Christian Gierds Jan Sürmeli (Eds.)

Services und ihre Komposition

Proceedings of the 2nd Central-European Workshop on Services and their Composition, ZEUS 2010

Berlin, Germany, February 25/26

CEUR Workshop Proceedings Vol. 563

2. Zentral-europäischer Workshop über Services und ihre Komposition
2nd Central-European Workshop on Services and their Composition

ZEUS 2010

Christian Gierds and Jan Sürmeli, Editors

Humboldt-Universität zu Berlin
Department of Computer Science
Unter den Linden 6
10099 Berlin, Germany

{gierds | suermeli}@informatik.hu-berlin.de

ISSN 1613-0073 (CEUR Workshop Proceedings)
http://CEUR-WS.org/Vol-563/

BibTeX

```
@proceedings{zeus2009,
  editor    = {Christian Gierds and Jan S\"urmeli},
  title     = {Proceedings of the 2nd Central-European
               Workshop on Services and their Composition,
               ZEUS 2010, Berlin, Germany,
               February 25--26, 2010},
  booktitle = {Services und ihre Komposition},
  publisher = {CEUR-WS.org},
  series    = {CEUR Workshop Proceedings},
  volume    = {563},
  year      = {2010},
  url       = {http://CEUR-WS.org/Vol-563/}
}
```

Vorwort

Der Zentraleuropäische Workshop über Services und ihre Komposition (ZEUS) hat zum ersten Mal im März 2009 in Stuttgart statt gefunden. Im Vordergrund hat die gemeinsame Diskussion neuer Ideen für den Servicebereich (im Gegensatz zu fertigen Forschungsergebnissen) mit anderen Nachwuchswissenschaftlern aus Universitäten und Firmen gestanden.

Auf Grund des großen Erfolges dieses Konzepts haben wir entschieden, es auch für den zweiten ZEUS-Workshop, der am 25. und 26. Februar 2010 an der Humboldt-Universität zu Berlin statt gefunden hat, zu übernehmen. Wir freuen uns darüber, dass unter den Teilnehmern nicht nur Teilnehmer von ZEUS 2009, sondern auch einige neue Gesichter zu finden gewesen sind. Letzteres sehen wir als deutlichen Indikator dafür, dass wir dem zweiten Ziel des ZEUS Workshops, dem Aufbau eines Netzwerks für Nachwuchswissenschaftler aus dem Servicebereich im zentraleuropäischen Raum, mit dem zweiten ZEUS Workshop einen Schritt näher gekommen sind.

Von den eingereichten Beiträgen haben wir 18 in das Programm aufgenommen. Das aus akademischem Bereich und Wirtschaft stammende Programmkommittee hat vorher alle Beiträge auf Relevanz hin überprüft. Ziel der Begutachtung ist es jedoch, den Autoren detaillierte Hinweise und Anregungen zu Inhalt und Qualität ihres Beitrags zu geben. In Folge einer zweiten Bearbeitungs- und Begutachtungsrunde werden schließlich 15 Beiträge elektronisch als Vol. 563 der CEUR-WS-Reihe veröffentlicht.

Auf dem Workshop hat Mathias Weske (Hasso-Plattner-Institut Potsdam) einen eingeladenen Vortrag gehalten. Wir haben den Eindruck, dass dieser und alle anderen Vorträge genügend Stoff für rege Diskussionen geboten haben, neue Kontakte über die eigene Forschungsgruppe hinaus entstehen ließen und jeder Teilnehmer wertvolle Anregungen mit nach Hause genommen hat. Dies bestätigte auch die sich an den Workshop anschließende Umfrage unter den Teilnehmern.

März 2010 Christian Gierds, Jan Sürmeli

Preface

In March 2009, the first Central-European Workshop on Services and their Composition (ZEUS) took place in Stuttgart. Discussing new ideas (instead of full-fledged results) in the area of services with fellow young researchers from universities and companies was the focus of the workshop.

Based on the success of the first ZEUS workshop, we had decided to choose the same conception for the second ZEUS workshop which took place at the Humboldt-Universität zu Berlin on February 25 and 26, 2010. We were glad that not only ZEUS 2009 participants but also a number of newcomers decided to participate in ZEUS 2010. We understand this as a clear indicator that ZEUS 2010 helped to achieve the second goal of ZEUS: to establish a scientific network for young researchers in central Europe.

We selected 18 submissions for the workshop program. A program committee consisting of members of academics and economy checked submissions for relevance. However, the main goal of the reviewing process was to provide the authors with useful hints and feedback about the quality of their submissions. Additionally, 15 papers were published electronically in Vol. 563 of the CEUR-WS series, after a second editing and reviewing phase following the workshop.

Mathias Weske (Hasso Plattner Institute Potsdam) gave a keynote speech at the workshop. We are convinced that this and all other talks provided a good basis for rich discussions and for establishing new contacts between the participants as well as equipped each participant with valuable feedback and ideas to take home. The survey following the workshop confirmed our impression.

February 2010 Christian Gierds, Jan Sürmeli

Organizers

Christian Gierds, Humboldt-Universität zu Berlin
Jan Sürmeli, Humboldt-Universität zu Berlin

Program Committee

Gero Decker, Signavio
Christian Gierds, Humboldt-Universität zu Berlin
Katharina Görlach, University of Stuttgart
Oliver Kopp, University of Stuttgart
Frank Michael Kraft, SAP
Niels Lohmann, University of Rostock
Jan Mendling, Humboldt-Universität zu Berlin
Christian Stahl, Eindhoven University of Technology
Vladimir Stantchev, Technische Universität Berlin
Jan Sürmeli, Humboldt-Universität zu Berlin
Hagen Völzer, IBM Research Zurich
Matthias Weidlich, Hasso Plattner Institute Potsdam

Additional Reviewers

Anja Monakova, University of Stuttgart
David Schumm, University of Stuttgart
Tobias Unger, University of Stuttgart

Contents

Communication models for services

Niels Lohmann[1,2]

[1] Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, P. O. Box 513, 5600 MB Eindhoven, The Netherlands
[2] Universität Rostock, Institut für Informatik, 18051 Rostock, Germany
niels.lohmann@uni-rostock.de

Abstract. Communication is an essential aspect of services. Services do not only realize simple request-response scenarios, but increasingly implement complex and stateful communication protocols. Such a protocol specifies the order in which messages are sent and received by a service and is an essential part of a service description. Services are usually not executed in isolation, but as a collaboration which is composed of several services. The behavior of such a collaboration is not only determined by the communication protocol of each participating service, but also by the way messages are exchanged. A *communication model* specifies the properties of the message channels between the services and defines the way how messages are sent and received. This paper studies and classifies several dimensions of communication models and describes their impact to the behavior of service compositions.

1 Introduction

The paradigm of service orientation aims at replacing complex monolithic systems by a composition of several simpler, yet logically or geographically distributed components, called services. This distributed nature introduces new problems, because independent services need to collaborate to reach a common goal. One way to achieve correct collaboration is to offer *standardized* service descriptions. The most prominent example is the language WS-BPEL [3] which emerged as standard to specify executable Web services as well as to describe communication protocols on an abstract level. WS-BPEL received much attention from academia, and there exists a variety of formalizations of the communication protocol of a WS-BPEL process; van Breugel and Koshkina [7] present a survey of hundreds of papers. However, these formalizations usually focus on the execution order of the service's activities and provides an answer to the question *when* communication takes place, but give little details on the way *how* messages are exchanged.

Although a communication protocol specifies many dependencies between activities — be it internally or between one service's send activity and another service's receive activity — other details remain unspecified. Is the message exchange atomic, or is sending and receiving decoupled? Can the receiver block the sender? Can messages be buffered, and if yes, how many of them? Each of these questions is not addressed by the WS-BPEL specification. However, they still have an impact on the behavior of the overall collaboration.

We observed that — apart from a vague distinction between *synchronous* and *asynchronous* message exchange — there does not exist a common understanding how communication should be modeled. This in turn makes it hard to compare existing results on the formalization and analysis of services. To this end, this paper studies *communication models*. A communication model specifies the aspects of the message exchange such as those previously sketched. It may also include the occurrence of *faults*. A fault is an undesired and abnormal scenario (i. e., a buffer overflow) for that the subsequent behavior is unspecified. A communication model thereby can be seen as the missing piece to completely describe and reason about the behavior of a service composition. Note that communication models are not tied to any specific service description language or formalization.

This paper is not a survey on existing service description languages or service formalizations; we refer the interested reader to survey [11,16]. Actually, Kazhamiakin et al. [16] already study a wide spectrum of communication models and investigate questions related to their expressive power. Compared with that approach, this paper aims at providing an intuitive classification of communication models which is not guided by expressive power or a concrete formalization; the dimensions studied in Sect. 2 are more general than those of Kazhamiakin et al. Furthermore, this paper does not discuss the suitability of certain communication models nor tries to compare formalisms or service description languages with respect to their assumed communication model(s) as it is done by approaches such as the *service interaction patterns* [4]. Instead, we want to highlight certain effects and consequences which arise with the choice of a certain communication model. We do this by examining in Sect. 3 the service's behavior from the point of view of partner services. Section 4 concludes the paper and lists several directions of future work.

2 Dimensions of communication models

The most apparent impact of the choice of a communication model becomes visible in the level of abstraction of the message buffer; that is, the infrastructure which transfers messages between two services. In case of *synchronous communication*, message exchange is assumed to be instantaneous: Messages are not buffered and no intermediate state in which a sent message is pending is modeled. Thereby, the meaning of the term "synchronous" is closer to "isochronous" rather than "synchronization", because the latter can also be realized with two messages modeling a handshake.

On the other hand, sending of a message can be decoupled from receiving; that is, sent messages are buffered until they are received and, as a consequence, communication is *asynchronous*. Even though considering intermediate states yields in an increased complexity, asynchronous communication allows for more efficient communication, because sender and receiver do not need to constantly synchronize, but can be executed more autonomously. Thereby, asynchronous communication naturally supports the distributed setting of services.

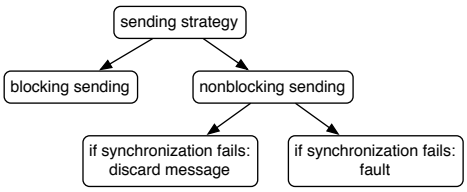


Fig. 1: Dimensions of synchronous communication models.

2.1 Synchronous message exchange

In the simplest communication model, message exchange is atomic. The sender waits until the receiver is ready for the message exchange, and then they synchronize by executing the sending and the receiving activity simultaneously. In this setting, the state in which the message is sent, but not yet received, is not modeled: the message channel is abstracted from. Consequently, the message exchange is executed in an all-or-nothing fashion. It is notable that the direction of the message exchange is irrelevant from a technical point of view, because there exists no distinguished initiator.

Variants of this synchronous communication loosen this symmetry between sender and receiver and allow for nonblocking execution of the sending activity. In case the sending activity can be executed independently from the receiving activity, the setting in which the receiving activity is not ready for execution (i. e., the synchronization fails) needs to be specified. Then, the message can be either discarded or a fault occurs. The former scenario is motivated by signal nets [17] which introduce one-sided synchronization of modules in the setting of control engineering. Desel [11] describes an application to services.

Figure 1 provides an overview of the different dimensions of synchronous communication models. The most prominent service model using synchronous communication is the "*Roman model*" [5] in which message transfer is specified by a synchronous communication model that assumes blocking sending. The same communication model is also used by earlier formalizations of interaction such as *I/O automata* [19] or *interface automata* [8].

2.2 Asynchronous message exchange

In contrast to synchronous message exchange, asynchronous communication models refine the message exchange and decouple the sending of a message from its receiving. This is usually motivated by performance issues, and the fact that the actual moment a message is received should be modeled independently of the moment of the sending. Consequently, the state in which the message is in transit (i. e., already sent but not yet received) is explicitly modeled. Consequently, an asynchronous communication model not only needs to specify the characteristics of the sending of a message, but also its transfer and its receipt. These characteristics can be grouped into properties of the employed message buffer (including how it can be accessed by the receiving activity) and sending strategies.

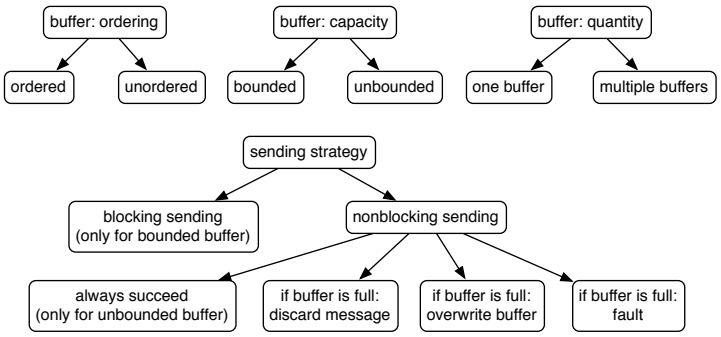


Fig. 2: Dimensions of asynchronous communication models.

Message buffers. An asynchronous communication model needs to specify three aspects of message buffers: their capacity, their organization, and their quantity.

The *capacity* of a message buffer determines the maximal number of messages which can be simultaneously buffered. In literature, the case in which a message buffer is unbounded is often considered. This absence of a capacity is typically motivated as a proper abstraction from a concrete, but unknown upper bound. Actually, the determination of the maximal capacity is not trivial, because it is not only influenced by the service under consideration, but also by those services which are composed to it. Nevertheless, the need of a finite and fixed capacity is motivated by the middleware which realizes the communication of services in reality. If the capacity is not known in advance, it may be approximated using capacity considerations or static analysis techniques, or it is chosen sufficiently large.

The second aspect specifies how buffered messages are *organized*. As Kazhamiakin et al. [16], we distinguish ordered and unordered buffers. In the first case, the order in which messages are added to the buffer is preserved, and the receiver only has access to the "oldest" (i. e., earliest sent) element. Ordered buffers are usually modeled by FIFO queues. In the second case, messages are buffered unordered, modeling a channel in which messages may overtake one another. Consequently, the receiver has access to all buffered messages. This scenario over-approximates any transfer delay an asynchronous medium can introduce.

Finally, communication can be organized using *multiple* buffers. In the setting of ordered buffers, this allows the receiver to access more messages and hence may enable more message-receiving activities compared with the case where only one message can be accessed. For the unordered case, multiple buffers do not introduce additional behavior. In addition, by organizing each message in a separate ordered buffer, unordered buffers can be simulated.

Sending strategies. Similar to synchronous message exchange, different parameters influence the way sending activities are executed. For unbounded buffers, a nonblocking sending is the simplest case. In case a buffer capacity is assumed,

the situation in which the buffer is full needs to be specified. Either, the message cannot be sent (blocking) or sending is nonblocking and the message is discarded, a buffered message is overwritten, or an error occurs.

Figure 2 diagrams the four dimensions of asynchronous communication models. Obviously, the buffer capacity and the buffer quantity need to be explicitly specified further than "bounded" and "multiple", respectively.

As an example from literature, *open nets* (previously called *open workflow nets*) use interface places to model a message buffer. Using Petri net places yields an asynchronous communication model with a single unordered buffer. The model itself does not pose a capacity of the buffer and hence open nets impose an "always succeed" sending strategy. Similarly, *concurrent automata* [2] assume a multiset as channel model, yielding a single unordered buffer. *Communicating FSM* [6] employ unbounded ordered buffers (i. e., FIFO queues) to model communication. Hence, sending is nonblocking. In case a service communicates with more than one other service, one FIFO queue is assumed for each pair of services.

3 Impact of communication models

The previous section sketched different dimension of communication models. In this section, we demonstrate how the choice of a communication model has an impact on the controllability of a service. A service is *controllable* [22] if there exists a partner service such that their composition can always eventually reach a final state in which the message channels (if modeled) are empty. To visualize service models, we use BPMN [21] as graphical notation.

As a first example, consider the service in Fig. 3(a). It controllable if we assume synchronous communication. An asynchronous communication model would need to specify a buffer capacity of at least 2 in case messages are not discarded, because a communication partner cannot observe the receipt of the A message. Figure 3(b) shows compatible partner for bounded channels with a "discard" strategy.

Figure 3(c) demonstrates the impact of ordering of buffered messages: this service can only be controlled with synchronous communication or ordered buffers, because reordering of messages A and B results in a situation in which a partner needs to "guess" whether to send a C or D message, and any guess could yield unreceived messages. A similar situation occurs in the interaction with the service in Fig. 3(d). This service cannot be controlled with an asynchronous communication model at all, because the result internal choice (modeled by an XOR gateway) cannot observed by a communication partner. In contrast, a partner with a synchronous communication model with blocking sending, however, can control the service.

Figure 3(e) shows an example of the impact of the buffer quantity. If we assume a single ordered buffer, this service is controllable, for instance by the service in Fig. 3(f). If we assume the transfer of message E takes very long (e. g., in case of a large video file), we cannot speed up the service composition by sending it concurrently to other messages as it is done in Fig. 3(g) which would

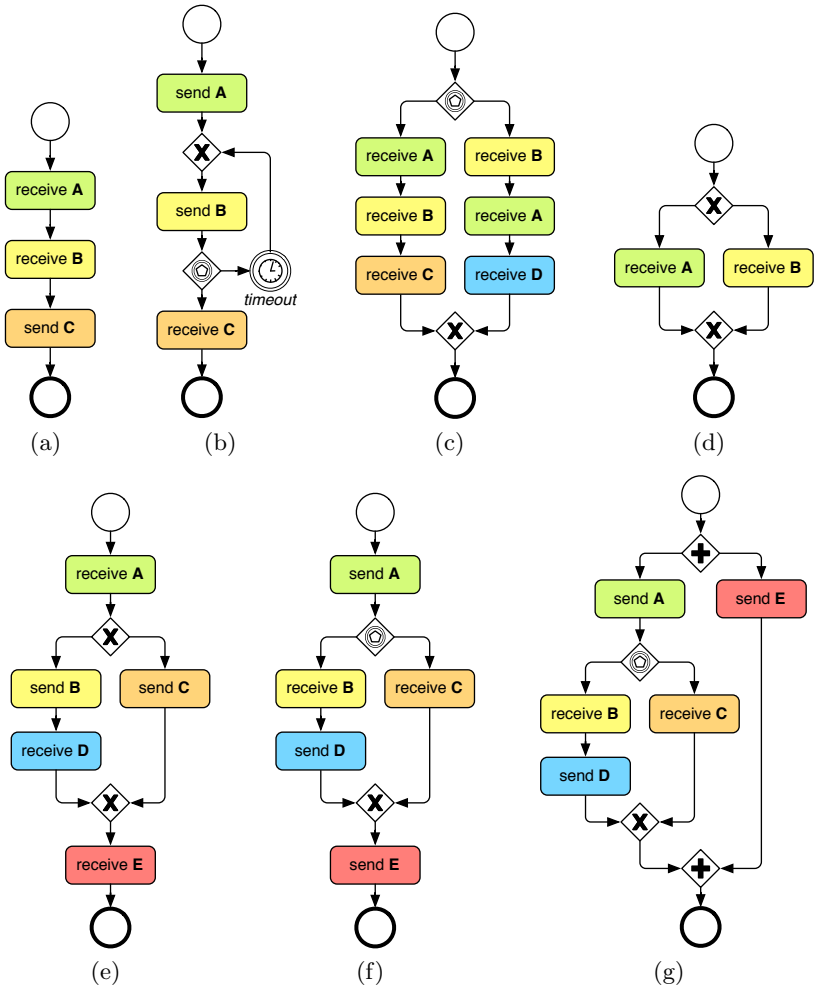


Fig. 3: Impact of communication models to service behavior.

be a correct partner in case we assume a distinct buffer for message E or an unordered buffer.

The presented examples show that the communication model has an impact to the overall behavior of a service. In particular, message buffers may introduce "new" behavior by enabling message-receiving activities. This additional behavior may yield in concurrent and more flexible service compositions on the one hand, but also in undesired problems such as deadlocks or unreceived messages on the other hand. In the context of unordered asynchronous communication, we already studied the reasons which may lead to uncontrollable service models [18]. Further impacts of synchronous communication on controllability are reported by Wolf [23] and Wolf [22].

4 Conclusion

We briefly studied communication models. By sketching different dimensions, we refined and systematized the vague synchronous/asynchronous distinction which can be found in the literature. Furthermore, we demonstrated the impact of different communication models with respect to controllability. We showed that one and the same service model (given as BPMN diagram) can be controllable or uncontrollable depending on the chosen communication model.

Future Work

Directions of future work are manifold. First, we plan to extend the dimensions of the communication models with other aspects related to instantiation semantics [10], lossy channels, or topologies. Second, this more detailed distinction can be used to classify and categorize further service formalisms from literature with respect to their assumed communication model. This would not only help to better compare and transfer results, but also to understand which communication model is required to study certain properties. We already sketched the impact to controllability in Sect. 3. Similarly, Kazhamiakin and Pistore [15] study the impact of communication models to choreography realization [12] and provide an algorithm which finds the "simplest" communication model under which a given choreography can be realized.

There already exist several approaches to translate between different communication models. Fu et al. [13] investigate necessary conditions for *synchronizability*, viz. when it possible to safely abstract from channels. This abstraction is motivated by the availability of more efficient verification techniques for synchronously communicating services. The converse direction from synchronous to asynchronous communication, called *desynchronizability*, is studied by Decker et al. [9]. They report of problems that may arise if a synchronous service choreography is implemented by asynchronously communicating services. Here, the transformation is motivated by the statement that atomic synchronous communication is an unrealistic assumption in the area of inter-organizational business processes.

Finally, the classification further needs to be further differentiated against many approaches to characterize several aspects in the area of business processes and services in terms of*patterns*. Whereas the control flow of a service can be investigated using *workflow patterns* [1], other approaches such as *enterprise integration patterns* [14], *service interaction patterns* [4], or *PAIS patterns* [20] also take interaction into account. Consequently, we plan to investigate similarities between interaction models and these patterns.

References

1. Aalst, W.M.P.v.d., Hofstede, A.H.M.t., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases 14(1), 5–51 (2003)
2. Alur, R., Etessami, K., Yannakakis, M.: Inference of message sequence charts. IEEE Trans. Software Eng. 29(7), 623–633 (2003)

3. Alves, A., et al.: Web Services Business Process Execution Language Version 2.0. OASIS Standard, OASIS (2007)
4. Barros, A.P., Dumas, M., Hofstede, A.H.M.t.: Service interaction patterns. In: BPM 2005. pp. 302–318. LNCS 3649, Springer (2005)
5. Berardi, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Mecella, M.: Automatic composition of e-services that export their behavior. In: ICSOC 2003. pp. 43–58. LNCS 2910, Springer (2003)
6. Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM 30(2), 323–342 (1983)
7. Breugel, F.v., Koshkina, M.: Models and verification of BPEL (2006), unpublished manuscript, available at `http://www.cse.yorku.ca/~franck/research/drafts/tutorial.pdf`
8. de Alfaro, L., Henzinger, T.A.: Interface automata. In: ESEC 2001. pp. 109–120. ACM (2001)
9. Decker, G., Barros, A., Kraft, F.M., Lohmann, N.: Non-desynchronizable service choreographies. In: ICSOC 2008. pp. 331–346. LNCS 5364, Springer (2008)
10. Decker, G., Mendling, J.: Instantiation semantics for process models. In: BPM 2008. pp. 164–179. LNCS 5240 (2008)
11. Desel, J.: Controlling Petri net process models. In: WS-FM 2007. pp. 17–30. LNCS 4937, Springer (2007)
12. Fu, X., Bultan, T., Su, J.: Conversation protocols: a formalism for specification and verification of reactive electronic services. Theor. Comput. Sci. 328(1-2), 19–37 (2004)
13. Fu, X., Bultan, T., Su, J.: Synchronizability of conversations among Web services. IEEE Trans. Software Eng. 31(12), 1042–1055 (2005)
14. Hohpe, G., Woolf, B.: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley Professional (2003)
15. Kazhamiakin, R., Pistore, M.: Analysis of realizability conditions for Web service choreographies. In: FORTE 2006. pp. 61–76. LNCS 4229, Springer (2006)
16. Kazhamiakin, R., Pistore, M., Santuari, L.: Analysis of communication models in web service compositions. In: WWW 2006. pp. 267–276. ACM (2006)
17. König, R., Quäck, L.: Petri-Netze in der Steuerungstechnik. Verlag Technik, Berlin (1988)
18. Lohmann, N.: Why does my service have no partners? In: WS-FM 2008. pp. 191–206. LNCS 5387, Springer (2009)
19. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann (1996)
20. Mulyar, N.: Patterns for process-aware information systems: an approach based on colored Petri nets. Ph.D. thesis, Technische Universiteit Eindhoven, Eindhoven, The Netherlands (2008)
21. OMG: Business Process Model and Notation, V1.1. OMG Available Specification, Object Management Group (2008)
22. Wolf, K.: Does my service have partners? LNCS T. Petri Nets and Other Models of Concurrency 5460(2), 152–171 (2009)
23. Wolf, M.: Synchrone und asynchrone Kommunikation in offenen Workflownetzen. Studienarbeit, Humboldt-Universität zu Berlin, Berlin, Germany (2007), (in German)

Anforderungen an ein Metamodell für SOA-Repositories

Stephan Buchwald[1], Julian Tiedeken[1,2], Thomas Bauer[1], Manfred Reichert[2]

[1]Abteilung für Daten- und Prozessmanagement, Daimler AG,
{stephan.buchwald, julian.tiedeken, thomas.tb.bauer}@daimler.com
[2]Institut für Datenbanken und Informationssysteme, Universität Ulm
{manfred.reichert, julian.tiedeken}@uni-ulm.de

Abstract: Service-orientierte Architekturen (SOA) gewinnen in Unternehmen zunehmend an Bedeutung. Insbesondere die lose Kopplung von Services verspricht mehr Flexibilität. Durch die Vielzahl an Services und Prozessen in unterschiedlichen Varianten sowie deren gleichzeitige Verwendung durch Service-Nutzer, entstehen hohe Kosten für Betrieb und Wartung. Services, die nicht mehr genutzt werden, sollten deshalb zeitnah „abgeschaltet" werden. Um solche nicht mehr benötigten Services identifizieren zu können, muss u.a. bekannt sein, welche Services aktuell von wem benutzt werden. Zudem entstehen durch unterschiedliche Versionen von Services komplexe Abhängigkeiten, die durch eine geeignete Informationsverwaltung im SOA-Repository beherrscht werden müssen. Dieser Beitrag stellt die in diesem Zusammenhang bestehenden Anforderungen an ein Metamodell dar.

1 Motivation

Service-Orientierung ist ein wichtiges Architekturprinzip für Unternehmen. Die Ziele einer Service-orientierten Architektur (SOA) [1, 7, 9] sind vielschichtig. Sie reichen von einer Erhöhung der Flexibilität durch Adaptierbarkeit der Geschäftsprozesse an geänderte Rahmenbedingungen [5, 13] bis hin zu kürzeren Entwicklungszeiten sowie reduzierten Kosten für Service- und Prozess-orientierte Informationssysteme. Generell beschreibt eine SOA nicht nur ein technisches Paradigma. Vielmehr versucht sie das Zusammenspiel zwischen Fachbereichen und IT-Abteilungen zu verbessern, was in der Literatur auch als Business-IT-Alignment bezeichnet wird [3]. D.h. Informationssysteme sollen fachliche Anforderungen exakter treffen als bisher.

In einer SOA wird die Geschäftsprozessmodellierung durch fachliche Services und fachliche Datenobjekte unterstützt, deren Dokumentation aber abstrakt gehalten ist. Sie werden auf Implementierungsebene ggf. durch mehrere technische Services und Datenobjekte verfeinert. Das Zusammenspiel und die Beziehung zwischen fachlichen und technischen Services, Prozessen und Datenobjekten sind in der Praxis meist nicht ausreichend dokumentiert. Deshalb entstehen zahlreiche Probleme: So kann bei Außerbetriebnahme eines Services nicht immer nachvollzogen werden, welche Prozesse oder Applikationen davon betroffen sind. Dadurch ist es schwierig, sicherzustellen, dass die Abschaltung nicht zu unerwarteten Fehlern führt. Zusätzliche Komplexität entsteht dadurch, dass sowohl fachliche als auch technische Services (und Datenobjekte) in unterschiedlichen Versionen existieren. Um die Metainformation zu all die-

sen Objekten sowie relevante Beziehungen zwischen ihnen beherrschen zu können, ist ein Repository notwendig, welches die Daten speichert. Durch deren logisch zentrale Verwaltung wird mehr Transparenz zu Objektabhängigkeiten geschaffen, was Inkonsistenzen nach Änderungen an Objekten (z.B. fachliche und technische Services oder Datenobjekte) erkennbar macht sowie eine Reaktion darauf zulässt. Außerdem muss die Entwicklung neuer Applikationen dahingehend unterstützt werden, dass relevante Informationen in jeder Phase des Entwicklungsprozesses durch das SOA-Repository bereitgestellt werden. Um möglichst schnell von fachlichen Anforderungen zu deren IT-Implementierung zu gelangen, ist eine durchgängige Modellierungsmethodik [2] notwendig, die ebenfalls durch das SOA-Repository unterstützt werden muss.

Dieser Beitrag stellt erstmalig und vollständig Anforderungen an das Metamodell eines zentralen SOA-Repositories aus Praxissicht vor. Dies legt die Basis zur Entwicklung eines Gesamt-Metamodells für SOA-Repositories. Dazu betrachten wir in den Kapiteln 2 bis 4 Ziele einer SOA und leiten daraus Anforderungen an das Metamodell ab. Kapitel 5 diskutiert verwandte Arbeiten, bevor mit einer Zusammenfassung geschlossen wird.

2 Nutzung angebotener Services

Eine zentrale Idee von SOA besteht darin, die Services verschiedener Anbieter einheitlich zu nutzen. Services abstrahieren dabei Funktionalitäten, die von anderen Applikationen verwendet werden können. Durch ihre Nutzung entsteht eine lose Kopplung zwischen Anbieter (engl.: *provider*) und Nutzer (*consumer*). Damit Service-Nutzer die richtigen Services finden und anschließend verwenden können, müssen Service-Informationen geeignet dokumentiert werden.

Anforderung 1: Service-Publikation und -Kontrakt

Bevor ein Service genutzt werden kann, sollte er in einem SOA-Repository publiziert werden. Anschließend kann ein potentieller Nutzer im Repository nach Services suchen, z.B. unter Verwendung von Suchkriterien wie angebotene Funktionalität oder verwendete Datenobjekte. Zur Ausführungszeit muss die physische Lokation eines Services mittels des Repositories ermittelbar sein, damit der Service-Aufruf durchgeführt werden kann. Der Einsatz eines Proxies (bspw. Enterprise Service Bus) ermöglicht die Entkopplung des Service-Aufrufs, d.h. Service-Nutzer rufen nicht direkt den konkreten Service, sondern einen Proxy-Service. Letzterer ermöglicht ein dynamisches Binden sowie eine Endpunktauswahl des tatsächlichen Services.

Für die Verwendung eines Services ist eine Nutzungsvereinbarung (*contract*) zwischen Anbieter und Nutzer notwendig. Diese klärt Ansprüche und Pflichten und beinhaltet Informationen zu Nutzungskosten, Antwortzeiten, Verfügbarkeit und Sicherheitsaspekten. Dem entsprechend sind die im folgenden ER-Diagramm in Min-Max-Notation skizzierten Objekte und Beziehungen im SOA-Repository zu verwalten (Attribute nur teilweise dargestellt):

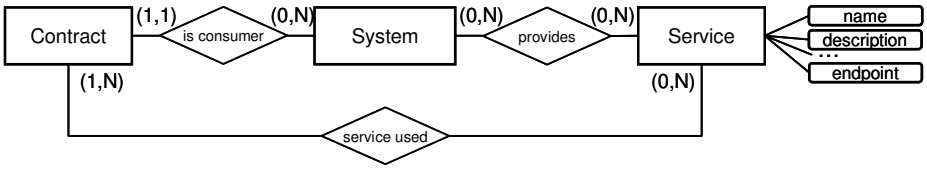


Abbildung 1: Kontrakt zwischen Service und System (Anbieter und Nutzer)

Ein *System* (Service-Anbieter bzw. Service-Nutzer) innerhalb eines Unternehmens kann sowohl ein Prozess oder eine Applikation (etwa J2EE [14]) sein, welches seine atomaren Funktionalitäten in Form von Services anbietet oder benötigte Funktionalität über Services konsumiert. Ein *Contract* ist dabei stets exakt einem *System* zugeordnet, wohingegen ein *System* beliebig viele *Contracts* hält.

Anforderung 2: Domänen zur Strukturierung

Die Untergliederung eines Unternehmens in Domänen auf organisatorischer Ebene dient u.a. dazu, einen Überblick über Funktionalität und Geschäftsobjekte verschiedener Fachbereiche zu erhalten [6]. Dabei werden unternehmensrelevante Funktionen sowie bestehende fachliche und technische Services der verschiedenen (Sub-) Domänen erfasst. Dadurch wird transparent, welche Funktionalität durch welche (Sub-) Domänen realisiert wird und wo Redundanzen bestehen.

Um Services auch Domänen-basiert suchen zu können, muss das SOA-Repository explizit die Beziehungen zwischen (Sub-)Domänen und angebotenen Services speichern. Damit sich Änderungen an Services effizient koordinieren lassen, werden Domänenverantwortliche im Repository verwaltet.

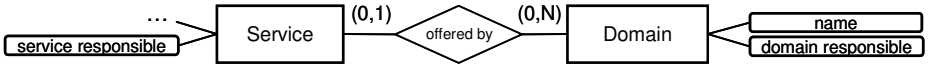


Abbildung 2: Service-Zuordnung zu Domänen

Anforderung 3: Informationserzeugung und -verwendung

Bei der Entwicklung eines Services sind neben unterschiedlichen Personengruppen verschiedene Werkzeuge im Einsatz. Diese dienen der fachlichen Beschreibung bzw. technischen Implementierung von Services. Sie erzeugen neue Informationen, verwenden aber auch existierende Dokumente aus dem SOA-Repository. Während der fachlichen Prozessmodellierung etwa werden neue fachliche Services erzeugt und bereits vorhandene weiterentwickelt. Diese verwenden Geschäftsobjekte, welche die Ein- und Ausgabeparameter der fachlichen Services darstellen. Die fachliche Service-Spezifikation (Fachspezifikation) bildet die Grundlage für die technische Service-Implementierung.

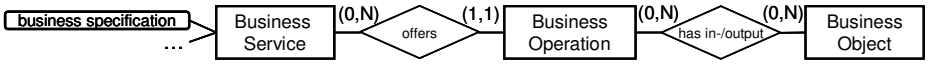


Abbildung 3: Fachliche Beschreibung von Services

Analog wird eine technische Service-Spezifikation (WSDL) inklusive technischer Operationen und den verwendeten Parametern (Datenschemata z.B. als XSD) im Repository gespeichert. Als Attribute werden zu einem technischen Service ein Nutzungshandbuch und nach der Implementierung und Installation ein Endpunkt bereitgestellt.

3 Gewährleistung der langfristigen Stabilität einer SOA

Auch im Kontext von Änderungen der Service-Landschaft müssen Service-nutzende Applikationen zuverlässig funktionieren. Jede freigegebene Änderung an Repository-Objekten sollte deshalb wieder zu einer gültigen Service-Landschaft führen: Es dürfen keine Inkonsistenzen entstehen, etwa durch Abschalten eines Services, für den eine Nutzung noch vertraglich garantiert wird. Im Folgenden betrachten wir Anforderungen an das Repository-Metamodell, welche die Stabilität einer SOA unterstützen.

Anforderung 4: Service-Lifecycle-Management

Während der Entwicklung einer Prozessapplikation durchlaufen Services unterschiedliche Zustände. Sie beschreiben z.B., ob für einen Service bereits eine Fachspezifikation vorliegt oder ob er bereits realisiert bzw. freigegeben ist. Die einzelnen Zustände, die ein Service durchlaufen kann, werden in einem Service-Lebenszyklus (*Service Lifecycle*) dokumentiert und beschrieben [9]. Ein Wechsel des Service-Zustands erfolgt erst, wenn alle Vorraussetzungen an den Nachfolgezustand erfüllt sind, etwa das Vorliegen einer Fachspezifikation oder eines Entwicklungshandbuchs. Die Kontrolle und Überwachung der Zustandsübergänge wird durch Governance-Prozesse realisiert. Sie regeln unter anderem das Änderungsmanagement von Services und Prozessen, indem sie festlegen, von wem Änderungen zu genehmigen sind. Entscheidungsgrundlage dafür ist die im SOA-Repository gespeicherte Information.

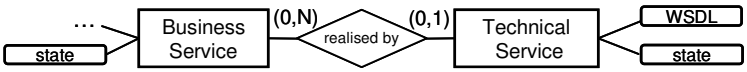


Abbildung 4: Speicherung von Service-Zuständen

Zur Sicherstellung der Qualität von Services werden Compliance-Prüfungen während ihrer Entwicklung durchgeführt, die eine korrekte und kontrollierte Realisierung gewährleisten sollen. Die dazu notwendigen Dokumente, etwa Fachspezifikationen oder WSDL-Beschreibungen, sind im SOA-Repository abzulegen.

Anforderung 5: Speicherung der Beziehungen zwischen Repository-Objekten

Fachliche und technische Artefakte, wie Service- oder Datenobjekte, stehen in direkter Beziehung zueinander: Ein fachlicher Service wird durch einen oder mehrere technische Services realisiert. Dabei verwendet eine Operation eines fachlichen Services Business-Objekte, die technisch auf ein oder mehrere Datenobjekte abgebildet werden. Um nach Änderungen an fachlichen Services, Operationen oder Business-Objekten festzustellen, ob bzw. welche technischen Artefakte angepasst werden müssen, sollten die Beziehungen zwischen diesen Objekten im Repository explizit verwaltet werden:

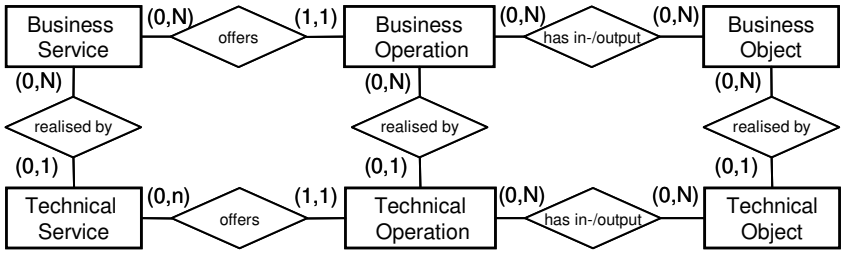


Abbildung 5: Abhängigkeiten zwischen fachlichen und technischen Services

Anforderung 6: Service-Versionen und Plantermine

Services werden in einer SOA kontinuierlich weiter entwickelt, wodurch neue Service-Versionen (sowohl fachlich als auch technisch) entstehen und veraltete „abgeschaltet" werden müssen. Durch die Vielzahl Service-konsumierender Applikationen ist eine Abschaltung jedoch nicht immer einfach durchzuführen, sondern macht für viele Applikationen eine aufwendige Anpassung erforderlich. Um Inkompatibilitäten zwischen Service-Anbieter und -Nutzer transparent zu machen, sollen neben den Abhängigkeitsbeziehungen auch Plantermine für die Abschaltung von Service-Versionen im Repository dokumentiert werden. Dadurch kann die Konsistenz überprüft und frühzeitig auf geplante Abschaltungen reagiert werden. Analog dazu können Plantermine eingeführt werden, welche die Nutzbarkeit einer neuen Service-Version anzeigen. Neue Service-Nutzer können dann z.B. prüfen, ob die neue Service-Version rechtzeitig zur Verfügung stehen wird oder ob eine ältere Version zu verwenden ist.

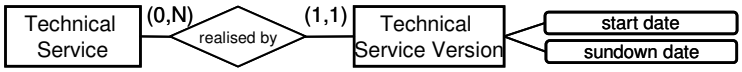


Abbildung 6: Versionierung von technischen Services

4 Permanente und durchgängige Speicherung von Modelldaten

Fachliche Prozesse und Services werden mit Werkzeugen (z.B. ARIS bzw. erweiterten Ereignisgesteuerten Prozess-Ketten) modelliert, wohingegen technische Prozesse häufig in CASE-Tools mit UML-Unterstützung dokumentiert werden. Infolge der unterschiedlichen Kenntnisse von Fach- und IT-Personal, werden fachliche Anforderungen oft unzureichend umgesetzt und es entsteht eine Lücke zwischen Fachbereich und IT-Abteilung. Deshalb ist es unabdingbar, die modellierte Information dauerhaft zu archivieren und diese Lücke zu schließen (Business-IT-Alignment [3]).

Anforderung 7: Langfristige Archivierung von Prozess- und Service-Modellen

Die langfristige Dokumentation von fachlichen und technischen Services sowie von Prozessen auf den Ebenen Fachmodell und technischem Modell ist essentiell. Hierdurch kann später nachvollzogen werden, welche konkreten Anforderungen und Fachprozesse durch eine Applikation bereits realisiert sind. Diese Information ist die Ausgangsbasis für eine Überarbeitung der Applikation (Application-Reengineering), nachdem sich Anforderungen geändert haben.

