T_B) \rightarrow \mathbf{E}(A,T_A) \wedge T_A < T_B$
	succession	$\mathbf{H}(A,T_A) \rightarrow \mathbf{E}(B,T_B) \wedge T_B > T_A$ $\mathbf{H}(B,T_B) \rightarrow \mathbf{E}(A,T_A) \wedge T_A < T_B$
	negation response	$\mathbf{H}(A,T_A) \rightarrow \mathbf{EN}(B,T_B) \wedge T_B > T_A$

a constraint leads either to a response or precedence constraint, depending on what kind of ordering is imposed between the two activities. For example, response states that if activity A has been performed, then B must be performed afterward; the "after" ordering can be modeled in $\mathcal{S}$CIFF by putting a "greater than" CLP [12] constraint among the execution time associated to B and the one associated to A, i.e. $T_B > T_A$. The precedence constraint is modeled in a similar way, by inverting the constraint to express a "before" relationship.

Finally, ConDec supports also negative constraints, i.e., constraints used to forbid the execution of certain activities. They are mapped to $\mathcal{S}$CIFF similarly to positive relation constraints but imposing negative expectations instead of positive ones (see, for example, the negation response constraint in Table 1, which states that after activity A it is not possible to execute B anymore, being T_B universally quantified with scope the disjunct where it appears).

$\mathcal{S}$CIFF can be used not only to formalize ConDec, but also to support different extensions to the language, such as: (i) considering conjunction of events in relationships (e.g., to model synchronizing responses, namely responses which trigger only when two or more events occur); (ii) involving quantitative temporal constraints, such as deadlines and delays; (iii) constraining also data involved in the activities execution, such as originators.

2.3 Running Example

In order to explain how the declarative mining approach works, we use a process model that is inspired to [17] as a running example. This model describes a simple process of renting rooms and services in a hotel and spa. Every process instance starts with the registration of the client name and her preferred way of payment (e.g., credit card). Data can also be altered at later time (e.g the client may decide to use another credit card). During her stay, the client can require one or more room, laundry and massage services. Each service, identified by a code, is followed by the respective registration of the service costs into the client bill. Of course, each service cost must be registered only if the service has been effectively provided to the client and only one time. Moreover, if the client chooses a shiatzu massage, the spa presents her a special offer. The cost related to nights spent in the hotel must be billed. It is possible for the total bill to be charged at several stages during the stay.

This process was modeled by using eleven activities and eleven constraints. Activities *register_client_data*, *check_out* and *charge* are about the check-in/check-out of the client and expenses charging. Activities *room_service*, *laundry_service*, and *massage_service* log which services have been accessed to by the client, while billings for each service are represented by corresponding activities. For each activity, a unique identifier is introduced to correctly charge the clients with the billings for the services they effectively made use of. Moreover, for the massage related activities, an additional parameter is used to specify the massage type (aromatic or shiatzu). Finally, the activity *shiatzu_offer* maps the business policy of offering a special packet/discount to clients interested in shiatzu massages.

Business related aspects of our example are represented as follows:

- (C.1) every process instance starts with activity *register_client_data*. No limits on the repetitions of this activity are expressed, hence allowing alteration of data;
- (C.2) *bill_room_service* must be executed after each *room_service* activity, and *bill_room_service* can be executed only if the *room_service* activity has been executed before;
- (C.3) *bill_laundry_service* must be executed after each *laundry_service* activity, and *bill_laundry_service* can be executed only if the *laundry_service* activity has been executed before;
- (C.4) *bill_massage_service* must be executed after each *massage_service*, and *bill_massage_service* can be executed only if the *massage_service* activity has been executed before;
- (C.5) *shiatzu_offer* must be executed after a *massage_service* activity with type shiatzu;
- (C.6) *check_out* must be performed in every process instance;
- (C.7) *charge* must be performed in every process instance;
- (C.8) *bill_nights* must be performed in every process instance.
- (C.9) *bill_room_service* must be executed only one time for each service identifier;
- (C.10) *bill_laundry_service* must be executed only one time for each service identifier;
- (C.11) *bill_massage_service* must be executed only one time for each service identifier;

