Daniel Eichhorn Agnes Koschmider Huayu Zhang (Eds.)

Services und ihre Komposition

Proceedings of the 3rd Central-European Workshop on Services and their Composition, ZEUS 2011 Karlsruhe, Germany, February 21/22

CEUR Workshop Proceedings Vol. 705

3. Zentral-europäischer Workshop über Services und ihre Komposition
3nd Central-European Workshop on Services and their Composition

ZEUS 2011

Daniel Eichhorn, Agnes Koschmider and Huayu Zhang, Editors
Karlsruhe Institute of Technology
Institute of Applied Informatics and Formal Description Methods
Building 05.20
76128 Karlsruhe, Germany
{daniel.eichhorn | agnes.koschmider | huayu.zhang}@kit.edu

http://CEUR-WS.org/Vol-705/

BIBTEX

```
@proceedings{zeus2011,
     editor = {Daniel Eichhorn and Agnes Koschmider and Huayu
               Zhang},
     title = {Proceedings of the 3rd Central-European
     Workshop on Services and their Composition,
     ZEUS 2011, Karlsruhe, Germany,
     February 21--22, 2011},
     booktitle = {Services und ihre Komposition},
     publisher = {CEUR-WS.org},
     series = {CEUR Workshop Proceedings},
     volume = {705},
     year = {2011},
     url = {http://CEUR-WS.org/Vol-705/}
}
```

Vorwort

Der Zentral-europäische Workshop über Services und ihre Komposition (ZEUS) hat zum ersten Mal im März 2009 in Stuttgart stattgefunden. 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 wurde ZEUS fortgeführt und fand als zweiter ZEUS-Workshop im Februar 2010 an der Humboldt-Universität zu Berlin statt.
Um das inzwischen bewährte Konzept fortzuführen, wurde der dritte Zentral-europäische Workshop am 21.- 22. Februar in Karlsruhe veranstaltet.
Von den eingereichten Beiträgen haben wir 17 in das Programm aufgenommen. Die eingereichten Beiträge wurden in einem Begutachtungsprozess von jeweils drei Gutachtern bewertet. Das aus akademischem Bereich und Wirtschaft stammende Programmkomitee hat alle Beiträge auf Relevanz hin überprüft. Ziel der Begutachtung war es, den Autoren detaillierte Hinweise und Anregungen zu Inhalt und Qualität ihres Beitrags zu geben.
Das ausgewählte Programm bat genügend Stoff für rege Diskussionen. Jedem Teilnehmer wurden wertvolle Anregungen mit nach Hause gegeben und das Forum hat neue Kontakte über die eigene Forschungsgruppe hinaus entstehen lassen.

Für Ihre Beteiligung am diesjährigen ZEUS-Workshop möchten wir uns bei allen Autoren bedanken. Bei Prof. Dr. Stefan Tai (Institut für Angewandte Informatik und Formale Beschreibungsverfahren, Karlsruher Institut für Technologie) möchten wir uns für seine Keynote besonders bedanken.

März 2011 Daniel Eichhorn, Agnes Koschmider und Huayu Zhang

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 upon the success of the first ZEUS workshop, it was decided to stick to this conception for the second ZEUS workshop, which took place at the Humboldt-Universität zu Berlin. To keep the successful concept running the third ZEUS workshop took place on February 21 and 22 2011 in Karlsruhe.
We selected 17 submissions for the workshop program. Each submission has been peer reviewed by three members of the program committee. The program committee consisting of members of academics and economy checked submissions with respect to relevance. However, the main goal of the reviewing process was providing the authors useful hints and feedback on the quality of their submissions.
We are convinced that all talks provided a good basis for valuable discussions and the workshop will allowed establishing new contacts between the participants as well as equipped each participant with valuable feedback and ideas to take home.

We would like to thank each participant for attending ZEUS 2011. A special thank goes to Prof. Dr. Stefan Tai (Institute of Applied Informatics and Formal Description Methods, Karlsruhe Institute of Technology) for giving a keynote speech at the workshop.

March 2011 Daniel Eichhorn, Agnes Koschmider und Huayu Zhang

Organizers

Daniel Eichhorn, Karlsruher Institut für Technologie
Huayu Zhang , Karlsruher Institut für Technologie

Program Committee

Sudhir Agarwal, KSRI - Karlsruhe Service Research Institute
Gero Decker, signavio
Daniel Eichhorn, Karlsruher Institut für Technologie
Christian Gierds, Humboldt-Universität zu Berlin
Thomas Hornung, Albert-Ludwigs-Universität Freiburg
Oliver Kopp, University of Stuttgart
Agnes Koschmider, University of Pretoria
Niels Lohmann, University of Rostock
Christian Stahl, Eindhoven University of Technology
Jan Sürmeli, Humboldt-Universität zu Berlin
Matthias Weidlich, Hasso Plattner Institute Potsdam
Huayu Zhang, Karlsruher Institut für Technologie

Contents

Session 1

Session 2

Session 3

Session 4

Session 5

Many-to-Many: Some Observations on Interactions in Artifact Choreographies

Dirk Fahland, Massimiliano de Leoni,
Boudewijn F. van Dongen, and Wil M.P. van der Aalst

Eindhoven University of Technology, The Netherlands
(d.fahland|m.d.leoni|b.f.v.dongen)@tue.nl, w.m.p.v.d.aalst@tm.tue.nl

Abstract. *Artifacts* have been proposed as basic building blocks for complex processes that are driven by life-cycle aware data objects. An *artifact choreography* describes the interplay of several artifacts from which the process "emerges". By design, an artifact choreography is tightly coupled to the process' underlying data model which gives rise to complex interactions between artifacts. This paper presents a simple model for these interactions and outlines open challenges in artifact choreographies.

Keywords: artifacts, choreography, interaction, synchronous, asynchronous

1 Introduction

The *artifact-centric* approach emerged in the last years as an alternative approach for precisely describing complex inter-organizational processes in a modular way [1–4]. The approach assumes that a process is driven by its data objects, called *artifacts*. Each artifact has its own *life-cycle* and can interact with other artifacts. In a service-oriented setting, each artifact's state can be updated by other artifacts via a well-defined *interface*, and the entire inter-organizational process follows from a *choreography* of its artifacts [3, 4].