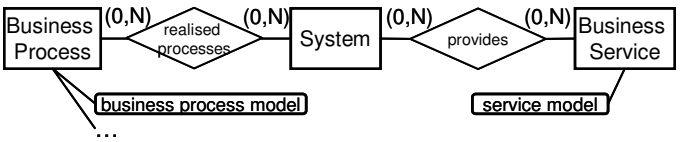


Abbildung 7: Archivierung von Modellinformationen

Anforderung 8: Beziehungen zwischen fachlichen und technischen Aktivitäten

Werden im SOA-Repository einzelne Aktivitäten der fachlichen und technischen Prozessmodelle und deren Beziehungen verwaltet, erhöht sich die Nachvollziehbarkeit, da Beziehungsrelationen zwischen fachlichen Anforderungen, fachlichen Services und ihrer Implementierung dokumentiert sind. Hierbei muss für jede Aktivität des fachlichen Modells (und damit für ihre Eigenschaften und Anforderungen) dokumentiert werden, durch welche Aktivität des technischen Modells diese realisiert wird [2]. Dadurch ist bei einer späteren Änderung fachlicher Aktivitäten leicht erkennbar, welche technischen Aktivitäten bzw. Services angepasst werden müssen. Zudem ist die Speicherung des Prozessmodells (*Business Process Model*) möglich, auch wenn die eigentliche Systemauswahl noch nicht abgeschlossen ist.

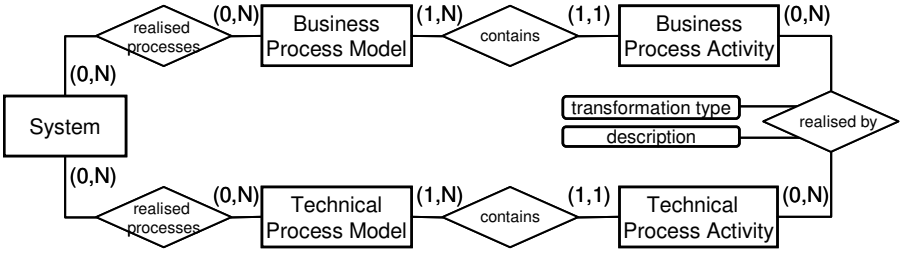


Abbildung 8: Beziehung zwischen fachlichen und technischen Aktivitäten

5 Verwandte Arbeiten

Heutige Repositories legen ihren Schwerpunkt entweder auf die Speicherung und Verwaltung technischer Artefakte (z.B. UDDI [12], WSRR [4]) oder Software- Komponenten [10, 11]. Eingeschränkt betrachtet werden dabei die Möglichkeiten zur konsistenten und durchgängigen Modellierung sowie Dokumentation von fachlichen und technischen Services. So ist etwa die ARIS-interne Datenbank [8] in der Lage, fachliche und technische Artefakte zu verwalten. Die Beziehungen zwischen fachlichen und technischen Services werden im Regelfall aber nicht explizit verwaltet.

Einige existierende Repositories bieten Erweiterungsmöglichkeiten zur Realisierung eines umfassenden Metamodells. [7, 9] betonen die Notwendigkeit eines Repositories in einer SOA und fordern Funktionalitäten, wie die Bereitstellung von Service-Endpunkten für einen Enterprise Service Bus (ESB) oder die Möglichkeit für das Suchen und Auffinden von Services. Nicht betracht wird, welche konkreten Anforderungen an ein SOA-Repository existieren, d.h. welche konkreten Objekte in einem SOA-Repository verwaltet werden sollen und wie diese untereinander in Beziehung stehen.

6 Zusammenfassung und Ausblick

In diesem Beitrag haben wir wichtige Anforderungen an ein SOA-Repository diskutiert, die für eine konsistente Modellierung, Dokumentation, Verwaltung und Speicherung von fachlichen und technischen Services unverzichtbar sind. Aus diesen Anforderungen werden wir im Projekt *Enhanced Process Management by Service Orientation (ENPROSO)* ein umfassendes und in sich konsistentes Gesamt-Metamodell für SOA-Repositories ableiten. Bei der Realisierung eines SOA-Repositories sollten aufgrund der mannigfaltigen wechselseitigen Abhängigkeiten zwischen Objekten ggf. nicht alle vorgestellten Anforderungen in der ersten Version des SOA-Repositories realisiert werden. Eine Umsetzung kann etwa durch eine relationale Datenbank oder durch eine Erweiterung vorhandener SOA-Repositories (bspw. WSRR [4]) realisiert werden.

Literatur

1. S. Buchwald, T. Bauer, R. Pryss: IT-Infrastrukturen für flexible, service-orientierte Anwendungen – Ein Rahmenwerk zur Bewertung; In. Proc. 13. GI-Fachtagung Datenbanksysteme in Business, Technologie und Web, S. 524-543, Münster; 2009
2. S. Buchwald, T. Bauer, M. Reichert: Durchgängige Modellierung von Geschäftsprozessen in einer Service-orientierten Architektur; In. Proc. GI-Fachtagung Modellierung'10, Klagenfurt, 2010 (forthcoming)
3. H.-M. Chen: Towards Service Engineering, Service Orientation and Business-IT Alignment; In. Proc. 41st Hawaii Inf. Conf. on System Sciences; 2008
4. C. Dudley et al.: WebSphere Service Registry and Repository Handbook; 2007
5. P. Dadam, M. Reichert: The ADEPT Project: A Decade of Research and Development for Robust and Flexible Process Support. Springer, 2009
6. G. Engels et al.: Quasar Enterprise: Anwendungslandschaften serviceorientiert gestalten, dpunkt.verlag, 2008.
7. T. Erl: SOA: Concepts, Technology, and Design; Prentice Hall, 2005
8. IDS Scheer: ARIS SOA Architect - Geschäftsprozesse als Grundlage für Serviceorientierte Architekturen; IDS Scheer, 2008
9. N. M. Josuttis: SOA in Practice; The Art of Distributed System Design. O'Reilly, 2007
10. B. Li et al.: Building Interoperable Software Components Repository Based on MMF, Grid and Cooperative Computing (GCC) Workshops, 2004
11. M.J. Morel, J. Faget: The REBOOT Environment. Proceedings of the 2nd International Workshop on Software Reusability Advances in Software, 1993
12. OASIS: Universal Description, Discovery, and Integration (UDDI), Version 3.0, 2002
13. M. Reichert, D. Stoll: Komposition, Choreograhpie und Orchestrierung von Web Services: Ein Überblick. EMISA Forum, Vol. 24, No. 2, pp. 21-32, 2004
14. J. Keogh. J2ee: The Complete Reference. Osborne/McGraw-Hill, Berkeley, CA, USA, 2002.

On Designing a People-oriented Constraint-based Workflow Language*

Frank Leymann, Tobias Unger, and Sebastian Wagner

Institute of Architecture of Application Systems
University of Stuttgart, Universitätsstraße 38, 70569 Stuttgart, Germany
{leymann, unger, wagner}@iaas.uni-stuttgart.de

Abstract The control-flow of business workflows is characterized by the strict execution order of the activities that is already defined at design time. This well-structured control-flow is for instance absolutely necessary if the workflows have to be performed fully automatically. However, this rigidity is not always appropriate for people-oriented workflows. Especially in scenarios where real world processes are only semi-structured humans should have more freedom to decide in which order they want to perform the activities. In this paper, we suggest an approach to design people-oriented workflows via constraints to make them more flexible.

1 Introduction

Many workflow languages address the importance of human interactions (e.g. BPEL [1] with the BPEL4People [2] extension) by providing means to model human activities (in the following called tasks). They also address issues like the assignment of people to tasks. Nevertheless, they do not take the control-flow of a business process with human interaction into consideration. Especially workflows that consist solely of activities that are performed by humans (called people-oriented workflows in this paper) should be more flexible than automated business workflows to give them more freedom to decide in which order they want to perform certain tasks. In the following, we propose a declarative approach of modeling people-oriented workflows. We suggest different classes of constraints to model them. This also encompasses constraints that cover scenarios where people have to collaborate. Additionally, techniques are sketched to validate the workflow models and to verify that the constraints are met during workflow execution.

2 Requirements

As mentioned before, people-oriented workflows should not be as strict as automated business workflows because the human factor makes them more difficult to

* This work is partially funded by the ALLOW project. ALLOW (http://www.allow-project.eu/) is part of the EU 7th Framework Programme (contract no. FP7-213339).

predict. Of course, also these workflows have to reflect real-world constraints (e.g. a customer can only be charged after she bought a certain product). However, each human has his own preferred way to work. They are also skilled in scheduling tasks [3], which allows us to relax the model of a workflow. Moreover, humans are aware of their environment. In a classical workflow the execution of the whole workflow is interrupted when resources are missing to execute a certain task. The person who has to perform the task is idle, even if she knows that she possesses all information to perform one or more of the succeeding tasks. This lack of flexibility can decrease the productivity of humans and lead to the fact that they reject a workflow model.

3 Declarative Workflows

Our approach to overcome the rigidity of imperative workflows are *declarative workflows* [4]. In this modeling paradigm a workflow is defined by using constraints that specify what must be done or which conditions must be satisfied during the execution of the workflow. These constraints approximate the desired behavior of the workflow. Any execution order of tasks is allowed as long as it does not violate one of the constraints. This results in the fact, that the user has much more freedom to decide in which order she wants to perform the tasks.

Before discussing the different constraints that we have identified, a simple example is presented that shows the benefits of modeling a workflow by using the declarative approach. A travel agency provides a workflow where customers can book a trip. The imperative version of the workflow consists of the three consecutive task models `Book Flight`, `Book Hotel` and `Pay`. This implies, that the customer has to perform the tasks in this predefined order. A disadvantage of the strict execution order is for instance that the customer can not book the hotel first if she only plans to fly if her favorite hotel has free rooms available. With the declarative approach this could be modeled much less restrictive. It would contain only one constraint that ensures that the task *Pay* is the last task that is executed. The execution order of the other two tasks is not modeled. Consequently, the customer can decide if she books the hotel or the flight first.

3.1 Constraints

This section gives a brief overview about control-flow constraints that can be used along with task models to design a people-oriented workflow. A more comprehensive description of the constraints can be found in [5].

On each constraint a so-called *activation condition* can be defined to support scenarios where constraints should be only enabled when a certain condition holds (e.g. when a process variable has a specific value). If the condition is met the constraint has to be satisfied during workflow execution. Otherwise, the constraint is not enabled and it is ignored. A similar approach is described in [6].

Unary constraints: In [6] several constraints were introduced that can be defined on a single task model, they are called *unary constraints* here. An example

for these unary constraints is the *existence* constraint. It defines a lower and/or upper bound concerning the number of instances of a task model that must be executed. We extend this class by the *disable* constraint. If the *disable* constraint is defined on a task model no instances of this task model must be executed. The usage of this constraint makes only sense if an activation condition was defined on it. This prevents the execution of a task until the activation condition does not hold anymore.

Choice constraints: Another class of constraints suggested in [6] are the *choice constraints*. These constraints are used to specify that from a given set of task models a certain subset has to be chosen for instantiation and execution. If the constraint defines for instance that 2 instances from the set of task models $\{A, B, C\}$ have to be executed, one of the subsets $\{A, B\}$, $\{A, C\}$, $\{B, C\}$ of task models must be chosen for execution. However, the choice constraint defines only a lower bound concerning the size of the subsets, i.e. the execution of the task models $\{A, B, C\}$ is also valid. A more restrictive version is the *exclusive choice constraint* that was also introduced in [6]. When using this constraint the lower bound acts additionally as upper bound for the size of subsets of different task models that can be chosen for execution. If this constraint defines that 2 instances of the task models `A`, `B` and `C` must be performed, the instantiation and execution of the task model set $\{A, B, C\}$ is not permitted. A customer of an online book store has for instance the possibility to pay either by credit card, debit card or wireless transfer (each payment method is represented by a task model). To ensure that exactly one payment method is used an exclusive choice has to be defined that specifies that only one task model from the set of task models can be instantiated and performed.

Relation constraints: The class of *relation constraints* defines the execution order of the instances of two task models. In [6] several sequential relation constraints are described. They can be used to restrict the sequential execution order of two tasks. The *A precedence B* constraint is an example for these kind of constraints. It imposes the restriction that an instance of a task model `A` has always to be executed before an instance of task model `B`.

However, with sequential relation constraints it is not possible to model the requirement that two tasks must be performed exactly or at least partly at the same time. These kind of requirements usually emerge through collaborative work, i.e. for achieving a certain goal several tasks have to be executed simultaneously and each of them is performed by a different person. For instance when the pair programming technique is applied to develop software, one programmer writes the code (task A) and the other one has to review the typed code (task B) simultaneously. We introduce parallel relation constraints to model these kind of restrictions. Each of these constraints can be used to determine the degree of simultaneous execution (e.g. partly or completely). The constraints base on the interval relations of Allen's interval algebra [7] (briefly called IA). The IA was chosen because firstly, the interval relations proposed there cover all possible parallel relations between two intervals and secondly, algorithms exist for this algebra to verify that interval relations are consistent. An example for these kind

of constraints is the *A during B* constraint which defines that task *A* has to be started and completed during the execution of task *B*. To realize the pair programming example an *equals* constraint has to be defined between the task models `Review` and `Write Code`.

In [5] the relation constraints are extended by time parameters. These time parameters can be used to reflect temporal restrictions between the tasks (e.g. that a task has to be executed within a certain time after its predecessor task was completed). Furthermore, for each relation constraint also a corresponding *negation constraint* exists. These constraints provide the capability to prohibit a certain execution order. The negation constraints are also described in [5].

4 Workflow Model Validation

When creating a declarative workflow model contradictions between the constraints can emerge that are hard to discover at the first glance. This is especially true if many relation constraints were defined. To validate the consistency of the relation constraints we follow an approach similar to the one suggested in [8]. The relation constraints are transformed to interval relations of the IA and each task model is represented by an interval. The different relations between the intervals form a constraint network like the one that is illustrated in Figure 1. On the constraint network reasoning techniques that were introduced in [7] and in [9] are performed. These reasoning techniques infer new relations between all intervals and discover contradictions between them. For instance in Figure 1, it can be inferred from the relations "`C` has to be executed during the execution of `B`" and "`B` has to be executed before `D`" that `C` must be executed before `D`. Moreover, it can be also inferred that a contradiction exists in the network since `A` can not be performed after `D` but before `B`.

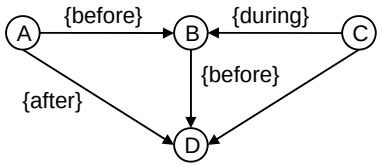


Figure 1. Example Constraint Network

5 Workflow Execution

During the execution of a workflow instance the workflow engine has to verify that all constraints are met. For the unary and choice constraints this can be done very easily. To check if the relation constraints were met we utilize the

constraint networks introduced in section 4. Since a network contains the interval relations that must hold between all pairs of tasks the engine can determine if a violation has occurred. If for instance task C in Figure 1 is not performed during the execution of task B the workflow is obviously violated. To reduce these violations the engine should actively decide which tasks can be started by the user. Based on the example before, the engine would not schedule task C for execution when task B is not executed. In [5] more detailed information about the workflow instance verification is provided.

The possible constraint violations have to be also reflected by the the lifecycle model of a declarative workflow. In Figure 2 an extended lifecycle model for declarative workflows is proposed. It bases on the lifecycle for imperative workflows that is suggested in [10]. For the sake of shortness, only the states are discussed that are different from those described in [10]. Here, the *running* state consists of the two substates *satisfied* and *temporarily violated* (these states are inspired by the constraint states proposed in [4]). A workflow instance is in the state *running satisfied* if the workflow is running and no constraints are violated. If a constraint is violated but the violation is still resolvable (e.g. by executing another task) the workflow is put into the *temporarily violated* state. An instance transitions into the *violated* state if it is permanently violated, i.e. if a constraint violation can not be resolved. In this state the workflow execution is interrupted and a process stakeholder can either stop the workflow or, if possible, perform corrective actions (e.g undo a task execution). As declarative workflows are not modeled by a directed acyclic graph there are no end-tasks that cause the workflow to be completed. Hence, declarative workflows have to be completed explicitly by an authorized user. As soon as the request to complete the workflow has been received it is put into the *completing* state. In this state no new tasks can be started anymore and all running tasks have to be completed. Since there is still the chance that constraints are violated during the completion of the workflow it can also transition into the *violated* state instead of the *completed* state.

6 Related Work

In [4], [6] and [11] a declarative approach for modeling and executing workflows is discussed. There is also a declarative workflow management system presented which is called Declare. As mentioned in 3.1, we extend the constraints proposed in [6] by time parameters to add support for temporal restrictions.

To provide a more flexible way for executing workflows in [12] a paradigm called *case handling* is proposed. The central concept behind this paradigm is the *case* which denotes a product that is created during the process execution (e.g. a document). A case consists of different data objects. The data objects are linked to one or more activities and an activity can be only started when its data objects are present. This means, that not control-flow related information drive the process but the state of the case, i.e. the existence of data objects. In this approach the user is focused on the whole case and not only on one activity like in traditional workflows.

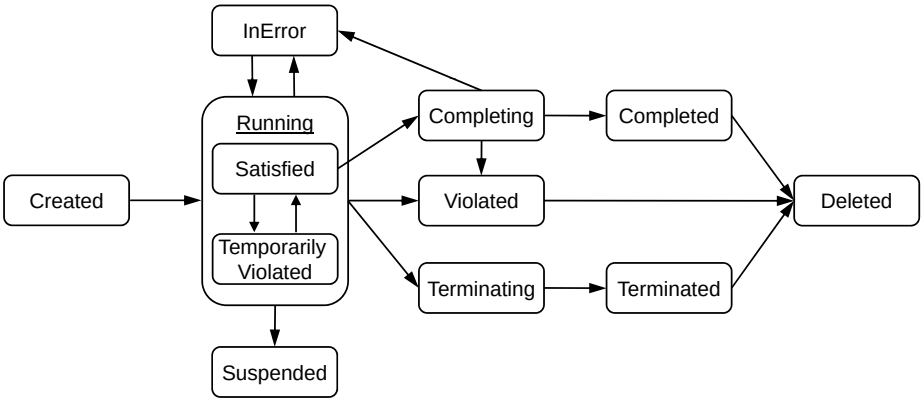


Figure 2. Declarative Workflow Lifecycle

BPEL4People [2] adds support for human interactions to BPEL. It extends BPEL by introducing *people activities* to enable human interactions with the BPEL process. It focuses on human interaction patterns, like the 4-eyes principle, escalation handling and nominations. The people activities are implemented by human tasks that are defined in the WS-HumanTask specification [13]. The concept of human tasks that is described there covers the most important aspects of human activities and could be used in conjunction with the constraints to model people-oriented workflows. In [5] an approach is presented that shows how this can be done.

7 Conclusion & Future Work

In this paper, we proposed a constraint-based language for modeling people-oriented workflows because the imperative paradigm tends to overspecify workflows and to be to restrictive. We introduced constraint networks to validate the consistency of these workflow models and to verify that the constraints were met during workflow execution.

A downside of the declarative approach is that these workflows are more difficult to understand by people because unlike in imperative workflows there is no directed graph that connects the different tasks. This is especially true when the workflow contains a lot of tasks and constraints. On the one hand, it is more sophisticated to observe the overall progress of a running workflow instance and on the other hand, it can be challenging to comprehend the transitive effects of the constraints. In future work, we plan to develop a proper visualization for the declarative workflows to make them more understandable for the end-users.

References

1. OASIS: Web Services Business Process Execution Language Version 2.0. Committee specification, OASIS WSBPEL TC (January 2007)
2. OASIS: WS-BPEL Extension for People Specification Version 1.1, Committee Draft 06. (2009)
3. Card, S.K., Newell, A., Moran, T.P.: The Psychology of Human-Computer Interaction. L. Erlbaum Associates Inc., Hillsdale, NJ, USA (1983)
4. van der Aalst, W.M.P., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. Computer Science - R&D **23**(2) (2009) 99–113
5. Wagner, S.: A Concept of Human-oriented Workflows. Diploma thesis, University of Stuttgart, Germany (January 2010)
6. Pesic, M.: Constraint-Based Workflow Management Systems: Shifting Control to Users. PhD thesis, Eindhoven University of Technology (2008)
7. Allen, J.F.: Maintaining knowledge about temporal intervals. Commun. ACM **26**(11) (1983) 832–843
8. Lu, R., Sadiq, S.W., Padmanabhan, V., Governatori, G.: Using a temporal constraint network for business process execution. In Dobbie, G., Bailey, J., eds.: ADC. Volume 49 of CRPIT., Australian Computer Society (2006) 157–166
9. Vilain, M.B., Kautz, H.A.: Constraint propagation algorithms for temporal reasoning. In: AAAI. (1986) 377–382
10. Leymann, F., Roller, D.: Production Workflow - Concepts and Techniques. PTR Prentice Hall (Januar 2000)
11. Pesic, M., Schonenberg, M.H., Sidorova, N., van der Aalst, W.M.P.: Constraint-based workflow models: Change made easy. In: OTM Conferences (1). (2007)
12. van der Aalst, W.M.P., Weske, M., Grünbauer, D.: Case handling: a new paradigm for business process support. Data Knowl. Eng. **53**(2) (2005) 129–162
13. OASIS: Web Services Human Task Specification Version 1.1, Committee Draft 06. (2009)

3D-Darstellung von Ressourcenattributen bei der Geschäftsprozessmodellierung

Daniel Eichhorn, Agnes Koschmider

Institut für Angewandte Informatik und Formale Beschreibungsverfahren (AIFB)
Karlsruher Institut für Technologie (KIT)
76128 Karlsruhe
{daniel.eichhorn|agnes.koschmider}@kit.edu

Abstract. 3D-Technologien eröffnen neue Möglichkeiten zur Modellierung von Geschäftsprozessen. In diesem Beitrag wird die dritte Dimension genutzt, um das Ablaufmodell mit dem Ressourcenmodell zu verknüpfen und um Ressourcenattribute kompakter darzustellen. Die dreidimensionale Ansicht von Ressourcenattributen kann zudem für die Simulation und Analyse von Ressourcen verwendet werden.

1 Einleitung

Modellierungswerkzeuge für Geschäftsprozesse unterstützen die Erstellung von Prozessmodellen in einer bestimmten Modellierungssprache. Die Unterstützung beschränkt sich derzeit noch vielfach auf syntaktische Vorgaben, eine kompakte Visualisierung verschiedener Prozess- und Dateninformationen wird vernachlässigt. 3D-Technologien eröffnen neue Möglichkeiten zur Modellierung von Geschäftsprozessen. Sie bieten eine höhere Anschaulichkeit und beseitigen bzw. reduzieren einige Defizite der herkömmlichen 2D-Prozessmodellierung, wie zum Beispiel die Begrenzung des Informationsumfangs in einem Prozessmodell.

Die Idee der Nutzung der dritten Dimension zur Prozessmodellierung ist nicht neu. [6] verwenden die dritte Dimension im Rahmen einer Fallstudie, um mehr Informationen miteinander kombinieren zu können. In [11] wurde diskutiert, wie die dritte Dimension genutzt werden kann, um Benutzer bei einer kompakten Darstellung und Animation von Geschäftsprozessen zu unterstützen. [13] beschreibt die Verwendung der dritten Dimension während der Simulation von Geschäftsprozessen.

In diesem Beitrag wird die dritte Dimension verwendet, um das Ablauf- und das Organisationsmodell miteinander zu verknüpfen und um den Einsatz von Ressourcen kompakt darzustellen. Damit sollen entsprechende Engpässe oder kritische Bereiche von Ressourcenattributen für den Anwender leichter erfassbar und analysierbar sein.

In den aktuell verfügbaren Modellierungssprachen ist eine Zuordnung von Ressourcen zu Aktivitäten möglich, allerdings sind weitere Modelle notwendig, um Ressourcenattribute zu erfragen.

In der Business Process Modeling Notation (BPMN) werden sogenannte Pools und Lanes verwendet, um die Ressourcenmodellierung mit der Ablaufmodellierung zu verbinden. Die Zuordnung von Ressourcen zu Aktivitäten ist ersichtlich, allerdings

werden Zusatzinformationen wie z.B. in welchem Maße fällt eine zeitliche oder finanzielle Belastung durch eine Aktivität an, in einem anderen Diagramm dargestellt.

Vor diesem Hintergrund wird in diesem Beitrag die dritte Dimension genutzt, um das Ablaufmodell mit dem Ressourcenmodell kompakter zu verbinden.

Der Beitrag gliedert sich wie folgt. Im Kapitel 2 werden Ressourcenattribute und ihre 3D-Darstellung beschrieben. Kapitel 3 schlägt eine dreidimensionale Ansicht zur besseren Analyse des Ressourceneinsatzes vor. Das letzte Kapitel fasst den Ansatz zusammen.

2 3D-Repräsentation von Ressourcen

In diesem Kapitel wird zunächst eine Klassifizierung von Ressourcen und Ressourcenattribute skizziert. Anschließend wird eine 3D-Repräsentation für Ressourcenattribute vorgestellt, die eine leichtere Erfassbarkeit der Zusammenhänge zwischen Attributen und Aktivitäten ermöglichen soll.

2.1. Kategorisierung von Ressourcen

In der Literatur lassen sich unterschiedliche Kategorisierungen von Ressourcen finden. Ressourcen können entsprechend ihrer *Funktionen* bzw. ihrer *Position* in einer Organisation eingeteilt werden [2]. Außerdem können Ressourcen in *menschliche* und *nicht-menschliche* Ressourcen eingeteilt werden [3], [4]. *Nicht-menschliche* Ressourcen können beständige oder verbrauchbare Ressourcen sein. Eine weitere Kategorisierung von Ressourcen erfolgt durch die *Zugriffsart* auf die Ressourcen [4]. Dabei wird in *geteilte* (z.B. ein Netzwerkdrucker) und *private* Ressourcen (z.B. der PC eines Mitarbeiters) unterschieden. Bei *geteilten* Ressourcen kann eine weitere Unterteilung entsprechend der Art des Zugriffes erfolgen [4]. Geteilte Ressourcen können entweder einen *simultanen* Zugriff (z.B. ein Netzwerkdrucker) oder *nicht simultanen* Zugriff (z.B. die Ausfüllung eines Schecks) zulassen. Der Ressourcenzugriff auf einen Prozess (eine Aktivität) wird in *Resource Workflow Patterns* beschrieben [3]. In unserem Ansatz werden die Eigenschaften bzw. Fähigkeiten, welche Ressourcen bzgl. einer Aktivität besitzen, visualisiert, die in dem Resource Workflow Pattern „Capability-based Allocation“ beschrieben sind. Dieses Resource Workflow Pattern beschreibt den Zugriff bzw. die Zuordnung von Ressourcen entsprechend ihrer Fähigkeiten. Die Belastung einer Ressource gilt nur für langlebige Ressourcen, da verbrauchbare Ressourcen bei der Durchführung einer Aktivität ausgeschöpft werden und somit keine Belastung für die Ressource entstehen kann. Dieser Beitrag beschränkt sich auf menschliche bzw. langlebige, nicht-menschliche, geteilte Ressourcen, die belastet werden.

Fähigkeiten von Ressourcen
Ressourcen werden zur Ausführung von Aktivitäten eines Prozesses benötigt [5]. Beziehungen zwischen Ressourcen und ihre Zugehörigkeit zu Ressourcenklassen[1] werden in einem Organisationsmodell abgebildet. Die Zuordnung der Ressource zu einer Tätigkeit in einem Prozessmodell erfolgt anhand ihrer Fähigkeiten (z.B. Sending vs. Request). Abbildung 1 zeigt die Einteilung von Ressourcen entsprechend ihrer Fähigkeiten und die Zusammenfassung von Ressourcenklassen gleicher Fähigkeiten. Ressourcen mit der Fähigkeit „Senden" werden in der Klasse „Sending Resource" zusammengefasst. Ressourcen, über die eine Anfrage gestellt werden kann, werden in der Klasse „Requesting Resource" zusammengefasst. Weiterhin gibt es Ressourcen, welche keiner Ressourcenklasse zugeordnet werden können (z.B. Printer und Cheque).

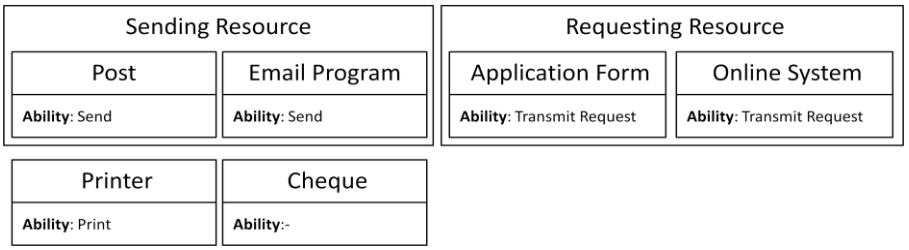


Abb. 1.: Kategorisierung von Ressourcen anhand ihrer Fähigkeiten

Die Ausführung einer Aktivität durch eine Ressource setzt eine bestimmte Eigenschaft bzw. Fähigkeit einer Ressource bzgl. der Aktivität voraus. So können einzelne Ressourcen unterschiedliche Zeiten zur Ausführung einer Aktivität benötigen oder es fallen unterschiedliche Kosten für die Ausführung der Aktivität an. Die Eigenschaften einer Ressource bezüglich der Aktivitätsausführung werden als Attribute bezeichnet. Eine Ressource kann die folgenden Attribute besitzen: (1) die Auslastung bzw. die zeitliche Verfügbarkeit einer Ressource, (2) Zugriffshäufigkeit auf eine Ressource, (3) Kosten, die für die Ausführung einer Aktivität durch die Ressource anfallen, (4) Fähigkeit bzw. die Eignung einer Ressource, eine entsprechende Aktivität durchzuführen und (5) die Arbeitsgeschwindigkeit, mit der eine Ressource eine Aufgabe erledigt.
Die Ressource kann durch ein mehrflächiges dreidimensionales Objekt dargestellt werden. Dabei entspricht jede Fläche einem Attribut. Dadurch ist es möglich, beliebig viele Attribute darzustellen, welche dann durch Drehung des Objektes zur Ansicht gebracht werden. Dieser Beitrag beschreibt beispielhaft die Darstellung der drei Attribute *Verfügbarkeit*, *Zugriff* und *Kosten*. In einer 3D-Ansicht sind Attribute miteinander kombinierbar und gleichzeitig darstellbar. So können bei der Analyse von Kosten sowohl geteilte als auch private Ressourcen betrachtet werden. Zur Ausnutzung der dreidimensionalen Sicht werden in diesem Beitrag nur diejenigen Arten von Ressourcen betrachtet, die alle drei Attribute nutzen. Beispielsweise kann bei privaten Ressourcen kein Zugriffskonflikt entstehen, da private Ressourcen nur für eine bestimmte Aktivität zur Verfügung stehen. Bei Ressourcenklassen mit exklusivem Zugriff kann ebenfalls kein Konflikt entstehen, weshalb private Ressourcen und

[1] Bei einer solchen Einteilung bezeichnet man die Ressourcenklasse auch als Rolle [5]. In diesem Beitrag wird zwischen Ressource und Ressourcenklasse unterschieden.

Ressourcen mit exklusivem Zugriff nicht weiter betrachtet werden. Es werden nur Ressourcen mit simultanem Zugriff betrachtet.

2.2. 3D-Darstellung der Attribute

Die dreidimensionale Darstellung der Attribute wird anhand eines Prozessmodells für Zahlungsanforderungen veranschaulicht. Das Prozessbeispiel stammt aus [7] und wurde in [4] um Ressourcen erweitert. In unserem Beitrag haben wir das Beispiel als Petri-Netz modelliert [8]. Damit die Attribute für den Benutzer erfassbar sind, müssen sie mit Hilfe von grafischen Objekten dargestellt werden, welche unterscheidbare Flächen aufweisen. Aus diesem Grund werden Ressourcen als stilisierte Figuren visualisiert, z.B. wird die Ressource *Computer* durch ein Computerbild dargestellt. Andere Ressourcen werden entsprechend mit passenden stilisierten Bildern der Ressourcen visualisiert. In unserem Beitrag verwenden wir zur Darstellung der Person eine stilisierte Figur, welche aus einer Kugel als Kopf und einem mehrseitigen Würfel als Körper besteht (siehe Abb. 2).

Die stilisierte Figur ist durch mehrere Seiten definiert. Dabei entspricht jede Seite einem Attribut. Die Fläche einer Seite gibt das Maximum des entsprechenden Attributwertes an und ist initial weiß dargestellt. Diese zusätzliche Visualisierung der Maxima ermöglicht eine schnelle Erfassung des noch möglichen Wertzuwachses der Attribute (um wie viel können Kosten, Belastung und Konflikte noch anwachsen). Die farbliche Füllung einer Seite spiegelt den momentanen Wert des Attributes wider.

Die Steigerung der Arbeitsgeschwindigkeit ist nicht linear und hängt von der Arbeitsbelastung ab. Bei niedriger und hoher Belastung wird die Arbeitsgeschwindigkeit schlechter [9]. Die Arbeitsgeschwindigkeit wiederum ist ausschlaggebend wie lange eine Aktivität von einer Ressource bearbeitet wird und welche Kosten dadurch für eine Aktivität anfallen. Der Anwender hat die Möglichkeit, die ihm am sinnvollsten erscheinenden Arbeitsgeschwindigkeitsformel abhängig von der Belastung der Ressource anzugeben. Denkbar ist hier einerseits ein lineares Geschwindigkeit-Belastungsverhältnis, d.h. mit steigender Belastung steigt auch die Geschwindigkeit. Andererseits ist eine Kurve entsprechend dem Yerkes-Dodson Gesetz[2] [10] denkbar. Nach dieser Formel kann der Belastungsverlauf, und die daraus resultierende Überlast errechnet werden.

Zur weiteren visuellen Unterstützung werden die Farben entsprechend [11] eingesetzt. Durch den Einsatz von Farben ist der Status einer Ressource schneller ersichtlich. Die Farbe Grün wird verwendet, um anzuzeigen, dass der Attributwert keine kritischen Grenzen überschritten hat. Gelb, um anzuzeigen, dass eine kritische Grenze überschritten wurde und Rot, um anzuzeigen, dass durch den Wert ein Problem verursacht wird. Für Betrachter mit Rot/Grün-Schwäche ist der Einsatz von anderen Farben denkbar.

Eine Ressource dient im Normalfall nicht nur dazu, eine einzelne Aktivität auszuführen [12], sondern wird mehreren Aktivitäten zugeordnet, bzw. mehrere

[2] Das Yerkes-Dodson Gesetz besagt, dass bei zunehmender Stimulation auch die Leistung zunimmt. Dies gilt jedoch nur bis zu einem gewissen Punkt. Ab diesem Punkt wird die Stimulation als Überforderung wahrgenommen und die Leistung sinkt.

Aktivitäten sind der Ressource zugeordnet. Die Ressource wird bei jeder Aktivität, die sie benötigt, dargestellt.
Abbildung 2 zeigt einen Auszug des Zahlungsanforderungs-Prozesses. Dieser Prozess beschreibt die Anordnung einer Zahlung, welche durch die Ressource *Department director* und der Benutzung der Ressource *Application Form* ausgeführt wird. In der Abbildung wird die Ressource *Department director* gezeigt[3]. An der Fülllinie der Vorderseite ist erkennbar, dass die Ressource durch diese Aktivität bisher nicht stark ausgelastet ist. Jedoch übersteigen die (Ausführungs-)Kosten, die für die Bearbeitung der Aktivität durch Ressourcen anfallen, einen kritischen Wert. Die finanzielle Überlastung ist an dem Füllbereich durch die Farbe Rot und durch die Kostenfülllinie gekennzeichnet. Durchs das Überfahren (hoovern) eines Füllbereiches mit dem Mauszeiger wird textuell angezeigt, welches Attribut der Füllbereich darstellt.

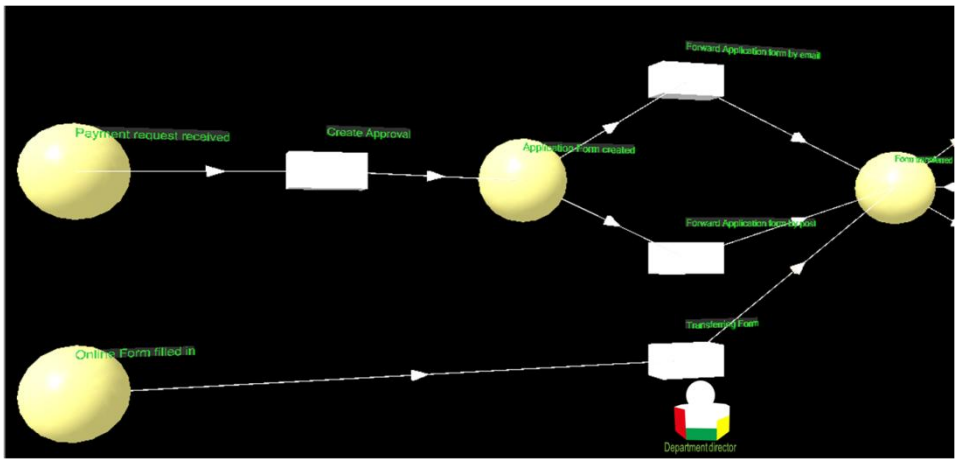


Abb. 2.: 3D-Darstellung von Attributen

3 Analyse von Ressourcenattributen

Die Darstellung der Attribute der Ressourcen/Aktivitäten-Beziehung eignet sich, um schnell kritische Stellen zu erkennen. Der Prozess kann anhand von Simulationen analysiert werden. Vor der Simulation müssen mehrere Daten manuell eingegeben werden. Es müssen die anfallenden Kosten eingetragen werden, welche bei der Ausführung einer Aktivität durch eine Ressource entstehen. Eine Annotation ist auch für die Belastung notwendig, also die Zeit, während der eine Aktivität eine Ressource benötigt. Hier muss, wie in Abschnitt 2.2. erwähnt, die Arbeitsgeschwindigkeitsformel vom Benutzer eingetragen werden. Um den kritischen Wert eines Attributs zu ermitteln, müssen entsprechende Bereichsgrenzen angegeben werden. Während der Simulation des Prozesses werden Aktivitäten mit Hilfe von Ressourcen ausgeführt. Dadurch findet eine

[3] Die Ressource *Application Form* ist als Verbrauchsgut zu sehen, da das Formular für eine bestimmte Zahlungszulassung nur einmal benutzt wird.

Belastung der Ressourcen statt. Weiterhin verursacht die Verwendung von Ressourcen Kosten. Die vorher eingegebenen Werte der Kosten und Belastung werden jeweils additiv erfasst und es werden die Zugriffskonflikte während der Simulation protokolliert. Bei Ausführung einer Aktivität werden dann jeweils die Werte der Attribute der zugehörigen Ressource aktualisiert.

Zur Darstellung der Simulationsergebnisse wird ein Würfel vorgeschlagen. Jede Seite des Würfels stellt dabei ein anderes Ressourcenattribut dar. Es sind immer drei Attribute (Würfelseiten) sichtbar. Auf der Y-Achse werden Ressourcen und auf der X-Achse Aktivitäten abgebildet. Wurde während der Simulation eine Ressource von einer Aktivität genutzt, wird auf dem Schnittpunkt zwischen der X- und Y-Achse der entsprechende Wert des Ressourcenattributes ein Objekt dargestellt. Um eine gute Übersichtlichkeit zu gewährleisten, besteht die Möglichkeit einzelne Ressourcen und dazugehörige Aktivitäten auszuwählen (dabei werden die übrigen Elemente kleiner und in grauer Farbe dargestellt). Die Simulationsergebnisse können genutzt werden, um auftretende kritische Werte zu erkennen und Gegenmaßnahmen zu ergreifen. Einige mögliche Gegenmaßnahmen werden im Folgenden kurz skizziert.

Zur Reduzierung einer Überlast kann eine andere Ressource, welche im Organisationsmodell der gleichen Ressourcenklasse (s. Abschnitt 2.1.) zugeordnet ist, genutzt werden. Die überlastete Ressource wird durch eine Ressource mit gleichen Fähigkeiten unterstützt[4]. Hierbei sollte die kostengünstigste Ressource verwendet werden. Eine Überlast kann auch reduziert werden, indem eine Aktivität in mehrere (Unter-)Aktivitäten aufgeteilt wird und dann unterschiedlichen Ressourcen zugeordnet wird. Der simultane Zugriff unterschiedlicher Aktivitäten auf eine Ressource, die keinen simultanen Zugriff auf sich erlaubt, kann kritisch sein und sollte durch das Hinzufügen einer weiteren Ressource gleichen Typs bzw. einer anderen Ressource der gleichen Klasse vermieden werden (siehe Abschnitt 2.1.). Ein Konflikt kann ebenso durch eine Überlastung der Ressource (erzwungener simultaner Zugriff) entstehen. Dieser kann dann durch die Maßnahmen, welche zur Behebung der Überlast durchgeführt werden, beseitigt werden. Die Auflösung einer Überlast und eines Ressourcenzugriffskonflikt ist durch den Einsatz einer größeren Anzahl von Ressourcen möglich. Beim Einsatz einer größeren Anzahl von Ressourcen können die Aktivitäten öfters ausgeführt werden. Hierdurch könnte zwar die Überlast und der Ressourcenzugriffskonflikt beseitigt werden, allerdings können die Kosten, welche für eine einzelne Aktivität anfallen dürfen, überschritten werden. Somit wird durch den höheren Einsatz von Ressourcen ein Zugriffs- bzw. Überlastproblem auf ein Kostenproblem verlagert. Beim Auftritt eines Kostenproblems müssen die Kosten gesenkt werden.

4 Zusammenfassung und Ausblick

In diesem Beitrag wurde eine Verknüpfung zwischen der Ablauf- und Organisationsmodellierung diskutiert. Zusätzlich wurde eine kompakte Darstellung von Ressourcenattributen vorgeschlagen. Diese kompakte Darstellung soll eine effiziente

[4] Diese Lösung wäre auch schon zur Laufzeit der Simulation denkbar. Dies ist abhängig von dem verwendeten Ressource Pattern [3].