The $\mathcal{S}$CIFF representation is composed by the following ICs:

$$\begin{array}{rl}
(C.1) & true \\
& \rightarrow E(register\_client\_data, Trcd)) \wedge Trcd = 1. \\
(C.2) & H(room\_service(rs\_id(IDrs)), Trs) \\
& \rightarrow E(bill\_room\_service(rs\_id(IDbrs)), Tbrs) \wedge \\
& IDrs = IDbrs \wedge Tbrs > Trs. \\
& H(bill\_room\_service(rs\_id(IDbrs)), Tbrs) \\
& \rightarrow E(room\_service(rs\_id(IDrs)), Trs) \wedge \\
& IDbrs = IDrs \wedge Trs < Tbrs. \\
(C.3) & H(laundry\_service(la\_id(IDls)), Tls) \\
& \rightarrow E(bill\_laundry\_service(la\_id(IDbls)), Tbls) \wedge \\
& IDls = IDbls \wedge Tbls > Tls. \\
& H(bill\_laundry\_service(la\_id(IDbls)), Tbls) \\
& \rightarrow E(laundry\_service(la\_id(IDls)), Tls) \wedge \\
& IDbls = IDls \wedge Tls < Tbls. \\
(C.4) & H(massage\_service(ma\_id(IDms), type(TYms))), Tms) \\
& \rightarrow E(bill\_massage\_service(ma\_id(IDbms), type(TYbms)), Tbms) \wedge \\
& IDms = IDbms \wedge TYms = TYbms \wedge Tbms > Tms. \\
& H(bill\_massage\_service(ma\_id(IDbms), type(TYbms))), Tbms) \\
& \rightarrow E(massage\_service(ma\_id(IDms), type(TYms)), Tms) \wedge \\
& IDbms = IDms \wedge TYbms = TYms \wedge Tms < Tbms. \\
(C.5) & H(massage\_service(ma\_id(IDms), type(TYms)), Tms) \wedge TYms = shiatzu \\
& \rightarrow E(shiatzu\_offer, Tbms) \wedge Tbms > Tms. \\
(C.6) & true \\
& \rightarrow E(check\_out, Tco). \\
(C.7) & true \\
& \rightarrow E(charge, Tch). \\
(C.8) & true \\
& \rightarrow E(bill\_nights, Tbn). \\
(C.9) & H(bill\_room\_service(rs\_id(IDbrs1)), Tbrs1) \\
& \rightarrow EN(bill\_room\_service(rs\_id(IDbrs2)), Tbrs2) \wedge \\
& IDbrs1 = IDbrs2 \wedge Tbrs2 > Tbrs1. \\
(C.10) & H(bill\_laundry\_service(la\_id(IDbls1)), Tbls1) \\
& \rightarrow EN(bill\_laundry\_service(la\_id(IDbls2)), Tbls2) \wedge \\
& IDbls1 = IDbls2 \wedge Tbls2 > Tbls1. \\
(C.11) & H(bill\_massage\_service(ma\_id(IDbms1), type(TYbms1))), Tbms1) \\
& \rightarrow EN(bill\_massage\_service(ma\_id(IDbms2), type(TYbms2)), Tbms2) \wedge \\
& IDbms1 = IDbms2 \wedge TYbms1 = TYbms2 \wedge Tbms2 > Tbms1.
\end{array}$$

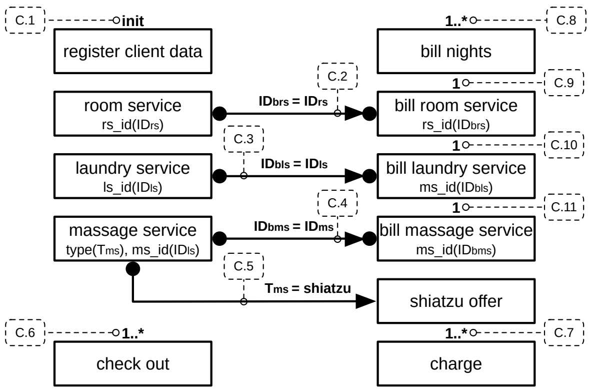


Fig. 2. A ConDec model augmented with a data-related perspective

One thing to observe is that, for constraint ($C.1$), the ConDec init constraint has been mapped in $\mathcal{S}$CIFF by imposing that the *"register_client_data"* activity is expected to happen at time 1 (the first activity in an execution trace).

As described in Section 2.2, $\mathcal{S}$CIFF can be used not only to formalize ConDec, but also to support different extensions to the language. These extensions are useful in the formalization of the hotel and spa process model. Let us consider the following constraints: after having chosen a massage service, this service must be billed to the client and a massage service must be billed only if the client has effectively received the service; if the client has chosen a shiatzu massage, then she can also take advantage of a special offer. In order to link each different service with its specific bill, we attach to the execution of these activities an identifier. Moreover, the second statement deals with a specific execution of the massage service, namely the one in which the client has actually chosen a shiatzu massage; so we attach a type attribute to massage services. This information has been included into the terms representing events in the $\mathcal{S}$CIFF language and the two sentences mapped to the integrity constraints ($C.4$) and ($C.5$).