What pushes artifact-centric choreographies beyond service choreographies is their tight coupling to the process' underlying data model. A process typically exhibits *many-to-many relationships* between its different data objects. For example, an order at an online-shop may be delivered in several packages where each package is delivered in a different truck. In turn, each truck usually delivers several packages of different orders. An artifact choreography inherits these many-to-many relations as a first-class concept: it describes how *several instances* of one artifact (e.g., order) interact with *several instances* of another artifact (e.g., deliveries).

This paper is devoted to explaining the subject of many-to-many relationships in artifact choreographies in more detail, and to highlighting specific challenges that arise in this setting. Using an instructive example, we present in Sect. 2 a *minimal* extension of service models that expresses *cardinality constraints* between artifact instances. This simple extension yields behavioral phenomena that only arise in the artifact-centric setting. We study these phenomena in Sect. 3 and note that a complete artifact choreography also must describe *stateful*

interaction protocol between artifact instances, and *which* instances interact with each other. We then show that the interaction between artifact instances can itself be expressed as a meaningful *coordinating artifact* that becomes part of the choreography. We conclude the paper in Sect. 4 by outlining two open research problems: (1) an automated generation of coordinating artifacts, and (2) ways to fully specify dynamic synchronization of artifact instances.

2 The Artifact-Centric Approach

The artifact-centric approach [1, 2] aims at a "more natural" approach of describing complex inter-organizational processes. Any process materializes itself in the artifacts (i.e., objects) that are involved in the process, and the artifacts' states. Examples of artifacts are a paper form, an electronic order, a package, or a delivery truck. State changes of an artifact usually follow a specific *life-cycle*: an artifact is *instantiated*; the state of an instance *changes* only via actions provided by the artifact; each artifact instance eventually reaches a *goal state* (e.g., a form gets signed, or a package is delivered). The key idea of the artifact-centric approach is that by just describing the artifacts' life-cycles and relations between artifacts, the process simply "emerges" from interactions of its artifact instances.

To better understand the subject, we consider the following example of an online shop's delivery process driven by 2 artifacts: *order* and *delivery* tour. The shop splits each order into several packages based on the availability of the ordered items. Several packages from different orders are then delivered in one tour. In case a package cannot be delivered, it is scheduled for another delivery tour or returned to the shop as undeliverable. The order is billed to the customer once all packages are processed. This behavior can be described by (1) for each artifact

We formally describe this process in an *artifact choreography* by describing the life-cycles of the artifacts *order* and *delivery* and how instances of these interact with each other. Many techniques are available for this purpose [5, 2–4]. Here, we employ *proclets* [5] as a formal model. Proclets minimally extend operational service models, e.g., [4], with *cardinality constraints* to express relations between artifact instances.

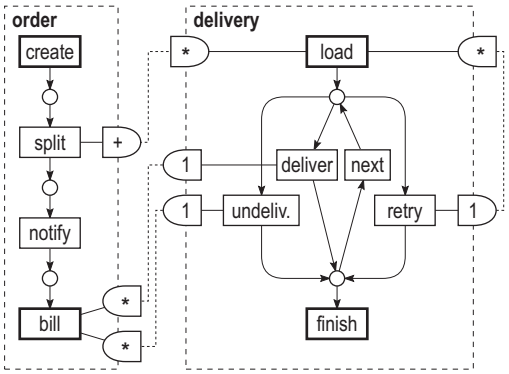


Fig. 1. An artifact choreography of a delivery process where orders can be split into multiple deliveries that can be retried and that can fail.

Artifact life-cycles, ports, and choreographies. To begin with, one proclet describes the life-cycle of one artifact as a *Petri net*. Figure 1 shows the life-cycles of an *order* and of a *delivery* tour inside the respective dashed boxes; the additional modeling

elements will be explained subsequently. A customer creates an order that is split into several packages by the availability of the ordered items; the order completes by notifying the customer about the order and sending the bill. A delivery tour begins with loading a delivery truck with all packages of the tour: each package is delivered, rescheduled for another another tour (retry), or declared as undeliverable before the next package is processed, until the tour finishes.

As the overall process follows from an *interaction* of orders and delivery tours, each proclet exposes some of its actions to other proclets via a *port*. A *proclet choreography* defines *channels* between proclet ports which describe how proclet instances interact with each other, e.g., by exchanging messages. The decisive difference to a service choreography comes by port *annotations* $(1, +, *)$ which specify how many messages an action sends to or receives from other proclet instances.

Formally, a proclet is a Petri net extended by ports; a choreography is a set of proclets with channels between ports.

Definition 1 (Proclet). *A* proclet $P = (N, ports)$ *is a Petri net* $N = (S, T, F)$ *with ports* $\subseteq 2^T \times \{in, out\} \times \{1, *, +\}$ *where each port* $p = (T_p, dir, card)$

1. *is associated to a set* $T_p \subseteq T$ *of transitions;*
2. *has a direction of communication (in: incoming port, the associated transitions receive a message, out: outgoing port, the associated transitions send a message);*
3. *has a* cardinality *card* $\in \{1, *, +\}$ *specifying how many messages may or have to be sent or received upon an occurrence of one* $t \in T_p$.

Definition 2 (Artifact choreography).
An artifact choreography $(\{P_1, \ldots, P_n\}, C)$ *consists of a finite set* $\{P_1, \ldots, P_n\}$ *of proclets together with a set* C *of channels s.t. each channel* $(p, q) \in C$ *is a pair of ports* $p, q \in \bigcup_{i=1}^{n} \text{ports}_i$ *with direction of* p *being in and direction of* q *being out.*

In Fig. 1, a half-round shape denotes a port and a dashed line a channel between two ports, e.g., there is a channel from split to load.

Artifact instances. During a process execution, artifacts of the choreography are *instantiated* and instances change their states according to the artifact lifecycle. We generally assume that each proclet P has a unique transition with an empty pre-set (no incoming arcs), and a unique transition with an empty post-set, which describe the creation and termination of instances, respectively. For example, an occurrence of create instantiates a new order of Fig. 1, an occurrence of bill terminates the instance; an instance of delivery is created by load and terminated by finish.