Analyse von Ressourcen unterstützen. Der in diesem Beitrag vorgestellte Analyseansatz für Ressourcenattribute ist erweiterbar um weitere Attribute, wie z.B. die Fähigkeit einer Ressource, eine entsprechende Aktivität durchzuführen oder ihre Arbeitsgeschwindigkeit.

Literaturverzeichnis

1. Oberweis, A.: Modellierung und Ausführung von Workflows mit Petri- Netzen. Teubner Verlag, Wiesbaden (1996)
2. Aalst, W. M. P., Hee, K.V.: Workflow Management: Models, Methods, and Systems. The MIT Press, Cambridge, MA (2002)
3. Russell, N., Hofstede, A.H.M ter, Edmond, D., Aalst, W. M.P van der.: Workflow Resource Patterns. BETA Working Paper Series, WP 127, Eindhoven University of Technology, Eindhoven (2004)
4. Li, H.; Yang, Y.; Chen, T. Y.: Resource constraints analysis of workflow specifications, In: The Journal of Systems and Software, Volume 73, S. 271-285. Elsevier, (2004)
5. Aalst, W.M.P. Van der: The Application of Petri Nets to Workflow Management. The Journal of Circuits, Systems and Computers, 8(1), S. 21-66 (1998)
6. Schönhage, B., Ballegooij, A. van, Eliens, A.: 3D Gadgets for Business Process Visualization – a case study-, In: Proceedings of the Fifth Symposium on Virtual Reality Modeling Language , S. 131- 138, Monterey, California, United States (2000).
7. Sadiq, W., Orlowska, M. E.: Analysing Process models using graph reduction techniques. Information Systems, Volume 25; S. 117-134 (2000)
8. Reisig, W.: Petrinetze - Eine Einführung, 2. Auflage. Springer (1986)
9. Nakatumba, J., Aalst, W.M.P. Van der: Analyzing Resource Behaviour Using Process Mining. In: Proceedings of the 7th International Conference of BPM, S.8-19. Ulm, Germany (2009)
10. Yerkes, R.M., Dodson, J.D.: The relation of strength of stimulus to rapidity of habit-formation. Journal Comparative Neurology and Psychology, 18, S. 459-482 (1908)
11. Eichhorn, D., Koschmider, A., Li, Y., Oberweis, A., Stürzel, P., Trunko, R.: 3D Support for Business Process Simulation. In: Proceedings of the *33rd Annual IEEE International Computer Software and Applications Conference*, S. 73-80, Seattle, Washington, USA (2009)
12. Aalst, W.M.P. Van der: Business Process Simulation: How to get it right. In: Proceedings of the 6th International Conference of BPM, Milan, Italy (2008)
13. Betz, S., Eichhorn, D., Hickl, S., Klink, S., Koschmider, A., Li, Y., Oberweis, A., Trunko, R.: 3D Representation of Business Process Models. In Loos, Peter; Nüttgens, Markus; Turowski, Klaus; Werth, Dirk, Proceedings of Modellierung betrieblicher Informationssysteme, LNI, Köllen Verlag (2008)

Structural and Behavioural Commonalities of Process Variants

Matthias Weidlich and Mathias Weske

Hasso-Plattner-Institute, University of Potsdam, Germany
{matthias.weidlich,weske}@hpi.uni-potsdam.de

Abstract. A common business process might exist in multiple variations in an enterprise, due to different legal requirements in different countries, deviations in the supporting IT infrastructure, or differences in the organisational structure. In order to explore and control such variability, we argue that the notion of a core process, the invariant nucleus of all process variants, might be applied. In this paper, we discuss the spectrum of structural and behavioural aspects that might be leveraged to define such a process.

1 Introduction

In large enterprises, it can be observed that a common business processes exists in many variations across different parts of the organisation. These variations may stem from different legal requirements in different countries, deviations in the supporting IT infrastructure, or differences in the organisational structure. Consequently, the existence of such *process variants* is often inevitable and needed to meet the concerns of a certain organisational unit. Still, excessive generation of variations of a common business process is a serious obstacle for enterprise-wide process initiatives, such as the roll-out of new IT infrastructure or an organisational restructuring. Thus, effective management of process variations is crucial for keeping the deviations within reasonable limits.

The creation of process variants might be controlled in the way that a base process is adapted following on well-defined change patterns (cf., [1, 2]) or configurations (cf., [3]). However, enforcement of such a controlled variability of processes is hard to achieve once process variants are created independently in different countries or organisational units. Addressing such scenarios of decoupled process variants, we focus on the use case of *exploration of process variability*. Given a set of independently created process variants, such an analysis assumes the knowledge about corresponding activities of these variants and tries to answer the following questions: Are there outliers, i.e., process variants that deviate significantly from the other variants? Are there redundant process variants, i.e., very similar variants that might be merged? Obviously, both questions are relative to a notion of commonality of process variants. We refer to this commonality as the *core process*, the invariant nucleus of all variants. Given the core process, the deviation between it and a dedicated variant is quantified. Based thereon, conclusions can be drawn regarding outliers and redundant process variants.

The notion of a core process is not new, but has been advocated under the term *greatest common divisor* (GCD) of a set of process models for solving other *'notorious problems'* [4]. For instance, the adaptation of a standard software solutions to the needs of an enterprise might be controlled by the GCD of the organisational processes and the processes implemented by an IT system. This use case imposes rather strict requirements, such that the proposed notion of a GCD is based on behaviour inheritance [4]. While this seems to be a reasonable choice for the aforementioned use case, process variants that are rather loosely coupled often do not have to comply to such a strict relation. Therefore, we aim at a framework that allows for flexibility in the definition of a core process. Starting with a very weak notion of a core process, the notion can be strictened in a step-wise approach in the course of variability exploration.

In this paper, we take the work on similarity of process models as a starting point. Recently, various structural and behavioural abstractions of process models have been proposed in order to search for process models in repositories [5, 6] or quantify process model consistency [7]. We show that the spectrum of these abstractions allows for a multitude of definitions of a core process. That allows for a fine-granular analysis of the degree of variability in a set of process variants.

The remainder of this paper is structured as follows. Section 2 introduces the spectrum of structural and behavioural abstractions that might be leveraged for the definition of a core process. Subsequently, Section 3 illustrates their application for quantifying variability. Section 4 reviews related work, while Section 5 concludes the paper.

2 Characterisation of a Core Process

In this section, we approach the question of how a core process can be characterised in terms of structure and behaviour given a set of process variants. Apparently, process variants might also differ with respect to the labelling of the process elements and the assumed level of abstraction. That is, corresponding activities have to be identified and refinements of activities have to be treated accordingly. However, we abstract from these aspects in the remainder of this paper and focus on the structural and behavioural dimension. Note that we also restrict our discussion to rather generic process models consisting of activities and means for routing the control flow. Therefore, we focus on the commonalities of most process modelling languages. For illustration purposes, we use the Business Process Modeling Notation (BPMN).

First, Section 2.1 discusses the structural aspects of any characterisation of a core process. Second, we elaborate on behavioural aspects in Section 2.2.

2.1 The Structural Dimension

As activities are the basic functional blocks of a process model, we argue that any characterisation of a core process should be guided by the activities. Thus, activities are selected for containment in the core process based on a certain

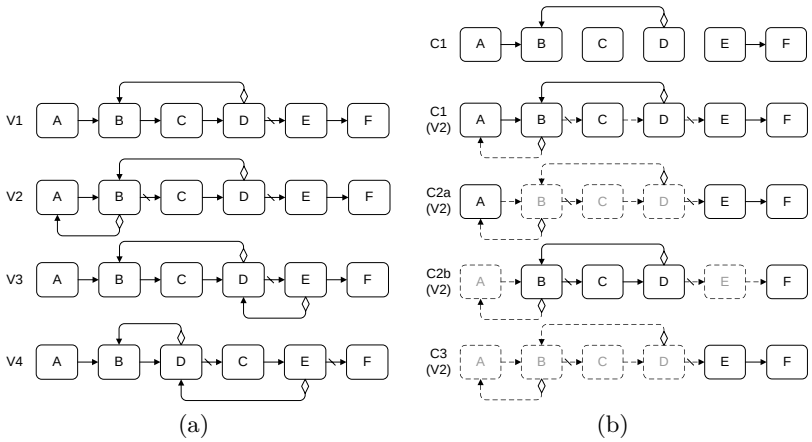


Fig. 1. (a) set of process variants, (b) different structural characterisations of a core process for the variants of (a) along with their projection on variant V2

criterion. Still, paths can be considered in the core process definition, such that all direct paths between selected activities (i.e., there is no other selected activity in between) shared by all process variants should also appear in the core process.

In order to select activities for the core process definition, we might leverage the following structural aspects.

Activity Occurrence. In the most simple case, activities of the core process are those that are contained in every process variant. However, one might also think of a relaxed interpretation that requires activities of the core process to appear in a certain share of the process variants. For instance, activities contained in at least 70% of the process variants qualify for being part of the core process.

Path Existence. If a first set of activities has been selected, the existence of paths between these activities can be used an additional filter for the selection. The rationale behind is that activities should occur in the core process, only if their dependencies in terms of the presence or absence of a path between them is consistent throughout the set of process variants. Again, the criterion might be relaxed in the sense that the path dependency is required to hold solely in the majority or a certain share of the process variants.

Subgraph Isomorphism. Selecting activities for a core process when considering the existence of paths between them might still neglect structural differences that appear throughout the process variants. Therefore, not only the presence or absence of paths, but a strict structural criterion in the sense of a graph isomorphism might be applied. Thus, we select the activities that are part of a subgraph for which there is an isomorphism across all process variants.

We illustrate these structural aspects along with their implications for the definition of a core process using the four process variants depicted in Fig. 1(a). Note that all variants share the same set of activities, but show different execution dependencies. Besides normal sequence flow, the models contain conditional flow (arc with a small diamond) that is activated based on a condition, and default

flow (arc with a slash marker) that is activated if none of the other outgoing flows can be taken. A first structural characterisation of a core process might be based on the activities and paths that all variants have in common. The model C1 in Fig. 1(b) illustrates this case. Note that the resulting model is not a complete process model, but rather a set of structural aspects that is invariant across all process variants. In order to illustrate the deviation of a certain process variant, we also depicted the projection of the core process C1 on the process variant V2 in Fig. 1(b) (dashed lines highlight parts that are not contained in the core process). Applying the aforementioned criterion of path existence results in a different characterisation of a core process. In particular, there are two core process definitions, C2a and C2b, both illustrated as projections on process variant V2 in Fig. 1(b). Both clusters of activities, {A, E, F} and {B, C, D, F}, satisfy the requirement of a consistent path existence or absence between the respective activities, while there is no subsumption relation between both sets. Note that multiple definitions of a core process (or GCD, respectively) have also been observed in [4]. Finally, core process C3 illustrates the case that a graph isomorphism is required to hold between the respective activities in all process variants. Obviously, trivial core process definitions containing just one activity would also satisfy this requirement.

Note that the structural aspects that might characterise a core process differ significantly regarding their computational complexity. While the pure occurrence of an activity can be checked in a straight-forward manner, the path existence criterion implies additional overhead. Here, computation of the transitive closure of the flow relation requires already low polynomial time. Moreover, the subgraph isomorphism problem is known to be NP-complete.

2.2 The Behavioural Dimension

Activities that are part of the core process may be required to show the same behavioural dependencies in all process variants. As in case of the structural dimension, there are various interpretations of the notion of *equal behavioural dependencies*. In the following paragraphs, we discuss four interpretations, i.e., behavioural aspects that can be utilised to characterise a core process.

Optionality of Execution. Evidently, the fact whether a certain activity is optional or mandatory for the completion of the process can be considered. That is, solely activities that are mandatory for completion in all process variants qualify to be part of the core process. As discussed for structural, this criterion might be relaxed, such that activities of a core process have to be mandatory in the majority (or a certain share) of the process variants.

Causal Footprints. In order to take constraints regarding the order of execution into account, causal footprints can be used as a behavioural abstraction. They have been introduced for verification of processes [8] and later been applied as a similarity measure [6]. The basic idea of causal footprints is to capture the behaviour of a process by means of two relations between activities. So called *look-back links* associate a set of activities to an activity, such that any execution of the latter activity is preceded by at least one of the sources of the link. Similarly,

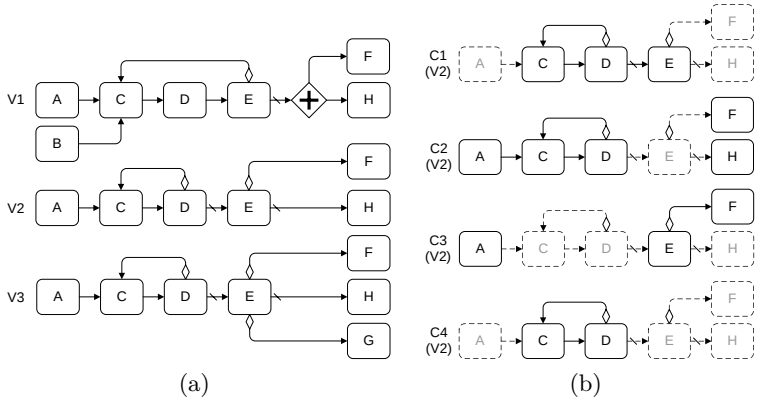


Fig. 2. (a) set of process variants, (b) different behavioural characterisations of a core process for the variants of (a) along with their projection on variant V2

look-ahead links relate an activity to a set of activities, such that an execution of the former activity is followed by at least one of the targets of the link. Starting with neighbouring activities, the causal closure of these links can be computed. In our setting, we can leverage causal footprints as follows. In order to qualify for containment in the core process, any set of activities has to show a consistent causal footprint throughout all process variants.

(Causal) Behavioural Profiles. A similar behavioural abstraction are behavioural profiles that have been introduced to quantify the consistency of model correspondences [7]. Such a profile comprises relations between pairs of activities based on the observable behaviour. That is, two activities are considered to be exclusive, in strict order, or in observation concurrency to each other. Together with reversed strict order, these relations partition the Cartesian product of activities. While the behavioural profile focussed on the *order of potential execution*, the *causal* behavioural profile takes optionality of activity execution and, therefore, also causality between activities into account. That is, an asymmetric co-occurrence relation is defined over pairs of activities An activity is co-occurring with another activity, if any complete trace of the process containing the first activity contains the second one as well. Both behavioural abstractions (causal and non-causal behavioural profile) can be applied as discussed for causal footprints, i.e., activities in the core process must be consistent w.r.t. to the behavioural abstraction throughout all process variants.

Behaviour Equivalence. The multitude of behavioural equivalences in the linear-time – branching-time spectrum [9] can also be leveraged. That is, activities of the core process are required to be, for instance, trace equivalent throughout the set of process variants. To this end, activities that are contained in some process variants only, might either be blocked or hidden yielding the well-known notions of behavioural inheritance for trace semantics, i.e., protocol inheritance (blocked) or projection inheritance (hidden) [10, 11].

Again, we illustrate the different behavioural characterisations of a core process using exemplary process variants as depicted in Fig. 2(a). First and

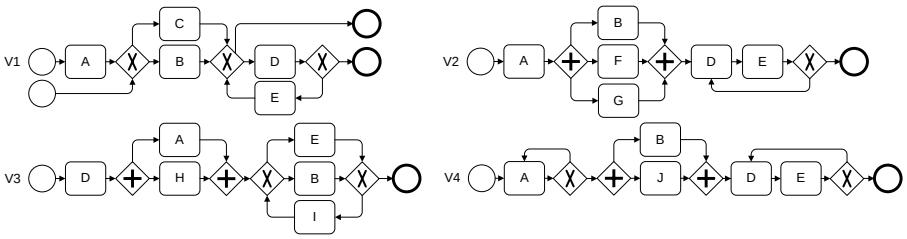


Fig. 3. Example process variants used for analysis

foremost, only common activities that are mandatory for the completion in all process variants are selected for the core process C1, which is depicted as the projection on process variant V2 in Fig. 2(b). Second, we selected the activities of the core process C2, such that their causal footprints are invariant across all process variants. Here, C2 reflects only one definition of the core process, as the set {A, E, F, H} of activities would also satisfy the respective property. For the case of behavioural profiles as the applied behavioural abstraction, there are multiple core processes as well, one depicted as model C3. Finally, C4 shows the only core process that is derived when trace equivalence is applied in the sense of projection inheritance. Protocol inheritance, in turn, would result in a core process that contains the activities {A, C, D}.

Regarding the computational complexity implied by the discussed behavioural aspects, general conclusions can hardly be drawn as it highly depends on the model characteristics. For instance, the optionality of activity execution might be determined for a block-structured model with a dedicated process start and process end quite efficiently by traversing the containment hierarchy of blocks and checking their respective type. However, we are not aware of any efficient approach to check optionality of execution for an arbitrarily structured model. Causal footprints, in turn, can be enriched step-wise. One causal footprint (such a footprint is not unique) can be derived efficiently using the algorithm proposed in [8]. Behavioural profiles might be derived in low polynomial time for sound free-choice models. The same holds true for the causal behavioural profiles as long as certain requirements are met for unstructured regions of the process model, cf., [7]. Behaviour equivalences are known to be exponentially hard in the general case.

3 Application of a Core Process for Analysis

We illustrate how the different characterisations of a core process can be applied for analysis by means of the example process variants depicted in Fig. 3. First, we apply a structural core process definition that is based on the occurrence of activities and the path existence criterion that has to be consistent for 70% of the process variants. That yields a core process definition that comprises four activities (A, B, D, E) and path relations from A to {B, C, E}, B to {C, E}, C to E, and E to C. Based thereon, we quantify the deviation from the core process for each process variant.

- The share of activities that are part of core process: $\frac{4}{5} = 0.8$ for V1, $\frac{2}{3} \approx 0.67$ for V2 and V3, and 0.8 for V4.
- The share of path relations for these activities (neglecting paths between an activity and itself) that is consistent with the core process: $\frac{12}{12} = 1$ for V1, V2, and V4, and $\frac{6}{12} = 0.5$ for V3.

Although not observable when looking solely at the overlap of activities, process variant V3 clearly shows a strong structural deviation when the path relation is also considered. Obviously, the question whether such an 'outlier' should be allowed for can only be judged by taking the business semantics into account. Assuming that the variants show a process implementation in four different countries, such a strong deviation might be explained, for instance, by the used technical infrastructures, such that V1, V2, and V4 are implemented on similar systems, whereas variant V3 is not. Still, identification of such structural deviation can trigger according actions for the respective variants.

Focussing on the three structurally similar variants V1, V2, and V4, we refine the analysis with a behavioural characterisation of the core process. That is, the core process is characterised by the four activities shared by these variants along with the relations of the (causal) behavioural profile for the Cartesian product of activities. To this end, the core process contains the relation that most process variants have in common. Behavioural deviation is quantified as follows.

- The relations of the behavioural profile that are consistent with the core process: $\frac{16}{16} = 1$ for V1 and V2, and $\frac{15}{16} \approx 0.94$ for V4.
- The relations of the causal behavioural profile that are consistent with the core process: $\frac{21}{32} \approx 0.66$ for V1, $\frac{32}{32} = 1$ for V2, and $\frac{31}{32} \approx 0.97$ for V4.

We see that the constraints w.r.t. the order of potential execution as defined by the behavioural profile are rather consistent throughout the three variants. When the constraints on causality of activity execution as defined by the causal behavioural profile are taken into account, we derive a different result. That is, V1 differs from V2 and V4, which, in turn, show nearly no behavioural deviation. Again, conclusions can only be drawn if the business semantics of these processes are considered. However, the high degree of behavioural similarity between V2 and V4 hints at potentially redundant variants. Thus, we might decide to merge both variants, such that the overall number of process variants is decreased.

4 Related Work

The question of a core process, i.e., the GCD of a set of process models, is closely related to the notion of the *least common multiple* (LCD) of a set of process models. That is, a process model might be derived that subsumes the behaviour of all process variants, cf., [12]. Such a model might be derived by mining process variants [13]. As mentioned above, LCDs can also be applied as reference models that allow for the derivation of process variants via configuration [3].

Further on, the work in [14] illustrates how structural and behavioural similarity of process variants can be leveraged to identify preferred variants, i.e., process variants that are observed frequently in an execution environment.

5 Conclusion

In this paper, we argued that the degree of variability of process variants might be explored using the notion of a core process. Such a process captures structural and behavioural aspects that are invariant across all process variants. We discussed the spectrum of structural and behavioural aspects that can be leveraged for a core process definition and sketched its application.

In future work, we aim to extend the presented ideas towards a framework that allows for a flexible analysis of variability aspects. In addition, the question of identifying process variants in a model repository has to be addressed.

References

1. Reichert, M., Rinderle, S., Kreher, U., Dadam, P.: Adaptive process management with adept2. In: ICDE, IEEE Computer Society (2005) 1113–1114
2. Weber, B., Reichert, M., Rinderle-Ma, S.: Change patterns and change support features - enhancing flexibility in process-aware information systems. Data Knowl. Eng. **66**(3) (2008) 438–466
3. van der Aalst, W.M.P., Dumas, M., Gottschalk, F., ter Hofstede, A.H.M., Rosa, M.L., Mendling, J.: Correctness-preserving configuration of business process models. In: FASE. Volume 4961 of LNCS., Springer (2008) 46–61
4. van der Aalst, W.M.P.: Inheritance of business processes: A journey visiting four notorious problems. In: Petri Net Technology for Communication-Based Systems. Volume 2472 of LNCS., Springer (2003) 383–408
5. Dijkman, R.M., Dumas, M., García-Bañuelos, L.: Graph matching algorithms for business process model similarity search. [15] 48–63
6. van Dongen, B.F., Dijkman, R.M., Mendling, J.: Measuring similarity between business process models. In: CAiSE. Volume 5074 of LNCS., Springer (2008) 450–464
7. Weidlich, M., Polyvyanyy, A., Mendling, J., Weske, M.: Efficient computation of causal behavioural profiles using structural decomposition. In: ICATPN. (2010) accepted for publication.
8. van Dongen, B.F., van der Aalst, W.M.P., Verbeek, H.M.W.: Verification of epcs: Using reduction rules and petri nets. In: CAiSE. Volume 3520 of LNCS., Springer (2005) 372–386
9. van Glabbeek, R.: The linear time - brancing time spectrum I. The semantics of concrete, sequential processes. In: Handbook of Process Algebra. Elsevier (2001) 3–99
10. Basten, T., Aalst, W.: Inheritance of Behavior. JLAP **47**(2) (2001) 47–145
11. Schrefl, M., Stumptner, M.: Behavior-consistent specialization of object life cycles. ACM Trans. Softw. Eng. Methodol. **11**(1) (2002) 92–148
12. van der Aalst, W.M.P., Basten, T.: Identifying commonalities and differences in object life cycles using behavioral inheritance. In: ICATPN. Volume 2075 of LNCS., Springer (2001) 32–52
13. Li, C., Reichert, M., Wombacher, A.: Discovering reference models by mining process variants using a heuristic approach. [15] 344–362
14. Lu, R., Sadiq, S.W.: On the discovery of preferred work practice through business process variants. In: ER. Volume 4801 of LNCS., Springer (2007) 165–180
15. Dayal, U., Eder, J., Koehler, J., Reijers, H.A., eds.: Business Process Management, 7th International Conference, BPM 2009. Volume 5701 of LNCS., Springer (2009)

Safira: Implementing the Set Algebra for Service Behavior

Kathrin Kaschner

Universität Rostock, Institut für Informatik, 18051 Rostock, Germany
safira@service-technology.org

Abstract. An important property of interacting services is their observable behavior. There exist different formalisms to describe service behavior (e.g. BPEL, BPMN, Petri nets, automata). Based on an extension of automata, in previous work we proposed a compact representation to characterize the behavior of sets of services and introduced a set algebra on it. In this paper, we present with Safira a tool which implements the fundamental set operations for such set of services.

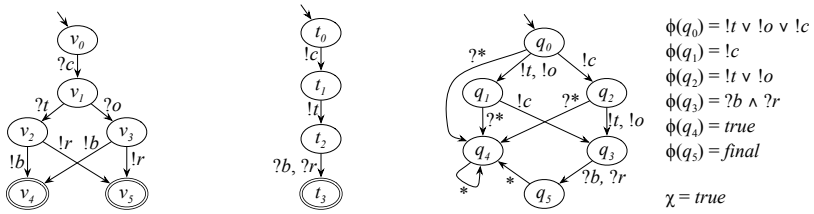
1 Introduction

The observable behavior is an important aspect of interacting *services*. For studying correct interaction, the concept of *operating guidelines* [1] was introduced. An operating guideline is an *annotated automaton* which represents the set of all correctly interacting partners of a given service. In [2] the concept of annotated automata has been extended such that fundamental set operations can be realized for sets of services. These operations have a number of useful applications, including reasoning about substitutability, behavioral constraints, and organizing a service registry [2].

This paper is devoted to our tool Safira which implements the fundamental set operations *complement*, *intersection* and *union* for sets of services. Section 2 introduces the basic formalisms. In Sect. 3 we define the algorithms of these set operations and highlight interesting issues of their implementation in Safira. Section 4 gives an overview of how to obtain and use Safira. Section 5 presents experimental results. Finally, we conclude the paper and give directions to future work in Sect. 6.

2 Background

To model the behavior of a single service, we use *service automata* [1]. A service automaton is a finite state automaton whose edges are labeled with asynchronous message events. As an example, Fig. 1(a) and Fig. 1(b) show two service automata. In the graphical representation, sending events are preceded by "!" and receiving events are preceded by "?". Multiple labels on one edge are a short hand notation for multiple transitions – each one being labeled with one of the events. Initial states have an incoming arc from nowhere. Final states are double-lined. The service automaton V models a simple vending machine expecting a customer to insert coins ($?c$) and to press a button for iced tea ($?t$), or orange juice ($?o$). If enough money has been inserted, the vending machine returns the beverage



(a) V (b) T (c) Q

Fig. 1: The service automata in (a) and (b) model the behavior of a vending machine and a single customer. The extended annotated service automaton in (c) represents a set of customers for the vending machine in (a).

($!b$). If not enough money has been inserted, the customer does not receive the beverage. Instead, the money is returned ($!r$). The service automaton T represents the behavior of a customer who orders iced tea. Note that sending events of T are receiving events of V and vice versa.

For the finite representation of a (possibly infinite) *set* of service automata, we use *extended annotated automata* [2]. They are also finite automata whose edges are labeled with asynchronous message events. To be able to characterize a set of services, we additionally annotate each state with a local Boolean formula ϕ and add a global Boolean formula χ to the automaton. In comparison to service automata, an extended annotated automaton does not have final states. Instead, the proposition *final* in the local formulas may define final states of the represented service automata.

Each service automaton S belonging to the set of service automata characterized by the extended annotated automaton A fulfills the following requirements. First, S is a subautomaton of A (including an initial state). Further, each state q of S satisfies the Boolean formula ϕ of its corresponding state in A and the global formula χ is evaluated to *true*. Thereby, ϕ determines which outgoing edges must be present in state q and χ defines which combinations of states are allowed in S. The interested reader is referred to [2].

Figure 1(c) depicts an extended annotated service automaton Q which characterizes the set of all customers of the vending machine V. An edge labeled with "*" means that there is a transition for each label of Q. "?*" is a placeholder for all receiving events of Q. The customer T in Fig. 1(b) is a subautomaton of Q. The corresponding states are (t_0, q_0), (t_1, q_2), (t_2, q_3) and (t_3, q_4). Furthermore, all states of T satisfy the local formulas of the corresponding states in A. For example, t_0 evaluates the formula $!c \vee ?t \vee ?o$ of q_0 to *true*, because there is a leaving edge labeled with $!c$. Since the global formula χ is equal to *true*, there are no additional constraints to the states of T. Thus, T is characterized by Q. As *all* possible customers of V are represented, Q is also called an *operating guideline* [1] of V.

3 Set Operations on extended annotated automata

In this section, we introduce the algorithms for the fundamental set operations. This means, given two extended annotated service automata representing sets

M_1 and M_2 of services, we show how to compute an extended annotated service automaton that represents the *complement* $\overline{M_1}$, the *intersection* $M_1 \cap M_2$ and the *union* $M_1 \cup M_2$. Since the result of each operation is again an extended annotated automaton, arbitrary nested structures are possible.

Due to the page limit we only sketch the algorithms (see [2] for a detailed description), but focus on the interesting details of their actual implementation in our tool Safira. At the end of each subsection, we also discuss the complexity of the operation.

3.1 Complement

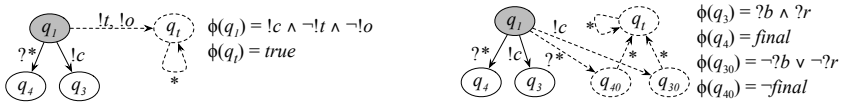
Theory. The most challenging of the three operations is the complement. To compute the complement, we first normalize the extended annotated automaton by applying two transformations. Both transformations modify the shape of the extended annotated automaton, but do not change the represented set of service automata. Having a normalized extended annotated automaton the actual complement operation turns out to be very simple.

The first transformation *totalizes* the extended annotated automaton A. That means, in each state of the totalized A there exists at least one outgoing edge for each label of A. To compute a total extended annotated automaton without changing its semantics, we insert missing edges with label x but explicitly forbid their usage by adding a conjunction with $\neg x$ to the corresponding local Boolean formula. Each inserted edge is connected to a trap state t that contains a self-loop for every label of A.

Figure 2(a) illustrates this procedure for state q_1 of the extended annotated automaton Q in Fig. 1(c). The set of the labels is $\{!t, !o, !c, ?b, ?r\}$. State q_1 has only three outgoing edges labeled with $!c$, $?b$ and $?r$. Therefore, we insert two new edges labeled with $!t$ and $!o$ connecting q_1 with the trap state q_t and set the Boolean formula of state q_1 to $!c \wedge \neg !t \wedge \neg !o$.

In the second transformation, we *complete* the extended annotated automaton such that for each state q and all labels y the disjunction of the formulas of all states q' with $q \xrightarrow{y} q'$ is equivalent to *true*. In Fig. 2(b), we illustrate the completion of state q_1 of the annotated automaton Q in Fig. 1(c). State q_1 has only one $!c$-labeled edge leading to state q_3. Since the Boolean formula $\phi(q_3) = ?b \wedge ?r$ is not equivalent to *true* we add an edge labeled with $!c$ to a new state q_{30}. State q_{30} has for every label an outgoing edge to the trap state q_t. The local formula of q_{30} is set to $\neg(?b \wedge ?r)$. Now, the disjunction of the local formulas of all $?c$-successors is equivalent to *true*. The remaining successor states of state q_1 are treated likewise. To avoid a change in the semantics of Q by inserting the new states q_{30} and q_{40} we explicitly forbid their usage by setting the global formula to $\chi = true \wedge \neg q_{30} \wedge \neg q_{40}$. That means, as soon as a service automaton S covers one of these new states, the global formula is evaluated to *false*. Consequently, S is correctly classified as a non represented service automaton.

After applying the two transformations to an extended annotated automaton A, every service automaton with the same interface as A is a subautomaton of A. Only the global formula χ decides wether or not the service is represented by A.



(a) making total (b) making complete

Fig. 2: In (a), the procedure of making the automaton total is shown for state q_1 of Q (from Fig. 1(c)). (b) illustrates the procedure of completion for the same state. The dashed parts are those that are inserted in the course of a transformation.

Consequently, the complement operation for a total and complete automaton turns out to be very simple: only the global formula χ has to be negated.

Figure 3(b) depicts the extended annotated automaton $\overline{Q}$ that represents the complement set of service automata represented by Q in Fig. 1(c). Figure 3(a) shows a service automaton T' which is represented by $\overline{Q}$. As both the button for iced tea and orange juice are pressed, it is not a customer of Q.

Implementation. Overall, Safira follows the procedure described above. As both transformations can be executed independently for every node, we store the nodes in a list, which is traversed. The totalization of the automaton can easily be done by checking the outgoing edges for every node. In contrast, the completion is more complicated, as boolean formulas have to be evaluated. Therefore, we integrated the open-source SAT solver Minisat[1] as a library into our tool Safira.

To use Minisat, every question concerning Boolean formulas must be converted into a satisfaction problem. Thus, to proceed with the completion, we negate the disjunction f of the formulas of all y-successors for each node q and ask Minisat, wether the formula $\neg f$ is satisfiable. If the answer is 'yes', then f is not equivalent to *true* and an additional y-edge for q is inserted.

Although, the satisfaction problem is NP-complete, the experimental results in the next section show that the complement can be computed efficiently. As there usually are only a small number of labels and the length of the formulas is rather small, the satisfaction problems we formulate for the complement generation are not challenging Minisat.

Our experimental results in Sect. 5 show that the complement of an extended annotated automaton can be computed. The applications mentioned in the first section of this paper require a further use of the resulting automaton. Thus, Safira also implements an optimization algorithm aiming at reducing the size of the result.

The main idea of this optimization is to merge added states with the same Boolean formula. This can be done, because all successors of each added state lead to the same state – the trap state q_t. To decide if a newly computed state can be merged with another state, we build a special decision tree during the completion operation. Each inner node of the tree contains an assignment over the labels of the automaton and has two outgoing edges labeled with 'yes' and 'no'. Each leaf represents a state, which was already added to the automaton by the completion operation. Figure 4 depicts the decision tree which was built during

[1] See the website of Minisat at `http://minisat.se`.

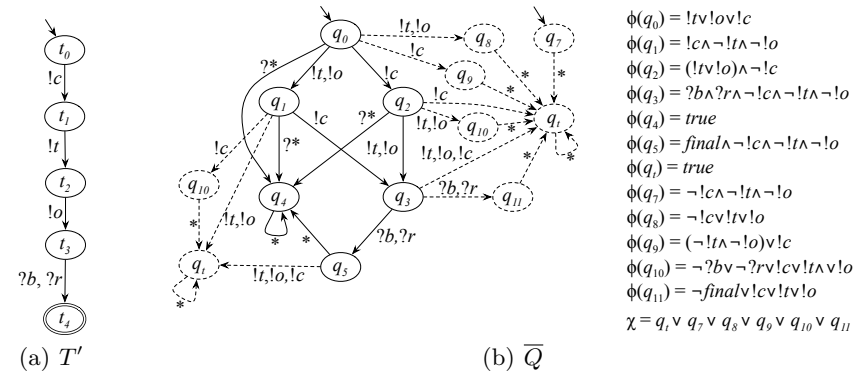


Fig. 3: The extended annotated service automaton $\overline{Q}$ in (b) is the Complement of Q in Fig. 1(c). For reducing the number of edge crossings, we depicted two copies of state q_{10} and the trap state q_t. (a) depicts a service automaton T', represented by $\overline{Q}$.

the complement generation for the vending machine Q. Each leaf represents one of the states $q_7, ..., q_{11}$ which was inserted into the automaton by the completion operation (cf. Figure 3(b)).

Consider a leaf l representing state p with its local Boolean formula g. The decision tree is constructed such that the assignment β of each inner node n along the path leading to l holds the following condition: β is a satisfying assignment of g if the outgoing edge of n leading to l is labeled with 'yes' and is a non-satisfying assignment, otherwise. As an example, see Figure 4. The assignments β_0 and β_1 both do not satisfy the Boolean formula of state q_7.

At the beginning of the completion operation we start with an empty decision tree. When we insert the first new state q_0 to the automaton, we also add q_0 to the decision tree. Thus, at this moment the decision tree contains only a leaf which represents q_0. Then, we proceed as follows: Suppose, we want to connect an existing state of the automaton with a new state q by edge e. Let f be the local Boolean formula of q. To find out if there already exists a state which is annotated with the same Boolean formula, we traverse the decision tree. At each inner node n, we check if the assignment β of n satisfies f. If this is the case, we follow the outgoing edge labeled with 'yes'. Otherwise, we follow the outgoing edge labeled with 'no'. Arriving at a leaf representing a state p, we check if its formula h is equivalent to f. If the answer is 'yes', no new state has to be inserted. Instead, the new edge e is directed to the existing state p. Otherwise, there is an assignment α that satisfies f, but not h or vice versa. In the decision tree, at the place of the leaf representing q, we insert a new inner node, containing α. Its outgoing edges are then directed to two leafs representing q and p, respectively.

In this manner, the decision tree is built successively during the completion operation. Note that for the totalization of the automaton, no additional nodes are necessary. Therefore the decision tree is not needed during this procedure.

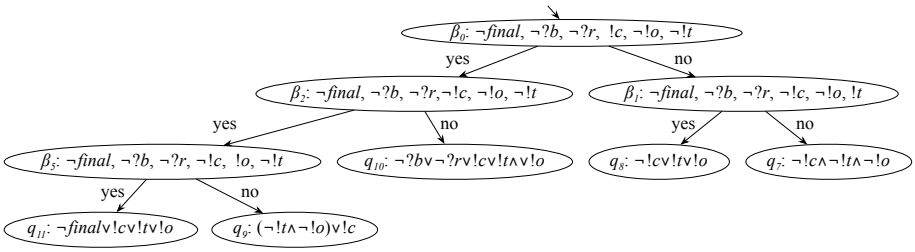


Fig. 4: The decision tree of $\overline{Q}$.

Complexity. Due to our optimization, for each state in the given automaton at most one new state (with the negated formula) is added. Thus, in the worst case the size complexity for completion is linear in the number of the states. The time complexity for the completion depends mainly of the shape of the decision tree. Assuming the decision tree is a balanced tree, the automaton is completed in $\mathcal{O}(n \cdot log\, n \cdot l)$, whereas n is the number of the states and l the number of labels. For the transformation to a total automaton, the worst case complexity for both space and time is $\mathcal{O}(n \cdot l)$.

3.2 Intersection

Theory. The idea of implementing the intersection of two extended annotated automata is to construct the product automaton known from classical automata theory. A product automaton implements the idea that both constituents run synchronized – in every step executing transitions with the same label. The states in the product automaton are annotated with the conjunction of the local Boolean formulas of the constituents [3]. The global formulas are connected by $\wedge$.

Implementation. The algorithm for the intersection is well known from classical automata theory. It was just adapted to the specific characteristics of extended annotated automata.

Complexity. The size of the resulting automaton mainly depends on the degree of similarity of the given automata. The worst case complexity regarding both space and time of the intersection algorithm is in the product of the number of states of the two involved automata.

3.3 Union

Theory. Given the two operations of intersection and complement from the previous subsections, the implementation of union is trivial by using De Morgan's rule: $M \cup N = \overline{\overline{M} \cap \overline{N}}$.

Implementation. We already proofed that the product of two total and complete automata is again total and complete [2]. Thus, we do not need to transform the product automaton of $\overline{M} \cap \overline{N}$ to generate the complement. Instead, we only have to negate the global formula of the automaton representing $\overline{M} \cap \overline{N}$.

Complexity. The complexity of the union results by the complexity of the underlying operations complement and intersection.

4 Obtaining and using Safira

Safira is free software[2] and can be downloaded at `http://service-technology.org/safira`. It is written in C++ and uses the GNU build system to compile the binary. Thus, it is available for a several platforms, including Linux, Microsoft Windows (using cygwin) and Mac OS X.

Safira is a command-line tool implementing the following use cases. We assume that two extended annotated automata are given in files "coffeeVendor.og" and "juiceVendor.og".

- Complement. Call Safira with
  `safira -opdf -t --complement coffeeVendor.og`
- Intersection. Call Safira with
  `safira -opdf -t --intersection coffeeVendor.og juiceVendor.og`
- Union. Call Safira with
  `safira -opdf -t --union coffeeVendor.og juiceVendor.og`

Option `-t` measures the time for computing the result and option `-o` triggers the output of Safira. In the examples above, a PDF file is generated which shows the graphical representation of the resulting extended annotated automaton. For a full description of the command line parameters, type `safira --help`.

5 Experimental results

In this section, we study how the algorithms for computing the complement automaton scale in practice. We do not examine the intersection, because the algorithm is of no issue and the size of the result mainly depends on the degree of similarity of the given automaton.

To generate the input automata for our experiments , we used the tool Wendy[3]. Wendy computes the operating guideline of a given service automaton. The examples are industrial service models and have been extracted from real BPEL processes using the tool BPEL2oWFN[4] (except the first one).

We executed the examples on a computer with a 1.83 GHz Intel Core 2 Duo processor and 2 GB of memory. The results are listed in Tab. 1. As expected, the number of states of the complement automaton computed by the optimization algorithm A_2 (usage of decision tree) is significantly smaller than the number of states of the automaton obtained by the basic algorithm A_1 (no usage of decision tree). The memory usage shows a similar result. There is one case (service "Car Analysis") in which the results concerning the memory usage diverge. This can be explained by the structure of the resulting decision tree. Compared with the decision trees generated for the other automata, this tree is not well balanced. Moreover, A_2 consumes more time than A_1. This can be explained by the additional data structure – the decision tree – which has to be built up and is traversed when a new state of the complement automaton is

[2] GNU Affero Public License Version 3, `http://gnu.org/licenses/agpl.html`.
[3] Available at `http://service-technology.org/wendy`.
[4] Available at `http://service-technology.org/bpel2owfn`.