We could incorporate such a data-related perspective directly at a graphical level, by representing activities together with their data and annotating the ConDec constraints with data condition (such as $TYms = shiatzu$ in $C.5$). Figure 2 shows how the model discussed above can be graphically rendered with annotations.

3 Learning Models

In this section, we describe the approach adopted for mining ConDec models. We first briefly review some concepts of Inductive Logic Programming and the ICL algorithm in particular, then we discuss how ICL has been applied to learning $\mathcal{S}$CIFF constraints and finally we illustrate the DecMiner ProM plug-in for mining $\mathcal{S}$CIFF and ConDec constraints.

3.1 Inductive Logic Programming Techniques

The idea of exploiting Inductive Logic Programming (ILP) for declarative process mining comes form the similarities between learning a $\mathcal{S}$CIFF theory, composed by a set of Social Integrity Constraints, and learning a clausal theory as described in the learning from interpretation setting of ILP [15]. Besides the fact that both $\mathcal{S}$CIFF and clausal theories can be used to classify a set of atoms (i.e., an interpretation) as positive or negative, they have strong similarities in the structure of the logical formula composing the theory. Then, thanks to the mapping of ConDec into $\mathcal{S}$CIFF rules, it is possible to learn ConDec models.

A clause C is a formula in the form $b_1 \wedge \cdots \wedge b_n \rightarrow h_1 \vee \cdots \vee h_m$ where b_i are logical literals and h_i are logical atoms. A formula is ground if it does not contain variables. An interpretation is a set of ground atoms. Let us define $head(C) = \{h_1, \ldots, h_m\}$ and $body(C) = \{b_1, \ldots, b_n\}$. Sometimes we will interpret clause C as the set of literals $\{h_1, \ldots, h_m, \neg b_1, \ldots, \neg b_n\}$.

The clause C is true in an interpretation I iff, for all the substitutions θ grounding C, $(I \models body(C)\theta) \rightarrow (head(C)\theta \cap I \neq \emptyset)$. Otherwise, it is false. A set of clauses (i.e. a theory) is true in an interpretation I iff all the clauses are true in I.

Sometimes we may be given a background knowledge B with which we can enlarge each interpretation I by considering, instead of simply I, the interpretation given by $M(B \cup I)$ where M stands for a model, such as the least Herbrand model of Clark's completion [5]. By using a background knowledge we are able to encode each interpretation parsimoniously, by storing only once the rules that are not specific to a single interpretation but are true for every interpretation.

The learning from interpretation setting of ILP is concerned with the following problem: given a clausal language $\mathcal{L}$, a set P of positive interpretations, a set N of negative interpretations and a definite clause background theory B, we want to find a clausal theory $H \in \mathcal{L}$ such that for all $p \in P$, H is true in the interpretation $M(B \cup p)$, and for all $n \in N$, H is false in the interpretation $M(B \cup n)$. Given a disjunctive clause C (theory H) we say that C (H) *covers* the interpretation I iff C (H) is true in $M(B \cup I)$. We say that C (H) *rules out* an interpretation I iff C (H) does not cover I.

The clausal language $\mathcal{L}$ is used in order to restrict the search space. It is usually described in an intensional way using a specific representation language. The description of $\mathcal{L}$ in this language is called *language bias* (LB).

An algorithm that solves the above problem is ICL [6]. In it, a function named Inductive-Constraint-Logic performs a covering loop in which negative interpretations are progressively ruled out and removed from the set N. At each iteration of the loop, a new clause is added to the theory and the negative examples excluded by it are removed from N. The loop ends when N is empty or when no clause is found.

The clause to be added in every iteration of the covering loop is returned by another procedure (namely, Find-Best-Clause). It looks for a clause by using beam search with $p(\ominus|\overline{C})$ as a heuristic function, where $p(\ominus|\overline{C})$ is the probability that an example interpretation is classified as negative given that it is ruled out by the clause C. This heuristic is computed as the number of ruled out negative interpretations over the total number of ruled out interpretations (positive and negative). Thus we look for clauses that cover as many positive interpretations as possible and rule out as many negative interpretations as possible. The search starts from an initial beam composed of the most specific clauses present in the language bias that is returned by the function MostSpecific(LB). The clauses in the beam are then gradually generalized. The maximum number of generalization steps is a user-defined parameter.