Data model and cardinality constraints. Yet, the notion of an artifact *instance* is much more crucial in artifact choreographies than in service choreographies. The artifacts describe the objects that drive the process. The process' *underlying data model* determines how many instances of one artifact (e.g., *order*)

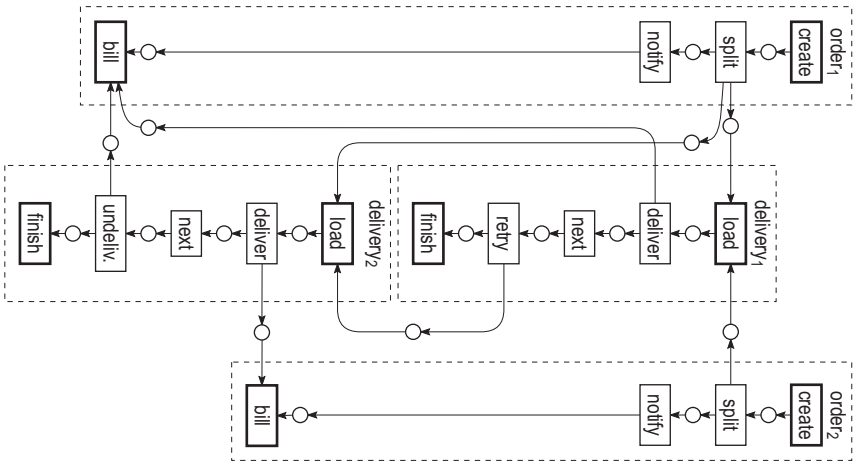


Fig. 2. A partially ordered run of the artifact choreography of Fig. 1.

may or must be related to how many instances of another artifact (e.g., *delivery*). For example, each *order* is delivered in one or more *delivery* tours (because it can be split), each *delivery* tour handles packages of several *orders*, and a delivery attempt of a tour can have a *follow-up* delivery tour.

In the artifact-centric setting, the process is driven by its artifacts. Hence, any two artifact instances that are related to each other also have to *interact* as the process evolves. The decisive contribution by proclets [5] is to incorporate this underlying data model of the process in the interaction specification. The *annotations* $(1, +, *)$ at the source and target ports of a channel from proclet A to proclet B specify how many instances of A interact with how many instances of B via the channel. This way, the port annotations at a channel define *cardinality constraints* on artifact instances.

For example, an order instance is split into one or more packages, each being handled by a different delivery instance (annotation $+$). Conversely, each delivery instance loads on a delivery truck packages from several $(*)$ order instances and from several $(*)$ previous delivery instances. The packages are delivered one by one: in case of success a single notification (1) is sent to the order instance; in case of failure the single package (1) is either handed over to the order instance or a follow-up delivery instance. These instances in turn collect all incoming notifications or packages $(*)$ before proceeding.

Figure 2 shows an execution of the process as a *partially ordered run* [6]. The execution involves two instances of order and two instances of delivery; order_1 is split into two packages, one handled by $\mathsf{delivery}_1$ and one by $\mathsf{delivery}_2$; the only package of order_2 cannot be delivered in the first attempt and hence is rescheduled to participate in $\mathsf{delivery}_2$ that also handles the second package of order_1. Behavior of this kind naturally arises in an artifact-centric setting and cannot be expressed with service choreographies.

3 Interaction in Artifact Choreographies

When relating the partially ordered run of Fig. 2 to the artifact choreography of Fig. 1 we see that the run satisfies all requirements of the choreography model. Yet, the run also exhibits crucial properties that are not reflected in the model.

(1) The choreography allows a variant of the run of Fig. 2 where the undelivered second package of order_1 is just dropped and not handed over to order_1. In another variant $\mathsf{delivery}_2$ could send 6 messages to order_2 instead of 1. Both kinds of runs are intuitively undesired and should be excluded. Intuitively, not only each artifact instance has a life-cycle to complete, but also *each artifact interaction has a life-cycle* to complete; such a life-cycle is not specified in the choreography.

(2) The choreography would also allow for a run that hands the undelivered package of $\mathsf{delivery}_2$ over to order_2 instead of its original order_1. Although the interaction completes, it completes with the wrong participants. Likewise, one can easily think of a process where such a forwarding of packages to another artifact instance is required. Currently, the choreography does not specify *which* instances interact with each other, but only *how many*.

In other words, the many-to-many relations between artifact instances require a more detailed artifact model than just expressing cardinality constraints.

In particular, the language has to describe (1) the life-cycle of an interaction between artifact instances, and (2) which instances synchronize with each other.

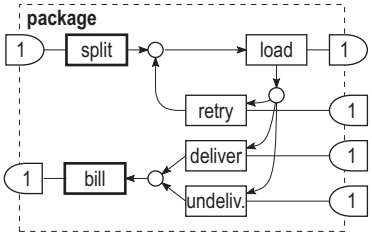


Fig. 4. The artifact package models the life-cycle of an interaction between an instance of order and an instance of delivery (Fig. 1), but not which instances synchronize with each other.

Artifact interaction life-cycle. In the following, we show that the desired interaction between artifact instances can easily be described by the life-cycle of a new, meaningful artifact. Figure 3 decomposes the desired run of Fig. 2 in a specific way. Instead of considering an *asynchronous interaction* between orders and deliveries, Fig. 3 describes exactly the same behavior as Fig. 2 in terms of *synchronous interaction* of orders with packages and packages with deliveries. The dashed lines describe which transitions occur *synchronously*, e.g., split of order_1, $\mathsf{package}_1$, and $\mathsf{package}_3$ occur *synchronously*.