Table 1: Experimental results showing the number of state of the complement automaton, the execution time and the memory usage for both the optimization algorithm A_2 and the basic algorithm A_1.

service	states of automaton	states of complement		time [sec]		memory [MB]	
		A_1	A_2	A_1	A_2	A_1	A_2
Vending Machine	6	23	12	0.0	0.0	0.5	0.5
Purchase Order	169	887	338	0.1	0.1	1.3	0.8
Car Analysis	733	3,931	987	0.6	6.51	1.9	2.9
SMTP	3,307	19,995	3,460	2.9	29.4	10.9	10.6
Quotation1	7,937	61,571	14,594	14.8	208.0	24.7	15.5
Quotation2	11,265	88,323	22,539	19.2	286.1	132.8	36.1

calculated. The overall time, however, is not a crucial measure because in our application settings the complement automaton is usually calculated at a point in time in which the calculation time is not an issue. The number of states of the complement automaton is more critical. In most settings, the complement automaton is used as an input for complex, nested operations such as union and intersection, for instance.

6 Conclusion and Future Work

In this paper, we presented the tool Safira implementing the set operations complement, intersection, and union. With the help of experimental results, we demonstrated that the algorithms used for the calculation of the complement scale well. The implementation of the set operations is an important step towards their applications: substitutability, behavioral constraints, and organizing a service registry. For their realization, the decision problem membership $S \in M$ for a service S and a set M of services, and also the emptiness check $M = \emptyset$ still need to be implemented. For both, we already introduced the theory [2]. The complexity for the membership problem is linear in the size of S so that we do not expect problems during the implementation. In contrast, we could proof that the emptiness check is NP-complete. Consequently, we have to find heuristics to decide this problem efficiently in practice.

References

1. Lohmann, N., Massuthe, P., Wolf, K.: Operating guidelines for finite-state services. In: ICATPN 2007. LNCS 4546, Springer (2007) 321–341
2. Kaschner, K., Wolf, K.: Set algebra for service behavior: Applications and constructions. In: BPM 2009. LNCS 5701, Springer (2009) 193–210
3. Lohmann, N., Massuthe, P., Wolf, K.: Behavioral constraints for services. In: BPM 2007. LNCS 4714, Springer (2007) 271–287

An efficient approach to detect lack of synchronization in acyclic workflow graphs

Cédric Favre

IBM Research — Zurich,
ced@zurich.ibm.com

Abstract. Control-flow analysis of business process models requires to check the absence of lack of synchronization. We use workflow graphs, which may contain inclusive OR gateways, to represent the control-flow of business process models. We structurally characterize lack of synchronization in an acyclic workflow graph. Based on this characterization, we show how to detect lack of synchronization in quadratic time.

1 Introduction

A business process model can be simulated, used for automated code generation, or directly executed by a workflow engine. With the increasing popularity of these use cases, ensuring that a process is free of control-flow errors is essential. A process is *sound* [1] if and only if it is free of two control-flow errors: the *deadlock* and the *lack of synchronization*. The control-flow of a process can be modeled by a workflow graph or a workflow net [1]. We use workflow graphs with the inclusive OR (IOR) gateways.

In this paper, we show how to efficiently check the absence of lack of synchronization in acyclic workflow graphs with IOR gateways. In Section 3, we show that a lack of synchronization in acyclic workflow graphs with IOR gateways can be fully characterized in term of *a path with a handle*. In Section 4, we show how to detect a path with a handle by using an extension of the approach of Perl and Shiloach [2]. The time and space complexity of this approach is in $O(|N| \cdot |E|)$, where $|N|$ is the number of nodes and $|E|$ the number of edges of the workflow graph.

Related work: In case a workflow graph does not contain IOR gateways, it can be easily converted into a Petri net and vice versa [1]. However, it is not clear if a workflow graph that contains IOR gateways can be converted into a Petri net. Our notion of handles is similar to the one of Esparza and Silva [3] for nets. If we restrict to workflow graphs without IOR gateways, one of the directions of our characterization follows from a result of Esparza and Silva [3]. The converse direction does not directly follow. Our notion of handles has been described by Aalst [4] who shows that, given a Petri net N, the absence of some type of path

with handle in N is a sufficient condition to the existence of an initial marking i of N such that (N, i) is sound. He points out that path with handles can be computed using a maximum flow approach. Various algorithms exist to compute the maximum flow (see [5] for a list). The complexity of these algorithms ranges between $O(|N| \cdot |E|^2)$ and $O(|N| \cdot |E| \cdot log(|N|))$. The existence of a handle can be checked by applying a maximum flow algorithm to each pair of transition and place of the net. Therefore, the complexity of detecting handles with such approach is at best $O(|N|^3 \cdot |E| \cdot log(|N|)$. Moreover, algorithms to determine a maximum flow are significantly more complex to implement than our approach, especially the ones with the lowest complexity.

2 Preliminaries

We use number of known concepts from set theory, graph theory, and the workflow graph verification which are defined below:

Let U be a set. A *multi-set* over U is a mapping $m : U \to \mathbb{N}$. We write $m[e]$ instead of $m(e)$. For two multi-sets m_1, m_2, and each $x \in U$, we have : $(m_1 + m_2)[x] = m_1[x] + m_2[x]$ and $(m_1 - m_2)[x] = m_1[x] - m_2[x]$. By abuse of notation, we sometimes use a set $X \subseteq U$ in a multi-set context by setting $X[x] = 1$ iff $x \in X$ and $X[x] = 0$ otherwise.

A *directed graph* $G = (N, E)$ consists of a set N of *nodes* and a set E of ordered pairs (s, t) of nodes, written $s \to t$. A *directed multi-graph* $G = (N, E, c)$ consists of a set N of nodes, a set E of *edges* and a mapping $c : E \to (N \cup \{\text{null}\}) \times (N \cup \{\text{null}\})$ that maps each edge to an ordered pair of nodes or null values. If c maps $e \in E$ to an ordered pair $(s, t) \in N$, then s is called the *source* of e, t is called the *target* of e, e is an *outgoing* edge of s, and e is an *incoming* edge of t. If $s = \text{null}$, then we say that e *has no source*. If $t = \text{null}$, then we say that e *has no target*. For a node $n \in N$, the set of incoming edges of n is denoted by $\circ n$. The set of outgoing edges of n is denoted $n\circ$. If n has only one incoming edge e, ${}^\circ n$ denotes e ($\circ n$ would denote $\{e\}$). If n has only one outgoing edge e', n° denotes e'.

A *path* $p = \langle x_0, ..., x_n \rangle$ *from* an element x_0 *to* an element x_n *in* a graph $G = (N, E, c)$ is an alternating sequence of elements x_i in N and in E such that, for any element $x_i \in E$ with $c(x_i) = (s_i, t_i)$, if $i \neq 0$ then $s_i = x_{i-1}$ and if $i \neq n$ then $t_i = x_{i+1}$. If x is an element of a path p we say that p *contains* x. A path is *trivial*, when it is contains only one element. A *cycle* is a path $b = \langle x_0 \dots x_n \rangle$ such that $x_0 = x_n$ and b is not trivial. If there exists a path from an element x_1 to an element x_2 of a graph, we say that x_1 *precedes* x_2, denoted $x_1 < x_2$.

A *workflow graph* $W = (N, E, c, l)$ consists of a multi-graph $G = (N, E, c)$ and a mapping $l : N \to \{AND, XOR, IOR\}$ that associates a *logic* with every node $n \in N$, such that: 1. An edge with null as source is a *source* of the workflow graph and an edge with null as target is a *sink* of the workflow graph. 2. The workflow graph has one source and at least one sink. 3. For each node $n \in N$,

there exists a path from the source to one of the sinks that contains n. W is *cyclic* if there exists a cycle in W.

Figure 1 depicts an acyclic workflow graph. A diamond containing a plus symbol represents a node with AND logic, an empty diamond represents a node with XOR logic, and a diamond with a circle inside represents a node with IOR logic. A node with a single incoming edge and multiple outgoing edges is called a *split*. A node with multiple incoming edges and single outgoing edge is called a *join*. For the sake of simplicity, we use workflow graphs composed of only splits and joins. This syntactic restriction does not reduce the expressiveness of workflow graphs. We usually label the source of the workflow graph s and use workflow graphs with a unique sink labeled t.

Let, in the rest of this section, $W = (N, E, c, l)$ be an acyclic workflow graph.

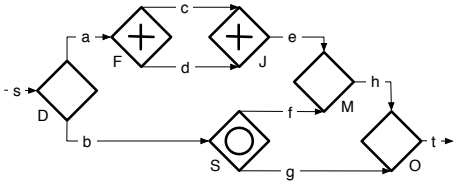


Fig. 1. A workflow graph .

The semantics of workflow graphs is, similarly to Petri nets, defined as a token game. If n has AND logic, executing n removes one token from each of the incoming edges of n and adds one token to each of the outgoing edges of n. If n has XOR logic, executing n removes one token from one of the incoming edges of n and adds one token to one of the outgoing edges of n. If n has IOR logic, n can be executed if and only if at least one of its incoming edges is marked and there is no marked edge that precedes a non marked incoming edge of n. When n executes, it removes one token from each of its marked incoming edge and adds one token to a non-empty subset of its outgoing edges. The outgoing edge or set of outgoing edges to which a token is added when executing a node with XOR or IOR logic is non-deterministic, by which we abstract from data-based or event-based decisions in the process model. In the following, this semantics is defined formally.

A *marking*, $m : E \to \mathbb{N}$, of a workflow graph with edges E is a multi-set over E. When $m[e] = k$, we say that the edge e *is marked with* k *tokens* in m. When $m[e] > 0$, we say that the edge e *is marked*. The *initial marking* m_s of W is such that the source edge s is marked by one token in m_s and no edge, other than s, is marked in m_s.

Let m and m' be two markings of W. A tuple (E_1, n, E_2) is called *transition* if $n \in N$, $E_1 \subseteq {}^\circ n$, and $E_2 \subseteq n^\circ$. A transition (E_1, n, E_2) is *enabled* in a marking m iff for each edge $e \in E_1$ we have $m[e] > 0$ and any of the following proposition:

- $l(n) = AND$, $E_1 = {}^\circ n$, and $E_2 = n^\circ$.
- $l(n) = XOR$, there exists an edge e such that $E_1 = \{e\}$, and there exists an edge e' such that $E_2 = \{e'\}$.

– $l(n) = IOR$, E_1 and E_2 are not empty, $E_1 = \{e \in {\circ}n \mid m(e) > 0\}$, and, for every edge $e \in {\circ}n \setminus E_1$, there exists no edge e', marked in m, such that $e' < e$.

A transition T can be *executed* in a marking m iff T is enabled in m. When T is executed in m, a marking m' results such that $m' = m - E_1 + E_2$.

An *execution sequence* of W is an alternate sequence $\sigma = \langle m_0, T_0, m_1, T_1 ... \rangle$ of markings m_i of W and transitions $T_i = (E_i, n_i, E_i')$ such that, for each $i \geq 0$, T_i is enabled in m_i and m_{i+1} results from the execution of T_i in m_i. An *execution* of W is an execution sequence $\sigma = \langle m_0, ..., m_n \rangle$ of W such that $n > 0$, $m_0 = m_s$ and there is no transition enabled in m_n. As the transition between two markings can be easily deduced, we often omit the transitions when representing an execution or an execution sequence, i.e., we write them as sequence of markings.

Let m be a marking of W, m is *reachable from* a marking m' of W iff there exists an execution sequence $\sigma = \langle m_0, ..., m_n \rangle$ of W such that $m_0 = m'$ and $m = m_n$. The marking m is a *reachable marking* of W iff m is reachable from m_s.

A *lack of synchronization* is a reachable marking m of W such that there exists an edge $e \in E$ that carries more than one token in m.

3 Handles and Lack of Synchronization

To characterize the lack of synchronization, we follow the intuition that potentially concurrent paths, i.e., paths starting with an IOR-split or an AND-split, should not be joined by XOR-join. In the following, we formalize this characterization and show that such structure always leads to lack of synchronization in deadlocks free acyclic workflow graphs.

Definition 1 (Path with a AND-XOR or a IOR-XOR handle). *Let $p_1 = \langle n_0, ..., n_i \rangle$ and $p_2 = \langle n_0', ..., n_j' \rangle$ be two paths in a workflow graph $W = (N, E, c, l)$.*

The paths p_1 and p_2 form a path with a handle[1] *iff p_1 is not trivial, $p_1 \cap p_2 = \{n_0, n_i\}$, $n_0 = n_0'$, and $n_i = n_j'$. We say that p_1 and p_2 form a path with an handle* from n_0 to n_i. *The paths p_1 and p_2 form a* path with a AND-XOR handle *iff they form a path with a handle, n_0 is an AND-split, and n_i is an XOR-join. The paths p_1 and p_2 form a* path with a IOR-XOR handle *iff they form a path with a handle, n_0 is an IOR-split, and n_i is an XOR-join. In the rest of this document, we use handle instead of path with a AND-XOR handle or path with a IOR-XOR handle. The node n_0 is the* start node of the handle *and the node n_i is the* end node of the handle.

[1] Strictly speaking, one path is the handle of the other path and *vice versa*.

Theorem 1. *In an acyclic workflow graph that contains no deadlock, there is a lack of synchronization iff there is a handle.*

The outline of the only if direction of the proof of Theorem 1 is that, whenever there is a handle, this handle can be 'executed' in the sense that there exists an execution such that a token reaches the incoming edge of the start node of the handle and then two tokens can be propagated to reach two incoming edges of the end node of the handle to create a lack of synchronization. We believe that, due to its direct relationship with an erroneous execution, the handle is an adequate error message for the process modeler. Note that it is necessary that the workflow graph is deadlocks free in order to show that the handle can be executed and thus a lack of synchronization be observed. However, even if the workflow graph contains a deadlock, a handle is a design error because, once the deadlock is fixed, the handle can be executed and a lack of synchronization can be observed.

4 Handle Detection

Given an acyclic directed graph $G = (N, E)$ and four different nodes $s_1, s_2, t_1, t_2 \in N$, Perl and Shiloach [2] show how to detect two node disjoint paths from s_1 to t_1 and from s_2 to t_2 in $O(|N| \cdot |E|)$. We extend their algorithm in order to detect two edge disjoint paths between two nodes of an acyclic workflow graph.

Our intuitive view of the approach proposed by Perl and Shiloach [2] is to try to move two tokens from a start node to an end node of a handle without allowing the two tokens to visit the same edge, i.e., without allowing the tokens to follow paths that overlap. To achieve this, we build a so called *state graph* which records the state space of the possible combinations of marked edges and a transition relation. Note that the size of the state graph is quadratic with respect to the number of edges of the original graph because we only consider combinations of two edges. Checking the existence of a handle in the workflow graph reduces to check the existence of a special path in the state graph. We ensure that the special path in the state graph does not allow the two tokens to visit the same edge in two ways: 1. We do not represent the states where two tokens are on the same edge. 2. We restrict the transition relation in a way that, if two tokens visit paths that overlap, they visit the overlapping edge synchronously (which is ruled out by 1). This synchronization is achieved by only allowing the token that is the most upstream in the graph to move.

In the remainder of the this section, we describe how to obtain a *line graph*: a directed graph in which the edges are represented as nodes. Based on the line graph and a numbering allowing us to detect which edge is upstream from another, we show how to build the state graph. Finally, we describe how to detect handles using the state graph.

Computing a line graph: Perl and Shiloach [2] describe how to detect two node disjoint paths in a directed graph whereas we want to detect two edge disjoint paths in a workflow graph; a directed multi-graph. To do so, we transform the workflow graph into its *line graph* [6]. A line graph G' of a graph G represents the adjacency between edges of G. The nodes of G' are the edges of G.

For the purpose of checking handles, there are two types of nodes that are of interest: 1. The start nodes of a possible handle $S = \{x \mid x \in N \wedge\ x$ is an AND-split or an IOR-split$\}$. 2. The end nodes of a possible handle $T = \{x \mid x \in N \wedge$ x is an XOR-join$\}$. We include these nodes in the line graph.

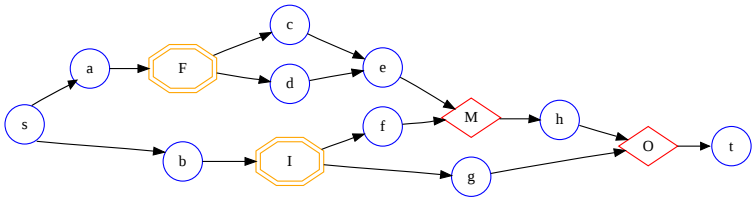


Fig. 2. The line graph of Figure 1.

In the following, for any edge e of a directed graph $t(e)$ references the target node of e and $s(e)$ references the source node of e.

Definition 2 (Line graph). *Let $W = (N, E, c, l)$ be an acyclic workflow graph. The* line graph $L = (N', E')$ *of W is a directed graph such that:*

$$N' = E \cup S \cup T$$
$$E' = \{a \to b \mid a, b \in E \wedge t(a) = s(b)\}$$
$$\cup\ \{a \to b \mid a, b \in N' \wedge (\ (a \in S \cup T \wedge a = s(b)) \vee (b \in S \cup T \wedge t(a) = b)\ )\}$$

Figure 2 shows the line graph of the workflow graph of Figure 1. The orange octagons with double line are nodes in S. The red diamonds are the nodes in T.

Generating the state graph: We build a *state graph* from the line graph of an acyclic workflow graph. Each node of the state graph is a multi-set containing two nodes of the line graph, i.e., two edges of the original workflow graph. Such multi-set represents a *state* of the line graph where the two nodes carry a token. The edges of the state graph represent the allowed token moves. In order to determine which node of the line graph is upstream of another, i.e., to determine the allowed token moves, we compute a value $v(n)$ given by the reverse post order numbering of the nodes [7]. The token on the node with the lowest reverse post order value is allowed to move. Similarly to Yang *et al.* [8], we perform a straightforward extension of the algorithm of Pearl and Shiloach [2]: We add to the state graph the the states where there are two tokens on a node in S or two tokens on a node in T. This allows us to check for two disjoint paths between

one pair of node in S and T instead of two pairs of nodes as originally described by Perl and Shiloach.

Definition 3 (State graph). *A* state graph *of an acyclic simple directed graph* $L = (N', E')$ *is an acyclic directed graph* $F = (N'', E'')$ *such that:*

$$N'' = \{[x, y] \mid x, y \in N \wedge x \neq y\} \cup \{[x, x] \mid x \in S \cup T\}$$
$$E'' = \{[x, y] \rightarrow [x', y] \mid x \rightarrow x' \in E' \wedge v(x) \leq v(y)\}$$

Figure 3 shows the state graph of the line graph of Figure 2. As shown by Perl and Shiloach [2] the number of edges $|E''|$ in the state graph is in $O(|N| \cdot |E|)$ in terms of the original graph.

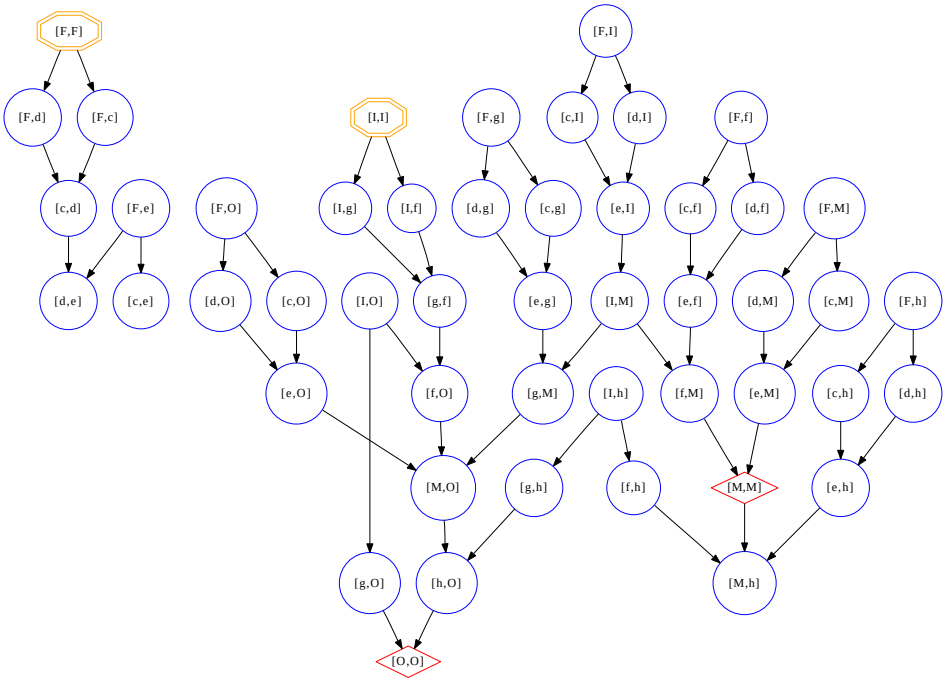


Fig. 3. The state graph of Figure 2 .

Checking for handles: In order to detect the existence of a handle in the original workflow graph, we have to check if there exists a path in the state graph from a state with two tokens on a node in S to a state with two tokens on a node in T. This is achieved by traversing the graph from each unvisited node

in S until reaching a node in $t \in T$ or having visited all the reachable nodes. In the worst case, the graph traversal visits each node of the graph once. The edges that belong to the handle in the workflow graph can be easily retrieved: They are the edges that are on the path from the state with two tokens on a node $s \in S$ to the state with two tokens on a node $t \in T$ of the handle in the state graph. On Figure 3 we observe that there is a path between $[I, I]$ and $[O, O]$ which indicates that there is an handle between I and O in workflow graph of Figure 1.

5 Conclusion

We propose an intuitive structural characterization for lack of synchronization in acyclic workflow graphs that contain inclusive OR logic: the handle. The handle is an adequate error message to the process modeler. We show how to check for handles by building a state graph. The size of the state graph is quadratic with respect to the original graph. All other operations are linear either with respect to the size of the original graph or the state graph. Thus, our approach requires quadratic time and space to check for handles. Note that the approach can be easily adapted to detect potential concurrency between two elements of the process. This has multiple applications. For example, we can detect two tasks accessing concurrently the same data store.

References

1. W. van der Aalst, A. Hirnschall, and H. Verbeek, "An alternative way to analyze workflow graphs," *Lecture Notes in Computer Science*, pp. 535–552, 2002.
2. Y. Perl and Y. Shiloach, "Finding two disjoint paths between two pairs of vertices in a graph," *J. Assoc. Comput. Mach*, vol. 25, no. 1, p. 9, 1978.
3. J. Esparza and M. Silva, "Circuits, handles, bridges and nets," *Advances in Petri nets*, vol. 483, pp. 210–242, 1990.
4. W. van der Aalst, "Workflow verification: Finding control-flow errors using Petri-net-based techniques," *Lecture Notes in Computer Science*, pp. 161–183, 2000.
5. A. V. Goldberg and R. E. Tarjan, "A new approach to the maximum-flow problem," *J. ACM*, vol. 35, no. 4, pp. 921–940, 1988.
6. F. Harary, "Graph Theory. 1969."
7. J. Gross and J. Yellen, *Graph theory and its applications*. CRC press, 2006.
8. B. Yang, S. Zheng, and E. Lu, "Finding two Disjoint Paths in a Network with Normalized α–Min-Sum Objective Function," 2005.

On the Behavioural Dimension of Correspondences between Process Models

Matthias Weidlich and Mathias Weske

Hasso-Plattner-Institute, University of Potsdam, Germany
{matthias.weidlich,weske}@hpi.uni-potsdam.de

Abstract. Enterprise-wide process harmonisation initiatives require the analysis of commonalities of existing business process models. That is, correspondences between activities are identified, such that the behavioural equivalence of the models can be assessed thereafter. Due to refinements, these correspondences can relate sets of activities to each other, i.e., there are complex 1:n or n:m correspondences. In this paper, we discuss how notions of behaviour inheritance can be applied in this context. In addition, we elaborate on how structural information can be leveraged to identify violations of behaviour inheritance.

1 Introduction

Enterprises model their business processes for various purposes, e.g., documentation of business operations, staff planning, or process automation. As a consequence, process models might show deviations in terms of level of detail, control flow, and activity labels, even if they capture a common real-world process.

In order to detect inconsistencies between related process models, their commonalities and differences have to be identified [1]. As a first step, this requires the identification of activities that *correspond* to each other. Note that this might not imply semantic equivalence of the activities (e.g., 'Get Quote' might correspond to 'Enter Quote Details' although both activities are not semantically equivalent). Such correspondences can be elementary 1:1 matches between activities, or complex 1:n (or even n:m) matches due to refinements of activities. Correspondences may stem from a system analyst inspecting the models or from automatic matching, cf., [2, 3]. As a second step, the behavioural equivalence of process models aligned by correspondences is assessed to detect inconsistencies and guide process harmonisation efforts.

Although not in a straight-forward manner, existing notions of behaviour inheritance might be applied for this purpose even in the presence of complex correspondences between two process models. However, we argue that certain violations of behaviour inheritance can be detected structurally. In this paper, we show that a structural decomposition of a sound free-choice WF-system might provide valuable hints at correspondences that violate behaviour inheritance. In particular, we focus on the notion of projection inheritance in the context of trace semantics, i.e., an adapted version of the trace equivalence criterion [4].

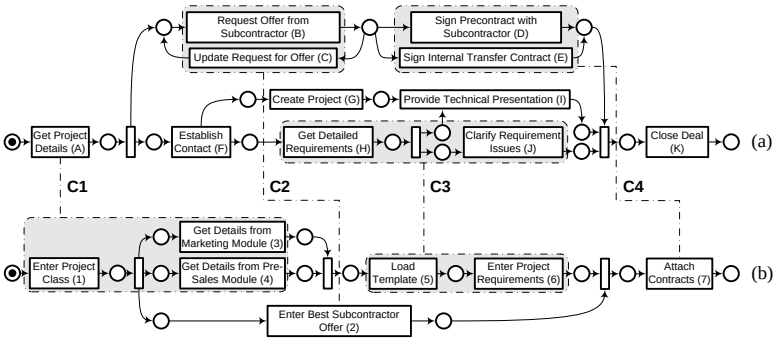


Fig. 1. Two exemplary process models, (a) focussing on the organisational perspective, (b) depicting the respective technical process, along with complex correspondences

Fig. 1 illustrates the problem addressed by this paper using two Petri nets that model a project planning process. Model (a) assumes an organisational perspective, whereas (b) shows the processing according to a supporting IT system. There are four complex correspondences highlighted using dash-dotted lines. Note that τ-transitions are never considered to be part of any correspondence. Clearly, the execution dependencies for transitions that are part of correspondences are not always consistent in both models. For instance, transitions D and H might be enabled concurrently in model (a), whereas their corresponding transitions, e.g., 5 and 7, are causally related in model (b). The question of whether and how such inconsistencies can be detected structurally is addressed in this paper.

The remainder of this paper is structured as follows. Section 2 gives preliminaries for our work. We discuss structural properties of correspondences in Section 3. Section 4 elaborates on behavioural analysis of correspondences. Finally, Section 5 reviews related work, before Section 6 concludes the paper.

2 Preliminaries

We recall basic definitions for workflow (WF-) systems based on [5, 6]. A *net* is a tuple $N = (P, T, F)$ with P and T as finite disjoint sets of places and transitions, and $F \subseteq (P \times T) \cup (T \times P)$ as the flow relation. We write $X = (P \cup T)$ for all nodes and F^+ for the transitive closure of F. For a node $x \in X$, $\bullet x := \{y \in X \mid (y, x) \in F\}$, $x\bullet := \{y \in X \mid (x, y) \in F\}$, $in_N(x) := \{(n, x) \mid n \in \bullet x\}$, and $out_N(x) := \{(x, n) \mid n \in x\bullet\}$. A net N is *free-choice*, iff $\forall\ p \in P$ with $|p\bullet| > 1$ holds $\bullet(p\bullet) = \{p\}$. N is conntected, if $\forall\ x_1, x_2 \in X\ [\ x_1 F^+ x_2\ \wedge\ x_2 F^+ x_1\ ]$. A *workflow (WF-) net* is a net $N = (P, T, F)$, such that there is exactly one place $i \in P$ with $\bullet i = \emptyset$, exactly one place $o \in P$ with $o\bullet = \emptyset$, and the short-circuit net $N' = (P, T \cup \{t_c\}, F \cup \{(o, t_c), (t_c, i)\})$ is connected. $M : P \mapsto \mathbb{N}$ is a *marking* of N and M_p denotes the marking that puts a token on p and no token elsewhere for a place $p \in P$. Given a WF-net N with M_i as its initial marking, we refer to the tuple $S = (N, M_i)$ as a *WF-system* and assume N to be defined always as $N = (P, T, F)$. We do not recall semantics for WF-systems but refer to [5, 6]. (N, M_i) is *sound*, iff the short-circuit system (N', M_i) is live and bounded [7].

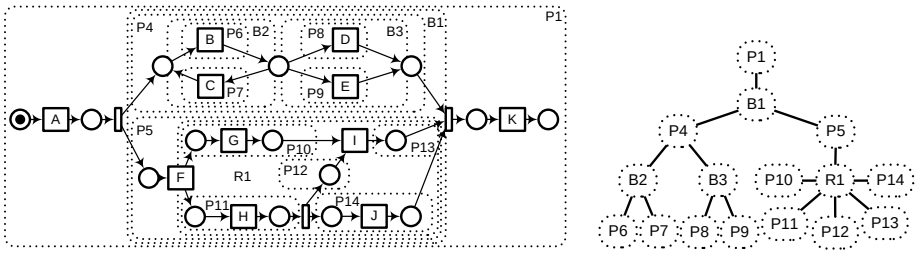


Fig. 2. (a) WF-system and its canonical fragments, (b) its RPST (without T-fragments)

We apply the structural decomposition technique introduced as the RPST in [8] to parse a WF-system into a hierarchy of fragments with a single entry node and a single exit node. Then, the RPST is a containment hierarchy of canonical fragments of the system that can be computed in linear time.

Definition 1 (Fragments, RPST). *Let $N = (P, T, F)$ be a WF-net with initial place i and final place o.*

- *A node $x \in X'$ of a connected subnet $N' = (P', T', F')$ of a net N is a* boundary node*, if $\exists\ e \in in_N(x) \cup out_N(x)\ [\ e \notin F'\ ]$. If x is a boundary node, it is an* entry *of N', if $in_N(x) \cap F' = \emptyset$ or $out_N(x) \subseteq F'$, or an* exit *of N', if $out_N(x) \cap F' = \emptyset$ or $in_N(x) \subseteq F'$.*
- *Any connected subnet ω of N, is a* fragment*, if it has exactly two boundary nodes, one entry and one exit.*
- *A fragment $\omega = (P_\omega, T_\omega, F_\omega)$ is* canonical *in a set of fragments Ω of N, iff $\forall\ \gamma = (P_\gamma, T_\gamma, F_\gamma) \in \Omega\ [\ \omega \neq \gamma \Rightarrow (F_\omega \cap F_\gamma = \emptyset) \vee (F_\omega \subset F_\gamma) \vee (F_\gamma \subset F_\omega)\ ]$.*

The RPST of N is a tuple $\mathcal{R}_N = (\Omega, \chi, t)$, where:

- *Ω is a set of all canonical fragments of N,*
- *$\chi : \Omega \to \mathcal{P}(\Omega)$ is a function that assigns to fragment its child fragments,*
- *$t : \Omega \to \{T, P, B, R\}$ is a function that assigns a type to a fragment.*

Fig. 2 shows the canonical fragments for one of the WF-systems of our example along with the corresponding RPST. Note that each canonical fragment can be classified as being either a trivial (T) fragment (a single edge), a polygon (P) fragment (a sequence of fragments), a bond (B) (fragment collection with common boundary nodes), or a rigid (R) fragment (any other structure).

3 Structural Properties of Correspondences

This section clarifies the notion of a correspondence between two WF-systems. Following on the terminology known from the domain of schema matching or ontology matching [9], a correspondence is defined by two sets of transitions of the WF-systems.

Definition 2 (Correspondence). *Let (N', M_i') and (N'', M_i'') be two WF-systems. The* correspondence relation $\sim\ \subseteq \wp(T') \times \wp(T'')$ *associates corresponding*

transitions of both systems to each other. Let $T_1 \subseteq T'$ *and* $T_2 \subseteq T''$*. If* $T_1 \sim T_2$ *then* (T_1, T_2) *is referred to as a* correspondence*.* (T_1, T_2) *is called* elementary*, iff* $|T_1| = |T_2| = 1$*, and* complex *otherwise.*

Further on, we discuss the structural relation between multiple correspondences between two systems by means of two structural properties. First, correspondences might be overlapping, i.e., they share a transition in at least one of the WF-systems.

Definition 3 (Overlapping Correspondences). *Let* (N', M_i') *and* (N'', M_i'') *be two WF-systems and* $C_1 = (T_1, T_2)$ *and* $C_2 = (T_3, T_4)$ *two correspondences between them, such that* $T_1, T_3 \subseteq T'$ *and* $T_2, T_4 \subseteq T''$*.* C_1 *and* C_2 *are called* overlapping*, iff* $T_1 \cap T_3 \neq \emptyset$ *or* $T_2 \cap T_4 \neq \emptyset$*.*

Regarding the example in Fig. 1, we see that all correspondences are non-overlapping.

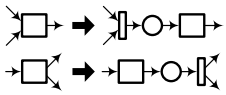


Fig. 3. Pre-processing

Second, we provide an additional property for the structural relation between two correspondences, which is based on the RPST. Here, we have the assumption that all transitions that are part of a correspondence have a single incoming and single outgoing flow arc. If not, a behaviour preserving pre-processing is done, cf., Fig. 3. Note that the systems in Fig. 1 are pre-processed. The property requires that two correspondences are not overlapping in terms of their lowest enclosing fragments in the RPST.

Definition 4 (Structurally Independent Correspondences). *Let* (N', M_i') *and* (N'', M_i'') *be two WF-systems,* $\mathcal{R}'_{N'} = (\Omega', \chi', t')$ *the RPST of* N'*, and* $C_1 = (T_1, T_2)$ *and* $C_2 = (T_3, T_4)$ *with* $T_1, T_3 \subseteq T'$ *and* $T_2, T_4 \subseteq T''$ *two correspondences.* C_1 *and* C_2 *are* structurally independent *in* N'*, iff* $|T_1| = |T_3| = 1$ *or* $\forall\ \omega = (P_\omega, T_\omega, F_\omega) \in \Omega'\ [\ T_1 \cap T_\omega \neq \emptyset \wedge T_3 \cap T_\omega \neq \emptyset \Rightarrow T_1 \subseteq T_\omega \wedge T_3 \subseteq T_\omega\ ]$*.*

Our example in Fig. 1 illustrates correspondences that are structurally independent in one or both systems. For instance, the pair of correspondences $C1$ and $C2$ is independent in model (a), as every fragment containing transition A and either B or C contains all three transitions, cf., Fig. 2. However, the very same pair of correspondences is not independent in model (b), as there is a bond fragment that represents a subnet comprising transitions 3, 4, and 5, but which does not contain transition 1. In contrast, $C1$ and $C4$ are an example for a pair of correspondences that are structurally independent in both models (a) and (b).

4 Behavioural Analysis of Correspondences

Based on the structural properties of correspondences introduced in the previous section, this section discusses behavioural analysis of correspondences. First, we elaborate on the behavioural ambiguity of overlapping correspondences. Second, the application of common notions of behaviour inheritance in the context of complex correspondences is discussed. Subsequently, we introduce heuristics to detect violations of behaviour inheritance for structurally independent correspondences.

Overlapping Correspondences. Overlapping correspondences raise various questions regarding their intended meaning. That is, it is unclear whether an occurrence of a transition that is part of both overlapping correspondences should be considered for one or for both correspondences. For instance, consider the systems depicted in Fig. 4. Here, the trace $A1, X$ in the lower system might correspond to both, the occurrence of transition A only, or the occurrence of both transitions, A and B, in the upper system. Thus, the semantic ambiguity of such overlapping correspondences has to be addressed as a prerequisite for any behavioural analysis.

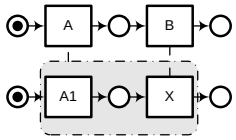


Fig. 4. Overlapping Correspondences

Behaviour Inheritance. In order to judge about behaviour equivalence in the context of correspondences between process models, first and foremost, the treatment of transitions without counterparts has to be addressed. To this end, two notions of behaviour inheritance can be distinguished, i.e., protocol inheritance or projection inheritance, cf., [10, 11]. That is, transitions in one model that are without counterpart in the second model are either blocked (protocol inheritance) or hidden (projection inheritance). Although these notions have been defined based on branching bisimulation, they can easily be transferred to trace-based behaviour analysis. It is worth to mention that the term inheritance is slightly misleading in our context as it implies a directed relation between two models (one model is a subclass of the other model). As illustrated by our example in Fig. 4, such a directed relation might not exist between aligned models.

The choice between protocol inheritance or projection inheritance is independent of the structural characteristics of the correspondences. However, we focus on projection inheritance in the context of this paper. In particular, this notions seems to be more appropriate when comparing models that focus on different modelling perspectives (e.g., organisational vs. technical) as intermediate process steps that are mandatory in one perspective may be without counterpart in the other model. For instance, transition F of model (a) in Fig. 1 is not part of any correspondence even though its execution is required for completion of process (a). Thus, any notion of protocol inheritance would be bound to failure.

In the general case, projection inheritance in terms of trace semantics between two models is assessed as follows. The language of both process models is checked for equivalence, when all correspondences are *resolved*. In case of 1:1 elementary correspondences this resolution is straightforward as an occurrence of one transition in one model corresponds to the occurrence of another transition in the other model. For the case of complex correspondences, this resolution involves a replacement of all valid subtraces involving the transitions of the correspondence in the one model with all valid subtraces comprising the corresponding transitions in the other model. Consider the example in Fig. 1 and neglect correspondence $C2$ and $C3$ for the time being. Then, the traces A, D and A, E in model (a) (all other transitions are hidden) would be rewritten as follows. According to the correspondences, an occurrence of A is replaced by the subtraces $1, 3, 4$ or $1, 4, 3$, while the subtraces D and E are replaced by an occurrence of transition 7. That

yields the two traces $1, 3, 4, 7$ and $1, 4, 3, 7$. As both traces described the language of model (b) when taking only transitions of correspondences $C1$ and $C4$ into account, we conclude that both correspondences do not violate projection inheritance. Note that during this analysis, the interleaving of transitions belonging to different correspondences has to be considered. Based thereon, each pair of correspondences can be assessed. If all correspondences show pairwise projection inheritance, projection inheritance for the two models and their alignment in terms of a set of correspondences can be concluded.

Structural Characterisation of Violations. While the aforementioned approach of comparing the set of all resolved traces can always be taken, violations of projection inheritance may already be discovered using a heuristic approach. It uses the RPST decomposition technique for sound free-choice WF-systems and is applicable for correspondences that are structurally independent in both systems (cf., Section 3). In previous work, we showed that the RPST of sound free-choice WF-systems can be annotated with behavioural details. That is, bond fragments are either transition bordered or place bordered, but there cannot be a mix of place and transition boundary nodes [12]. The former represents a parallel block, whereas the latter represents either an exclusive block or a loop fragment.

This information, in turn, can be leveraged to identify violations of projection inheritance that are induced by correspondences. Given two correspondences we proceed as follows.

1. Determine the lowest common ancestor (LCA) in the respective RPST of all trivial T fragments for which the entry node is a transition aligned by one of the two correspondences. Derive the LCA in both models.
2. Compare the type of the LCA fragments.
   - If one LCA fragment is of type P, the other LCA fragment has to be of type P as well and the order of the respective child fragments (transitively) containing all T fragments of the correspondences has to coincide in both LCA fragments.
   - If one LCA fragment is of type B and not cyclic, the other LCA fragment has to be of type B and acyclic as well. In addition, the fragments are either both place bordered or both transition bordered.
   - If one LCA fragment is of type B and cyclic, the other LCA fragment has to be of type B and cyclic as well.

This observation can be explained as follows. As both correspondences are structurally independent, the LCA of all T fragments that are related to the transitions of the correspondence must have at least two children, one (transitively) containing all T fragments related to one correspondence and the other one (transitively) containing all T fragments related to the other correspondence. Therefore, the type of the LCA induces a constraint on the potential order of occurrence between all transitions of one correspondence and all transitions of the other correspondence. This constraint has to hold in the second model as well. For instance, if there is a sequential order (LCA is P fragment) between the respective T fragments (and, therefore, the transitions) of the two correspondences in one model, there cannot be a potential concurrent enabling (LCA is B fragment

that is transition bordered) of the corresponding transitions in the other model. Such a setting will never show projection inheritance. It is important to see two limitations of this approach. On the one hand, conclusions cannot be drawn, if one of the two LCAs is a rigid (R) fragment, as the constraints on the order of potential execution cannot be derived directly. On the other hand, this check is solely a heuristic that formulates a necessary, but not a sufficient condition for a violation of projection inheritance. Even if the type of the LCA is equal, projection inheritance might be violated by the two correspondences. For instance, if both LCAs are of type P, it might still be the case that the occurrence of all transitions of one correspondence is optional in one model, whereas it is mandatory in the other model.

Besides $(C1, C2)$ and $(C1, C3)$, all pairs of correspondences in Fig. 1 are structurally independent in both models, such that the types of the respective LCA fragments can be compared. That, in turn, yields the following result. For the pairs $(C1, C4)$, $(C2, C3)$, and $(C2, C4)$ the LCA fragments show a consistent type in both models. In contrast, for correspondence $(C3, C4)$ the LCA fragment is of type B (transition bordered) in model (a) and of type P in model (b). Thus, the structural analysis revealed that correspondences $C3$ and $C4$ violate projection inheritance between both models. Transitions of both correspondences might be enabled concurrently in model (a), which is not possible in model (b).