The generality order that is used is θ-subsumption [19], a relationships between two clauses that can be checked syntactically and is stronger than implications. Generalizations of a clause C are obtained by adding a literal to the body or an atom to the head of C. The language bias of ICL defines the literals that can be added to clauses. Moreover, the language bias defines also the set of most specific clauses.

3.2 Application of ICL to $\mathcal{S}$CIFF Learning

ICL has been effectively used to learn $\mathcal{S}$CIFF ICs in Declarative Process Model Learner (DPML) [14]. Each IC is seen as a clause that must be true in all the positive interpretations (compliant execution traces) and false in some negative interpretations (non-compliant execution traces). A theory, composed of a set of ICs, must be such that all the ICs are true when considering a compliant trace and at least one IC is false when considering a non-compliant one.

If we define a generality order and a generalization operator for ICs, we can apply an algorithm similar to ICL for learning ICs. The generality order can be defined in this way: an IC C is more general than an IC D (written $C \geq D$) if there exists a substitution θ for the variables of $body(D)$ such that $body(D)\theta \subseteq body(C)$ and for each disjunct d in the head of D: if d is positive, then there exist a positive disjunct c in the head of C such that $d\theta \supseteq c$; if d is negative, then there exist a negative disjunct c in the head of C such that $d\theta \subseteq c$.

A generalization of an IC C can be obtained in the following ways: adding a literal to the body, adding a disjunct to the head, removing a literal from a positive disjunct in the head or adding a literal to a negative disjunct in the head. The language bias takes the form of a set of assertions that are couples (BS, HS): BS is a set that contains the literals that can be added to the body and HS is a set that contains the disjuncts that can be added to the head. Each element of HB is a couple $(Sign, Literals)$ where $Sign$ is either + for a positive disjunct or - for a negative disjunct, and $Literals$ contains the literals that can appear in the disjunct.

When adding a disjunct to the head, the generalization operator behaves differently depending on the sign of the disjunct: in the case of a positive disjunct, the disjunct formed by the **E** literal plus all the literals in the language bias for the disjunct is added; in the case of a negative disjunct, only the **EN** literal is added.

3.3 Learning ConDec Models and $\mathcal{S}$CIFF Rules: The DecMiner Plug-in

DPML has been further extended in this work in order to be able to learn both ConDec models and $\mathcal{S}$CIFF ICs and re engineered as a mining plug-in of the ProM [23] process mining framework, named DecMiner.

DecMiner learns a ConDec model, by first learning $\mathcal{S}$CIFF ICs and then translating them into ConDec constraints using the mapping introduced in Section 2.2. In order to ease the translation, we provide DecMiner with a special language bias that allows only ICs that can be translated into ConDec. We generate this language bias automatically starting from a set of general templates, one for each ConDec constraint, that can be instantiated to generate specific assertions for the language bias. The number of all possible assertions can be huge, while the user could be interested to models defined only by a small, yet meaningful set of ConDec constraints. For this reason, we let the user the possibility of selecting a subset of activities A and a subset of ConDec constraints T. Then, our approach uses only the instantiation of these constraints with the selected activities for learning the model. Besides providing as output a model that fits the user requirements, smaller constraint sets allow also better performances of the learning algorithm.

The accuracy and learning time depends on the choice of these subsets. They influence the accuracy of the learned model because an activity relation discriminating between compliant and non-compliant execution traces cannot be learned if the appropriate template and/or activities were not chosen. The time complexity is linear in the number of traces and in the number of constraints. With respect to the number of activities, it is quadratic if there are binary constraints, and linear if there are only unary constraints.

An advantage of mining ConDec constraints through $\mathcal{S}$CIFF is that the approach can be extended to induce constraints involving more than two activities, for example constraints having a conjunction of preconditions or a disjunction of postconditions, and constraints with conditions over data.

DecMiner implements all the data preparation and learning phases of the mining process described above and guides the user by means of its graphical user interface. In the first phase, named "Classification", the user uses the graphical interface shown in Figure 3 to browse the execution traces and label some of them as compliant (positive) or not compliant (negative). In the second phase, named "Activities", the user can choose among all the activities and their associated parameters the information that she considers important for learning the declarative model. In the third phase, named "Templates", the user uses the graphical interface shown in Figure 4 to choose the set of existence, relation and negation ConDec templates to be used in the mining phase. The fourth phase, named "Mining", is started when the "Start mining" button is pressed. In this phase the language bias for ICL is generated, by instantiating the chosen templates with the chosen activities, and the learning algorithm is applied, producing the declarative model. In the fifth phase, named "Results", the learned