The synchronous interactions caught in the packages describe how we expect the artifact-interactions to complete. These can easily be modeled as a separate artifact package as shown in the proclet of Fig. 4. The package interacts on either side with *exactly one* instance of order and delivery, i.e., it describes the life-cycle of one interaction between two related instances. The choreography of Fig. 1 can be refined to reflect this artifact-interaction life-cycle by placing proclet package between order and delivery. The refinement comes with a paradigm shift: a channel

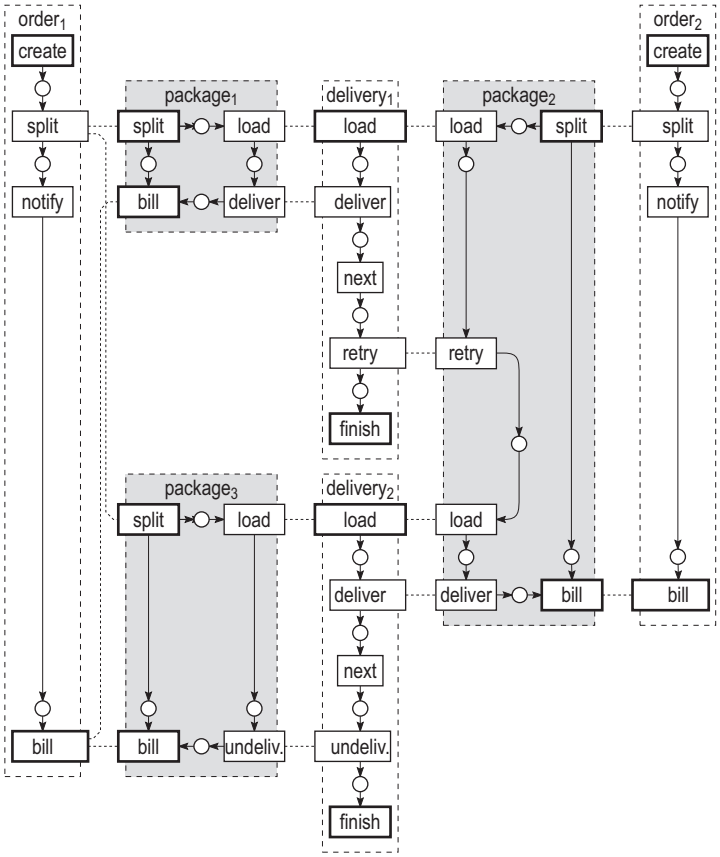


Fig. 3. The messages exchanged between the artifacts in Fig. 2 follow the packages handled in the process.

between two transitions now specifies *synchronous occurrences of transitions* instead of message exchange.

4 Conclusion: Data Specification at the Interaction Level

In this paper, we have shown that artifact choreographies naturally describe behavior that cannot be expressed by services. By lifting the underling data model to the behavioral specification, artifact choreographies particularly express many-to-many relations between artifact instances. Section 3 showed that a *complete* artifact choreography requires to specify *life-cycles of artifact interactions*. To this end, a choreography can be refined with further artifacts.

Two main challenges remain open in this context. First, a choreography description language needs to describe *which* instances interact with each other. It particularly needs to express that an instance A_1 of an artifact A synchronizes with instances $B_1, \ldots, B_k$ of an artifact B which possibly have not been created yet. In our example, a *delivery* is only instantiated *after* all participating *orders*

have been split. A possibility could be to adapt WS-BPEL's *correlation handling* mechanism [7] to the artifact-centric setting.

Second, as artifact interactions can be very complex, it may be reasonable to *synthesize* the artifacts that describe artifact interaction life-cycles. An approach from controller synthesis allows to automatically complete a given choreography in case of 1-to-1 relations [4]. It is worth exploring whether the approach can be leveraged to many-to-many relations.

Alternatively, *process mining* techniques [8] might be applied in this context. Process mining comprises techniques to discover process models from observed behaviors. Such behaviors are extracted from the execution logs of running systems. For an artifact-centric setting, the recorded executions would contain events of artifact-life cycles as well as of artifact interactions. So, execution logs could be an alternative source of information to obtain artifact interaction life-cycles which then lead to a complete artifact choreography.

In controller synthesis, as well as in process mining, the open problem is concerned with the fact that these techniques assume service instances to work in isolation w.r.t. other instances for the same service. In this paper, we have shown that artifact choreographies introduce many-to-many relations among artifacts, which do not exist in traditional service-oriented approaches. As a consequence, the definition of a case concept needs to be rethought. For instance, coming back to the working example, there is no evident preference to consider a diverse case for each order, rather than for each delivery. Every order is associated to several delivery, but also every delivery is associated to several orders.

Acknowledgements. The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement n° 257593 (ACSI).

References

1. Nigam, A., Caswell, N.: Business artifacts: An approach to operational specification. IBM Systems Journal **42** (2003) 428–445
2. Cohn, D., Hull, R.: Business artifacts: A data-centric approach to modeling business operations and processes. IEEE Data Eng. Bull. **32** (2009) 3–9
3. Fritz, C., Hull, R., Su, J.: Automatic construction of simple artifact-based business processes. In: ICDT'09. Volume 361 of ACM ICPS. (2009) 225–238
4. Lohmann, N., Wolf, K.: Artifact-centric choreographies. In: ICSOC 2010. Volume 6470 of LNCS., Springer (2010) 32–46
5. van der Aalst, W., Barthelmess, P., Ellis, C., Wainer, J.: Proclets: A Framework for Lightweight Interacting Workflow Processes. Int. J. Cooperative Inf. Syst. **10** (2001) 443–481
6. Engelfriet, J.: Branching processes of Petri nets. Acta Informatica **28** (1991) 575–591
7. Web Services Business Process Execution Language Version 2.0, 11 April 2007. OASIS Standard (2007)
8. van der Aalst, W., Reijers, H., Weijters, A., van Dongen, B., Medeiros, A., Song, M., Verbeek, H.: Business Process Mining: An Industrial Application. Information Systems **32** (2007) 713–732

Do We Need a Refined Choreography Notion?

Andreas Schönberger

Distributed and Mobile Systems Group,
University of Bamberg
Bamberg, Germany
andreas.schoenberger@uni-bamberg.de

Abstract. Since the term *choreography* for capturing the publicly observable message exchanges between integration partners was coined, choreography technology evolved significantly. Today, the diversity of choreography languages is high. Up to now, choreography languages have been categorized by distinguishing between implementation specific and implementation independent choreographies as well as interaction and interconnection choreographies.
In this work, we review important characteristics of choreography technologies to find out whether a refined choreography notion is needed. The fact that choreography classes that are almost orthogonal to existing categorizations as well as several selective choreography characteristics can be identified suggests this need.