Regarding correspondences that are not structurally independent in both systems, we already discussed the case of correspondences $C1$ and $C2$ of our example in Fig. 1. These correspondences obviously violate projection inheritance. However, our example also illustrates that correspondences that are structurally dependent do not necessarily lead to violations. Consider, for instance, correspondences $C1$ and $C3$, which are not structurally independent in model (b), due to fragments that represent subnets containing transitions $3, 4, 5, 6$, but not 1. Nevertheless, the order between all transitions of correspondence $C1$ and $C3$ as imposed by model (a) is also reflected in model (b). Albeit beyond the scope of this discussion, we foresee that necessary conditions for violations can also be specified for the case of correspondences that are not structurally independent.

5 Related Work

We already discussed related work in the field of behaviour equivalence [4] and behaviour inheritance [10, 11]. Further on, work on trace-based assessment of similarity is related to our work. For instance, the set n-grams of possible traces two models can be compared [13]. In other work, a trace is replayed in a model and the number of valid replay steps is leveraged for measuring similarity [14].

Moreover, the assessment of behaviour inheritance in the presence of complex correspondences is related to work on behaviour preserving model refinements, see [15] for a thorough survey. A transition refinement is defined by replacing a transition with a block-structured refinement net with a dedicated set of initial and final transitions. However, other approaches allow refinement nets to also show so-called distributed input or distributed output places [15].

6 Conclusion

In this paper, we addressed the challenge of assessing behaviour equivalence for process models aligned by complex correspondences. In particular, we discussed how the notion of behaviour inheritance can be applied for complex correspondences and also introduced necessary conditions for violations of this notion by a pair of correspondences based on structural decomposition techniques for sound free-choice workflow systems.

As a next step, the discussed application of behaviour inheritance in the context of complex correspondences needs to be formalised. In addition, we want to further investigate the structural characterisation of correspondences and their relation to the notions of behaviour inheritance. Here, the goal would be the specification of structural characterisations that are not only a necessary, but also a sufficient condition for a violation of behaviour inheritance.

References

1. Dijkman, R.M.: Diagnosing differences between business process models. In: BPM. Volume 5240 of LNCS., Springer (2008) 261–277
2. Dijkman, R., Dumas, M., García-Bañuelos, L., Käärik, R.: Aligning business process models. In: EDOC. (2009) 45–53
3. Nejati, S., Sabetzadeh, M., Chechik, M., Easterbrook, S.M., Zave, P.: Matching and merging of statecharts specifications. In: ICSE, IEEE Computer Society (2007) 54–64
4. van Glabbeek, R.: The linear time - brancing time spectrum I. The semantics of concrete, sequential processes. In: Handbook of Process Algebra. Elsevier (2001) 3–99
5. Aalst, W.: The application of Petri nets to workflow management. Journal of Circuits, Systems, and Computers **8**(1) (1998) 21–66
6. Desel, J., Esparza, J.: Free Choice Petri Nets. Cambridge University Press (1995)
7. Aalst, W.: Verification of workflow nets. In: ICATPN. Volume 1248 of LNCS. (1997) 407–426
8. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. In: BPM. Volume 5240 of LNCS. (2008) 100–115
9. Euzenat, J., Shvaiko, P.: Ontology Matching. Springer-Verlag (2007)
10. Basten, T., Aalst, W.: Inheritance of Behavior. Journal of Logic and Algebraic Programming **47**(2) (2001) 47–145
11. Schrefl, M., Stumptner, M.: Behavior-consistent specialization of object life cycles. ACM Trans. Softw. Eng. Methodol. **11**(1) (2002) 92–148
12. Weidlich, M., Polyvyanyy, A., Mendling, J., Weske, M.: Efficient computation of causal behavioural profiles using structural decomposition. In: ICATPN. (2010) accepted for publication.
13. Wombacher, A.: Evaluation of technical measures for workflow similarity based on a pilot study. In: OTM (1). Volume 4275 of LNCS., Springer (2006) 255–272
14. de Medeiros, A.K.A., van der Aalst, W.M.P., Weijters, A.J.M.M.: Quantifying process equivalence based on observed behavior. Data Knowl. Eng. **64**(1) (2008) 55–74
15. Brauer, W., Gold, R., Vogler, W.: A survey of behaviour and equivalence preserving refinements of petri nets. In: ICATPN. Volume 483 of LNCS., Springer (1989) 1–46

Structural Abstraction of Process Specifications

Artem Polyvyanyy

Business Process Technology Group
Hasso Plattner Institute at the University of Potsdam
Prof.-Dr.-Helmert-Str. 2–3, D-14482 Potsdam, Germany
Artem.Polyvyanyy@hpi.uni-potsdam.de

Abstract. Software engineers constantly deal with problems of designing, analyzing, and improving process specifications, e.g., source code, service compositions, or process models. Process specifications are abstractions of behavior observed or intended to be implemented in reality which result from creative engineering practice. Usually, process specifications are formalized as directed graphs in which edges capture temporal relations between decisions, synchronization points, and work activities. Every process specification is a compromise between two points: On the one hand engineers strive to operate with less modeling constructs which conceal irrelevant details, while on the other hand the details are required to achieve the desired level of customization for envisioned process scenarios. In our research, we approach the problem of varying abstraction levels of process specifications. Formally, developed abstraction mechanisms exploit the structure of a process specification and allow the generalization of low-level details into concepts of a higher abstraction level. The reverse procedure can be addressed as process specialization.

Keywords: Process abstraction, process structure, process connectivity, process modeling

1 Introduction

Process specifications represent exponential amounts of process execution scenarios with linear numbers of modeling constructs, e.g., service compositions and business process models. Nevertheless, real world process specifications cannot be grasped quickly by software engineers due to their size and sophisticated structures, leading to a demand for techniques to deal with this complexity. The research topic of process abstraction emerged from a joint research project with a health insurance company. Operational processes of the company are captured in about 4 000 EPCs. The company faced the problem of information overload in the process specifications when employing the models in use cases other than process execution, e.g., process analysis by management. To reduce the modeling effort, the company requested to develop automated mechanisms to derive abstract, i.e., simplified, process specifications from the existing ones. The research results derived during the project are summarized in [1].

Abstraction is the result of the generalization or elimination of properties in an entity or a phenomenon in order to reduce it to a set of essential characteristics.

Information loss is the fundamental property of abstraction and is its intended outcome. When working with process specifications, engineers operate with abstractions of real world concepts. Process specifications are special types of entities that describe principles of observed or intended behavior. In our research, we develop mechanisms to perform abstractions of formal process specifications. The challenge lies in identifying what the units of process logic suitable for abstraction are, and then performing the abstraction afterwards. Once abstraction artifacts are identified, they can be eliminated or replaced by concepts of higher abstraction levels which conceal, but also represent, abstracted detailed process behavior. Finally, individual abstractions must be controlled in order to achieve an abstraction goal—a process specification that suits the needs of a use case.

Fig. 1 shows an example of two process specifications, given as BPMN process models, which are in the abstraction relation. The example is adapted from [2]. The model at the top of the figure is the abstract version of the model at the bottom. Abstract tasks are highlighted with a grey background; the corresponding concealed fragments are enclosed within the regions with a dashed borderline. The fragments have structures that result in an abstract process which captures the core process behavior of the detailed one. The abstract process has dedicated routing and work activity modeling constructs and conceals detailed behavior descriptions, i.e., each abstracted fragment is composed of several work activities. The research challenge lies in proposing mechanisms which allow examining every process fragment prior to performing abstraction and suggesting mechanisms which coordinate individual abstractions, i.e., assign higher priority to abstracting certain fragments rather than the others.

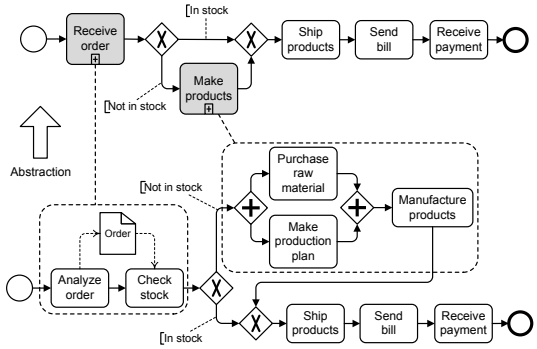


Fig. 1. Process abstraction

The rest of the paper is organized as follows: The next section presents the connectivity-based framework designed to approach the discovery of process fragments suitable for abstraction. Sect. 3 discusses issues relevant to the control of process abstraction. The paper closes with conclusions which summarize our findings and ideas on further research steps.

2 Discovery of Process Fragments

A necessary part of a solution for process abstraction is a mechanism for the discovery of process fragments, i.e., parts of process logic suitable for abstraction. The chances of making a correct decision on which part of a process specification to abstract from can only be maximized if all the potential fragment candidates

for conducting an abstraction are considered. To achieve this completeness, we employ the *connectivity* property of process graphs—directed graphs used to capture process specifications.

Connectivity is a property of a graph. A graph is *k-connected* if there exists no set of $k-1$ elements, each a vertex or an edge, whose removal makes the graph disconnected, i.e., there is no path between some pair of elements in a graph. Such a set is called a separating $(k-1)$-set. 1-, 2-, and 3-connected graphs are referred to as connected, biconnected, and triconnected, respectively. Each separating set of a process graph can be addressed as a set of boundary elements of a process fragment, where a boundary element is incident with elements inside and outside the fragment and connects the fragment to the main flow of the process. Let m be a parameter, the discovery of all separating m-sets (graph decomposition) of the process graph leads to the discovery of all process fragments with m boundary elements—potential abstraction candidates.

The *vertex (edge) connectivity* of a graph is the size of the smallest separating set of the graph that is composed only of vertices (edges). For an arbitrary graph it holds that its vertex connectivity is less than or equal to its edge connectivity [3]. Intuitively, removal of an edge, when testing the edge connectivity, can be substituted with removal of an incident vertex, which implies removal of all incident edges. In general, one can speak about (n, e)-*connectivity* of a graph. A graph is (n, e)-*connected* if there exists no set of n nodes and there exists no set of e edges whose removal makes the graph disconnected. Observe, an (n, e)-connected graph is $(n+e+1)$-connected. A lot of research was carried out by the compiler theory community to gain value from the triconnected decomposition of process specifications, i.e., the discovery of triconnected fragments in process graphs. The decompositions which proved useful were $(2, 0)$-decomposition, or the tree of the triconnected components, cf., [4], and $(0, 2)$-decomposition, cf., [5]. Triconnected process graph fragments form hierarchies of single-entry-single-exit (SESE) fragments and are used for process analysis, process comparison, process comprehension, etc. For these decompositions, linear-time complexity algorithms exist [6,7]. Recently, these techniques were introduced to the business process management community [8,9]. We employed triconnected process graph fragments to propose mechanisms of process abstraction [10,11]; we discover and generalize the triconnected process fragments to tasks of a higher abstraction level.

Fig. 2 shows an example of a process graph. Routing decisions and synchronization points can be distinguished by the degree of the corresponding vertex, i.e., the number of incident edges, and orientation of the incident edges, i.e., incoming or outgoing. Process starts (ends) have no incoming (outgoing) edges. More-

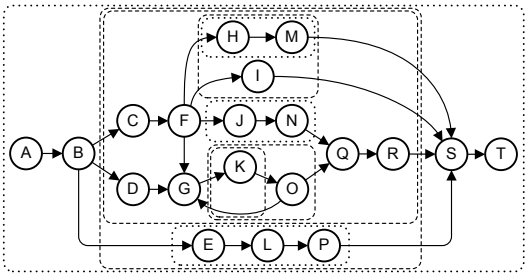
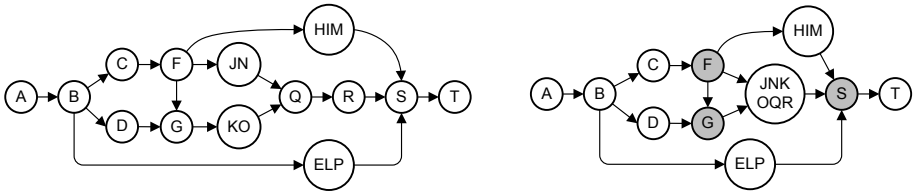


Fig. 2. Process graph, its SESE fragments



(a) (b)

Fig. 3. (a) Abstract process graph, (b) $(3, 0)$-connected fragment abstraction

over, Fig. 2 visualizes the triconnected fragments of the graph (SESE fragments). Each triconnected fragment is enclosed in the region and is formed by edges inside or intersecting the region. Fragments enclosed by regions with dotted borderlines can be discovered after performing $(0, 2)$-decomposition of the process graph, whereas regions with dashed borderlines define fragments additionally discovered by $(2, 0)$-decomposition. Observe that trivial fragments, the fragments that represent sequences composed of a single vertex, are not visualized.

The abstract process graph, obtained from the graph shown in Fig. 2, is given in Fig. 3(a). The abstraction is performed following the principles from [11], i.e., by aggregating the triconnected fragments into concepts of a higher abstraction level. The graph from Fig. 3(a) has a single separating pair B, S. Next abstractions of the triconnected fragments of the graph will result in aggregation of either the unstructured fragment composed of nodes $\{C, ..., R\} \setminus \{E, L, P\}$, or the fragment with entry node B and exit node S. To increase the granularity of process fragments used to perform abstractions, one can start looking for multiple-entries-multiple-exits (MEME) fragments within the triconnected fragments.

Fig. 4 visualizes a connectivity-based process graph decomposition framework, i.e., the scheme for process fragment discovery. In the figure, each dot represents a connectivity property of the process graph (process fragment) subject to decomposition, e.g., $(0, 0)$ means that the graph is connected if no nodes and no edges are removed. Edges in the figure suggest which decompositions can be performed for a graph with a certain connectivity level. For example, one can decompose a $(0, 0)$-connected graph by looking for a single node or an edge which renders the graph disconnected. Afterwards, obtained fragments can be treated as subjects for $(2, 0)$-, $(1, 1)$-, or $(0, 2)$-decomposition. By proceeding in this way, highly connected fragments get gradually decomposed.

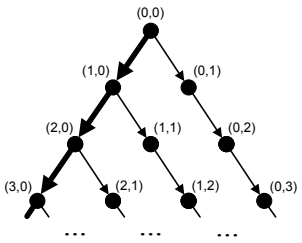


Fig. 4. Connectivity-based process graph decomposition framework

A careful reader might notice that Fig. 4 does not suggest all the possibilities for accomplishing decomposition. For instance, $(0, 2)$-connected fragments can be subjects for $(2, 0)$-decomposition, cf., Fig. 2. As the vertex connectivity of a graph is less than or equal to its edge connectivity, we propose to give the preference to the vertex-based decomposition over the edge-based one (refer to

the left-most path in Fig. 4). Such a strategy stimulates the fragments to be discovered "faster", i.e., by performing less decomposition steps, and "finer", i.e., by discovering more fragments. However, in the cases when it is important to achieve the granularity on the level of edges, we suggest to deviate from the main strategy only once by switching from the vertex-based to the edge-based decomposition strategy.

An (n, e)-decomposition ($n+e \geq 3$) allows to decompose unstructured process graphs into MEME fragments with $n + e$ entries and exits. A process graph in Fig. 3(b) is obtained by abstracting a $(3, 0)$-connected fragment defined by a separating set $\{F, G, S\}$ (F and G are the entries and S the exit of the fragment, highlighted with grey background). For reasonable $n + e$ combinations, it is possible to perform decomposition in low polynomial-time complexity. For example, the $(3, 0)$-decomposition of a $(2, 0)$-connected graph can be accomplished by removing a vertex from the graph and afterwards running the triconnected decomposition, for which the linear time complexity algorithm exists [7]. Each discovered separation pair together with the removed vertex form a separating triple of the original graph. The procedure should be repeated for each vertex of the graph. Hence, a square-time complexity decomposition procedure is obtained. Following the described rationale, one can accomplish $(k, 0)$-decomposition in $O(n^{k-1})$ time, where n is proportional to the size of the graph.

When performing the $(n + 1, e)$- or $(n, e + 1)$-decomposition of an (n, e)-connected graph, one can remove a vertex or, respectively, an edge from the graph and run the same algorithm that was used when obtaining the (n, e)-connected graph; the procedure should be repeated for each vertex or, respectively, edge. The $(2, 0)$-decomposition results in the hierarchy of $(2, 0)$-connected fragments organized in a tree structure [4]. In the case of the $(3, 0)$-decomposition, one obtains a forest, where each tree is the result of the $(2, 0)$-decomposition of the graph after the removal of a single vertex. The amount of trees in the forest is equal to the number of vertices in the graph. Such a forest can be treated as the structural characterization of the $(2, 0)$-connected graph.

By following the principles of the connectivity-based decomposition framework, we not only discover process fragments used to perform process abstraction, but also learn their structural characteristics. Structural information is useful at other stages of abstraction, e.g., when introducing control over abstraction. Initial work on classifying and checking the correctness of process specifications based on discovered process fragments was accomplished in [8,12].

3 Abstraction Control

The task of adjusting the abstraction level of process specifications requires intensive intellectual work and in most cases can only be accomplished by process analysts manually. However, for certain use cases it is possible to derive automated or to support semi-automated abstraction control mechanisms. The task of abstraction control lies in telling significant process elements from insignificant ones and to abstract the latter. In [1], work activities are classified as insignificant

if they are rarely observed during process execution. We were able to establish the abstraction control when investigated processes were annotated with information on the average time required to execute activities. Process fragments which contain insignificant activities get abstracted. Hence, we actually deal with the significance of fragments which represent detailed work specifications.

Significant and insignificant process fragments can be distinguished once a technique for fragment comparison is in place, i.e., a partial order relation is defined for process fragments. The average time required to execute activities in the process provides an opportunity to derive a partial order relation, i.e., fragments which require less time are considered insignificant. Other examples of criteria and the corresponding abstraction use cases are discussed in [13].

Once an abstraction criterion, e.g., the average execution time of activities, is accepted for abstraction, one can identify a minimal and a maximal values of the criterion for a given process specification. In our example, the minimal value corresponds to the most rarely observed activity of the process and the maximal value corresponds to the average execution time of the whole process. By specifying a criterion value from the interval, one identifies insignificant process fragments—those that contain activities for which the criterion value is lower than the specified value. Afterwards, insignificant fragments get abstracted. In [13], we proposed an abstraction slider as a mechanism to control abstraction. An abstraction slider is an object that can be described by a slider interval defined by a minimal and a maximal allowed values for an abstraction criterion, has a state—a value that defines the desired abstraction level of a process specification, and exposes behavior—an operation that changes its state.

4 Conclusion

In our research, we develop methods which allow the derivation of high abstraction level process specifications from detailed ones. In order to discover fragments suitable for abstraction, we employ the structure of process specifications, which are usually formalized as directed graphs. As an outcome, developed techniques can be generalized to any process modeling notation which uses directed graphs as the underlying formalism.

It is a highly intellectual task to bring a process specification to a level of abstraction that fulfills emergent engineering needs, and one without a single perfect solution. By employing the technique for the discovery of abstraction fragments, one can approach the problem as a manual engineering effort. Besides, when it is sufficient to fulfill certain use case, cf., [1], one can define principles for a semi-automated or fully automated composition of individual abstractions.

As future steps aimed at strengthening the achieved results, we plan to validate the applicability of the connectivity-based process graph decomposition framework for the purpose of process abstraction with industry partners and to look for process abstraction use cases for which automated control mechanisms can be proposed. Finally, studies regarding the methodology of abstractions need to complement the technical results.

References

1. Polyvyanyy, A., Smirnov, S., Weske, M.: Reducing complexity of large EPCs. In: Modellierung betrieblicher Informationssysteme. Volume 141 of Lecture Notes in Informatics., Saarbruecken, Germany (November 2008) 195–207
2. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer Verlag (2007)
3. Diestel, R.: Graph Theory. Springer Verlag (2005)
4. Tarjan, R.E., Valdes, J.: Prime subprogram parsing of a program. In: Proceedings of the 7th Annual ACM Symposium on Principles of Programming Languages, New York, NY, USA, ACM Press (1980) 95–105
5. Johnson, R.: Efficient Program Analysis using Dependence Flow Graphs. PhD thesis, Cornell University, Ithaca, NY, USA (1995)
6. Johnson, R., Pearson, D., Pingali, K.: The program structure tree: Computing control regions in linear time. In: Proceedings of the ACM Conference on Programming Language Design and Implementation, ACM Press (1994) 171–185
7. Gutwenger, C., Mutzel, P.: A linear time implementation of SPQR-trees. In: Proceedings of the 8th International Symposium on Graph Drawing, London, UK, Springer Verlag (2001) 77–90
8. Vanhatalo, J., Völzer, H., Leymann, F.: Faster and more focused control-flow analysis for business process models through SESE decomposition. In: Proceedings of the 5th International Conference Service-Oriented Computing. Volume 4749 of Lecture Notes in Computer Science., Springer Verlag (September 2007) 43–55
9. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. In: Proceedings of the 6th International Conference on Business Process Management. Volume 5240 of Lecture Notes in Computer Science., Milan, Italy, Springer Verlag (September 2008) 100–115
10. Polyvyanyy, A., Smirnov, S., Weske, M.: On application of structural decomposition for process model abstraction. In: Proceedings of the 2nd International Conference on Business Process and Service Computing. Volume 147 of Lecture Notes in Informatics., Leipzig, Germany, GI (2009) 110–122
11. Polyvyanyy, A., Smirnov, S., Weske, M.: The triconnected abstraction of process models. In: Proceedings of the 7th International Conference on Business Process Management. Volume 5701 of Lecture Notes in Computer Science., Ulm, Germany, Springer Verlag (September 2009) 229–244
12. Polyvyanyy, A., García-Bañuelos, L., Weske, M.: Unveiling hidden unstructured regions in process models. In: Proceedings of the 17th International Conference on Cooperative Information Systems. Volume 5870 of Lecture Notes in Computer Science., Vilamoura, Algarve-Portugal, Springer Verlag (November 2009) 340–356
13. Polyvyanyy, A., Smirnov, S., Weske, M.: Process model abstraction: A slider approach. In: Proceedings of the 12th International IEEE Conference on Enterprise Distributed Object Computing, Munich, Germany, IEEE Computer Society (September 2008) 325–331

Mapping Interconnection Choreography Models to Interaction Choreography Models

Oliver Kopp, Frank Leymann, and Fei Wu

Institute of Architecture of Application Systems, University of Stuttgart, Germany
lastname@iaas.uni-stuttgart.de

Abstract Choreographies offer a global view on interacting processes. There are two ways to capture this global view: interaction models and interconnection models. Although there is a mapping from interaction models to interconnection models, there is no mapping vice versa. This paper fills this gap and provides a first approach mapping interconnection models to interaction models: The presented approach transforms BPMN models into iBPMN models by using Petri nets as intermediate format.

1 Introduction

Service choreographies describe the interactions between multiple processes [1]. Currently, there are two modeling types available: interaction models and interconnection models. The basic building block of interaction models is the interaction activity. This activity describes a message exchange between two participants in a choreography. In the case of interconnection models, messaging activities of different processes are interconnected. These two metamodels can be seen as different views on the same choreography.

Decker and Weske [2, 3] show a way to generate interconnection models out of interaction models. Currently, there is no work to generate interaction models out of interconnection models. This paper presents a first idea for such a mapping. The motivation is to provide a high-level view on existing interacting services. For this paper, we assume that an interconnection BPMN model—a BPMN model having pools with activities and message flows—has already been constructed out of interacting services.

Consequently, this paper is structured as follows: First, Sect. 2 provides background information and presents related work in the field. Section 3 presents an overview on the approach and Sect. 4 discusses the approach. Finally, Sect. 5 concludes and provides an outlook on future work.

2 Background and Related Work

Figure 1 presents different viewpoints in service-oriented design, adapted to current languages in the Web services stack. (A) On the top service value networks are shown. Service value networks provide a high view on the relationship between services without giving details on the collaboration [4]. (B) Below the service value networks, we see interaction choreography models, such as BPMN 2.0

choreographies [5], iBPMN [6] or BPEL^{gold} [7]. They provide a high-level view on the collaboration protocol by modeling message exchanges as single activities. Most interaction choreography languages hide internal behavior [8]. (C) Interconnection choreography models model each participant as a separate process and may provide internal behavior. Samples for interconnection choreography languages are BPMN 2.0 collaborations, BPMN 1.2 models with more than one pool [9], BPMN^{+} [10] or BPEL4Chor [11]. (D) The behavior of each participant can be expressed as a single BPMN pool or an abstract BPEL process. Additional possibilities to specify behavior of a participant include the open net approach [12]. (E) At the lowest level of abstraction, executable BPMN models or executable BPEL models reside. Each of these models can be deployed on a workflow engine and be enacted.

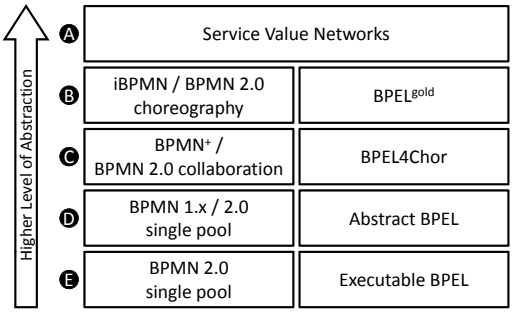


Figure 1. Different viewpoints

Different viewpoints in service-oriented design were first presented in [13], where especially the choreography view and the orchestration view are distinguished. The choreography view does not distinguish between interaction and interconnection models. Thus, it is unclear, whether interconnection models really are on a lower level than interaction models. That aspect should be investigated in future work. Barros et al. [14] propose role-based views, milestone models and scenario models as views on top of choreography modeling. They regard interaction models as the most abstract choreography models available. Using the technique presented in this paper interaction models can be derived out of interconnection models. The generated interaction models in turn may form a basis to extract a role-based view. This view can then be used to identify potentials for improvement.

The need to generate view on the protocol of running service interactions has been identified by Motahari-Netzhad et al. [15], too. One motivation for them is the evolution of services: in case the implementation of a service evolves, protocol discovery provides an up-to-date protocol definition. Motahari-Netzhad et al. call interaction models protocols and model them by finite state machines [16]. In contrast to our work, they generate interaction models out of audit logs and not out of interconnection models.

It has not yet been investigated whether interaction choreography models or interconnection choreography models are more suitable to capture choreographies. The expressiveness of the two modeling styles, however, is not equal: Decker and Weske [2,3] show that there are anti-patterns in BPMN which cannot be rendered in iBPMN. One example is anti-pattern “D2: Impossible data-based decisions”. For instance, a bidder deciding for himself whether he has won an auction—and not waiting for a decision message—is an instance of that anti-pattern. An additional distinction is that most interaction models do not support internal activities [8].

Regarding the mapping of interaction models to interconnection models, Decker et al. [17] study the property of “non-desynchronizability”: it is impossible to map all interaction models mapped to interconnection models by just splitting an interaction task to a

send task and a receive task. The reason is that interaction models assume synchronous communication, whereas interconnection models assume asynchronous communication. In our mapping, we go from asynchronous communication to synchronous communication. Our issue is then the property of "synchronizability", which has been studied by Fu et al. [18]: a set of interacting processes is synchronizable if they are (a) synchronous compatible and (b) autonomous with respect to the choice of the next action. A set of interacting processes is synchronous compatible if for each state where a send task is active, a receive task is reachable by internal transitions only. The autonomous condition demands that each process can only terminate, send or receive a message at one time. This conditions disallows the choice between sending and receiving a message.

A "realizable" choreography model is an interaction model which can be implemented by a set of processes where the composition exactly shows the specified message exchange behavior [19, 20]. Realizability is a property of an interaction choreography model. Our transformation generates a realizable interaction choreography model, since the input of the transformation is a set of processes which realizes the generated interaction choreography model.

Currently, following transformations between the languages presented in Fig. 1 are available: The interplay between Service Value Networks and choreographies is shown in [4]. The integration between BPMN 1.2 choreographies and BPEL4Chor has been investigated in [21]. A mapping from BPEL^{gold} to BPEL4Chor is presented in [7]. A mapping from executable BPEL processes to a BPEL4Chor description is presented in [22]. Currently, there is neither a mapping from BPMN 1.2 choreographies to iBPMN nor a mapping from BPEL4Chor to BPEL^{gold}. This paper presents a first approach of a possible mapping. In case the techniques are applied to a BPEL4Chor to BPEL^{gold} mapping, it is possible to generate an interaction choreography view out of executable BPEL processes.

Figure 2 presents a sample choreography for investments using the BPMN 1.2 notation. First, a financial advisor offers a product to a client. Subsequently, the client has 24 hours time to decide whether he accepts the investment proposal or rejects the proposal. The example is used in [23] to check choreography conformance during runtime. The equivalent interaction model is presented in Fig. 3. Here, the control flow is rendered between the pools and the local messaging activities have been replaced by interaction activities. The timer event and the data-based exclusive gateway are annotated with the controlling role customer indicating that the customer is responsible to control the time and the decision which message to send.

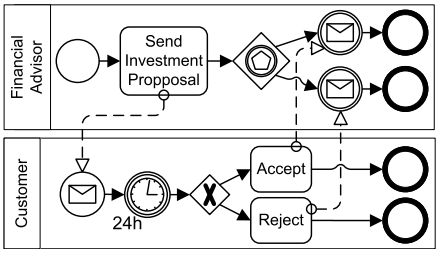


Figure 2. Interconnection model describing investment offers

Decker and Barros [6] introduce interaction Petri nets (iPNs) as formal foundation of iBPMN. Transitions are separated into interaction transitions, controlled silent transitions and uncontrolled silent transitions. Interaction transitions represent an interaction between two participants, controlled silent transitions represent decisions and timers controlled by a set of dedicated participants. Uncontrolled silent transitions are used solely for routing purposes. We use iPNs as an intermediate format to go from BPMN to iBPMN.

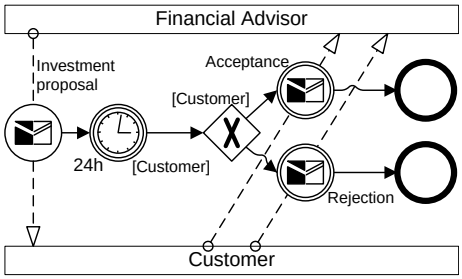


Figure 3. Interaction model describing investment offers

Reduction techniques for Petri nets are presented in [24] and [25]. Murata [24] presents basic techniques to collapse redundant structures, such as a sequence of places and transitions, where a transition only has one incoming edge and one outgoing edge. Berthelot et al. [25] introduce the definition of redundant places. We use their definition to reduce the Petri net resulted from the mapping.

Dijkman et al. [26] present a mapping from BPMN to classical Petri nets. The mapping supports multi-instance loops if the number n of instances is known at design time. In this case, the part of the Petri net representing the respective part of the loop is duplicated n times. As the semantics of the OR join in BPMN is not formally defined, the work does not map OR joins at all. The work of Dijkman et al. is an integral part of our transformation.

Lohmann et al. [27] present a survey on current transformation approaches from BPEL, BPMN and EPC process models to Petri nets. There, the work of Dijkman et al. is also regarded as the de facto approach to transform BPMN models to Petri nets.

3 Overview on the Approach

The goal of the transformation is to keep the ordering of the message exchanges and to provide an iBPMN model with as least nodes and edges as possible. Thus, we want to keep the set of traces as well as the branching behavior as Decker et al. did in their mapping from interconnection models to interconnection models [19]. Since this paper presents a first idea, we do not present a formal definition of the properties we want to preserve here.

We require the input BPMN model to be sound and safe. Furthermore, we require them to contain only following elements: sequence flows, plain start events, start message events, intermediate message events, tasks, data-based exclusive gateways, event-based exclusive gateways, parallel gateways, intermediate timer events as well as plain end events. Tasks may be configured as while or repeat until loop. Regarding message flows, we demand that tasks and events are only connected to at most one message flow and that each message flow has a source and a target. Thus, our work focuses on the positive control flow only and excludes exception, termination and compensation handling. Finally, we require the BPMN model to be free of the anti-patterns presented in [2, 3] and to be synchronizable.

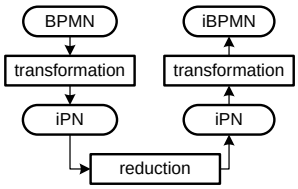


Figure 4. Overview

Figure 4 presents the overview on the approach. We start with a BPMN 1.2 model. This model is transformed directly into an interaction Petri net model. In this step, control flow structures are transformed using the approach presented in [26] and pairs of interaction activities are directly transformed to interaction transitions. The restriction on message flows in the input model ensures that this transformation is unambiguous. Tasks configured as loops are expanded to gateways surrounding the mapped content of the task.

The control flow surrounding the interaction transitions is not modified in the first step. To gain a proper interaction control flow, the resulting Petri net is reduced using the techniques of [24] and [25]. The reduced iPN model is then transformed into an iBPMN model.

In the following, we present the idea of the algorithm by transforming the interaction model presented in Fig. 2 into an iBPMN model. A detailed and formal description of the transformation is out of scope of this paper, but is presented in [28].

Figure 5 presents the interaction Petri net after mapping the BPMN choreography to iPNs. The nodes $p_0, t_0, p_1, t_1, p_2, t_2, t_7, p_3$ represent the financial advisor. p_2 in combination with t_2 and t_7 represents the event-based gateway. The nodes $p_4, t_3, p_5, t_1, p_6, t_4, p_7$, t_5, p_8, $t_2, t_6, p_9, t_7, p_{10}$ represent the customer. t_4 is the mapped timer event with the controlling role customer. We introduce an additional timer marking to controlled transitions to be able to correctly map timer events back to timer events. If the marker was not present, these transitions would have been mapped to a gateway. Finally, p_7, t_5, t_6 represent the data-based exclusive gateway.

A place p is redundant to a place $q \neq p$ if p is not marked in the initial marking and p is always marked if q is marked [25]. Thus, the analysis of the iPN leads to following results: 1) p_0, t_0 and p_4, t_3 can be removed, 2) then, p_1 and p_5 have no preceding transition and target the same transition. Hence, p_5 can be removed. 3) t_5, p_8 and t_6, p_9 can be removed, 4) p_2 is redundant to p_7 and can be removed, 5) p_3 is redundant to p_{10} and can be removed.

Result 4 cannot be gained by applying the structural reduction rules of [24] only, because the rules are only applicable for a transition between two places or a place between two transitions. The intermediary steps are not shown due to space limitations. The overall result is presented in Fig. 6.

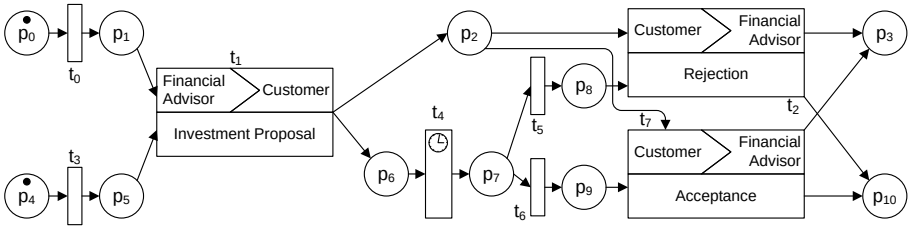


Figure 5. Interaction Petri net derived from the investment offer scenario

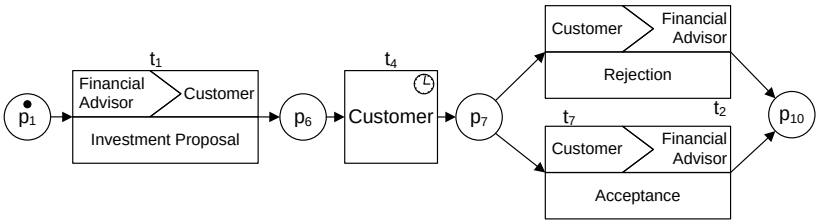


Figure 6. Reduced interaction Petri net

To map the reduced iPN to an iBPMN model, the iPN has to be modified to enable a pattern-based transformation. The modification ensures that all interaction transitions have exactly one incoming and one outgoing arc. If an interaction transition t_i has more than one incoming arc, a transition t is added before t_i and all arcs targeting t_i are retargeted to t and a new arc from t to t_i is added. A similar approach is taken for outgoing arcs. Now, parallel gateways are always generated out of a transition with multiple outgoing arcs.

iBPMN supports both event-based and data-based gateways to offer a choice between alternative message exchanges. iBPMN demands that an event-based gateway is used in the case a timer-event is involved in the current conversation and a data-based gateway in all other cases. In case the interaction transition is preceded by a place which is followed by multiple transitions, the place has to be transformed into a data-based gateway or to an event-based gateway depending on the type of the subsequent transitions. BPMN does not support mixed choices [17]. That means, the interactions following a place are always initiated by the same sender. In case a timer transition is present after the place, the place is transformed into an event-based gateway. Otherwise, the place is transformed into a data-based gateway with controlling *S*, where *S* is the sender in all interaction transitions following.

The remaining constructs are transformed by applying the rules of [2,3] "backward". That means, we map the constructs from iPN to iBPMN instead of mapping iBPMN to iPNs. The result is presented in Fig. 3.

4 Discussion of the Approach

Interconnection choreography models assume asynchronous communication, whereas interaction models assume synchronous communication. The latter means, the sender is blocked until the receiver consumes the message. In this paper, we assume that the asynchronous model is "synchronizable". Future work, however, has to provide a detailed investigation whether the synchronizability definitions given by Fu et al. [18] are applicable to interconnection BPMN models.

The approach uses Petri nets as intermediate format for the mapping. The approach cannot handle multi-instance loops without a priori knowledge of the number of instances. As the Petri net is not used for verification, the entry place of the loop can be labeled with a marker. That marking can later be used to transform the loop back to a multi-instance loop in iBPMN.

BPMN 1.2 does not foresee to mark participants as multi-instance participants. These extensions have been introduced in [29] to overcome this limitation. Thus, the start place of the participant can be marked if it is a multi-instance participant to enable a transformation to a multi-instance participant in iBPMN.

The idea of markings can also be used for OR joins. An OR join can be transformed into a transition marked with "OR". Then, the reduction part of the algorithm does not reduce that transition and the mapping from iPN to iBPMN transforms that transition to an OR gateway.

5 Conclusion and Outlook

This work presented a first mapping from BPMN to iBPMN using Petri nets as intermediate formalism. We mapped the positive control flow only and thus leaving the negative control flow for future work. As the algorithm was sketched, future work has to provide a formal presentation of the algorithm as well as a formal presentation of the properties the algorithm preserves and proofs of them.

Besides using Petri nets as intermediate formalism, it seems to be possible that BPMN can be directly mapped to iBPMN. The phases of that approach are: (i) transformation of pairs of messaging activities to one interaction activity and (ii) reduce the resulting iBPMN model. Thus, we currently investigate whether and how the reduction rules presented in [24, 25] can be applied to iBPMN models. Subsequently, we are planning a detailed investigation on the direct BPMN to iBPMN mapping and a comparison to the presented approach.

Acknowledgments We thank the anonymous reviewers for their valuable comments and new insights on the opportunities and limitations of the approach.

References

1. Decker, G., Kopp, O., Barros, A.: An Introduction to Service Choreographies. Information Technology **50**(2) (2008) 122–127
2. Decker, G., Weske, M.: Interaction-centric Modeling of Process Choreographies. (2010) in submission.
3. Decker, G.: Design and Analysis of Process Choreographies. PhD thesis, Hasso Plattner Institute, University of Potsdam (2009)
4. Bitsaki, M., et al.: An Architecture for Managing the Lifecycle of Business Goals for Partners in a Service Network. In: ServiceWave 2008, Springer (2008) 196–207
5. Object Management Group (OMG): Business Process Model and Notation (BPMN) Specification 2.0. (2009) v2.0 Beta 1.
6. Decker, G., Barros, A.P.: Interaction Modeling Using BPMN. In: 1st International Workshop on Collaborative Business Processes 2007, Springer (2007) 208–219
7. Engler, L.: BPEL^{gold}: Choreography on the Service Bus. Diploma thesis, University of Stuttgart, IAAS (2009)
8. Kopp, O., Leymann, F.: Do We Need Internal Behavior in Choreography Models? In: ZEUS 2009. Volume 438., CEUR-WS.org (2009) 68–73
9. Object Management Group (OMG): Business Process Modeling Notation (BPMN) Version 1.2. (January 2009) `http://www.bpmn.org/`.