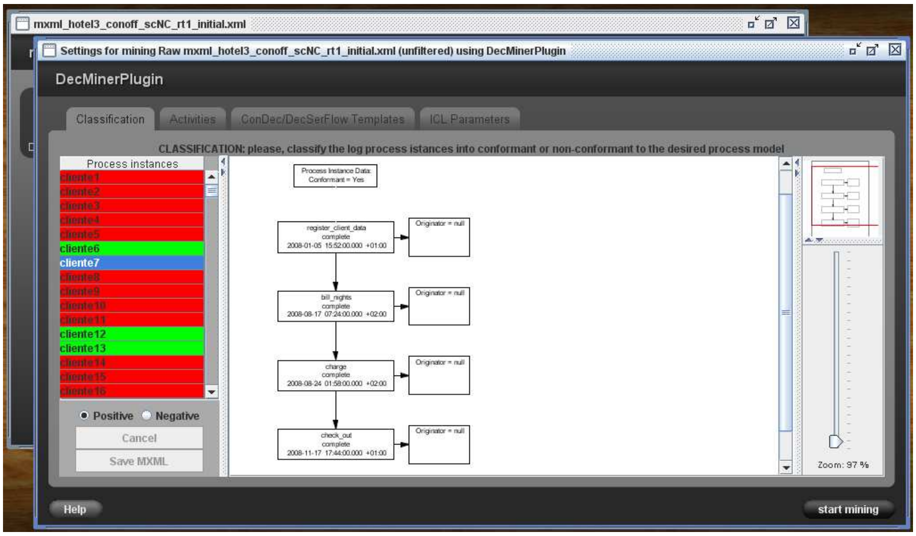

mxml_hotel3_conoff_scNC_rt1_initial.xml
Settings for mining Raw mxml_hotel3_conoff_scNC_rt1_initial.xml (unfiltered) using DecMinerPlugin
DecMinerPlugin
Classification
Activities
ConDec/DecSerFlow Templates
ICL Parameters
CLASSIFICATION: please, classify the log process istances into conformant or non-conformant to the desired process model
Process instances
cliente1
cliente2
cliente3
cliente4
cliente5
cliente6
cliente7
cliente8
cliente9
cliente10
cliente11
cliente12
cliente13
cliente14
cliente15
cliente16
Positive
Negative
Cancel
Save MXML
Process Instance Data: Conformant = Yes
register_client_data complete 2008-01-05 15:52:00.000 +01:00
bill_nights complete 2008-08-17 07:24:00.000 +02:00
charge complete 2008-08-24 01:58:00.000 +02:00
check_out complete 2008-11-17 17:44:00.000 +01:00
Originator = null
Zoom: 97 %
Help
start mining

Fig. 3. DecMiner plug-in: trace classification

$\mathcal{S}$CIFF rules and ConDec constraints are presented to the user[1]. The current version of the tool can be downloaded from the web[2].

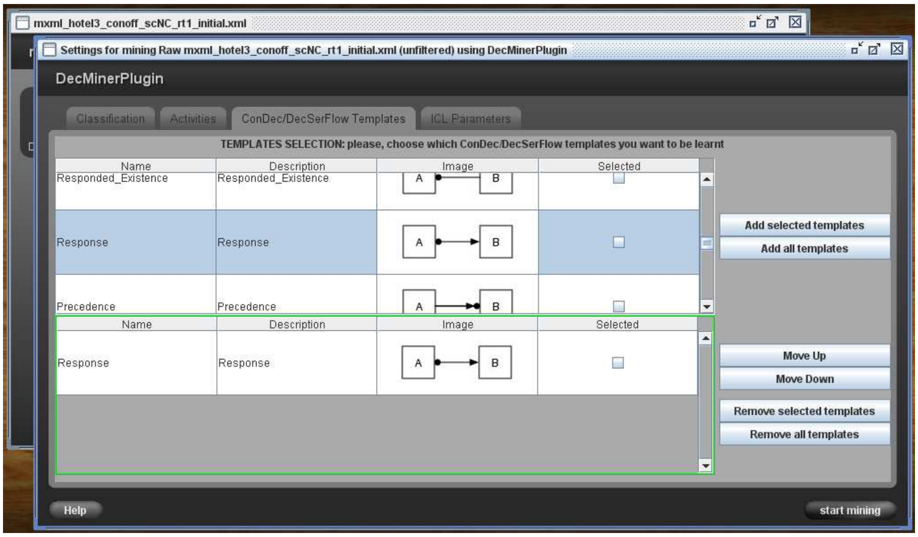
4 Experiments

In order to evaluate the effectiveness and robustness of the learning approach, we followed a typical machine learning experimentation methodology: first, we create an artificial process model, described in Section 2.3, which presents some difficulties for the learning approach to be tested; second, we randomly generate from such model several training and testing datasets; third, we apply the learning approach on the training datasets obtaining models; finally, we compare the learned models with the original one and compute the classification accuracy of the learned models on the testing datasets.