Keywords: B2Bi Choreography, Services Choreography, Conceptual Choreography

1 Introduction

In 2003, Chris Peltz coined the terms *Web Services Choreography and Orchestration* by distinguishing between *tracking the messages between* integration partners and the *executable local processes* of individual integration partners (cf. [11]). While Peltz tied the notion of *Choreography* to Web Services, today, there are a number of Web Services agnostic choreography languages such as UMM [18], ebXML BPSS (ebBP) [9] or Let's Dance [22]. However, capturing publicly visible messages between entities has remained as common characteristic of choreography languages.
Decker, Kopp and Barros [3] developed a categorization of choreographies based on two pivotal properties of choreography languages. First, they distinguish between *interconnection* choreographies that focus on the local send and receive actions of individual partners as well as the interconnection of corresponding send/receive actions and *interaction* choreographies that treat corresponding send and receive events as atomic actions and define sequences of these actions. Second, they distinguish between *implementation specific* choreographies that capture implementation level concepts like communication technology (say, Web Services) and *implementation independent* choreographies that are agnostic to

those concepts.
While this categorization for sure is pivotal it still captures languages with considerable differences in the same category. For example, Let's Dance and ebBP can both be characterized as implementation-independent interaction choreographies. However, ebBP targets at specifying the business document exchanges between enterprises while Let's Dance targets at supporting service interaction patterns [2] with a visual choreography language. Although these two goals overlap, they result in substantially different concepts. ebBP provides support for referencing existing business document libraries as provided by RosettaNet[1] and for specifying security and reliability requirements. Also, it assumes a protocol consisting of several message exchanges for implementing a business document exchange. Let's Dance, in turn, offers functionality for analyzing such protocols and provides a rich set of features for modeling service interactions.
These differences are a first hint that a refined choreography notion may be needed. This paper is dedicated to the investigation of that need. In Sect. 2, the analysis framework for conceptual modeling languages put forward by Wand and Weber [21] is used to derive *B2Bi/Services/Conceptual Choreographies* as distinct choreography classes that are largely orthogonal to the categorization presented in [3]. In Sect. 3, we identify 15 criteria that discriminate well between choreography categories. From these two results, we conclude that a refined choreography notion indeed is needed for helping practitioners and researchers in choosing a choreography language that fits their needs. Section 4 briefly discusses related work and Sect. 5 concludes and points out directions for future work.

2 Choreography Classes

In [21], Wand and Weber present an analysis framework for conceptual modeling. For comparing languages, the framework components *task factors* capturing the purpose of using a language as well as *modeling grammar* capturing the constructs and rules for creating models are relevant.
While choreography languages may lend itself to a variety of different purposes, it is striking that almost all choreography languages and approaches underline their relevance for Business-to-Business integration (B2Bi). Publications such as [7] and [15] that analyze the development phases of B2Bi and therefore are suited to identify *task factors* reveal that choreography technologies typically are used to fill the semantic gap between business process models (BPM) and orchestration models (OM). This can be done by refining BPMs or by abstracting OM concepts. For example, BPEL4Chor [4] reuses a considerable part of WS-BPEL concepts which corresponds to *abstracting OM concepts*. Conversely, ebBP uses so-called BusinessTransactions to specify requirements of message exchanges which corresponds to *refining the BPM layer*. Finally, for some choreography languages it is not easily decidable whether they are semantically more close to the BPM layer or to the OM layer. For example, IOWF-Nets [19] capture choreographies as interconnected Petri Nets. This resembles the concept of composing

[1] `http://www.rosettanet.org/`

a choreography by connecting orchestrations and therefore seems to imply a close relationship to the OM layer. However, the BPM layer may contain partner-local models as well and as IOWF-Nets do not have technology specific concepts, they could potentially be used for analyzing the BPM layer itself too.
These differences are also reflected in the core building blocks of the various choreography languages (cf. *modeling grammar* [21]). In BPEL4Chor, *communication activities* are used to capture the send and receive events of the individual partners. The WSDL interaction styles 'one-way' and 'request/response' are adopted to allow for *"higher similarity between participant behavior descriptions and orchestrations"* ([5], section 4.2). So, although BPEL4Chor is defined such that it does not *technically depend* on WSDL (by removing the *partnerLink*, *portType*, and *operation* attributes from BPEL communication activities), it can be concluded that BPEL4Chor is *designed for* services based interactions. In ebBP, a BusinessTransaction represents a B2Bi domain specific configuration of a business document exchange with B2Bi parameters for a lower-level execution protocol. Finally, the core building blocks of languages like IOWF-Nets or Let's Dance neither rely on Web Services concepts nor define B2Bi domain concepts.
On the basis of this analysis of *tasks factors* and *modeling grammar* as supposed by [21], at least three different classes of choreography languages (largely orthogonal to the categorization in [3]) can be identified:

B2Bi Choreographies that offer B2Bi specific concepts like configurable BusinessTransactions and which semantically are close to BPM models.
Services Choreographies that offer Web Services technology specific concepts and are close to the orchestration layer.
Conceptual choreographies that offer concepts driven by the purpose of analysis and may be used to complement/analyze the BPM layer as well as the OM layer.

3 Selective Criteria

While the last section shows that major different choreography classes can be distinguished by *task factors* and *modeling grammar elements*, this section identifies criteria that promise to discriminate between different categories of choreographies, i.e., which have the same value for some choreography languages but not for all. The criteria have been collected by leveraging two publications that postulate requirements for the important classes of services choreographies [5] and B2Bi choreographies [12] respectively and by reviewing design drivers of various choreography related publications, in particular [1,2,3,4,6,11,16,22].
The resulting criteria then have been filtered by removing criteria that do not discriminate well or can be derived almost functionally from other criteria. For evaluating selectiveness between choreography categories, the following representatives from the aforementioned choreography classes have been chosen that also represent all fields of the choreography categorization matrix from [3]:
IOWF-Nets [19] represent conceptual choreographies as well as implementation-

independent interconnection choreographies. Let's Dance [22] represents conceptual choreographies as well as implementation-independent interaction choreographies. ebBP [9] represents B2Bi choreographies as well as implementation-independent interaction choreographies. WS-CDL [20] represents services choreographies as well as implementation-specific interaction choreographies. Finally, BPEL4Chor [4] represents services choreographies as well as implementation specific interconnection choreographies.
While the identified criteria and the selected choreography languages for sure do not cover all aspects of choreography technology, the results described in Table 1 nonetheless demonstrate that there are several criteria that discriminate well between existing categories of choreographies. This, in turn, proves evidence for the fact that a refined choreography notion is needed. Note that the criteria of Table 1 are not intended as comparison framework for comparing in detail *choreography languages* of the same choreography class, but rather for distinguishing between *choreography categories*. While a single-author qualitative study (which always is biased to some extent) like the one at hand is sufficient for identifying the need for a refined choreography taxonomy, the development of a comprehensive choreography taxonomy calls for a joint effort of choreography researchers. Below, the individual criteria are presented:

1 Implementation Independence. Corresponds to the implementation specific/independent distinction as described in Sect. 1.
2 Communication Focus. Corresponds to the interconnection/interaction distinction as described in Sect. 1.
3 Core Design Driver. The core design driver of a choreography language can be inferred from its core building blocks and design rules and is frequently stated in related publications. For Let's Dance as well as BPEL4Chor, support for *Service Interaction Patterns* [2] is explicitly postulated as design driver in [22] and [5] respectively. For ebBP, *composition of BusinessTransactions* is the core design driver while *composition of interactions* apparently drove the design of WS-CDL. Finally, IOWF-Nets [19] result from extending the reach of WF-Nets to inter-organizational systems and therefore can be considered to be *formalism driven*. Obviously, choosing a core design driver does not uniquely determine language design.
4 Decomposability. Recursive decomposition of models is a frequent language design goal and is explicitly postulated in [12] and [22]. This criterion is represented as a yes/no value.
5 Distinction between participants and participant types. This criterion fosters *Service Interaction Patterns* support because it enables multiple instances of the same type of partner/service (cf. [22], [5]). We distinguish between *explicit* support and *no* support.
6 Domain. This criterion distinguishes between choreography languages that focus on an application domain such as *B2Bi* and *general* purpose languages.
7 Error Handling. This yes/no criterion is identified in both, [5] and [12], and is valued depending on the existence of explicit error handling concepts or

techniques.

8 Executability. Whether a model can be executed or not is a property of the particular model (or an according approach) and not of the language a model is composed from. However, there a different ways that choreography languages can be used. B2Bi choreographies are frequently used to just create a *cartography* of the types and scenarios of business document exchanges without intending to derive an implementation in a (semi-)automated manner. However, that does not exclude automated derivation of the control flow (cf. [13,16]). Moreover, a choreography may be used as a *blueprint* for identifying important parts of an implementation in a semi-automated manner [5]. Accordingly, *cartography*, *executable* and *blueprint* are possible values of this criterion.

9 Integration of Structural and Behavioral Views. Supporting the behavioral view on a system in the sense of constraining admissible message exchange sequences is a natural quality of choreography languages. However, some choreography languages additionally describe structural aspects such as the topology of services. Hence, *behavioral* and *integrated* can be assigned as values for this criterion.

10 Link Mobility. Some integration scenarios require the capability to pass on the endpoint of a communication partner to a third party. This capability, frequently denoted as link mobility, has been identified in [20] and [5] and is well-known from the π-calculus. This criterion is valued *explicit* or *no* support depending on the existence of dedicated link mobility constructs.

11 Processing Signals. For notifying a business document sender about the processability of the document, ebBP offers so-called Receipt-/AcceptanceAcknow -ledgements as *processing signals*. Identical concepts are also available in UMM [18] and the *Business Choreography Language* [23]. Note that processing signals are not first-class business messages as their existence depends on business document exchanges. This criterion is valued on a yes/no basis.

12 Protocol Abstraction. While Let's Dance or BPEL4Chor assume a one-to-one correspondence between a message exchange at the choreography level and its corresponding message exchange on the orchestration level, ebBP assumes a full communication protocol for implementing a single choreography exchange, i.e., a BusinessTransaction. *Protocol abstraction* captures the fact that a full protocol may be represented by a single exchange at the choreography level and accordingly is assigned a yes/no value.

13 Runtime Determination of Participants. [2] and [5] postulate the requirement for choreography languages to be able to leave the number of participant instances unspecified until runtime. This is different from *Link Mobility* as it does not necessarily require passing on communication endpoints. This criterion is valued *explicit* if the choreography language has explicit constructs for capturing that or *no* support otherwise.

14 Standardization. Although not a first-citizen property of languages, *standardization* of a language affects the selection of available constructs as well as its amenability to change. For industry and academia, whether or not a choreography language is a *standard* or a *research* prototype makes an important difference.

Table 1. Selective Criteria for Comparing Choreography Languages

Criterion	Let's Dance	ebBP	WS-CDL	BPEL4Chor	IOWF-Nets
1 Implementation Independence	independent	independent	specific	specific	independent
2 Communication Focus	interaction	interaction	interaction	inter-connection	inter-connection
3 Core Design Driver	interaction patterns	Business-Transaction composition	interaction composition	interaction patterns	formalism driven
4 Decomposability	yes	yes	yes	no	no
5 Distinction between participants and participant types	explicit	no	no	explicit	no
6 Domain	general	B2Bi	general	general	general
7 Error Handling	no	yes	yes	yes	no
8 Executability	cartography/ blueprint	cartography/ executable	cartography/ blueprint	blueprint	blueprint
9 Integration of Structural and Behavioral Views	behavioral	behavioral	integrated	integrated	integrated
10 Link Mobility	explicit	no	explicit	explicit	no
11 Processing Signals	no	yes	no	no	no
12 Protocol Abstraction	no	yes	no	no	no
13 Runtime Determination of Participants	explicit	no	no	explicit	no
14 Standardization	research	standard	standard	research	research
15 Transaction Safety	no	choice	choice	no	no

15 Transaction Safety. While ebBP by default assumes that business document exchanges are performed in a transactional manner, languages such as IOWF-Nets deliberately choose to separate sending messages from receiving messages and do not assume transaction safety. This is influenced by the fact that transaction safety for simple one-way or request/response interactions can easily be implemented using reliable messaging or distributed transaction features of the underlying middleware. Conversely, B2Bi business document exchanges at the choreography level may represent complex multi-message exchanges at the orchestration level [17,14] that reflect whether or not transaction safety is required explicitly. In [8], so-called *choreography spheres* are proposed to guarantee transaction safety for sets of choreography-level activities using advanced transaction features of the underlying BPEL engines. That approach can be applied on top of BPEL4Chor which does not provide built-in transaction safety support.