10. Pfitzner, K., Decker, G., Kopp, O., Leymann, F.: Web Service Choreography Configurations for BPMN. In: WESOA 2007, Springer (2007) 401–412
11. Decker, G., Kopp, O., Leymann, F., Weske, M.: Interacting services: from specification to execution. Data & Knowledge Engineering **68**(10) (2009) 946–972
12. Wolf, K.: Does my service have partners? LNCS T. Petri Nets and Other Models of Concurrency **5460**(2) (2009) 152–171
13. Dijkman, R., Dumas, M.: Service-oriented Design: A Multi-viewpoint Approach. International Journal of Cooperative Information Systems **13**(4) (2004) 337–368
14. Barros, A., Decker, G., Dumas, M.: Multi-staged and multi-viewpoint service choreography modelling. In: SEMSOA 2007. (2007) 1–15
15. Motahari-Nezhad, H.R., et al.: Deriving Protocol Models from Imperfect Service Conversation Logs. IEEE Transactions on Knowledge and Data Engineering **20** (2008) 1683–1698
16. Benatallah, B., Casati, F., Toumani, F.: Representing, analysing and managing web service protocols. Data Knowl. Eng. **58**(3) (2006) 327–357
17. Decker, G., Barros, A.P., Kraft, F.M., Lohmann, N.: Non-desynchronizable Service Choreographies. In: ISCOC 2008. (2008) 331–346
18. Fu, X., Bultan, T., Su, J.: Synchronizability of Conversations among Web Services. IEEE Trans. Softw. Eng. **31**(12) (2005) 1042–1055
19. Decker, G., Weske, M.: Local Enforceability in Interaction Petri Nets. In: Business Process Management 2007, Springer (2007) 305–319
20. Fu, X., Bultan, T., Su, J.: Conversation protocols: a formalism for specification and verification of reactive electronic services. Theor. Comput. Sci. **328**(1-2) (2004) 19–37
21. Decker, G., Kopp, O., Leymann, F., Pfitzner, K., Weske, M.: Modeling Service Choreographies using BPMN and BPEL4Chor. In: CAiSE '08, Springer (2008) 79–93
22. Steinmetz, T.: Generierung einer BPEL4Chor-Beschreibung aus BPEL-Prozessen. Student Thesis, University of Stuttgart, IAAS (2007) in German.
23. Kopp, O., van Lessen, T., Nitzsche, J.: The Need for a Choreography-aware Service Bus. In: YR-SOC 2008. (2008) 28–34
24. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE **77**(4) (1989) 541–580
25. Berthelot, G., Lri-Iie: Checking properties of nets using transformations. In: Advances in Petri Nets 1985, Springer (1986) 19–40
26. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Information and Software Technology **50**(12) (2008) 1281–1294
27. Lohmann, N., Verbeek, E., Dijkman, R.: Petri Net Transformations for Business Processes – A Survey. In: ToPNoC, Springer (2009) 46–63
28. Wu, F.: Mapping interconnection choreography models to interaction models. Diploma thesis, University of Stuttgart, IAAS (2009)
29. Decker, G., Puhlmann, F.: Extending BPMN for Modeling Complex Choreographies. In: CoopIS 2007, Springer (2007) 24–40

Prozessumstrukturierung unter Berücksichtigung von Nachrichteninhalten

Thomas S. Heinze[1], Wolfram Amme[1], Simon Moser[2]

[1] Friedrich-Schiller-Universität Jena
{T.Heinze,Wolfram.Amme}@uni-jena.de
[2] IBM Entwicklungslabor Böblingen
smoser@de.ibm.com

1 Einführung und Motivation

Bei der informationstechnischen Umsetzung von Geschäftsprozessen mit Hilfe von verteilten, service-orientierten Architekturen existieren mehrere sowohl wissenschaftlich als auch industriell relevante Fragestellungen. Neben grundsätzlichen Fragen zur Korrektheit und Kompatibilität verteilter Prozesse spielt auch das Auffinden geeigneter Dienste zur Dienstkomposition eine wichtige Rolle. Grundlage einer solchen Dienstsuche bilden öffentliche Schnittstellenbeschreibungen, die zumeist aus einer Liste der implementierten Operationen der Dienste bestehen. Um Fehler bei der Dienstkomposition zu vermeiden, sollten insbesondere bei zustandsbehafteten Diensten (wie zum Beispiel Geschäftsprozessen) zusätzlich Informationen zu der Abfolge der Operationen vorhanden sein. Ein formaler Ansatz zur Modellierung und Analyse der durch einen (verteilten) Geschäftsprozess realisierten Abfolge von Operationen, in Form gesendeter und empfangener Nachrichten, wird in [6, 7] beschrieben. Darin wird das Prozessverhalten aus Sicht eines möglichen Partners wiedergegeben und auf diese Weise eine Art *Bedienungsanleitung* für den Prozess bereitgestellt. Diese kann dann zur Schnittstellenbeschreibung genutzt werden und so die Dienstsuche unterstützen [7].

Abbildung 1 zeigt das Fragment eines Geschäftsprozesses, in diesem Fall der Sprache WS-BPEL [8], zusammen mit der das zugehörige Verhalten beschreibenden Bedienungsanleitung. Die dargestellte Aktivität `BiddingSequence` setzt ein Veräußerungsverfahren um, bei dem in einer Schleife (`RepeatUntil`) fortlaufend eine Ausschreibung veröffentlicht wird (`BidRequest`), zu der ein Gebot abgegeben werden kann (Nachricht `Bid`). Die Schleife wird verlassen und das Verfahren mit einer Mitteilung beendet (`BidClosure`), falls ein Gebot den vordefinierten Mindestverkaufswert übersteigt. Darüber hinaus kann das Verfahren auch durch den Veräußerer vorzeitig abgebrochen werden (Nachricht `Abort`). Die Ausführung der Schleife wird dabei durch drei Variablen kontrolliert (`$break or $currentBid` $>$ `$threshold`). Zum einen enthalten `$currentBid` und `$threshold` Informationen zu dem zuletzt abgegebenen Gebot und dem Mindestverkaufswert (`1000`). Andererseits wird der vorzeitige Abbruch des Verfahrens durch `$break` gesteuert. Wie ebenfalls in Abbildung 1 dargestellt, ist die Bedienungsanleitung eines Geschäftsprozesses ein Automat [7]. In diesem

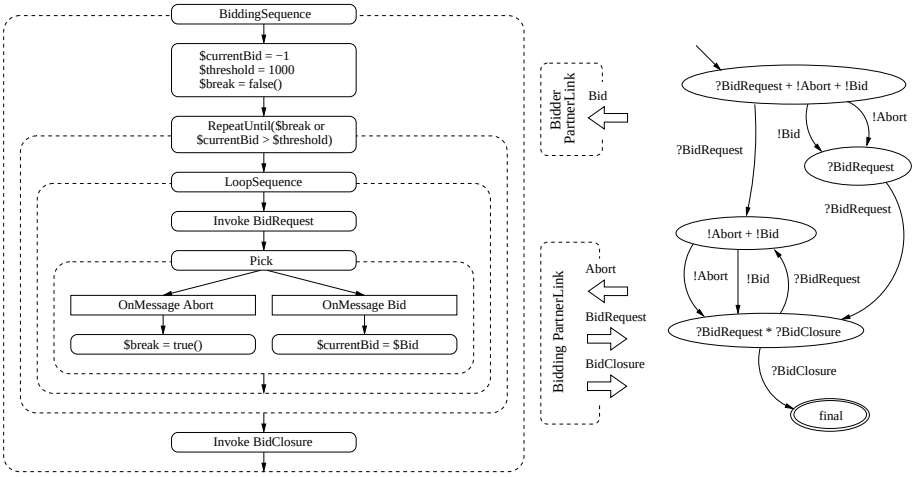


Abb. 1. `BiddingSequence` (links) und zugehörige Bedienungsanleitung (rechts)

repräsentiert ein Pfad eine mögliche Ausprägung des von außen sichtbaren Prozessverhaltens, wobei die Kanten des Automaten den gesendeten und empfangenen Nachrichten aus Sicht des jeweiligen Prozesspartners entsprechen. Zusätzlich finden sich Integritätsbedingungen an die möglichen Partnerprozesse in Form von Zustandsannotationen.[1] Wie leicht zu erkennen ist, beschreibt die angegebene Bedienungsanleitung jedoch nicht alle denkbaren Partner der Aktivität `BiddingSequence`. So sind beispielsweise Partnerprozesse, die zunächst mehrere Gebote (Nachricht `Bid`) senden, und erst im Anschluss die zugehörigen Ausschreibungen (`BidRequest`) empfangen, nicht mit erfasst. Auch ist das Ende der Veräußerung nach einem den Mindestverkaufswert übersteigenden Gebot oder einem vorzeitigen Abbruch (Nachricht `Abort`) nicht explizit wiedergegeben.

Die Begründung dafür ist in dem zur Ableitung der Bedienungsanleitung angewandten Verfahren zu suchen. Dieses beruht auf einer petrinetzbasierten Modellierung der Geschäftsprozesse, in der Prozessdaten zugunsten einer durchführbaren Analyse nicht wiedergegeben werden [5, 6]. Insbesondere wird der bedingte Kontrollfluss auf nichtdeterministische Konflikte abgebildet, und damit eine Abstraktion verwendet, die zu der skizzierten Ungenauigkeit in der abgeleiteten Bedienungsanleitung führen kann. In [4] haben wir eine *Umstrukturierungsmethode* vorgestellt, mit der sich bestimmte Arten des bedingten Kontrollflusses so transformieren lassen, dass Daten- in Kontrollabhängigkeiten umgewandelt werden. Im Ergebnis können die Datenabhängigkeiten, namentlich die Verzweigungs- oder Schleifenbedingungen, entfernt und somit die Anzahl der benötigten nichtdeterministischen Konflikte reduziert werden. Die Klasse der in [4] betrachteten Schleifen und Verzweigungen ist dabei durch Bedingungen charakterisiert, die zur Prozesslaufzeit nur auf durch Konstanten definierte Variablen zugreifen.

[1] Die durch Annotationen ausgeschlossenen Pfade der Bedienungsanleitung, dass heißt Pfade die zu fehlerhaftem Verhalten führen können, sind nicht mit abgebildet.

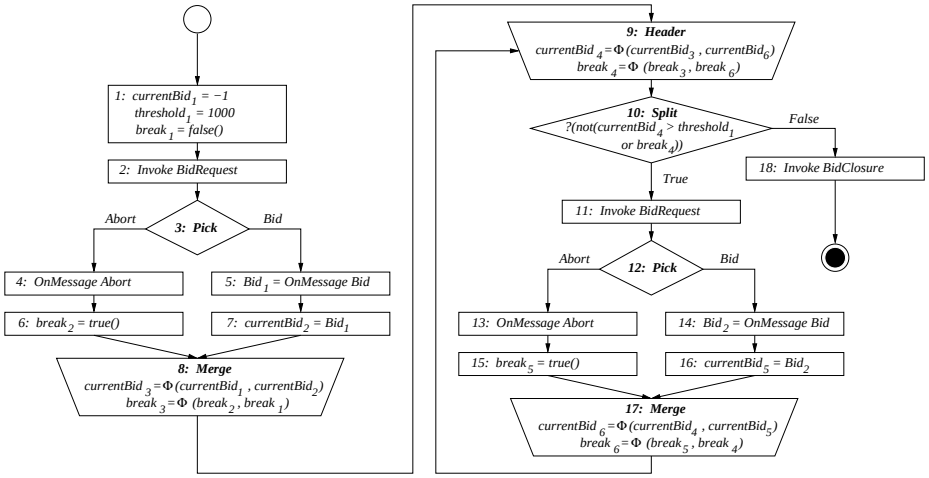


Abb. 2. Erweiterter Workflow-Graph für `BiddingSequence`

Im Folgenden soll ein Ansatz zur Erweiterung der Umstrukturierungsmethode am Beispiel `BiddingSequence` beschrieben werden, mit der sich auch solche Verzweigungs- und Schleifenbedingungen eliminieren lassen, deren Variablen zusätzlich zu Konstanten durch eingehende Nachrichten definiert sind.

2 Prozessumstrukturierung

Die Umstrukturierungsmethode basiert auf einer Prozessrepräsentation durch *erweiterte Workflow-Graphen* und ist damit grundsätzlich auf Prozesse aller Spezifikationssprachen für (strukturierte) Geschäftsprozesse anwendbar, zu denen eine Abbildung auf Workflow-Graphen existiert (außer für WS-BPEL existiert ein Abbildungsverfahren [1] für BPMN [9]). Erweiterte Workflow-Graphen erlauben zusätzlich zu der durch herkömmliche Workflow-Graphen unterstützten Abbildung des Kontrollflusses auch die Wiedergabe der Prozessdaten. Insbesondere lassen sich die während der Umstrukturierung benötigten Datenabhängigkeiten innerhalb dieses Repräsentationsformats auf einfache Weise identifizieren. In Abbildung 2 ist der erweiterte Workflow-Graph für das Beispiel `BiddingSequence` abgebildet.[2] Darin repräsentieren Knoten die Aktivitäten und Kanten verbinden diese entsprechend dem Kontrollfluss. Spezielle Knoten (*Pick*, *Merge*, *Split*, *Header*) dienen der Aufspaltung und Vereinigung des Kontrollflusses im Fall der ereignisgesteuerten Verzweigung (`Pick`) und der Schleife des Prozessfragments. Wie bereits erwähnt, erfolgt die Repräsentation der Prozessdaten derart, dass eine einfache Analyse der Datenabhängigkeiten möglich ist. Zu diesem Zweck wur-

[2] Aus technischen Gründen wurde die eigentlich fußgesteuerte Schleife (`RepeatUntil`) in `BiddingSequence` durch Herausziehen der ersten Iteration und Negierung der Schleifenbedingung in eine äquivalente kopfgesteuerte Schleife umgewandelt.

den die Variablen so umbenannt, das jede statische Variablendefinition einen eigenen Namen erhält (so $break_1, break_2, \ldots, break_6$ für `$break`). Weiterhin wurde eine Φ-Funktion zur Zusammenfassung konkurrierender Definitionen eingefügt, falls mehrere Definitionen einer Variablen auf verschiedenen Pfaden des Kontrollflusses zusammentreffen (vergleiche $break_4 = \Phi(break_3,\ break_6)$ in *Header*). Der Wert einer solchen Funktion entspricht gerade der Definition, gegeben als Operand, die zur Laufzeit ausgeführt wurde.

Ausgehend von dieser Prozessrepräsentation lassen sich die Datenabhängigkeiten des bedingten Kontrollflusses analysieren. Für `BiddingSequence` fällt dabei auf, dass die Bedingung der darin enthaltenen Schleife nur auf Variablen zugreift, die entweder allein durch Konstanten (`$break`, `$threshold`) oder zusätzlich durch Nachrichten (`$currentBid`) definiert sind. Im ersteren Fall wird von *statisch quasi-konstanten Variablen* gesprochen, und für Bedingungen die nur auf diese Art von Variablen zugreifen von *statisch quasi-konstanten Bedingungen*. Der Wert einer solchen Bedingung hängt nur vom zur Prozesslaufzeit gewählten Kontrollflusspfad ab, da jeder Pfad zur Bedingung einer Belegung der darin vorkommenden Variablen mit Konstanten entspricht. Daher kann die Bedingung durch Einfügen geeigneter Kontrollabhängigkeiten ersetzt werden.

Die Umstrukturierungsmethode zur Erzeugung der Kontrollabhängigkeiten erfolgt in zwei Schritten. Zunächst wird eine Schleife oder Verzweigung mit statisch quasi-konstanter Bedingung in eine *Normalform* überführt. Diese zeichnet sich dadurch aus, das alle (statischen) Pfade des Kontrollflusses, die für eine Bedingungsvariable unterschiedliche Werte definieren, aufgetrennt sind. Zu diesem Zweck werden, mit Ausnahme des Kopfs einer Schleife, alle Knoten in denen unterschiedliche Definitionen aufeinandertreffen nacheinander aufgelöst. Im Anschluss kann für eine Verzweigung die Verzweigungsbedingung auf allen Pfaden ausgewertet und durch unbedingte Sprünge zu den Verzweigungszielen ersetzt werden. Für eine Schleife laufen nun hingegen alle Definitionen für Bedingungsvariablen im Schleifenkopf zusammen. Um auch diese aufzutrennen, wird die Schleife im folgenden Schritt durch mehrere *Schleifeninstanzen* ersetzt. Jede *Instanz* repräsentiert dabei die Ausführung des Schleifenrumpfs für eine mögliche Belegung der Bedingungsvariablen mit Konstanten. Demnach können die Schleifenbedingungen in den Instanzen ebenfalls ausgewertet, und durch unbedingte Sprünge zum Austrittsknoten der Schleife oder zu weiteren Instanzen ersetzt werden. Eine ausführliche Beschreibung der Methode ist in [4] zu finden.

3 Erweiterte Umstrukturierung

Nachstehend wird die Erweiterung der Umstrukturierungsmethode beschrieben. Diese soll auch die Elimination von Bedingungen erlauben, in denen Variablen durch eingehende Nachrichten definiert sind. Die Bedingung der Schleife in `BiddingSequence` enthält, neben den statisch quasi-konstanten Variablen `$break` und `$threshold`, auch eine solche Variable (`$currentBid`). Diese Art von Variablen, und Bedingungen die nur auf diese Variablen zugreifen, werden als *dynamisch quasi-konstant* bezeichnet. Um die Umstrukturierungsmethode

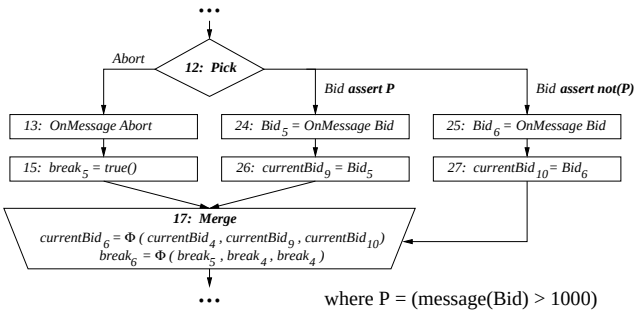


Abb. 3. Verfeinerte Abbildung der die Variable `$currentBid` definierenden Nachricht

auch auf Schleifen und Verzweigungen mit derartigen Bedingungen anwenden zu können, wird ein abstrakterer Instanzenbegriff benötigt. Der bisher verwendete Begriff beschreibt die Ausführung des Rumpfs einer Schleife bezüglich einer Belegung der Bedingungsvariablen mit Konstanten. Im Folgenden verstehen wir unter einer Schleifeninstanz die Ausführung des Schleifenrumpfs bezüglich einer allgemeinen *Zusicherung* für den Zustandsraum der Bedingungsvariablen. Zweifelsohne ist die erstere Begriffsbildung ein Spezialfall der letzteren.

Die Zusicherungen werden dabei aus der betrachteten Bedingung abgeleitet. Dazu wird diese zunächst in eine Klauselform transformiert, aus der die Grundprädikate der Bedingung bestimmt werden können, so ($currentBid_4 > threshold_1$) und ($break_4$) für die Bedingung in `BiddingSequence`. Zwei Typen von Prädikaten müssen unterschieden werden. Zum einen können Prädikate nur auf statisch quasi-konstante Variablen zugreifen ($break_4$). In diesen Fällen werden, wie bisher, Belegungen der Variablen mit Konstanten als Zusicherungen in den Instanzen verwendet. Davon zu trennen sind Prädikate, in denen auch dynamisch quasi-konstante Variablen vorkommen ($currentBid_4 > threshold_1$), da nun das Prädikat auch vom Inhalt eingehender Nachrichten abhängig ist. Um die dadurch gegebenen Datenabhängigkeiten mittels Kontrollabhängigkeiten repräsentieren zu können, werden Nachrichten nicht mehr nur als Ereignisse interpretiert, sondern zusätzlich Klassen möglicher Nachrichteninhalte unterschieden.[3] Dies erlaubt die Definition einer Variablen durch eine Nachricht mit Zusicherungen für die möglichen Werte der Nachricht zu verfeinern. Im erweiterten Workflow-Graphen werden dann die Prädikate mit dynamisch quasi-konstanten Variablen auf Zusicherungen für die diese Variablen definierenden Nachrichten zurückgeführt. In Abbildung 3 ist das Ergebnis für die Definition der Variablen `$currentBid` innerhalb der Schleife aus `BiddingSequence` angegeben. Wie zu erkennen, ist die ursprüngliche Definition der Variablen (Knoten 16 in Abbildung 2) dupliziert und durch zwei komplementäre Zusicherungen an die jeweils definierende Nachricht `Bid` ergänzt worden ($assert\ P$, $assert\ not(P)$). Die Zu-

[3] Die explizite Repräsentation von Nachrichteninhalten wurde bereits in [5] vorgeschlagen, basierend auf der Entfaltung höherer Petrinetze. Dies erfordert jedoch das Vorliegen von Bedingungen und Nachrichten mit endlichen Datenbereichen.

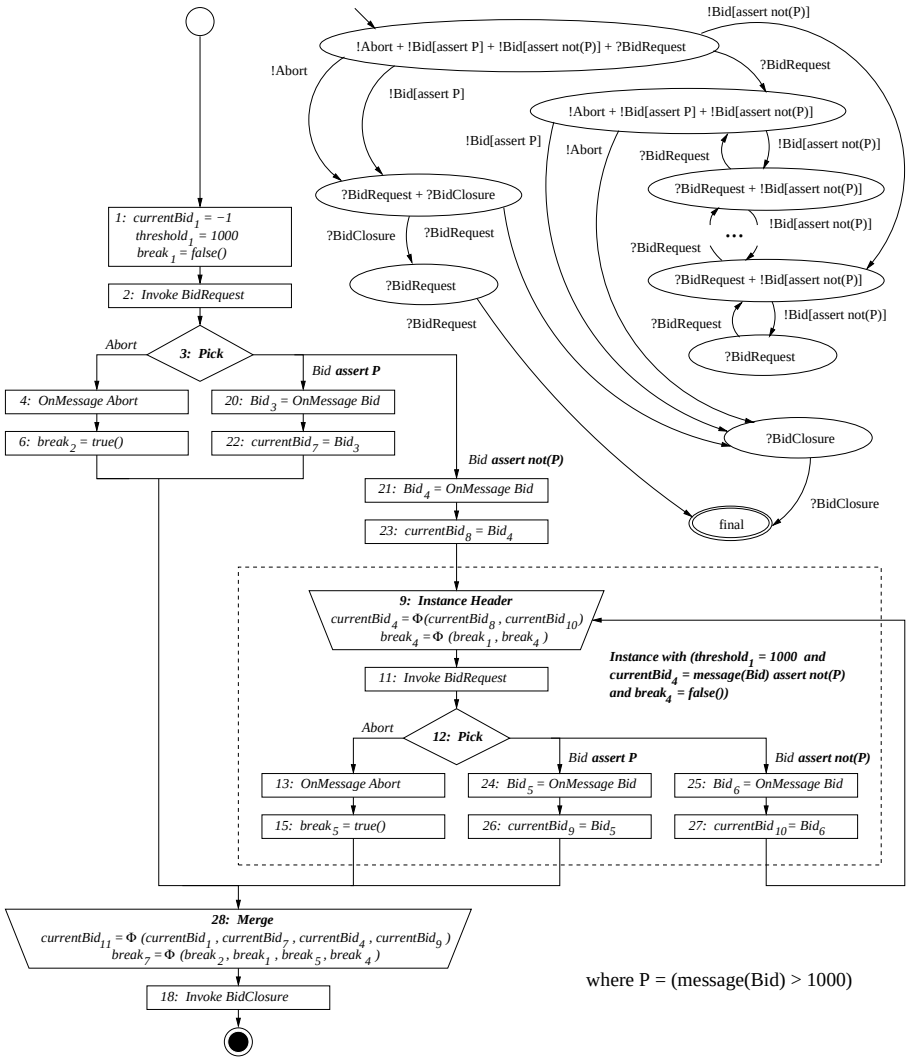


Abb. 4. Umstrukturierter Workflow-Graph und zugehörige Bedienungsanleitung

sicherungen entsprechen dabei der positiven und negativen Variante des die Variable `$currentBid` enthaltenen Prädikats der Schleifenbedingung, in denen die Variablen durch ihre Definitionen ersetzt wurden ($P = message(Bid) > 1000$).

Während der Umstrukturierung können die damit eingeführten Zusicherungen für relevante Nachrichten zur Beschreibung der Schleifeninstanzen genutzt werden. Das Resultat der so erweiterten Umstrukturierung ist für die Schleife aus `BiddingSequence` in Abbildung 4 dargestellt. Darin wurde lediglich eine einzige Instanz mit den Zusicherungen $break_4 = false()$, $threshold_1 = 1000$ und $currentBid_4 = message(Bid)\ assert\ not(P)$ erzeugt, da die Schleifenbe-

dingung nur für diese Zusicherungen erfüllt ist. Abbildung 4 zeigt ebenfalls die aus dem derart umstrukturierten Workflow-Graphen abgeleitete Bedienungsanleitung. Im Vergleich zur ursprünglichen Bedienungsanleitung aus Abbildung 1 sind darin nun auch solche Partner erfasst, die zwischen zwei Geboten (`Bid`) nicht eigens auf eine zugehörige Ausschreibung (Nachricht `BidRequest`) warten. Zudem sind die zwei Alternativen zum Beenden des Veräußerungsverfahrens nun explizit wiedergegeben. So kann ein Partner das Verfahren zum einen durch Senden der Nachricht `Abort` abbrechen (vergleiche die durch *!Abort* erreichbaren Zustände). Andererseits ist, durch die eingefügte Unterscheidung möglicher Inhalte der Nachricht `Bid` (*!Bid[assert message(Bid)>1000]*, *!Bid[assert not(message(Bid)>1000)]*), zudem auch das Ende nach einem den Mindestverkaufswert übersteigenden Gebot des Partners repräsentiert.

4 Diskussion und Zusammenfassung

In der vorliegenden Arbeit beschreiben wir einen Ansatz zur Erweiterung der in [4] vorgestellten Umstrukturierungsmethode für verteilte Geschäftsprozesse. Diese erlaubt bestimmte Schleifen und Verzweigungen so zu transformieren, dass deren Bedingungen eliminiert werden können. Auf diese Weise lassen sich die petrinetzbasierten Prozessmodelle bestehender Analysen präzisieren und dadurch genauere Analyseergebnisse erzielen. In [4] werden Bedingungen betrachtet, die nur auf durch Konstanten definierte Variablen zugreifen. Durch die Verwendung eines abstrakteren Begriffs der Schleifeninstanz und die Berücksichtigung von Nachrichteninhalten können wir die Methode nun auch auf Bedingungen anwenden, deren Variablen zusätzlich durch Nachrichten definiert sind.

Im selben Moment stellt sich aber auch die Frage nach der Notwendigkeit einer Präzisierung der Prozessmodelle. So bildet die in Abbildung 1 dargestellte Bedienungsanleitung bereits eine korrekte, wenn auch unvollständige, Beschreibung des von außen sichtbaren Verhaltens der Aktivität `BiddingSequence`. Daher kann der Nutzen einer präziseren Bedienungsanleitung infrage gestellt werden, insbesondere im Hinblick auf die darin veröffentlichte, möglicherweise vertrauliche, Information zum Mindestverkaufswert. Anders gestaltet sich die Situation jedoch, falls ein leicht modifizierter Prozess betrachtet wird. Ein Prozess, der beispielsweise keine Ausschreibungen sendet, dass heißt die Aktivität `Invoke BidRequest` nicht enthält, dafür aber jedes Gebot mit einer entsprechenden Nachricht bestätigt, sollte ebenfalls durch eine Bedienungsanleitung beschrieben werden können. Die Ableitung einer Bedienungsanleitung ist aber in diesem Fall nicht mehr möglich. Stattdessen kann aufgrund der nichtdeterministischen Abbildung der Schleifenbedingung unter Verwendung des herkömmlichen petrinetzbasierten Prozessmodells kein nicht verklemmender Partnerprozess gefunden werden. Eine Präzisierung des Prozessmodells, etwa durch Anwendung der vorgestellten Umstrukturierungsmethode, ist zur Ableitung einer Bedienungsanleitung für diesen Prozess zwingend erforderlich. Gleichzeitig muss diese nun auch die Information zum Mindestverkaufswert der Veräußerung bereitstellen.

In weiterführenden Arbeiten wollen wir den vorgestellten Ansatz zunächst evaluieren. Dazu soll sowohl eine Implementierung als auch eine Sammlung realistischer Beispielprozesse zur Beurteilung der praktischen Relevanz der Umstrukturierungsmethode verwirklicht werden. In diesem Zusammenhang stellt sich auch die Frage nach der Abgrenzung zu vergleichbaren Ansätzen der Prädikatenabstraktion [3]. Darüber hinaus sind wir, wie bereits in [4] angedeutet, an der schrittweisen Ausweitung des Ansatzes auf andere statisch auswertbare Bedingungen interessiert. Zu diesem Zweck erscheint uns insbesondere der hier eingeführte allgemeinere Begriff der Schleifeninstanz ein geeignetes Mittel. Durch eine präzisere Analyse der Datenabhängigkeiten von Schleifen- und Verzweigungsbedingungen ist die Ableitung entsprechender Zusicherungen denkbar. Als eine mögliche Analyse tritt dabei die in der Arbeit [2] beschriebene abstrakte Interpretation der in WS-BPEL-Prozessen genutzten XPath-Ausdrücke zur Abschätzung der Wertebereiche von Variablen hervor.

Literatur

[1] Favre, Cédric: *Algorithmic verification of business process models*, École Polytechnique Fédérale de Lausanne, Master's Thesis, 2008

[2] Görlach, Katharina: *Ein Verfahren zur abstrakten Interpretation von XPath-Ausdrücken in BPEL-Prozessen*, Humboldt-Universität zu Berlin, Diplomarbeit, 2008

[3] Graf, Susanne ; Saidi, Hassen: Construction of Abstract State Graphs with PVS. In: Grumberg, Orna (Hrsg.): *Computer Aided Verification, 9th International Conference, CAV'97, Haifa, Israel, June 22-25, 1997, Proceedings*, Springer-Verlag, 1997 (Lecture Notes in Computer Science 1254), S. 72–83

[4] Heinze, Thomas S. ; Amme, Wolfram ; Moser, Simon: A Restructuring Method for WS-BPEL Business Processes Based on Extended Workflow Graphs. In: Dayal, Umeshwar (Hrsg.) ; Eder, Johann (Hrsg.) ; Koehler, Jana (Hrsg.) ; Reijers, Hajo A. (Hrsg.): *Business Process Management, 7th International Conference, BPM 2009, Ulm, Germany, September 8-10, 2009, Proceedings*, Springer-Verlag, 2009 (Lecture Notes in Computer Science 5701), S. 211–228

[5] Lohmann, Niels: A Feature-Complete Petri Net Semantics for WS-BPEL 2.0. In: Dumas, Marlon (Hrsg.) ; Heckel, Reiko (Hrsg.): *Web Services and Formal Methods, 4th International Workshop, WS-FM 2007, Brisbane, Australia, September 28-29, 2007*, Springer-Verlag, 2007 (Lecture Notes in Computer Science 4937), S. 77–91

[6] Lohmann, Niels ; Massuthe, Peter ; Stahl, Christian ; Weinberg, Daniela: Analyzing Interacting WS-BPEL Processes Using Flexible Model Generation. In: *Data & Knowledge Engineering* 64 (2008), Nr. 1, S. 38–54

[7] Lohmann, Niels ; Massuthe, Peter ; Wolf, Karsten: Operating Guidelines for Finite-State Services. In: Kleijn, Jetty (Hrsg.) ; Yakovlev, Alex (Hrsg.): *Petri Nets and Other Models of Concurrency - ICATPN 2007, 28th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency, ICATPN 2007, Siedlce, Poland, June 25-29, 2007, Proceedings*, Springer-Verlag, 2007 (Lecture Notes in Computer Science 4546), S. 321–341

[8] *Web Services Business Process Execution Language Version 2.0.* OASIS Standard, Organization for the Advancement of Structured Information Standards, 2007

[9] *Business Process Model and Notation (BPMN) Version 2.0.* OMG Standard, Object Management Group / Business Process Management Initiative, 2009

Research Challenges on Person-centric Flows*

Tobias Unger, Hanna Eberle, and Frank Leymann

Institute of Architecture of Application Systems
University of Stuttgart, Universitätsstraße 38, 70569 Stuttgart, Germany
lastname@iaas.uni-stuttgart.de

Abstract Research in the domain of Workflow Management focuses increasingly on service orchestrations. Often the fact is neglected that a huge part of the activities of business processes are performed by people. Especially, in the domain of pervasive computing processes are describing sequences of real world activities which are invariably performed by people. Therefore we consider the role of people participating in workflows from a new perspective. The basic idea of this work is to transfer the workflow metaphor to people processing their tasks. Therefore, we introduce the concept of a person-centric flow, which denotes such an implicit flow scheduled and executed by a single person. Secondly, we provide a list on research challenges on person-centric flows.

1 Introduction

Most research in business process management focuses on the aspects of service orchestration. The fact, that in certain types of businesses a business process consists of many manual activities, has not been recognized as a major research item, yet. Manual activities are carried out by people. We call those activities in the following tasks to differentiate them from the other types of activities, such as an invoke activity that requests the execution of a Web Service. For example, pervasive computing processes are typically describing sequences of activities which are performed by people. People are involved in many business processes at the same time, which requires appropriate support from the underlying IT infrastructure to help organize the work they need to carry out. Among many other functions such a infrastructure requires capabilities for work prioritization, context-sensitive assistance, or the capability of structuring a task by dividing the task into smaller tasks and ordering them as required by the user. Several specifications such as [1] and WS-Humantask [2] have been developed to address the integration of people into the world of services. We claim that those specifications only partially provide the required support and that additional capabilities must be provided by workflow management system to better integrate people in the execution of business processes.

* This work is partially funded by the ALLOW project. ALLOW (http://www.allow-project.eu/) is part of the EU 7th Framework Programme (contract no. FP7-213339).

Since the cognitive capabilities of people do not allow to carry out many tasks in parallel, people order the different tasks in a way that is convenient for them to process the different tasks. Current state of the art in workflow management typically provide the notion of work lists (cf. [3]), which the user can sort in any desired way using the properties that are associated with each task. Incidentally [4] shows that the efficiency of users highly depends on the capability for individual ordering of the work list. A disadvantage of larger work lists is the increased probability of missing the schedule for a task.

Studies have shown that people not only use the basic ordering of the work lists to select the next set of tasks, but also use some correlations or similar context information between the different tasks to select them [4]. This is similar to traditional workflow paradigm where the execution of the different activities is driven through the context that is associated with a process instance. We propose in this paper to order the individual tasks via process templates that are dynamically generated from the available tasks via the user preferences and automatically instantiated.

We introduce the concept of a person-centric flow, which denotes such an implicit flow scheduled and executed by a single person. Basically, a person-centric flow consists of the tasks on a person's worklist extended by additional ordering information. However, the knowledge about the existence of such a implicit person-centric flow cannot be utilized. Thus, making the person-centric flow explicitly visible and observable opens great opportunities to support people in doing their work. For example recommendations on the task ordering can be made which helps people e.g. to save resources or time. The goal of this work is to list research challenges on person-centric flows, their modeling, and execution. The aim of person-centric flows is not to prescribe people an ordering for performing their tasks. In fact, we want to support people e.g. by calling their attention to possible errors or to the fact, that the chosen ordering is prohibited. Another advantage of person-centric flows is the possibility to adapt the environment to needs of a future task. This is particularly useful in the case of actions that need time consuming preparation (cf. [5]) For example the filling of the bathtub can be started in time, as soon as it is known when a nurse will start washing a patient. For simplification reasons this work assumes that all tasks are generated by a Workflow Management System (WfMS).

The outline of this paper is as follows: Section 2 presents a scenario which is used to explain some issues of person-centric flows. A brief definition of person-centric flows is presented in Section 3. Section 4 lists the research challenges concerning person-centric flows and related approaches. Finally, Section 5 concludes the paper.

2 Scenario

Our scenario is situated in the domain of Healthcare (cf. [6]). Figure 1 shows simplified versions of the workflows executed each morning for each patient. All steps of the two workflows have to be done by a nurse and all steps of the

medication flow has to be done by the same nurse. Imagine there are two patients Frank and Tobias situated in two different rooms and one nurse Hanna. Imagine the situation where Hanna has executed the *prepare medicine* and *fetch medicine* task for both patients. As a consequence, Hanna has four tasks to perform which all appear on her worklist. Her person-centric flow could be first to medicate and wash Frank and afterwards to medicate and wash Tobias. In another situation she may decide to medicate both patients before washing them. This can be considered as an adaptation of her flow.

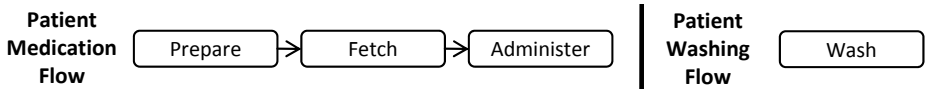


Figure 1: Secenario Flows

The scenario shows that a person-centric flow does not just follow a static flow model. It is highly dynamic due to context and changes of the worklist. Hanna implicitly forms her flow according to her actual situation. Her flow is not visible to the outside completely. Only the history of the flow is visible. Secondly, while executing a *prepare task* of an instance of the medication flow, by experience she schedules a fetch and a medication task even if these task are actually not on her worklist yet. By experience she knows that the a fetch medicine task is followed by a medication task. Hence, she considers the medication task right at the moment of scheduling her tasks, although only the fetch medication task is active.

3 Person-centric Flows

The aim of this section is to provide a first informal definition of the term and concept of person-centric flows. Later on the definition serves as foundation in order to identify research challenges. As mentioned before all tasks arise from flow instances executed by a WfMS which also provides a person's worklist. In the second part of this section we present the types of person-centric flows we identified during our research.

3.1 Definition

Definition 1 (Person-centric flow). *A person-centric flow defines a partial ordering over a set of tasks which have to be performed by one single person. Tasks of a person-centric flow can be classified in three tenses: past tasks, present tasks, and future tasks. Past tasks are completed either correct or incorrect and their ordering is known. Present tasks are tasks currently present at a person's worklist. Their ordering is planned by the person but can change dynamically. Future tasks are tasks which are assumed to be executed in the future. Both the set of future task as well as their ordering may change dynamically.*

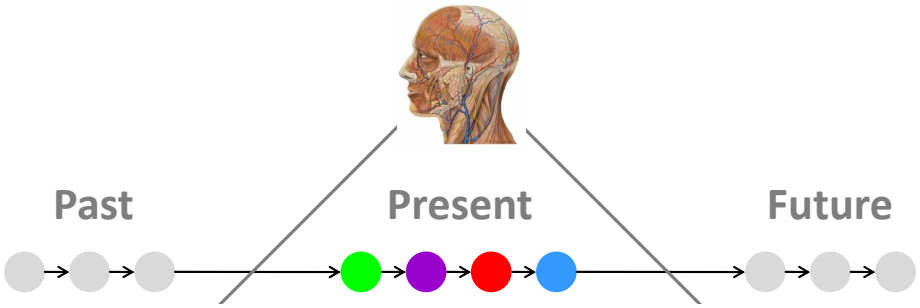


Figure 2: Person-centric Flow[1]

Figure 2 shows a simple graphical representation of a person-centric flow. Please note that present and future tasks can be mixed up within a person-centric flow. Only the actual executed task must be included in the set of present tasks. Since a person-centric flow is formed implicitly in a person's mind it must be predicted by the WfMS in order to utilize the ordering information. A person cannot be demanded to tell the WfMS its actual flow. Since tasks appear and disappear in a high frequency, this would be an additional stress factor. As a consequence predictions may be wrong. Furthermore, the person-centric flow paradigm is partly contrary to the existing workflow paradigm. For example a person-centric flow has no prescribed flow model since the set of tasks changes dynamically and consequently the ordering of the tasks. Many control patterns like loops are not needed in person-centric flows since each task is executed once. There also exists a single flow instance associated with exactly one person.

3.2 Classes of Person-centric Flows

During our research we recognized the following classes of person-centric flows:

Predicted flow: This is a predicted flow which corresponds to a person's flow with a high probability. Such flows are used for example to adapt the environment of a person. If washing a patient is supposed to be a nurse's next task the light in the shower could be switched on automatically. Regarding our scenario the knowledge of the nurse's flow helps to decide which patient will be washed first and consequently in which shower the light has to be switched on.

Guidance flow: A guidance flow is based on a predicted flow. However, a guidance flow may be extended by external ordering constraints. The aim of a guidance flow is to guide people actively through their flow. For example a constraint demands that a nurse must disinfect her hands between washing two patients. If she decides to perform both washing tasks without disinfecting her hands between, the constraint will be violated and the system will inform her.

[1] The figure uses an illustration by Patrick J. Lynch, medical illustrator and C. Carl Jaffe, MD, cardiologist.

Enforcement flow: An enforcement flow orders the tasks of a patient only based on external constraints. In this case the person-centric flow of a person is given by the environment. For example the fact that the medication must take place before washing can be enforced by an enforcement flow and the system can drive the person accordingly. A major disadvantage of this kind of person-centric flow is that it deprives people of the freedom to decide in which order they want to perform their tasks.

However, there is no clear distinction between the three classes of person-centric flows. A flow can comprise enforced as well as predicted as well as guided parts. Mostly, only a meaningful scenario-based combination of the three classes facilitates to reduce the cognitive load of a person necessary for scheduling task while retaining a high degree of freedom in scheduling tasks. Sometimes the characteristic of a person-centric flow may change over time. While a unexperienced nurse should be guided, with increasing experience the guidance should be reduced. As of that time the person-centric flow is only used for adapting the environment.

4 Research Challenges

In this section we present the identified research challenges on person-centric flows derived from the scenario analysis. The list is not exhaustive. Analyzing other scenarios may identify further issues.

Relation between implicit and explicit flow: As mentioned earlier a person-centric flow is implicit created by a person. In order to utilize the person-centric flow a explicit representation must be created. Like a hidden markov chain the observable behavior of the explicit flow may differ from the hidden implicit flow. Anyhow, the relation between these two flows must be described. The relation may be defined by describing the possible deviations of the predicted person-centric flow compared to the person-centric flow a person has in mind. Such an approach for modeling process variants is presented in [7]. Additionally, the predicted flow can be annotated with a maximum likelihood of its occurrence.

Rethinking the notion of flow model and instance: A person-centric flow has no predefined flow model. Since tasks emerge and disappear continuously, the person-centric flow has to be adapted very often. Thus, a language has to be defined that can deal with such a high degree of flexibility. Furthermore, the notion of an instance must be rethought. An instance of a common flow model has a defined beginning and a defined end. In contrast to traditional workflows a person-centric flow has no defined beginning and end. People are continuously performing tasks. Although there might be gaps in the flow where no tasks are performed.