Given the ConDec and $\mathcal{S}$CIFF process models described in Section 2.3, we generated five training and five testing datasets. The generation of each of them is made in two phases. In the first, a Java application randomly creates an execution trace. In the second phase, the $\mathcal{S}$CIFF Checker (presented in Section 2) is used to classify each trace as compliant or non-compliant with respect to the correct hotel process. The process is repeated until a dataset containing 2000 compliant traces and 2000 non-compliant traces has been generated.

[1] The ConDec model is shown by using the DECLARE tool `http://is.tm.tue.nl/research/declare/`

[2] `http://www.unife.it/dipartimento/ingegneria/informazione/informatica/processmining/`


mxml_hotel3_conoff_scNC_rt1_initial.xml
Settings for mining Raw mxml_hotel3_conoff_scNC_rt1_initial.xml (unfiltered) using DecMinerPlugin
DecMinerPlugin
Classification
Activities
ConDec/DecSerFlow Templates
ICL Parameters
Add selected templates
Add all templates
Move Up
Move Down
Remove selected templates
Remove all templates
Help
start mining

TEMPLATES SELECTION: please, choose which ConDec/DecSerFlow templates you want to be learnt

Name	Description	Image	Selected
Responded_Existence	Responded_Existence	A ●—— B	☐
Response	Response	A ●——▶ B	☐
Precedence	Precedence	A ——▶● B	☐

Name	Description	Image	Selected
Response	Response	A ●——▶ B	☐

Fig. 4. DecMiner plug-in: ConDec template selection

DecMiner has been then applied to each training dataset. By applying the learned models to the classification of the testing sets, we computed the *classification accuracy*, defined as the number of compliant traces that are correctly classified as compliant plus the number of non-compliant traces that are correctly classified as non-compliant divided by the total number of traces. The average accuracy achieved by DecMiner was 100%.

Comparing the original hotel process model with those learned by using the five training sets, we observe that sometimes there are differences in the learned constraints. This happens because some of the randomly generated training sets do not contain the traces that allow to distinguish the behaviors of similar constraints.

This is the case, for instance, of the precedence(A,B) and responded_existence(A,B) constraints. They share a common set of labeled execution traces (BA labeled as compliant, B labeled as compliant and A labeled as non-compliant) and cannot be distinguished until a trace containing AB is labeled as compliant or non-compliant. In the first case, DecMiner learns a responded_existence(A,B) constraint otherwise it learns a precedence(A,B). In real applications, this behavior can be considered an advantage because it allows dynamic adaptation and refinement of the learned process models when new traces are classified as compliant and non-compliant and added to the training set. If the traces distinguishing the behavior of two constraints are not present in the dataset, DecMiner learns the constraint that comes first in the language bias.

We also investigated the robustness of DecMiner to noise in the classification of traces: we repeated the experiments by considering training sets with an increasing portion of misclassified examples. Table 2 shows that the performances of DecMiner degrade gracefully with the increase of the amount of noise.

Table 2. Accuracy as a function of noise

Errors	Accuracy
2%	99.72 %
10%	98.85 %
20%	97.29 %

As for other ILP systems, the learning phase depends not only on the training set but also on the language bias: by restricting it to different subsets of the ConDec constraints, it is possible to learn different models. Each model corresponds to a different perspective about a real process, pointing out different aspects. The 100% accuracy achieved in the former experiment is a consequence of the chosen language bias: all the templates were added with those referring to the constraints in the original hotel and spa model put at the top of the language bias. We evaluated the influence of language bias on classification accuracy, by randomly mixing the ConDec templates in the bias. Despite this change, the accuracy achieved by DecMiner considering datasets without noise remains high (99.97%).

Results achieved by our approach on other real and artificial datasets (e.g., the cervical cancer screening process, the netbill e-commerce protocol and an auction protocol) are reported in [14] and [13].