4 Related Work

In [11], the distinction between choreography and orchestration first was described. However, an analysis of different choreography classes is not provided. In [3], a categorization based on implementation independence and the interconnection/interaction dichotomy was proposed. The work at hand shows that

this categorization can be extended and complemented. In [24], requirements and language concepts for modeling cross-organizational business processes are identified. However, the focus is not put on choreographies in particular. Instead, the BPM layer and the OM layer are considered as well. Consequently, only 1 out of 7 requirements and 2 out of 7 language concepts distinguish well between categories of choreographies (evaluated for the languages used in table 1).
Finally, there is an abundance of publications postulating requirements for languages for particular purposes such as [5] for supporting service interaction patterns or [12] for B2Bi. However, these requirement sets are aligned with the design purpose and not with the intent to distinguish between choreography categories.

5 Conclusion and Future Work

This work contributes to choreography research by extending and complementing existing choreography categorizations which implies the need for a refined choreography notion. Researchers and practitioners benefit from the identification of choreography classes and from a number of selective criteria that discriminate well between choreography categories. Note that these criteria are unlikely to be '*met*' by a single language. In so far, they are also design options for choosing a choreography language. Also, the identified classes and criteria can be used by new choreography technologies such as the BPMN 2.0 choreographies ([10], section 11) to clarify its scope.
While this work proves the need for a refined choreography notion the identification of choreography classes and criteria is not complete. In so far, the results of this paper also call for a *joint* effort on extending choreography taxonomies.

References

1. A. Barros, G. Decker, and M. Dumas. Multi-staged and multi-viewpoint service choreography modelling. In *Proc. of the Workshop on Software Engineering Methods for Service Oriented Architecture (SEMSOA)*, volume 244 of *CEUR-WS*, May 2007.
2. A. Barros, M. Dumas, and A. H. M. T. Hofstede. Service interaction patterns. In *Proceedings of the 3rd International Conference on Business Process Management (BPM), Nancy, France*, pages 302–318. Springer Verlag, 2005.
3. G. Decker, O. Kopp, and A. Barros. An introduction to service choreographies. *Information Technology*, 50(2):122–127, 2008.
4. G. Decker, O. Kopp, F. Leymann, and M. Weske. BPEL4Chor: Extending BPEL for modeling choreographies. In *Proc. of the 2007 IEEE Int. Conf. on Web Services (ICWS), July 9-13, 2007, Salt Lake City, Utah, USA*, pages 296–303, 2007.
5. G. Decker, O. Kopp, F. Leymann, and M. Weske. Interacting services: From specification to execution. *Data & Knowledge Engineering*, 68(10):946 – 972, 2009.
6. R. Dijkman and M. Dumas. Service-oriented Design: A Multi-viewpoint Approach. *International Journal of Cooperative Information Systems*, 13(4):337–368, 2004.
7. J. Dorn, C. Grün, H. Werthner, and M. Zapletal. A survey of B2B methodologies and technologies: From business models towards deployment artifacts. In *HICSS 2007: Proc. of the 40th Hawaii Int. Conf. on System Sciences, Hawaii*. IEEE.

8. O. Kopp, M. Wieland, and F. Leymann. Towards Choreography Transactions. In *Proc. of the 1st Central-European Workshop on Services and their Composition, ZEUS 2009, Stuttgart, Germany, March 2–3, 2009*, volume 438 of *CEUR-WS*, pages 49–54, 2009.
9. OASIS. *ebXML Business Process Specification Schema Technical Specification.* OASIS, 2.0.4 edition, December 2006.
10. OMG. *Business Process Model and Notation, v2.0.* OMG, January 2011.
11. C. Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, 2003.
12. A. Schönberger. The CHORCH B2Bi approach: Performing ebBP choreographies as distributed BPEL orchestrations. In *Proc. of the 6th World Congress on Services 2010 (SERVICES 2010), Miami, Florida, USA*. IEEE, July 2010.
13. A. Schönberger, C. Pflügler, and G. Wirtz. Translating shared state based ebXML BPSS models to WS-BPEL. *International Journal of Business Intelligence and Data Mining*, 5(4):398 – 442, 2010.
14. A. Schönberger and G. Wirtz. Using Webservice Choreography and Orchestration Perspectives to Model and Evaluate B2B Interactions. In *The 2006 Int. Conf. on Software Engineering Research and Practice (SERP'06), Las Vegas, USA*, 2006.
15. A. Schönberger and G. Wirtz. Taxonomy on consistency requirements in the business process integration context. In *Proc. of 2008 Conf. on Software Engineering and Knowledge Engineering (SEKE)*, Redwood City, California, USA, July 2008.
16. A. Schönberger and G. Wirtz. Towards executing ebBP-Reg B2Bi choreographies. In *Proceedings of the 12th IEEE Conference on Commerce and Enterprise Computing (CEC'10), Shanghai, China.* IEEE, November 10-12 2010.
17. A. Schönberger, G. Wirtz, C. Huemer, and M. Zapletal. A composable, QoS-aware and web services-based execution model for ebXML BPSS businesstransactions. In *Proceedings of the 6th 2010 World Congress on Services (SERVICES2010), Fourth International Workshop on Web Services and Cloud Services Testing (WS-CS-Testing 2010), Miami, Florida, USA*. IEEE, July 2010.
18. UN/CEFACT. *UN/CEFACT's Modeling Methodology (UMM): UMM Meta Model - Foundation Module Version 1.0.* UN/CEFACT, 1.0 edition, 10 2006.
19. W. M. P. van der Aalst and M. Weske. The P2P approach to interorganizational workflows. In *CAiSE '01: Proceedings of the 13th International Conference on Advanced Information Systems Engineering*, pages 140–156, London, UK, 2001.
20. W3C. *Web Services Choreography Description Language.* W3C, 1.0 edition, November 2005.
21. Y. Wand and R. Weber. Research commentary: Information systems and conceptual modeling–a research agenda. *Info. Sys. Research*, 13(4):363–376, 2002.
22. J. M. Zaha, A. P. Barros, M. Dumas, and A. H. M. ter Hofstede. Let's Dance: A language for service behavior modeling. In *Proc. of the 14th Int. Conf on cooperative information systems (CoopIS'06)*, Montpellier, France, 10 2006.
23. M. Zapletal, T. Motal, and H. Werthner. The business choreography language (BCL) - a domain-specific language for global choreographies. In *Proc. of the 5th 2009 World Congress on Services (SERVICES 2009 PART II), Bangalore, India.* IEEE, September 2009.
24. J. Ziemann, T. Matheis, and J. Freiheit. Modelling of cross-organizational business processes. In *Proc of the 2nd Int. Workshop on Enterprise Modelling and Information Systems Architectures (EMISA'07), St. Goar, Germany, October 8-9, 2007*, pages 87–100.