Imperative vs. declarative process modeling languages: Basically, two types of workflow languages exist: imperative languages (cf. e.g. [3]) and declarative languages (cf. e.g. [8]). While imperative languages are assumed to be easier to understand, declarative languages promise a higher degree of flexibility. Furthermore, different orderings of the same set of tasks are hard to model

using imperative languages. In this work we use [9, 10] as starting points for our discussion on declarative and imperative workflow modeling languages, since the authors aim to contribute to a rigorous, theoretical discussion of this topic. Since high flexibility is a requirements for person-centric flows we decided to use a declarative language for the first attemp to describe person-centric flows (cf. [11]). The language is based upon the interval relations of Allen's interval algebra [12]. In contrast to existing declarative languages (cf. [8, 13, 14]) we extended the constraints of our language by time parameters to add support for temporal restrictions. To validate the consistency of the relation constraints we follow an approach similar to the one suggested in [15]. Also case handling [16] provides more flexibility in executing workflows. In case handling the control-flow related information does not drive the process but the state of the process, i.e. the existence of data objects. Since our approach focuses on the tasks themselves and not on the state of a data object, case-handling is inappropriate for modeling person-centric flows.

Algorithms for prediction: Since we are not able to capture the person-centric flow a person has in mind, we need to find algorithms to determine the task orderings. Especially history based algorithms are promising since people often have behavior patterns which can be detected by history-based algorithms. But also algorithms operating e.g. on task deadlines without considering the history (e.g. scheduling algorithms) seem to be useful in certain scenarios. There are many approaches which can be used as a starting point. In [17] the authors present an approach, which can be used for predicting a person's next steps based on the flow history. The recommendation in this approach is made for a single process instance and doesn't consider the interleaving of different process instances. In [18] scheduling algorithms are used to order worklists. Such an approach is suitable, if the person-centric flow should be enforced. [19] provides an approach which can be used for mining behavior patterns of people. Algorithms for predicting a person's next location are presented in [20]. The latter two approaches can be used, if there are recurring behavior patterns. However, they have to be adapted in order to operate on workflow histories.

Granularity of tasks: Tasks in business process are often modeled on a higher level than they will be actually executed by the people. Hence, people divide tasks in a set of subtasks. For example the washing of a patient is divided in several sub-tasks by a nurse. Algorithms for predicting person-centric flows also have to consider the fact that people re-structure their tasks. In [21] a case study is presented showing how people re-structure tasks.

Parallelism of human actions: It has to be discussed whether it is reasonable to allow parallelism in person-centric flows. Hence, granularity implies that if task are divided in sub-tasks which might be executed interleaving with subtasks of other tasks urge for parallelism capabilities.

Visualization: If person-centric flows are made explicit they should also be visualized in order to present them to a person, e.g. in the case that the personal flow is a guidance flow. However, meaningful representations must be found. For example, the person-centric flow of a nurse may be integrated within a map of

the hospital. Such a scenario may be realized based on an approach for workitem visualization as presented in [22]. In [23] a workflow visualization framework is presented, which enables a flexible business-oriented process visualization (e.g. using a gant chart). This approach can also serve as foundation for a visualization discussion on person-centric flows.

Privacy: Person-centric flows contain a lot information about the specific behavior of a person, which should be protected. Hence, discussing privacy is an important issue. [24] gives an overview on the effects of combining workflow systems and context-aware systems and to discuss its implications for workplace privacy and human-oriented design.

5 Conclusion

In this paper we introduce the concept of person-centric flows. Since person-centric flows are orthogonal to the existing workflow concept the two concepts complement each other. The main goal of person-centric flows is to support a person performing her tasks for example by reducing the cognitive load necessary for scheduling their tasks. However, person-centric flows can also be used to enforce external constraints. Initially, we will use the declarative workflow modeling language proposed in [11] to representing person-centric flows. In future work, we investigate and implement a prediction algorithm to predict person-centric flows prototypically. The aim of the prototype is to guide unskilled nurses. The prototype is also used to improve the accuracy of activity sensing and context recognition. In that case the person-centric flow reduces the search space of the activity recognition system since two activities having the same characteristics of sensor data may be distinguished according to their position in the person-centric flow.

References

1. OASIS: WS-BPEL Extension for People Specification Version 1.1, Committee Draft 06. (2009)
2. OASIS: Web Services Human Task Specification Version 1.1, Committee Draft 06. (2009)
3. Leymann, F., Roller, D.: Production Workflow – Concepts and Techniques. Prentice Hall PTR (2000)
4. Card, S.K., Newell, A., Moran, T.P.: The Psychology of Human-Computer Interaction. L. Erlbaum Associates Inc., Hillsdale, NJ, USA (1983)
5. Nurmi, P., Martin, M., Flanagan, J.A.: Enabling Proactiveness through Context Prediction. In: In: CAPS 2005, Workshop on Context Awareness for Proactive Systems. (2005)
6. ALLOW: D3.1: Results of Scenario Analysis. (2009) Available online (`http://www.allow-project.eu/`).
7. Hallerbach, A., Bauer, T., Reichert, M.: Issues in Modeling Process Variants with Provop. In Ardagna, D., Mecella, M., Yang, J., eds.: Business Process Management Workshops. Volume 17 of Lecture Notes in Business Information Processing., Springer (2008) 56–67

8. van der Aalst, W.M.P., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between Flexibility and Support. Computer Science - R&D **23**(2) (2009) 99–113
9. Fahland, D., Mendling, J., Reijers, H., Weber, B., Weidlich, M., Zugal, S.: Declarative vs. Imperative Process Modeling Languages: The Issue of Maintainability. In: 1st International Workshop on Empirical Research in Business Process Management (ER-BPM'09), Ulm, Germany (September 2009) 65–76
10. Fahland, D., Lübke, D., Mendling, J., Reijers, H., Weber, B., Weidlich, M., Zugal, S.: Declarative versus Imperative Process Modeling Languages: The Issue of Understandability. In: Proceedings of the 14th International Conference on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD'09), Amsterdam, The Netherlands, Springer-Verlag (June 2009) 353–366
11. Wagner, S.: A Concept of Human-oriented Workflows. Diploma thesis, University of Stuttgart, Germany (January 2010)
12. Allen, J.F.: Maintaining Knowledge about Temporal Intervals. Commun. ACM **26**(11) (1983) 832–843
13. Pesic, M.: Constraint-Based Workflow Management Systems: Shifting Control to Users. PhD thesis, Eindhoven University of Technology (2008)
14. Pesic, M., Schonenberg, M.H., Sidorova, N., van der Aalst, W.M.P.: Constraint-Based Workflow Models: Change Made Easy. In: OTM Conferences (1). (2007)
15. Lu, R., Sadiq, S.W., Padmanabhan, V., Governatori, G.: Using a temporal constraint network for business process execution. In: ADC. (2006) 157–166
16. van der Aalst, W.M.P., Weske, M., Grünbauer, D.: Case handling: a new paradigm for business process support. Data Knowl. Eng. **53**(2) (2005) 129–162
17. Schonenberg, H., Weber, B., van Dongen, B.F., van der Aalst, W.M.P.: Supporting Flexible Processes through Recommendations Based on History. [25] 51–66
18. Han, R., Liu, Y., Wen, L., Wang, J.: A Two-Stage Probabilistic Approach to Manage Personal Worklist in Workflow Management Systems. In Meersman, R., Dillon, T.S., Herrero, P., eds.: OTM Conferences (1). Volume 5870 of Lecture Notes in Computer Science., Springer (2009) 24–41
19. Vrotsou, K., Ellegard, K., Cooper, M.: Everyday Life Discoveries: Mining and Visualizing Activity Patterns in Social Science Diary Data. In: IV, IEEE Computer Society (2007) 130–138
20. Petzold, J., Bagci, F., Trumler, W., Ungerer, T.: Comparison of Different Methods for Next Location Prediction. In Nagel, W.E., Walter, W.V., Lehner, W., eds.: Euro-Par. Volume 4128 of Lecture Notes in Computer Science., Springer (2006)
21. Schroeder, W.: New Tools for Task Workflow Analysis. In: CHI EA '09: Proceedings of the 27th international conference extended abstracts on Human factors in computing systems, New York, NY, USA, ACM (2009) 3877–3882
22. de Leoni, M., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Visual Support for Work Assignment in Process-Aware Information Systems. [25] 67–83
23. Bobrik, R., Reichert, M., Bauer, T.: View-Based Process Visualization. In Alonso, G., Dadam, P., Rosemann, M., eds.: BPM. Volume 4714 of Lecture Notes in Computer Science., Springer (2007) 88–95
24. Wieland, M., Längerer, C., Leymann, F., Siemoneit, O., Hubig, C.: Methods for Conserving Privacy in Workflow Controlled Smart Environments - A Technical and Philosophical Enquiry into Human-Oriented System Design of Ubiquitous Work Environments. In: UBICOMM 2009, Sliema, Malta, IEEE (October 2009)
25. Dumas, M., Reichert, M., Shan, M.C., eds.: Business Process Management, 6th International Conference, BPM 2008, Milan, Italy, September 2-4, 2008. Proceedings. In Dumas, M., Reichert, M., Shan, M.C., eds.: BPM. Volume 5240 of Lecture Notes in Computer Science., Springer (2008)

Prozessorientierte Koordination von Kooperationen in Sozialen Netzwerken

Agnes Koschmider, Andreas Oberweis, Huayu Zhang

Karlsruher Institut für Technologie (KIT),
Institut für Angewandte Informatik und Formale Beschreibungsverfahren (AIFB)
76128 Karlsruhe
{agnes.koschmider|andreas.oberweis|huayu.zhang}@kit.edu

Abstract. Soziale Netzwerke wie Facebook, LinkedIn und XING unterstützen den Aufbau von Kontakten und bieten Kommunikationsmöglichkeiten für Kooperationen. Allerdings bieten existierende soziale Netzwerke keine Koordinationsmechanismen für Kooperationen an. In diesem Beitrag wird ein Modell zur prozessorientierten Koordination von Kooperationen in sozialen Netzwerken vorgestellt. Der Ansatz basiert auf Aktivitätslisten von Netzwerkteilnehmern, anhand derer ein Prozessmodell generiert wird, das die Kooperation auf Basis der Netzwerkentwicklung und mit Hilfe der Analyseergebnisse der Netzwerkstruktur koordiniert. Der Ansatz wird anhand eines Anwendungsfalls veranschaulicht.

1 Einleitung

Soziale Netzwerke wie Facebook, LinkedIn und XING erfreuen sich großer Beliebtheit. Obwohl existierende soziale Netzwerke Basisfunktionalitäten für Kooperationen bieten (z.B. Nachrichtenaustausch und Eventplanung), werden soziale Netzwerke selten als Arbeitsplattformen verwendet [1][2]. Ein Grund dafür ist eine unzureichende Koordinationsunterstützung für Kooperationen, insbesondere die fehlende Unterstützung einer flexiblen Einbindung von Kooperationspartnern. Ineffektive Verwaltung der Kommunikation (z.B. Überwachung, Analyse und Anzeigen von Kommunikationsstatus) in sozialen Netzwerken verlangsamt auch die Initiierung einer Kooperation. Koordinationsmechanismen können helfen, die bestehenden Beziehungen zu analysieren, Kommunikationshürden hinsichtlich der zu erzielenden Kooperationsausgabe zu überbrücken und die Kooperationstätigkeiten effizient zu organisieren.

Dieser Beitrag beschreibt ein Modell für die Koordination von Kooperationen in sozialen Netzwerken. Die in diesem Beitrag vorgestellte Lösung kann zur Erweiterung bestehender sozialer Netzwerke genutzt werden, um die Koordination von Kooperationen zu unterstützen.

Abbildung 1 zeigt ein Szenario für Koordination von Kooperationen in sozialen Netzwerken. Teilnehmer B1 aus dem sozialen Netzwerk B beabsichtigt, einen Artikel mit weiteren Personen zu schreiben. Angenommen, ihm steht eine Quelle (z.B. eine

Wiki-Seite) zur Verfügung, die den Schreibprozess beschreibt. Alternativ kann der Netzwerkteilnehmer die Aktivitäten zum Schreiben eines Artikels selbst angeben. Basierend auf dieser Quelle bzw. Aktivitätsliste wird ein Prozessmodell generiert. Einige dieser Aktivitäten erfordern möglicherweise Kooperationspartner, die in sozialen Netzwerken gefunden werden können wie z.B. die Teilnehmer A3 und B9. Im Falle einer Kooperation kann das Prozessmodell von B1 durch die Aktivitäten der beiden Partner erweitert werden. Das Ziel der Verwendung eines Prozessmodells ist eine transparente Organisation der Kooperation, inkl. Kooperationsinitiierung und -durchführung sowie das Teilen und die Wiederverwendung von Kooperationserfahrungen, die im Modell enthalten sind.

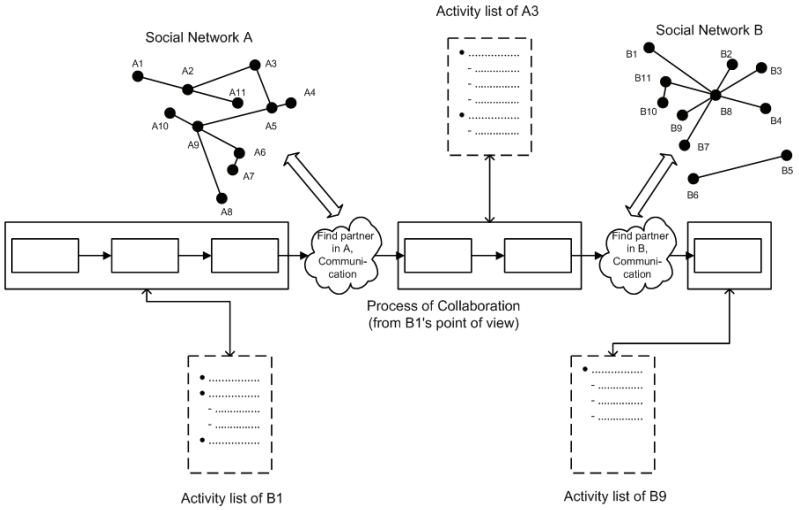


Abb. 1. Koordination eines Schreibprozesses.

Der Beitrag ist wie folgt gegliedert. Kapitel 2 beschreibt die Entwicklungsphasen einer Kooperation in sozialen Netzwerken und die zu koordinierenden Aktivitäten. Kapitel 3 stellt ein Modell zur prozessorientierter Koordination von Kooperationen vor. Der Ansatz wird anhand eines Anwendungsbeispiels in Kapitel 4 veranschaulicht. Der Beitrag schließt mit einem Vergleich mit verwandten Arbeiten sowie einem Ausblick auf zukünftige Arbeiten.

2 Kooperation in sozialen Netzwerken

Ein soziales Netzwerk (im Sinne der Soziologie) ist ein Netzwerk, dessen Knoten soziale Akteure (Personen oder Gruppen) sind, und dessen Kanten die Verhältnisse der Akteure zueinander abbilden [3]. Mit einem Soziogramm aus der Social Network Analysis (SNA) [4] kann die Netzwerkstruktur grafisch dargestellt werden. Eine Kooperation existiert frühestens in einem sozialen Netzwerk nachdem die ersten beiden Phasen einer Netzwerkentwicklung (Potential und Coalescing, siehe Abbildung 2)

durchlaufen worden sind. Nach der Beendigung der dritten Phase (Aktiv) stehen die Netzwerkteilnehmer nur noch gelegentlich im Kontakt. Abbildung 2 gibt einen Überblick über die Aktivitäten der einzelnen Phasen, die im Folgenden sind:

- Potenzial (Potential): Partnersuche angesichts gleicher und/oder komplementärer Interessen
- Vereinigend (Coalescing): Beziehungsaufbau (Kontaktaufnahme und Kommunikation), Abstimmung von Kooperationsausgaben
- Aktiv (Active): Kooperationsdurchführung (z.B. Schreiben einer Publikation, Prototypentwicklung oder Organisation eines gemeinsamen Workshops)

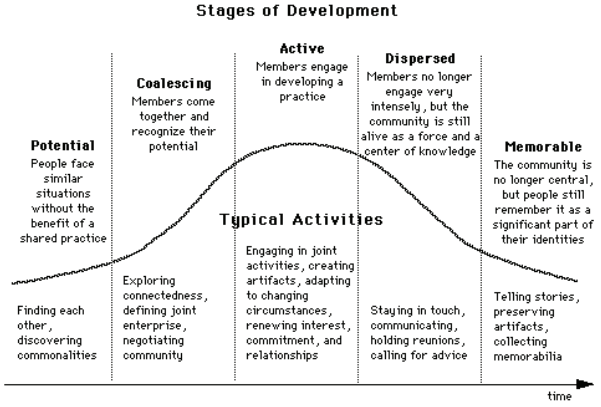


Abb. 2. Entwicklungsphasen eines sozialen Netzwerks [5].

In diesem Beitrag werden soziale Netzwerke betrachtet, die online verfügbar sind, beispielsweise Facebook, LinkedIn und XING.

3 Modell zur Beschreibung einer Kooperationskoordination in sozialen Netzwerken

Zur Koordination einer Kooperation in sozialen Netzwerken wird eine prozessorientierte Vorgehensweise gewählt. Ihre Vorteile sind insbesondere eine transparente Organisation und leichtere Analyse der Kooperation. Die Aktivitäten der Netzwerkteilnehmer zur Erreichung einer Kooperationsausgabe werden in einem Community-Prozess erfasst, der mit der Netzwerkstruktur verknüpft ist.

Ein *Community-Prozess* (CP) ist eine Menge von zusammenhängenden Aktivitäten, die zur Erreichung einer Kooperationsausgabe durchgeführt werden. Die Aktivitäten eines Community-Prozesses sind entweder Einzelaktivitäten (*Single-Activities*) oder Kooperationsaktivitäten (*Cooperation-Activities*). An einer Single-Activity ist nur ein oder kein (bzgl. einer automatischen Aktivität [6]) Netzwerkteilnehmer beteiligt. An einer Cooperation-Activity beteiligen sich mindestens zwei Netzwerkteilnehmer, die eine explizite Kooperationsbeziehung kennzeichnet. Die Aktivitäten

laufen sequenziell, parallel, iterativ oder alternativ ab. Jeder Community-Prozess hat genau einen Anfang und genau ein Ende. Ein Community-Prozess kann durch Unterprozesse verfeinert werden und hat mindestens eine Cooperation-Activity, von der die Unterprozesse Partnersuche (*Finding-Partners*), Beziehungsaufbau (*Building-Relationships*) und Kooperationsdurchführung (*Cooperation-Execution*) abgeleitet werden können. Beispiele für einen Community-Prozess sind „Kooperation an einem EU-Antrag" und „Kooperation bei der Veranstaltung eines Workshops". Ein anderes Beispiel wird in Kapitel 4 im Detail dargestellt.

Ein Community-Prozess ist mit einer Menge von Prozessressourcen verbunden, die als *Community-Prozess-Objekte* benannt werden. Ein Community-Prozess-Objekt ist entweder ein fließendes Objekt (*Flowing-Object*) oder ein nicht fließendes Objekt (*Non-flowing-Object*). Ein Flowing-Object beinhaltet Informationen und Daten, die von einer Aktivität zur anderen übertragen werden, sodass eine Aktivität ausgeführt werden kann. Ein Non-flowing-Object beinhaltet die Ressourcen, die der Aktivität direkt zugewiesen werden und nicht (auf andere Ressourcen) übertragbar sind.

Ein spezielles Non-flowing-Object ist der *Community-User* (CU), der einen Netzwerkteilnehmer durch ein Benutzerprofil beschreibt. Ein Community-User steht in Beziehung zu anderen Community-Users. Aus diesen Beziehungen lässt sich die Netzwerkstruktur ableiten. Ein Benutzerprofil enthält Informationen zu eigenen Kontaktdaten, Kenntnissen und Interessen usw. Eine Art von Flowing-Object ist der *Community-Content* (CC), der ein Container für einen Zeit-, Orts- oder Ereignisbeschränkten Kontext ist. Beispiele für Community-Content sind „Projektantrag", „Konferenz", „Termin" und „Kooperationsvereinbarung".

Eine Kooperation wird koordiniert, indem ein Netzwerkteilnehmer einen Community-Prozess, insbesondere die Cooperation-Activities, spezifiziert und Community-Prozess-Objekte dem Prozess zuordnet.

Zur grafischen Beschreibung eines Community-Prozesses werden Petri-Netze [7] mit entsprechenden Erweiterungen verwendet. Petri-Netze sind gut geeignet für die Modellierung, Analyse und Validierung von Prozessen. Allerdings sind zusätzliche grafische Darstellungselemente erforderlich, um menschlich zu steuernde Tätigkeiten, insbesondere das Kommunikationsverhalten des Community-Prozesses, zu beschreiben. Die grafischen Symbole der Notation sind in Tabelle 1 aufgelistet.

Tabelle 1. Modellierungsnotation des Community-Prozesses

Symbol	Benennung	Bedeutung / Bemerkungen
	Stelle	Zwischenablage für Flowing-Objects
	Transition	Single-Activity
	Kante	Die Kanten dürfen jeweils nur von Stellen zu Transitionen oder von Transitionen zu Stellen führen.

U	Beschriftung	Kooperationsbeziehung; Durch die Beschriftung einer Transition wird eine Cooperation-Activity gekennzeichnet.
F	F-Block	Repräsentation eines abstrakten Finding-Partners - Unterprozesses
B	B-Block	Repräsentation eines abstrakten Building-Relationships - Unterprozesses
C	C-Block	Repräsentation eines abstrakten Cooperation-Execution-Unterprozesses
— — →	Block-Kante	Diese Kanten dürfen jeweils nur von F-Block zu B-Block oder von B-Block zu C-Block führen.
Name	Mitglied	Repräsentation eines Community-Users mit Namen
— — —	Verbindung	Zuordnung eines Community-Users zu einer Single-Activity; Verbindungen können nur zwischen nicht durch Beschriftung gekennzeichneten Transitionen und Mitgliedern existieren

Außer der grafischen Erweiterung ist eine besondere Verfeinerungsregel für Cooperation-Activities in einem Community-Prozess wie folgt definiert: Bei der Verfeinerung einer Cooperation-Activity wird eine sequentielle Folge von abstrakten Finding-Partners-, Building-Relationships- und Cooperation-Execution-Unterprozessen erzeugt.

Für die Modellierung der Community-Prozess-Objekte können UML-Klassendiagramme verwendet werden. Abbildung 3 zeigt beispielsweise die Struktur des Community-Prozess-Objektes.

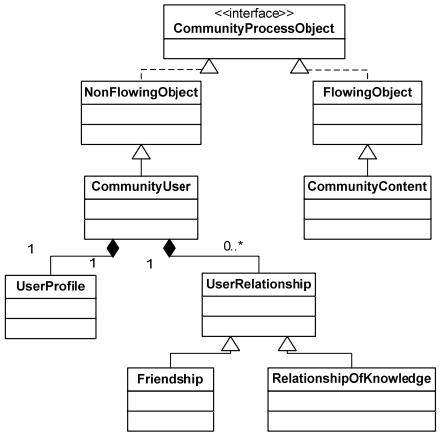


Abb. 3. Struktur des Community-Prozess-Objektes.

4 Anwendungsfall

In diesem Kapitel wird ein Anwendungsfall des Community-Prozesses für die Koordination von Forschungskooperation gezeigt. In Anlehnung an [8] wurde der Prozess „Kooperation bei der Erstellung einer gemeinsamen wissenschaftlichen Publikation“ ausgesucht. Abbildung 4 zeigt vereinfacht den Aufbau des Community-Prozesses.

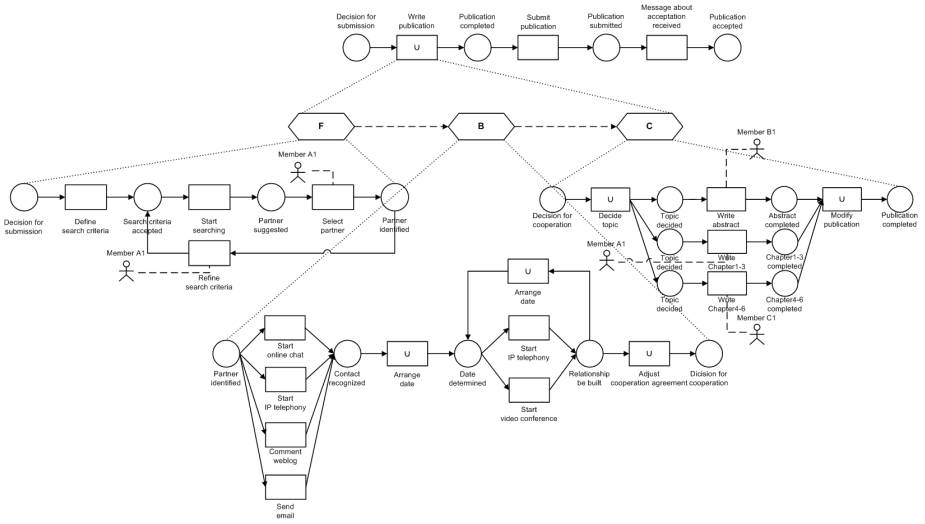


Abb. 4. Aufbau des Community-Prozesses

Der Ablauf der Prozessmodellierung beginnt immer mit der (1) Definition aller Community-Prozess-Objekte. Die Community-Users können z.B. durch die Analyse von Benutzer- und Kommunikationsdaten aus „event logs“ [9] oder E-Mails [10] bestimmt werden oder aus existierenden sozialen Netzwerken importiert werden. Basierend auf den Beziehungen kann die Netzwerkstruktur mit einem Soziogramm [4] (Abb. 5, links) dargestellt und analysiert werden. Die aus der Analyse gewonnenen Metriken wie *Centrality*, *Indegree/Outdegree* und *Transitivity* [4] können genutzt werden, um passende Kontaktpersonen oder Partner herauszufiltern und vorzuschlagen. Aus Übersichtsgründen wurden nur einige Kanten gewichtet, die die Kommunikationshäufigkeit zwischen jeweils zwei Netzwerkteilnehmern beschreiben. (2) Die Erzeugung der ersten Abstraktionsebene des Community-Prozesses erfolgt anhand einer Aktivitätsliste eines Netzwerkteilnehmers oder anhand von Wiki-Seiten [11]. (3) Im Anschluss daran werden die Community-Prozess-Objekte dem Prozess zugeordnet. (4) Darauf folgt die Verfeinerung des Community-Prozesses. Immer wenn eine Cooperation-Activity gekennzeichnet wird, erfolgt eine sequenzielle Konkretisierung der abstrakten Finding-Partners-, Building-Relationships- und Cooperation-Execution-Unterprozesse.

Die obigen Modellierungsschritte (3) und (4), evtl. auch (1) werden sowohl in der Designzeit als auch in der Laufzeit des Prozesses wiederholt, bis alle Cooperation-

Activities definiert sind und die Kooperationsausgabe im Laufe der Durchführung vom Community-Prozess erreicht wird. Dabei handelt es sich um Lazy/Late Modeling [12], bei der nicht vorhersehbare Entwicklungen oder Entscheidungen (z.B. Nutzung verschiedener Kommunikationskanäle im Building-Relationships-Unterprozess und Aufgabenzuweisung im Cooperation-Execution-Unterprozess) zur Laufzeit nachmodelliert werden.

Der Prozess wird top-down modelliert. Dabei kann der Netzwerkteilnehmer die Aktivitäten selbst modellieren oder ein Modellierungsunterstützungssystem verwenden, das ihn bei der Modellierung durch Vorschlag von passenden Prozessaktivitäten(-mustern) hilft [13].

Zur Laufzeit des Community-Prozesses werden relevante Kommunikationsdetails, z.B. Kommunikationsdauer und -häufigkeit, zwischen den Beteiligten (z.B. A1, A2, B1, B4 und C1, siehe Abb. 5) gesammelt, mit denen das ursprüngliche Soziogramm stets aktualisiert wird. Ebenso werden die häufig verwendeten Kommunikationskanäle durch Überwachung erkannt und es werden Hinweise für Netzwerkteilnehmer gegeben, die bei der Entscheidung nützlich sein könnten.

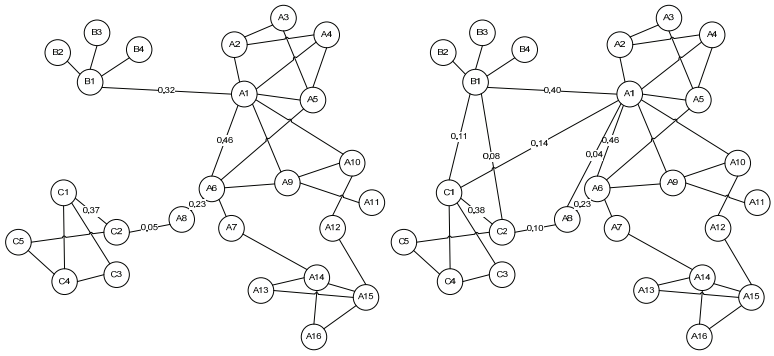


Abb. 5. Netzwerkstruktur vor (links) und nach (rechts) der Kooperation.

5 Fazit und Ausblick

Der vorliegende Beitrag schlägt ein Modell zur prozessorientierten Koordination von Kooperationen in sozialen Netzwerken vor. Die bisherigen Ansätze zur prozessorientierten Koordination von Kooperationen berücksichtigen Netzwerkentwicklungen nicht angemessen, sodass eine Kooperation nicht effizient erfolgen kann. In [14] beispielsweise wurde ein CSCW-Rahmenwerk für die wissenschaftliche Zusammenarbeit beschrieben, das eher technische Aspekte und nicht soziale Fragen berücksichtigt. [15] und [16] betrachten zwar die Einsatzmöglichkeit von sozialer Software im eCollaboration-Bereich zur Schaffung besserer Kommunikation, unterstützen aber keine Koordinationsformen.

Die Koordinierung auf Basis der Netzwerkentwicklung hat den Vorteil, dass Aktivitäten einfacher und gezielter zur Kooperationsinitiierung und -durchführung einge-

setzt werden können. Eine flexible Einbindung von Kooperationspartnern wird ermöglicht. Zur Steuerung der Kooperation wurde in diesem Beitrag der Community-Prozess vorgestellt, der Einzel- und Gemeinschaftsaktivitäten von Netzwerkteilnehmern koordiniert. Die Kooperation wird durch den Community-Prozess effektiver gesteuert, weil das Kommunikationsverhalten mit Partnern und der Ablauf der Ausführung transparent sind.

Im nächsten Schritt müssen alle Konzepte des Community-Prozesses für eine systemunterstützende Ausführung formalisiert werden. Ein grafischer Prozesseditor für ausgewählte soziale Netzwerke wird dem einzelnen Netzwerkteilnehmer bereit gestellt. Eine Evaluation soll die Effektivität des Systems untersuchen.

Referenzen

1. Döbler, T.: Potenziale von Social Software. FAZIT-Schriftenreihe, Forschungsbericht/Band 5, MFG Stiftung Baden-Württemberg (2007)
2. FAZIT-Kurzbericht Nr.1/2009, Zentrum für Europäische Wirtschaftsforschung GmbH http://www.fazit-forschung.de/fileadmin/_fazit-forschung/downloads/FAZIT_Kurzbericht_1_2009.pdf
3. Barnes, J.A.: Social Networks. Addison-Wesley, Reading Massachusetts (1972)
4. Wasserman, S., Faust, K.: Social Network Analysis: Methods and Applications. Cambridge University Press, Cambridge (1994)
5. Wenger, E.: Communities of practice: learning, meaning and identity. Cambridge University Press, Cambridge (1998)
6. Dogac, A., Sheth, A., Özsu, T., Kalinichenko L.: Workflow Management Systems and Interoperability. Springer-Verlag, Berlin (1998)
7. Reisig, W.: Petrinetze: Eine Einführung. Springer-Verlag, Berlin (1986)
8. Klink, S., Oberweis, A., Ried, D., Trunko, R.: A Service-oriented Information System for Collaborativ Research and Doctoral Education. In: IEEE International Conference on e-Business Engineering. IEEE Press (2006)
9. van der Aalst, W., Song, M.: Mining social networks: uncovering interaction patterns in business processes. In: 2th International Conference on Business Process Management. Springer-Verlag, Berlin (2004)
10. Yamakami, T.: Social Process Visualization in Regional Community Network Users. In: 3th Asia Pacific Conference on Computer Human Interaction. IEEE Press (1998)
11. Dengler, F., Lamparter, S., Hefke, M., Abecker, A.: Collaborative Process Development using Semantic MediaWiki. In: 5th International Conference of Professional Knowledge Management. Köllen Verlag (2009)
12. Petrie, C., Goldmann, S., Raquet, A.: Agent-based project management. Lecture notes in AI 1600, Springer-Verlag, Berlin (1999)
13. Hornung, T., Koschmider, A., Lausen, G.: Recommendation Based Process Modeling Support: Method and User Experience. In: 27th International Conference on Conceptual Modeling. Springer-Verlag, Heidelberg (2008)
14. Lubich, H.P.: Towards a CSCW Framework for Scientific Cooperation in Europe. Springer-Verlag, Heidelberg (1995)
15. Lattemann, C., Stieglitz, S., Kupke, S.: Deutsche Unternehmen auf dem Weg zum Web 2.0? In: HMD - Praxis der Wirtschaftsinformatik: eCollaboration, S. 18-26. dpunkt-Verlag, Heidelberg (2009)
16. Benlian, A., Hilkert, D., Hess, T.: eCollaboration mit Social Software in der globalen Software-Entwicklung. In: HMD - Praxis der Wirtschaftsinformatik: eCollaboration, S. 37-45. dpunkt-Verlag, Heidelberg (2009)

Sichere und Zuverlässige Prozessausführung in Serviceorientierten Architekturen

Dieter Schuller

Multimedia Communications Lab (KOM),
Technische Universität Darmstadt, Germany
Dieter.Schuller@KOM.tu-darmstadt.de

Zusammenfassung Prozesse in Serviceorientierten Architekturen lassen sich durch Komposition von Services realisieren. Die dabei verwendeten Services sind jedoch nicht notwendigerweise allesamt innerhalb der eigenen Unternehmung vorhanden, sondern über Organisationsgrenzen hinweg verteilt. Im Fall von fehlendem Einsatz entsprechender Sicherheitstechnologien sind die Nachrichten, die für die Invokation externer Services mit dem Service Provider ausgetauscht werden, möglicherweise das Ziel von Angreifern, die die Nachrichten manipulieren oder den Nachrichtenaustausch gänzlich verhindern können. Insofern sind Mechanismen erforderlich, die eine sichere und zuverlässige Serviceausführung ermöglichen. Der in dieser Arbeit vorgestellte Serviceüberwachungs- und Steuerungsansatz soll den Ausfall des gesamten Prozesses aufgrund von manipulierten, fehlerhaften oder ausgefallenen Services verhindern.

1 Einleitung

Geschäftsprozesse werden heutzutage nicht mehr ausschließlich innerhalb der Grenzen der eigenen Organisation ausgeführt (vgl. [1, 2]). Unternehmensübergreifende Prozesse gewinnen zunehmend an Bedeutung (vgl. [3]). Der Einsatz von Services – die je nach Granularität eine mehr oder weniger komplexe Funktionalität zur Verfügung stellen (vgl. [4]) – zur Realisation der Prozesse im Rahmen von Serviceorientierten Architekturen kann hier insofern nützlich sein, als er die Integration der verschiedenen Legacy Systeme und IT Systeme unterstützt. Externe Services, die von Geschäftspartnern entsprechend ihrer Kernkompetenzen angeboten werden oder auf Service-Marktplätzen vorhanden sind, können somit zur Effizienzsteigerung hinsichtlich der eigenen Prozessausführung eingebunden werden. Fällt einer der eingesetzten Services aus, kann aufgrund der Eigenschaft der losen Kopplung bspw. der fehlerhafte Service durch andere Services ersetzt werden, die die gleiche bzw. eine vergleichbare Funktionalität haben, sofern solche alternativen Services innerhalb der Unternehmung oder bei anderen Geschäftspartnern vorhanden bzw. auf Service-Marktplätzen verfügbar sind. Des Weiteren besteht jedoch die Gefahr, dass der für die Invokation externer Services notwendige Nachrichtenaustausch durch Angreifer manipuliert oder behindert wird (vgl. [5]). Um zu vermeiden, dass die Prozessausführung aufgrund der Invokation manipulierter Services erfolglos abbricht, sind Sicherheitsmechanismen erforderlich, die erkennen, ob ein Service manipuliert wurde.

Daher wird in Abschnitt 2 der in unserer Arbeit in [6] vorgestellte Ansatz zur Gewährleistung einer zuverlässigen Prozessausführung um Sicherheitsaspekte mit dem Ziel erweitert, eine sichere und zuverlässige Prozessausführung zu erreichen. Diesbezüglich wird einen Ansatz beschrieben, um *Sicherheit* für die Entscheidung zu quantifizieren, welcher Service für die Realisation eines bestimmten Prozessschrittes selektiert werden soll. Das entsprechende Optimierungsproblem wird in Abschnitt 3 formuliert und gelöst. Die Arbeit schließt mit einer Zusammenfassung und einem Ausblick.

2 Steuerungsansatz für sichere und zuverlässige Prozesse

Um einen Prozessausfall aufgrund von Servicefehlern zu verhindern, wurde ein dreistufiger Überwachungs- und Steuerungsansatz realisiert, der in Abbildung 1 dargestellt ist.

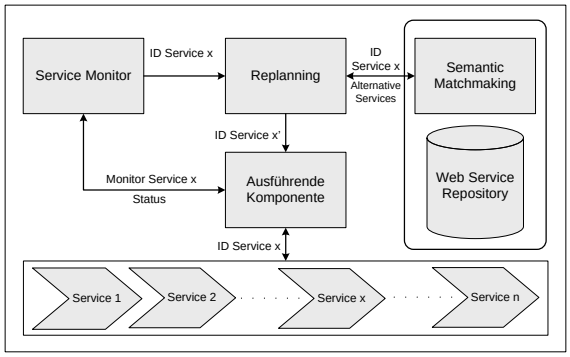


Abbildung 1: Replanning Zyklus

Der Service Monitor misst die Ausführungszeit des invokierten Services. Überschreitet dieser den in den Service Level Agreements (vgl. [7]) angegeben Wert, wird die Replanning-Komponente aufgerufen. Diese identifiziert unter Zuhilfenahme einer Semantic Matchmaking Komponente alternative Services (anhand von semantischen Annotationen), die im Web Service Repository gelistet sind. Um eine Auswahl zu treffen, welcher der alternativen Services den ausgefallenen ersetzen soll, stellt die Replanning-Komponente im Rahmen des Service-Selektions-Problems (vgl. [8–10]) das in Abschnitt 3 beschriebene Optimierungsproblem auf und löst es optimal hinsichtlich der nicht-funktionalen Service-Eigenschaften (Quality of Service – QoS). Dabei werden sowohl der bisher noch nicht erfolgreich ausgeführte Prozessschritt sowie alle nachfolgenden Prozessschritte des ursprünglichen Ausführungsplans für die Optimierung berücksichtigt.

Auf diese Weise ist jedoch noch nicht sichergestellt, dass die eingebundenen Services nicht von Angreifern manipuliert wurden. Insofern sind Sicherheitsmechanismen wie Verschlüsselung, Digitale Signaturen und Checksummen erforderlich,

die eine solche Manipulation erkennen, um den manipulierten Service durch einen alternativen Service (wie beschrieben) auszutauschen. Um im Rahmen des Service-Selektions-Problems diese qualitativen Sicherheitseigenschaften von Services als QoS-Parameter berücksichtigen zu können, ist eine Quantifizierung dieser Eigenschaften notwendig. Eine Voraussetzung stellt dabei ihre Erfassbarkeit dar. Andernfalls lässt sich keine nachvollziehbare, konsistente Zuordnung der qualitativen Eigenschaft auf einen numerischen Wert finden. Konsistent bedeutet dabei, dass einer erfassten Eigenschaft immer der gleiche, numerische Wert zugewiesen wird. Im Kontext von Sicherheitsmetriken definiert Jaquith in seiner Arbeit [11] eine gute Metrik als einen „konsistenten Standard für die Messung“. Er fordert von einer guten Metrik, dass ihre Messung konsistent möglich sein sollte und dass sie günstig (vorzugsweise automatisiert) erfassbar ist. Zudem sollte sie durch eine numerische Zahl ausgedrückt werden können und nicht durch qualitative Kennzeichnungen wie „high“, „medium“ oder „low“.