5 Related Works and Discussion

Process mining is an active research field. Notable works in such a field are [1,21,24,11,7,9]. Agrawal et al. [1] introduced the idea of applying process mining to workflow management. The authors proposed an approach for inducing a process representation in the form of a directed graph encoding the precedence relationships. van der Aalst et al. [21] presented the α-algorithm for inducing Petri nets from data and identified for which class of models the approach is guaranteed to work. The α-algorithm is based on the discovery of binary relations in the log, such as the "follows" relation. In [24] van Dongen and van der Aalst described an algorithm which derives causal dependencies between activities and uses them for constructing instance graphs, presented in terms of Event-driven Process Chains. [11] is a recent work where a process model is induced in the form of a disjunction of special graphs called workflow schemas.

We differ from these works because we use a representation that is declarative rather than procedural, without sacrificing expressiveness. Moreover, we learn from compliant and non-compliant traces, rather than from compliant traces.

[7,9] are closer to our work because they deal with mining (partially) declarative specifications. In [7] the learning starts from process *runs* that are high level specification of a set of process traces and are represented by means of Petri nets. Mining is performed by merging the different runs for the same process. The model that is obtained is hybrid, in the sense that it may contain sets of activities that must be executed but for which no specific order is required. We

differ from this work because we start from traces rather than runs: while runs specify already a partial order among activities, traces are simply a sequence of events representing activity executions. Therefore, runs are already very informative of the process model.

[9] related BPM to the field of planning in artificial intelligence: activities in business process are seen as planning operators with pre-conditions and post-conditions. Representing a process in this way requires the specification of *fluents* besides activities, i.e., properties of the world that may change their truth value during the execution of the process. The adoption of fluents allows to explicitly express pre-conditions and post-conditions of activities. Thus fluents introduce a new dimension to BPM that needs further explorations. Our work remains in the traditional domain of BPM in which the pre-conditions and post-conditions of activities are left implicit. The approach for learning process models of [9] involves iterating planning and operator refinement: given the current definition of the pre-conditions and post-conditions of the activities, a plan for achieving the business goal is generated and presented to the user which has to specify whether each activity of the plan can be executed. In this way the system collects positive and negative examples of activities executions that are then used in a learning phase. In order to avoid asking the user to classify activities, [10] proposed an approach for automatically generating negative events, i.e., events that are used as negative examples. In the future we plan to investigate the extension of this approach to the automatic generation of negative traces.

With respect to performance evaluation, a direct comparison with [21,24] is unfair since we adopted accuracy (since we have compliant and non-compliant traces in the test set) while they adopt fitness.

In this special issue, [22] and [8] face the problem of mining a process representation in the form of a Petri net, while [20] extracts metrics and patterns from collaborative processes in SOA-based environments.

6 Conclusions and Future Work

We propose a methodology for analyzing a log containing several traces labeled as compliant or non-compliant. From them we learn a set of declarative constraints expressed as $\mathcal{S}$CIFF rules able to accurately classify a new trace, and corresponding to a ConDec model.

The proposed methodology is based on Inductive Logic Programming and, in particular, on the ICL algorithm. Such an algorithm is adapted to the problem of learning integrity constraints in the $\mathcal{S}$CIFF language. By considering not only compliant traces, but also non-compliant ones, our approach can learn a model which expresses also what is forbidden. Furthermore, the learned $\mathcal{S}$CIFF ICs are easily mapped into ConDec constraints. We call the resulting system DecMiner.

In order to test the proposed methodology, we performed an experiment on a case study regarding the management of a hotel and spa. The results show that DecMiner nearly recovers the correct model. Other experiments have been documented in [14,13].

In the future, we plan to apply DecMiner to university students' careers, where positive traces are careers of students that graduated on time, and negative ones are careers of students who did not finish their studies in the prescribed time.

Moreover, we plan to investigate the development of a *mining-checking cycle*, in which learning is interleaved with classification of traces into positive or negative either manually by the user or automatically using the $\mathcal{S}$CIFF Checker plug-in with a user specified model. In this way the user can improve an initial model of the process by experimenting different languages biases.

Acknowledgments. This work has been partially supported by the FIRB project "TOCAI.IT" and FAR projects by University of Bologna and University of Ferrara. We would like to thank Pamela Zapparoli for her contribution to DecMiner development.