Die vorliegende Arbeit verfolgt das Ziel, einen nachvollziehbaren und konsistenten Ansatz zur Zuordnung von Sicherheitseigenschaften auf numerische Werte zu finden. Hierfür wird in Anlehnung an die in [12] definierten Schutzziele (Daten-) Integrität, (Informations-) Vertraulichkeit, Verfügbarkeit, Verbindlichkeit, Authentizität und Privatheit entsprechende *Sicherheits-Level* als QoS-Parameter definiert. Sind die mit externen Serviceanbiertern ausgetauschten Nachrichten bspw. verschlüsselt und/oder digital signiert, sind die Schutzziele Authentizität und Vertraulichkeit erfüllt. Zudem kann mithilfe von Checksummen die Integrität der erhaltenen Nachrichten überprüft werden. Je nachdem, welche Schutzziele von einem Service bzw. von einem Service Provider erfüllt werden, wird für den betreffenden Service ein gewisser Level gesetzt. D. h., dass bestimmte Sicherheits-Eigenschaften bestimmten Levels zugeordnet werden. Dies kann jedoch nicht generisch vorgenommen werden, sondern muss spezifisch für eine bestimmte Situation bzw. für ein bestimmtes Szenario durchgeführt werden. Bspw. könnte es in einem Business-Kontext wichtiger sein, dass zur Wahrung von Betriebsgeheimnissen die Informationen verschlüsselt sind, die zur Invokation eines externen Services als Eingabeparameter die Unternehmensgrenzen verlassen, als dass die zurück gelieferten Ergebnisse integer sind. Handelt es sich jedoch um erfolgskritische Prozesse (wie bspw. in einem Katastrophenszenario), sind Integrität und Autorisierung von essentieller Bedeutung. Allerdings ist die alleinige Zuordnung einer Sicherheits-Eigenschaft zu einem Sicherheitslevel nicht immer ausreichend. Bspw. könnte ein Angreifer das Ergebnis einer externen Service-Invokation abfangen und stattdessen andere Ergebnisse an den ursprünglichen Service-Aufrufer weiterleiten. Diese Ergebnisse sind zwar auch integer, sie stammen aber nicht von der Instanz, von der die Funktionalität des aufgerufenen Services erwartet wurde. Um hier ein Beispiel zu nennen, könnte ein Angreifer in einem Katastrophenszenario mit Hochwasser falsche Informationen über Pegelstände weiterleiten, sodass notwendige Maßnahmen nicht angemessen geplant werden können. In diesem Kontext muss also eine Kombination von Sicherheits-Eigenschaften vorliegen, um eine Zuordnung zu bestimmten Sicherheits-Levels vornehmen zu können. Als mögliche Kombinationen werden im Rahmen dieser Arbeit zunächst die in Abbil-

dung 2a und Abbildung 2b angegebenen Level vorgeschlagen. Die Level haben per Definition h ganzzahlige numerische Werte $s \in S = \{1, ..., h\}$. Dabei wird dieser Parameter als positiver QoS-Parameter definiert, sodass ein höherer Wert besser ist. Sind auf diese Weise Sicherheits-Level für die verwendeten Services erstellt worden, lässt sich dieser nun quantifizierte Parameter als QoS-Parameter für die Formulierung des Optimierungsproblems in Abschnitt 3 verwenden.

Level 1	Verbindlichkeit
Level 2	Integrität
Level 3	Vertraulichkeit
Level 4	Authentifikation

(a) Kombination 1

Level 1	Authentifikation
Level 2	Verbindlichkeit
Level 3	Integrität
Level 4	Vertraulichkeit

(b) Kombination 2

Abbildung 2: Sicherheits-Level

3 Optimierungsproblem

Die in Abschnitt 2 angesprochene Replanning-Komponente befasst sich mit dem Service-Selektions-Problem. D. h., sie erstellt einen optimalen Ausführungsplan, indem sie entscheidet, welcher Prozessschritt von welchem dafür infrage kommenden Service realisiert werden soll. Optimal bedeutet in diesem Zusammenhang, dass diejenigen Services selektiert werden, die vorgegebene Restriktionen einhalten und hinsichtlich ihrer QoS Eigenschaften am besten sind. In anderen Worten ist bei der Auswahl der Services zu beachten, dass die gebildete Servicekomposition einerseits bestimmte Restriktionen in Bezug auf ihre QoS-Parameter einhält, die in Modell 1 als Nebendingungen formuliert sind. Andererseits soll sie die in (2) angegebene Zielfunktion maximieren, um eine optimale Lösung darzustellen. Ein Beispiel für eine einzuhaltende Restriktion hinsichtlich der aggregierten Ausführungszeit der ausgewählten Services wäre eine obere Grenze von 20 Sekunden. Hinsichtlich der Zuverlässigkeit könnte eine untere Grenze bspw. 95%. Dabei hängen die unteren und oberen Schranken sowie die Zielfunktion von dem betrachteten Szenario ab.

Um das Systemmodell zu erstellen und das Optimierungsproblem zu formulieren, wird unser in [6] vorgestellte, generische Ansatz für die fünf QoS-Parameter Zuverlässigkeit r, Sicherheit s, Verfügbarkeit a, Ausführungszeit t und Kosten c, die in dieser Reihenfolge mit $k \in K = \{1, 2, 3, 4, 5\}$ nummeriert werden, erweitert. Dabei wird zunächst von einer sequenziellen Anordnung von n Prozessschritten ausgegangen. Prozessschritt $i \in I = \{1, ..., n\}$ wird vor Prozessschritt $i + 1$ ausgeführt. Für jeden Prozessschritt i gibt es m_i alternative Services $j_i \in J_i = \{1, ..., m_i\}$, wobei Prozessschritt i durch genau einen Service j_i realisiert wird. Diese Services j_i unterscheiden sich hinsichtlich ihrer fünf QoS-Parameter q_{ijk}. Ist ein höherer (geringerer) QoS-Wert besser, handelt es

sich um einen positiven (negativen) QoS-Parameter. Die Restriktionen für die QoS werden mit b_k bezeichnet. Die Entscheidungsvariablen $x_{ij} \in \{0, 1\}$ geben Auskunft darüber, ob Prozessschritt i durch Service j realisiert wird. Um die (unterschiedlichen) QoS-Parameter für die Zielfunktion aggregieren zu können, ist eine Normalisierung für alle Werte q_{ijk} erforderlich, die in Gleichung (1) angegeben ist. Andernfalls lässt sich bspw. die Sicherheit als QoS-Parameter nicht mit der Zuverlässigkeit eines Services verrechnen. Das zu lösende Optimierungsproblem wird in Modell 1 formuliert. Vor dem Hintergrund, eine sichere und zuverlässige Prozessausführung zu erreichen, werden die QoS-Parameter Zuverlässigkeit r, Sicherheit s und Verfügbarkeit a mit jeweils $33,3\%$ für die Zielfunktion gewichtet.

$$q_{ijk}^{norm} := \begin{cases} 1 - \frac{max\{q_{ijk}\} - q_{ijk}}{max\{q_{ijk}\} - min\{q_{ijk}\}} & \text{, falls k negativer QoS} \\ \frac{q_{ijk} - min\{q_{ijk}\}}{max\{q_{ijk}\} - min\{q_{ijk}\}} & \text{, sonst} \end{cases} \tag{1}$$

Modell 1 Nicht-lineares Optimierungsproblem

Zielfunktion

$$\text{maximiere } F(x) = \sum_{i=1}^{n} \sum_{j=1}^{m_i} \sum_{k=1}^{3} \frac{1}{3} q_{ijk}^{norm} x_{ij} \tag{2}$$

s.d.

$$\prod_{i=1}^{n} \sum_{j=1}^{m_i} r_{ij} x_{ij} \geq b_1 \tag{3}$$

$$\min \left\{ \sum_{j=1}^{m_i} s_{ij} x_{ij} \right\} \geq b_2 \tag{4}$$

$$\prod_{i=1}^{n} \sum_{j=1}^{m_i} a_{ij} x_{ij} \geq b_3 \tag{5}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{m_i} t_{ij} x_{ij} \leq b_4 \tag{6}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{m_i} c_{ij} x_{ij} \leq b_5 \tag{7}$$

$$\sum_{j=1}^{m_i} x_{ij} = 1 \qquad \forall i \in I \tag{8}$$

$$x_{ij} \in \{0, 1\} \quad \forall i \in I, \forall j \in J_i \tag{9}$$

Die zu maximierende Zielfunktion ist in (2) angegeben. In (3), (4), (5), (6) und (7) sind die Nebenbedingungen für die QoS-Parameter Zuverlässigkeit r,

Sicherheit s, Verfügbarkeit a, Ausführungszeit t und Kosten c angegeben. In (8) wird gefordert, dass für jeden Prozessschritt i genau ein Service ausgewählt wird. Die Ganzzahligkeitsbedingung in (9) stellt sicher, dass lediglich *ganze* Services selektiert werden.

Da es sich bei (3), (4), (5) um nicht-lineare Gleichungen handelt, wird für (3) und (5) die Approximation in (10) angewendet, die für z nahe 1 gute Werte liefert [13]. Des Weiteren lässt sich (4) durch (11) ersetzen.

$$\prod_{i=1}^{n}\sum_{j=1}^{m_i} z_{ij}x_{ij} \approx 1 - \sum_{i=1}^{n}(1 - \sum_{j=1}^{m_i} z_{ij}x_{ij}) \tag{10}$$

$$\sum_{j=1}^{m_i} s_{ij}x_{ij} \geq b_2 \qquad \forall i \in I \tag{11}$$

Das hierdurch erhaltene lineare Optimierungsproblem lässt sich mit Methoden des Operations Research optimal lösen, sofern eine Lösung existiert (vgl. [14]). Hierbei ist jedoch zu berücksichtigen, dass es sich bei dem beschriebenen Optimierungsproblem um ein NP-schweres Problem handelt [9, 15, 16]. Bei steigender Problemgröße lässt sich die Heuristik H1_RELAX_IP [17] verwenden, die bei großen Problemgrößen nicht signifikant schlechter abschneidet als das exakte Lösungsverfahren [18]. Hier wird die Ganzzahligkeitsbedingung relaxiert und das so entstandene gemischt-ganzzahlige lineare Optimierungsproblem optimal gelöst. Anschließend werden diejenigen Services ausgewählt, deren Werte in den Entscheidungsvariablen x_{ij} am größten sind.

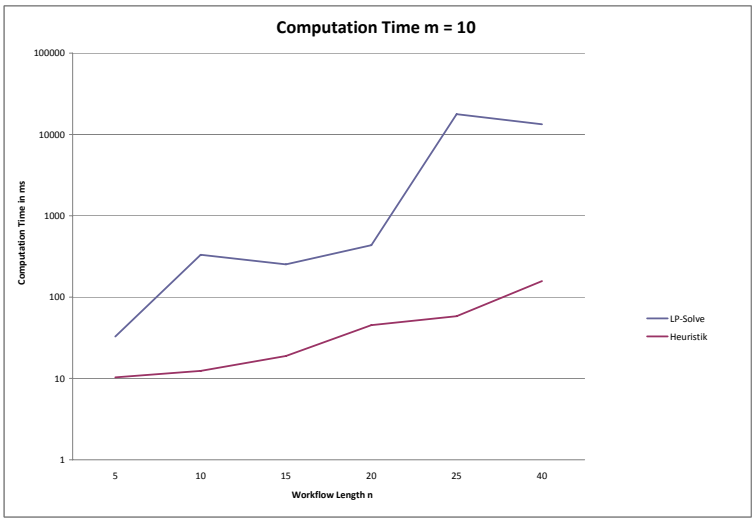


Abbildung 3: Berechnungszeit bei Variation der Prozesslänge

Um einen Einblick in die Laufzeitkomplexität zu geben, wird die Laufzeit für die Berechnung der optimalen Lösung (durch Einsatz von Branch-and-Bound

Verfahren) für das formulierte Optimierungsproblem in Abbildung 3 und Abbildung 4 mit der Laufzeit der Heuristik H1_RELAX_IP verglichen. Dabei erreicht die Heuristik zumeist eine Lösungsgüte von mehr als 95% der optimalen Lösung. In Abbildung 3 wird die Anzahl alternativer Services pro Prozessschritt m_i auf $m_i = 10$ fixiert und die Anzahl an Prozessschritten n variiert. In Abbildung 4 wird die Anzahl an Prozessschritten n auf $n = 10$ fixiert und die Anzahl alternativer Services pro Prozessschritt m_i variiert.

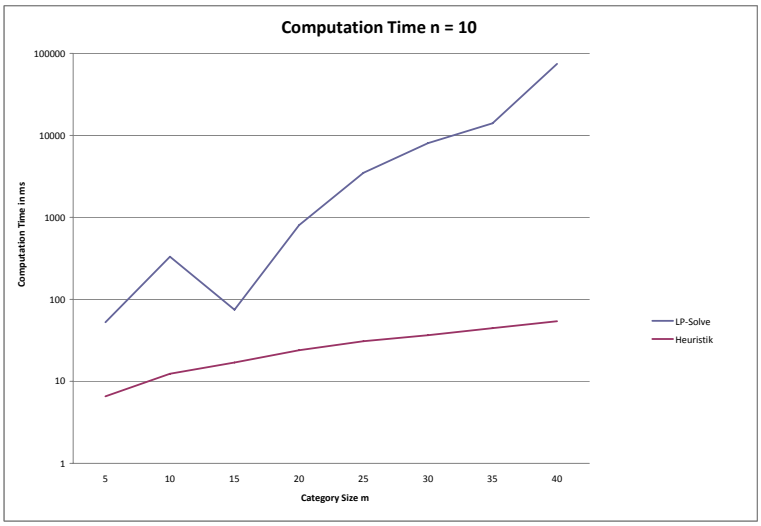


Abbildung 4: Berechnungszeit bei Variation der Anzahl alternativer Services pro Prozessschritt

4 Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde ein Serviceüberwachungs- und Steuerungsansatz vorgestellt, der durch eine optimale Lösung des Service-Selektions-Problems aufgrund der Gewichtung der QoS-Parameter Sicherheit, Zuverlässigkeit und Verfügbarkeit eine sichere und zuverlässige Prozessausführung gewährleisten soll.

Der Fokus für die zukünftige Arbeit wird einerseits auf die Formulierung von linearen Optimierungsproblemen für komplexe Prozesse gelegt, um den vorgestellten Ansatz auch für komplexe Prozessstrukturen verwenden zu können. Andererseits sollen Kombinationen der betrachteten Sicherheits-Eigenschaften hinsichtlich des erreichten Sicherheitsniveaus (Quality of Protection) analysiert werden, um ein Konzept für die Definition von sinnvollen Sicherheits-Levels zu entwickeln. Zudem wird die Evaluation des vorgestellten Ansatzes angestrebt.

Acknowledgements. Diese Arbeit wurde in Teilen durch das BMBF-finanzierte Projekt SoKNOS (http://www.soknos.de) und durch das E-Finance Lab e. V., Frankfurt am Main, Deutschland, (http://www.efinancelab.de) unterstützt.

Literatur

1. Bussler, C.: The Role of B2B Protocols in Inter-Enterprise Process Execution. In: International Workshop on Technologies for E-Services (TES). (2001) 16–29
2. Chen, Q., Chen, Q., Hsu, M., Hsu, M.: Inter-Enterprise Collaborative Business Process Management. In: International Conference on Data Engineering (ICDE). (2001) 253–260
3. Leymann, F., Roller, D.: Production Workflow: Concepts and Techniques. Prentice Hall PTR, Upper Saddle River, NJ, USA (2000)
4. Krafzig, D., Banke, K., Slama, D.: Enterprise SOA: Service-Oriented Architecture Best Practices. Prentice Hall PTR, Upper Saddle River, NJ, USA (2004)
5. Miede, A., Nedyalkov, N., Schuller, D., Repp, N., Steinmetz, R.: Cross-organizational Security – The Service-oriented Difference. In: International Conference on Service-Oriented Computing (ICSOC) – 2009 Workshops. (2009)
6. Schuller, D., Papageorgiou, A., Schulte, S., Eckert, J., Repp, N., Steinmetz, R.: Process Reliability in Service-oriented Architectures. In: Digital Ecosystems and Technologies (DEST). (2009) 606–611
7. Keller, A., Ludwig, H.: The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. Journal of Network and Systems Management **11**(1) (2003) 57–81
8. Jaeger, M.C., Mühl, G., Golze, S.: QoS-Aware Composition of Web Services: A Look at Selection Algorithms. In: Internationl Conference on Web Services (ICWS). (2005) 807–808
9. Yu, T., Lin, K.J.: Service Selection Algorithms for Composing Complex Services with Multiple QoS Constraints. In: International Conference on Service-oriented Computing (ICSOC). (2005) 130–143
10. Ardagna, D., Giunta, G., Ingraffia, N., Mirandola, R., Pernici, B.: QoS-Driven Web Services Selection in Autonomic Grid Environments. In: OTM Conferences (2). (2006) 1273–1289
11. Jaquith, A.: Security Metrics: Replacing Fear, Uncertainty, and Doubt. Addison-Wesley Professional (2007)
12. Eckert, C.: IT-Sicherheit. 5. edn. Oldenbourg Wissensch.Vlg (November 2007)
13. Heckmann, O.: A System-oriented Approach to Efficiency and Quality of Service for Internet Service Providers. PhD thesis, TU Darmstadt, Fachbereich Informatik (2004)
14. Domschke, W., Drexl, A.: Einführung in Operations Research. Springer Verlag, Heidelberg (2007)
15. Canfora, G., Penta, M.D., Esposito, R., Perfetto, F., Villani, M.L.: Service Composition (Re)Binding driven by Application-Specific QoS. In: 4th International Conference Service-Oriented Computing (ICSOC). (2006) 141–152
16. Zeng, L., Benatallah, B., Ngu, A.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-Aware Middleware for Web Services Composition. Transactions on Software Engineering **30**(5) (2004) 311–327
17. Berbner, R., Spahn, M., Repp, N., Heckmann, O., Steinmetz, R.: Heuristics for QoS-aware Web Service Composition. In: International Conference on Web Services (ICWS). (2006)
18. Berbner, R., Spahn, M., Repp, N., Heckmann, O., Steinmetz, R.: Dynamic Replanning of Web Service Workflows. In: Digital Ecosystems and Technologies (DEST). (2007)

Estimating costs of a service

Christian Gierds and Jan Sürmeli

Institut für Informatik
Humboldt-Universität zu Berlin
Unter den Linden 6, 10099 Berlin, Germany
{gierds|suermeli}@informatik.hu-berlin.de

Abstract. When designing a publicly available Web service, a service designer has to take care of costs and revenue caused by this services. In the very beginning possible partners might only be vaguely known, or the service behavior contains arbitrary repetitions. Then the estimation of costs for running this service is difficult and decisions based on them can hardly be made.
We propose a static analysis of the service's behavior. We over-approximate possible runs and therefore costs of the service. Our approach provides a basis for reasoning about nonfunctional properties as shown for costs.

Key words: Nonfunctional properties, state equation, open nets, service behavior

1 Motivation and approach

A *service provider* in a *service-oriented architecture* (SOA) [1] has only limited knowledge about the future interaction partners of its published service. Then determining nonfunctional properties such as estimating the costs of a service run (or any other performance measure) in the interaction with a *service requester* is complex, because they vary depending on the requester's behavior. A *service broker* may assure certain functional properties such as proper termination, thus giving starting points for narrowing down the possible interactions. For approximation of a service's costs we propose a static analysis approach on a constrained set of interaction partners.

For two services A and B, question is: How much does it cost the provider of A to interact with B? As a prerequisite, we assume that each action a of A has a distinct cost. However prior to execution, it is impossible to give one discrete overall cost: Both A and B may contain non-determinism resulting in different runs of their interaction. So we propose an *interval* for the overall cost instead. Naively, we can explore the whole state space to determine the minimal and maximal overall costs, taking only into account the maximal runs, that is those that are either infinite or end in a deadlock state. This however can be very complex and may be impossible, if the state space is infinite or too big to fit into memory.

We thus propose a simplification of the previous question: If both A and B terminate, how much did the interaction cost the provider of A? If we further assume that termination always means reaching a final state ω of A, the question boils down to: How much does it cost the provider of A to reach ω from its initial state α when A and B are coupled? We identify three core challenges:

(*i*) As for the overall cost, the cost of reaching ω is not one distinct number but a cost interval $C = (c_{\min}, c_{\max})$, because there typically exists not only one, but a set R of runs from α to ω, which can even be infinite. (*ii*) We find the nature of R not only being dependent on the behavior of A, but also on the behavior of B. (*iii*) ω is not necessarily reachable in the interaction of A and B, resulting in an undefined cost interval.

In this paper, we will tackle those three challenges with classical Petri net analysis. We translate the model of the composition $A \oplus B$ into a system of linear equations – the *state equation* [2] – and use *linear optimization* to minimize and maximize costs. The resulting bounds $X = (x_{\min}, x_{\max})$ will over-approximate C. Although we lose some precision with this technique, we expect the proposed approach to make cost estimation before execution feasible in practice.

If B is not given as a model, but only as a set Ψ of *constraints*, we build the state equation of A alone and augment it by the constraints in Ψ. Again, we use linear optimization which yields a valid over-approximation $X' = (x'_{\min}, x'_{\max})$ of C. We observe that X' is not as tight as X: X' is valid for a whole set of partner services $\mathcal{B}_\Psi$ (induced by Ψ) in contrast to only one explicitly given partner service B. However, computation of X' can be done in a less time critical phase, whereas X is computed after B has been selected and also requires a complete behavioral model of B. Furthermore, we attend the special case that $\Psi = \emptyset$, resulting in the complete set of partners of A, similarly to the work in [3].

The third challenge is harder to come by. In this paper we aim at ensuring, that *if* A reaches ω, *then* the found cost interval applies. The found cost intervals for each final marking can be *composed* for a global overview over costs. To avoid complexity from this source, we draft a method to directly find minimal and maximal costs for given sets of final markings.

The rest of the paper is structured as follows: In Sect. 2 we introduce the necessary formal notions. Section 3 describes the approach itself together with the above announced specializations and generalizations. Finally we will conclude and will give an outlook for application.

2 Notation

Let $f : A \to B$ be a function, then for $A' \subseteq A$, $f(A')$ is defined as $\{f(a) \mid a \in A'\}$. For the rest of this paper, let us assume a set $\mathcal{C}$ of *message channels* and message exchange to be *asynchronous*, messages may even overtake each other. Sending a message over channel a is denoted by $!a$ and receiving a message over channel a is denoted by $?a$. Then $\mathcal{E} = \{!c \mid c \in \mathcal{C}\} \cup \{?c \mid c \in \mathcal{C}\}$ denotes the set of all *sending* and *receiving events*, respectively. Furthermore, each service uses a channel in at most one direction: sending or receiving (viz. a service cannot unsent a message).

Now an *open net* $N = (P, T, F, \alpha, \Omega, ev)$ is a classical P/T-net [4] (P, T, F, α). $ev : T \to (\mathcal{E} \cup \{\tau\})$ is a *labeling function* indicating for each transition t if it is either communicating ($ev(t) \in \mathcal{E}$ with $ev(t) = ?c \Rightarrow !c \notin ev(T)$) or internal to the net ($ev(t) = \tau$). Additionally we define a set of final markings Ω, which indicate proper termination. The *interface* $\mathcal{E}(N) = ev(T) \setminus \{\tau\}$ of an open net N comprises all labels used by the transitions of N. Examples for open nets are depicted in Figs. 1 and 2. As

usual, places are depicted by circles, transitions by rectangles and the flow relation by arcs. Tokens are black dots, labels are written inside the transitions, omitting τ.

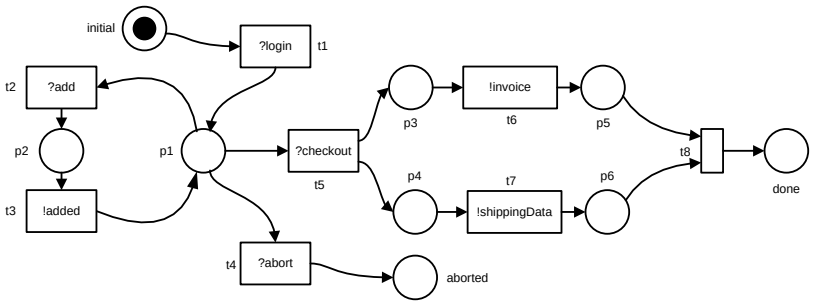


Fig. 1. Open net shop with $\Omega = \{[\mathsf{aborted}], [\mathsf{done}]\}$

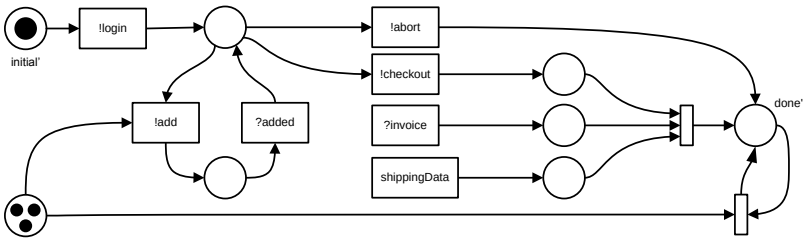


Fig. 2. Open net customer with $\Omega = \{[\mathsf{done}']\}$

The open net shop models an online shop. After logging in, a customer can decide to add products by sending messages via channel add which are confirmed via added. The customer can either decide to send an abort message or to checkout. After receiving a message via checkout, the shop sends information about shipping and an invoice. The two final markings [done] and [aborted] specify the two expected results of the interaction. The open net customer is a *partner* of shop: Their interfaces are compatible.

We assume the standard firing semantics for transitions, thus $\mu \xrightarrow{t} \mu'$ means a *step* from μ to μ' by *firing* t. A transition sequence $t_0 t_1 t_2 \dots$ is called *firing sequence of* N if there exists a sequence of markings $\mu_0 \mu_1 \mu_2 \dots$, such that $\mu_0 = \alpha$ and for each $i = 0, 1, 2 \dots$, $\mu_i \xrightarrow{t_i} \mu_{i+1}$ is a step of N. Note that for each firing sequence, there exists exactly one corresponding sequence of markings. We thus say that a finite firing sequence $t_0 t_1 t_2 \dots t_n$ *ends* in μ_{n+1}. We call a finite firing sequence *terminating*, if it ends in a final marking. The Parikh vector [5] of a transition sequence r is denoted as $\mathrm{occ}(r)$. $r_{|T'}$ denotes the restriction of r to elements of T', which we canonically extend

to Parikh vectors. The *behavior* of an open net is the set of all its firing sequences, denoted as $\mathrm{Beh}(N)$. The set of all firing sequences ending in a marking μ is denoted as $\mathrm{Beh}(N, \mu)$.

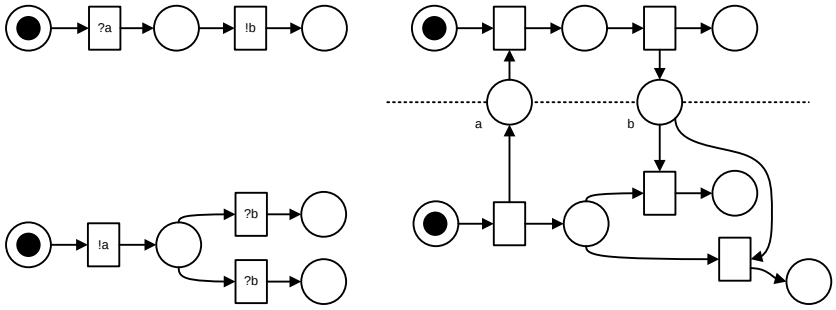


Fig. 3. Two open nets and their composition

The *composition* of two partners N and N' to the open net $N \oplus N'$ is realized by introducing buffer places and corresponding arcs as depicted in Fig. 3. Inspecting the structure of $N \oplus N'$, we find that a firing sequence of $N \oplus N'$ corresponds to one firing sequence of N as well as one of N', allowing us to analyze N in isolation and draw conclusions for the behavior of N in $N \oplus N'$.

Thus, let N, N' be partners and μ, μ' be markings of N, N' respectively. Then, $r \in \mathrm{Beh}(N \oplus N', \mu + \mu')$ implies $r_{|T} \in \mathrm{Beh}(N, \mu)$ and $r_{|T'} \in \mathrm{Beh}(N', \mu')$.

We now introduce *costs* for actions of services into our formal model. Each transition t has a *globally fixed integer cost*, denoted as $\mathrm{cost}(t)$. The cost of a transition sequence only depends on its Parikh vector: $\mathrm{cost}(r) = \mathrm{cost}(\mathrm{occ}(r)) = \sum_{t \in T} \mathrm{occ}(r)(t) \cdot \mathrm{cost}(t)$. Let μ be a marking of an open net N. If $\mathrm{Beh}(N, \mu) \neq \emptyset$, then the minimal and maximal cost of μ in N are defined as $\min(\{\mathrm{cost}(r) \mid r \in \mathrm{Beh}(N, \mu)\})$ and $\max(\{\mathrm{cost}(r) \mid r \in \mathrm{Beh}(N, \mu)\})$, denoted as $\bot(N, \mu)$ and $\top(N, \mu)$ respectively. Otherwise, the minimal and maximal cost are undefined. Let $T' \subseteq T$, then $\bot(N, \mu)_{|T'}$ and $\top(N, \mu)_{|T'}$ denote the minimal and maximal cost for μ by only taking into account transitions in T'.

As an example, consider that the transitions $t_1, \ldots, t_8$ of the open net shop in Fig. 1 have costs of $10, 5, 1, 10, 5, 20, 20, 1$ respectively. Then, the firing sequence $t_1 t_2 t_3 t_2 t_3 t_4$ has a cost of 32. For marking $[p_3]$ the cost interval is $(15, \infty)$.

3 Cost estimation

The *state equation* [2] is a proven tool in Petri net analysis: Let as usual ${}^\bullet x$ ($x^\bullet$) denote the *preset* (*postset*) of a node $x \in P \cup T$. The $P \times T$-matrix I_N, the *incidence matrix* of N, is defined as follows: $I_n(p, t) = -1$ if $p \in {}^\bullet t \setminus t^\bullet$, $I_n(p, t) = 1$ if $p \in t^\bullet \setminus {}^\bullet t$ and $I_n(p, t) = 0$, otherwise. Let y be a vector with $y(p) = \mu(p) - \alpha(p)$, $p \in P$. Then, $\mathrm{X}(N, \mu)$ denotes *all non-negative integer solutions* of the state equation $I_N \cdot x = y$ of N

with respect to marking μ. The state equation for the open net shop in Fig. 1 is displayed in Table 1.

Table 1. State equation of the open net shop (Fig 1) w.r.t. marking [done].

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	y
initial	-1								-1
p_1	+1			-1	-1				
p_2		+1	-1						
p_3					+1	-1			
p_4					+1		-1		
p_5						+1		-1	
p_5							+1	-1	
done								+1	+1
abort				+1					

The Parikh vector of any firing sequence ending in μ is a solution of the state equation. As an example, consider open net shop in Fig. 1: $t_1t_2t_3t_2t_3t_5t_6t_7t_8$ is a firing sequence ending in the marking [done]. Its Parik vector is $\left(1\ 2\ 2\ 0\ 1\ 1\ 1\ 1\right)$, which is a solution of the state equation. The reverse does not hold for the general case: By solving the state equation tokens can be "borrowed" from and "given back" to places, leading to an effect of 0, although no firing sequence with this solution as a Parikh vector exists. Obviously, $\mathrm{X}(N, \mu)$ is a valid over-approximation of $\mathrm{occ}(\mathrm{Beh}(N, \mu))$. Because the cost of a firing sequence is only dependent on its Parikh vector, the state equation is a valid mechanism to estimate costs for a given marking.

In this section, we present two approaches: First, we show how for *two given services* the costs for one provider can be estimated by solving the state equation of the composition of their models. Then, we explain how this approach can be generalized to the setting where the second partner is only represented by a set of *constraints*. For the second approach, we can shift the analysis effort to a time-uncritical phase. Ensuing, we have a critical look at both approaches, collecting results for the open net shop in Fig. 1. Finally, we draft how our general approach can be extended from estimating costs for a given state to a given (even infinite) set of states.

3.1 Estimating costs for one provider in a specific composite

In the following let N (with its transition set T) and N' be partners and ω, ω' be final markings of N and N', respectively. From the property of the state equation follows:

Lemma 1. $\mathrm{occ}(\mathrm{Beh}(N \oplus N', \omega + \omega'))_{|T} \subseteq \mathrm{X}(N \oplus N', \mu)_{|T}$.

Directly from set algebra follows that the state equation of $N \oplus N'$ provides a *valid over-approximation* for the costs of the provider of N in $N \oplus N'$:

Theorem 1. $\mathrm{Beh}(N \oplus N', \omega + \omega') \neq \emptyset$ *implies* $\min(\mathrm{cost}(\mathrm{X}(N \oplus N', \omega + \omega')_{|T})) \leq \bot(N \oplus N', \omega + \omega')_{|T} \leq \top(N \oplus N', \omega + \omega')_{|T} \leq \max(\mathrm{cost}(\mathrm{X}(N \oplus N', \omega + \omega')_{|T}))$.

Thus, the costs for the provider of N can be estimated by solving the linear problem for each final marking of $\Omega \times \Omega'$. From our experience, the number of final markings is very small, if not 1. As an example, evaluating the state equation for the composite $\mathsf{shop} \oplus \mathsf{customer}$, we find two cost intervals: $(56, 74)$ for the final marking $[\mathsf{done} + \mathsf{done}']$ and $(20, 38)$ for $[\mathsf{aborted} + \mathsf{done}']$, which we can compose to an *overall cost estimation* of $(20, 74)$. We observe, that this is the actual cost interval. Obviously, the approach is symmetrical: The costs for the provider of N' can be estimated analogously. We will now generalize our approach from a completely known open net N' to a set of constraints, describing a set of services.

3.2 Pre-estimating costs for one provider

Assume now that not an open net N' is given but a *set of constraints* of the following syntax and semantics: A constraint is an *inequality* or *equation* $l * r$ where $* \in \{\leq, \geq, =\}$, consisting of an *integer linear combination* l *of event labels* and an *integer value* r. An open net N' *solves* $l * r$ if for any terminating firing sequence in N', its event occurrence vector solves $l * r$. N' *solves* the set Ψ of constraints, if N' solves each $\psi \in \Psi$. We denote the set of open nets solving Ψ with $\mathcal{B}_\Psi$. As an example, the open net shop in Fig. 1 solves the constraints $?\mathsf{login} = 1$ and $?\mathsf{add} - !\mathsf{added} = 0$. [3] shows how such a set of constraints can be obtained from an open net by static analysis.

Intuitively, we take the state equation of N and *augment* it correspondingly with the given set of constraints Ψ: Because every message has been consumed in a final marking of the composite, we find that for each sent message of a partner of N, N has received the message and vice versa. Thus, for each message x sent (received) by N', a transition receiving (sending) x of N has fired once. This induces a set of constraints $\widehat{\Psi}(N) = \{\sum_{t \in T} l(ev(t)) \cdot t * r \mid l * r \in \Psi\}$ on the transition occurrence of N. As an example, consider the open net shop in Fig. 1 and assume $\Psi = \{!\mathsf{add} \leq 10\}$ as well as $N' \in \mathcal{B}_\Psi$. Then, in any terminating firing sequence of $N \oplus N'$ the only transition with label $?\mathsf{add}$, namely t_2 fires only up to 10 times, thus $\widehat{\Psi}(N) = \{t_2 \leq 10\}$.

Therefore, the state equation of N with respect to ω together with $\widehat{\Psi}(N)$, its solutions denoted as $\mathrm{X}_\Psi(N, \omega)$, over-approximates all firing sequences ending in $\omega + \omega$ of $N \oplus N'$ if $N' \in \mathcal{B}_\Psi$.

Lemma 2. *$N' \in \mathcal{B}_\Psi$ implies* $\mathrm{Beh}(N \oplus N', \omega + \omega')_{|T} \subseteq \mathrm{X}_\Psi(N, \omega)$.

Analogously to our approach with full knowledge of N', we can conclude that for a given Ψ, the costs for the provider of N for N interacting with an $N' \in \mathcal{B}_\Psi$ can be *over-approximated* by using the state equation:

Theorem 2. *$N' \in \mathcal{B}_\Psi$ and* $\mathrm{Beh}(N \oplus N', \omega + \omega') \neq \emptyset$ *implies* $\min(\mathrm{cost}(\mathrm{X}_\Psi(N, \omega))) \leq \bot(N \oplus N', \omega)_{|T} \leq \top(N \oplus N', \omega)_{|T} \leq \max(\mathrm{cost}(\mathrm{X}_\Psi(N, \omega)))$.

Thus, solving the linear program for each $\omega \in \Omega$ yields a *valid cost estimation*. Expecting $|\Omega|$ to be small, this approach seems feasible for use in practice. Furthermore, we draft in Sect. 3.4 how a valid cost estimation can directly be computed for a set of final markings.

3.3 A critical look

We have collected a small number of results in Table 2. The table shows for the open net shop in Fig. 1 how the *actual* cost intervals $(c_{\min}, c_{\max})$ and the *estimated* cost intervals $(x_{\min}, x_{\max})$ for a selected final marking ω correspond for partners specified by Ψ. The results were gained manually; there does not exist an implementation yet. In the example, the estimated cost interval is identical with the actual cost interval for any Ψ. We also see that open net customer in Fig. 2 solves $\{1 \leq \mathsf{!add} \leq 10\}$. Composing the second and fifth row in the table, we find an estimated cost interval of $(20, 116)$, which is not as tight as the former result $(20, 74)$, but still valid.

In the general case however, results are not necessarily this precise. A factor are *t-invariants*, transition vectors x, such that $I_N \cdot x = 0$. For the open net shop in Fig. 1, $\big(0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\big)$ is a t-invariant. Obviously, if y is a solution of state equation S and x is a t-invariant, then for any $n \in \mathbb{N}$ holds that $y + n \cdot x$ is also a solution of S. One can create an example with a t-invariant that never fires, intuitively an unmarked loop structure. Thus, if a t-invariant x exists and $\mathrm{cost}(x) > 0$ ($\mathrm{cost}(x) < 0$), cost estimation yields unbounded for the upper (lower) bound, unless there is an additional constraint bounding it. For the t-invariant $x = \big(0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\big)$, $\mathrm{cost}(x)$ is greater than zero. *Constraining* it through the constraint set $\{1 \leq \mathsf{!add} \leq 10\}$ bounds the cost interval because x can not be added arbitrarily often due to bounding the occurrence t_2. It is up to future work to find a way to handle t-invariants and to use them as a starting point for narrowing down interesting Ψ.

Table 2. Minimal/maximal costs for the provider of open net shop (Fig. 1)

Ψ	ω	$c_{\min}$	$c_{\max}$	$x_{\min}$	$x_{\max}$
$\{\}$	[done]	56	∞	56	unbounded
$\{0 \leq \mathsf{!add} \leq 10\}$	[done]	56	116	56	116
$\{\mathsf{!abort} = 1\}$	[done]	undefined	undefined	infeasible	infeasible
$\{\}$	[aborted]	20	∞	20	unbounded
$\{0 \leq \mathsf{!add} \leq 10\}$	[aborted]	20	80	20	80
$\{\mathsf{!abort} = 1\}$	[aborted]	20	∞	20	unbounded

3.4 Costs for sets of markings

Solving one linear program for *each* $\omega \in \Omega$ is not feasible for big or even infinite Ω. For the case that we can express or approximate the given set of markings as a *set of linear constraints*, we build a system of linear inequalities similar to the state equation and use the same techniques as described above.

4 Conclusion

In this paper, we presented an approach to *estimate* the costs for the provider of a service A for interacting with a service B. Thereby, we concentrated on two scenarios: Either

the partner B is known in detail, then the estimation process boils down to static analysis of the *composed* system $A \oplus B$. Or, B is not known and only narrowed down by a set of *constraints* Ψ. In this case, we analyze the behavior of A in *isolation*, constraining it according to Ψ. The downside of static analysis is a loss of precision, for which we identify t-invariants as an important factor. Still, we find that our approach tackles the first and the second challenge introduced in Sect. 1. For the case of a not reachable final marking, our approach might yield a cost interval although it is not defined. We consider this a low price to pay in contrast to the state space explosion problem.

5 Related and future work

In [6, 7] the authors analyze BPMN [8] models. For minimal and maximal costs [6] uses the *Dijkstra* algorithm with complexity $O(n \log n)$. Our approach is based on the *simplex* algorithm which is known to normally have linear runtime, although worst-case complexity is exponential. In the case of acyclic services, the state equation approach is even sufficient, so we also find strict bounds. The approach in [7] is pattern-based, an approach not applicable to arbitrary graph-based formalisms such as BPMN or open nets.

In this paper, we only considered a single fixed cost for each atomic action of a service. As a start it is intuitive that a certain step has always the same costs. A natural extension would be allow intervals [6] or even stochastic values [9]. However, as long as the cost functions are still linear, the extension is canonical. Additionally, costs might be history-dependent. An action might have some fixed and some execution costs, such that a consecutive execution of this action might be less expensive. Furthermore it would be interesting to use the approach as a decision help for service discovery. As a base we can imagine a cost profile stored in a service repository, including cost estimations and acceptable cost intervals, inducing a concept of compatibility under costs.

References

1. Papazoglou, M.P.: Web Services: Principles and Technology. Pearson - Prentice Hall, Essex (July 2007)
2. Lautenbach, K.: Liveness in Petri Nets. St. Augustin: Gesellschaft für Mathematik und Datenverarbeitung Bonn, Interner Bericht ISF-75-02.1 (1975)
3. Sürmeli, J.: Profiling services with static analysis. In Freytag, T., Eckleder, A., eds.: AWPN. (2009)
4. Reisig, W.: Petri Nets: An Introduction. Volume 4 of Monographs in Theoretical Computer Science. An EATCS Series. Springer (1985)
5. Parikh, R.: On context-free languages. J. ACM **13**(4) (1966) 570–581
6. Magnani, M., Montesi, D.: BPMN: How much does it cost? An incremental approach. In: BPM. (2007) 80–87
7. Sampath, P., Wirsing, M.: Computing the cost of business processes. In: UNISCON. (2009) 178–183
8. OMG: Business Process Model and Notation 2.0. (August 2009)
9. van Hee, K.M., Verbeek, H.M.W., Stahl, C., Sidorova, N.: A framework for linking and pricing no-cure-no-pay services. T. Petri Nets and Other Models of Concurrency **2** (2009) 192–207

Author Index