References

1. Agrawal, R., Gunopulos, D., Leymann, F.: Mining process models from workflow logs. In: Schek, H.-J., Saltor, F., Ramos, I., Alonso, G. (eds.) EDBT 1998. LNCS, vol. 1377, pp. 469–483. Springer, Heidelberg (1998)
2. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Verifiable agent interaction in abductive logic programming: the SCIFF framework. ACM T. Comput. Logic 9(4) (2008)
3. Chesani, F., Mello, P., Montali, M., Riguzzi, F., Sebastianis, M., Storari, S.: Checking compliance of execution traces to business rules. In: Ardagna, D., et al. (eds.) BPM 2008 Workshops. LNBIP, vol. 17, pp. 134–145. Springer, Heidelberg (2009)
4. Chesani, F., Mello, P., Montali, M., Storari, S.: Towards a decserflow declarative semantics based on computational logic. Technical Report DEIS-LIA-07-002, DEIS, Bologna, Italy (2007)
5. Clark, K.L.: Negation as failure. In: Logic and Databases. Plenum Press (1978)
6. De Raedt, L., Van Laer, W.: Inductive constraint logic. In: Zeugmann, T., Shinohara, T., Jantke, K.P. (eds.) ALT 1995. LNCS (LNAI), vol. 997, pp. 80–94. Springer, Heidelberg (1995)
7. Desel, J., Erwin, T.: Hybrid specifications: looking at workflows from a run-time perspective. Int. J. Computer System Science & Engineering 15(5), 291–302 (2000)
8. Bergenthum, R., Desel, J., Mauser, S., Lorenz, R.: Construction of process models from example runs. In: Jensen, K., van der Aalst, W. (eds.) ToPNoC II. LNCS, vol. 5460, pp. 243–259. Springer, Heidelberg (2009)
9. Ferreira, H.M., Ferreira, D.R.: An integrated life cycle for workflow management based on learning and planning. Int. J. Cooperative Inf. Syst. 15(4), 485–505 (2006)
10. Goedertier, S.: Declarative techniques for modeling and mining business processes. PhD thesis, Katholieke Universiteit Leuven, Faculteit Economie en Bedrijfswetenschappen (2008)
11. 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)
12. Jaffar, J., Maher, M.J.: Constraint logic programming: a survey. J. Logic Program. 19(20), 503–582 (1994)

13. Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Inducing declarative logic-based models from labeled traces. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 344–359. Springer, Heidelberg (2007)
14. Lamma, E., Mello, P., Riguzzi, F., Storari, S.: Applying inductive logic programming to process mining. In: Blockeel, H., Ramon, J., Shavlik, J., Tadepalli, P. (eds.) ILP 2007. LNCS, vol. 4894, pp. 132–146. Springer, Heidelberg (2008)
15. Muggleton, S., De Raedt, L.: Inductive logic programming: Theory and methods. J. Logic Program. 19(20), 629–679 (1994)
16. Pesic, M.: Constraint-Based Workflow Management Systems. PhD thesis, Technische Universiteit Eindhoven, Department of Technology Management (2008)
17. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: Declare: Full support for loosely-structured processes. In: 11th IEEE International Enterprise Distributed Object Computing Conference, pp. 287–300. IEEE Computer Society, Los Alamitos (2007)
18. Pesic, M., van der Aalst, W.M.P.: A declarative approach for flexible business processes management. In: Eder, J., Dustdar, S. (eds.) BPM Workshops 2006. LNCS, vol. 4103, pp. 169–180. Springer, Heidelberg (2006)
19. Robinson, J.A.: A machine-oriented logic based on the resolution principle. J. ACM 12(1), 23–41 (1965)
20. Truong, H.L., Dustdar, S.: Online interaction analysis framework for ad-hoc collaborative processes in SOA-based environments. In: Jensen, K., van der Aalst, W. (eds.) ToPNoC II. LNCS, vol. 5460, pp. 260–277. Springer, Heidelberg (2009)
21. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Trans. Knowl. Data Eng. 16(9), 1128–1142 (2004)
22. van Dongen, B., de Medeiros, A.K.A., Wen, L.: Process mining: Overview and Outlook of Petri net discovery algorithms. In: Jensen, K., van der Aalst, W. (eds.) ToPNoC II. LNCS, vol. 5460, pp. 225–242. Springer, Heidelberg (2009)
23. van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H.M.W., Weijters, A.J.M.M., van der Aalst, W.M.P.: The ProM framework: A new era in process mining tool support. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 444–454. Springer, Heidelberg (2005)
24. van Dongen, B.F., van der Aalst, W.M.P.: Multi-phase process mining: Building instance graphs. In: Atzeni, P., Chu, W., Lu, H., Zhou, S., Ling, T.-W. (eds.) ER 2004. LNCS, vol. 3288, pp. 362–376. Springer, Heidelberg (2004)

Author Index