A Proposal for Checking the Conformance of ebBP-ST Choreographies and WS-BPEL Orchestrations

Matthias Geiger, Andreas Schönberger and Guido Wirtz

Distributed and Mobile Systems Group,
University of Bamberg
Bamberg, Germany
{matthias.geiger,andreas.schoenberger,guido.wirtz}@uni-bamberg.de

Abstract. A common problem in applying choreographies and orchestrations is ensuring and enforcing the consistency of the models which is often referred to as "conformance checking". In this position paper, we introduce a concept for checking the conformance of WS-BPEL based orchestrations to ebBP-ST choreographies: First, the ebBP-ST and WS-BPEL models will be transformed into the process algebra CCS. Afterwards, the actual conformance check is performed by checking these CCS models for bisimulation equivalence.

Keywords: choreography, orchestration, conformance checking, WS-BPEL, ebBP

Today, the concepts of choreography and orchestration are well-known and widely accepted in the services community. While choreographies describe a scenario from a global point of view, orchestrations concentrate on the local implementations of each involved party. While the de-facto standard on the orchestration level is the *Web Services Business Process Execution Language* (WS-BPEL; [3]), there exist various choreography languages.

Especially in the business-to-business integration (B2Bi) domain, the *ebXML Business Process Specification Schema* (BPSS or ebBP; [4]) is a suitable choreography language as it provides B2Bi specific features. In [5], the authors present ebBP-ST as an ebBP modeling dialect that captures collaborations between exactly two partners as state-machine based choreographies. ebBP-ST choreographies define an interaction protocol between partners that defines admissible business document exchanges and the effect of those exchanges.

At the implementation level, this protocol has to be implemented for the involved partners. Figure 1 shows the basic architecture of these implementations: For each involved partner, the (existing) systems containing application logic are encapsulated as so-called backend systems. The actual orchestrations are realized by so-called control processes (one for each partner) which are realized in WS-BPEL. Within these WS-BPEL processes the automata structure defined in the choreography is reused in order to govern the control flow of the choreography.

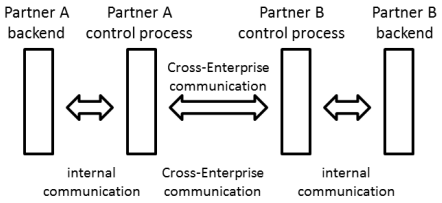


Fig. 1. Basic Architecture

Nevertheless, the actual decision which activity should be performed next is not initiated by the control process but by the backend systems resp. the interaction partner.

As control processes do not contain application logic, it is clear that a rather strict notion of conformance has to be used: A control process is not able to know how the backend systems or the partner control process will continue in the protocol execution and therefore all foreseen possibilities defined by the choreography have to be implemented. (Weak) bisimulation ([2]) is a suitable equivalence notion for this purpose.

To actually check the conformance between ebBP-ST choreographies and corresponding WS-BPEL orchestrations, the process definitions have to be transformed into a common formal basis. We propose the process algebra *Calculus of Communicating Systems* (CCS; [1]) as a suitable process representation because of its straightforward support for state machine like structures.

After specifying the conformance notion and the process algebra to use, the overall approach to check the conformance is as follows: As ebBP-ST choreographies describe binary collaborations, the ebBP-ST choreography has to be implemented by two WS-BPEL orchestrations - one for each involved partner. Therefore, the ebBP-ST model as well as the two WS-BPEL orchestrations have to be transformed to CCS, resulting in three CCS representations. Afterwards each of the CCS orchestration models has to be checked against the CCS choreography model regarding bisimulation equivalence using a model checking tool like the Edinburgh Concurrency Workbench (CWB).

References

1. R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.
2. R. Milner. Calculi for Synchrony and Asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.
3. OASIS. *Web Services Business Process Execution Language (WSBPEL) Version 2.0*, April 2007.
4. OASIS. *ebXML Business Process Specification Schema Technical Specification Version 2.0.4*, Oktober 2006.
5. A. Schönberger, C. Pflügler, and G. Wirtz. Translating Shared State Based ebXML BPSS models to WS-BPEL. *International Journal of Business Intelligence and Data Mining - Special Issue: 11th International Conference on Information Integration and Web-Based Applications and Services in December 2009*, 5(4), 2010.

Towards The Essential Flow Model

Oliver Kopp, Frank Leymann, Tobias Unger, and Sebastian Wagner*

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

Abstract Many of today's manufacturing projects are so complex that they cannot be conducted only by one company anymore. Current approaches for modeling inter-enterprise processes require an early decision on the way activities are connected. The modeler has to decide between control flow and message flow. This implies an early decision on the used IT-technology. We present a modeling approach where this decision is postponed to a later modeling phase. This enables modelers to concentrate on the essentials of the model.

1 Introduction

Many of today's manufacturing projects are so complex that they cannot be conducted only by one company anymore. These collaborations are mostly modeled and executed using business processes. State of the art in modeling collaborations is to model (1) a centralized process model, where the involved partners are represented by swimlanes and connected by sequence flows or to model (2) a choreography, where each partner is represented in a separate pool and connected via message flows. Thus, a modeler has to decide early on how the connection between two activities is established. This has also implications on the used IT infrastructure: In case a single process is used, a single workflow engine coordinates the activities. In case multiple processes are used, a workflow is executed at each participant in the choreography. These workflows have to exchange the agreed messages in order to coordinate. We argue that the decision whether to use a single workflow engine or multiple workflow engines should be taken after capturing the business process. We call such a model "Essential Flow Model".

The idea of an Essential Flow Model is illustrated in Fig. 1. First, an Essential Flow Model is modeled. In principle, this model may be implemented by one of the following infrastructure types: (1) As orchestration, where the workflow is executed by a single workflow engine and the activities are implemented by services or a human task manager [17], (2) as choreography, where a group of activities (typically one lane) becomes a participant in the choreography and thus is executed within a separate workflows engines, (3) using a distributed workflow engine, where the workflow engine is distributed across the different participants. A detailed explanation of each option is provided in the remainder of the paper.

We use BPMN2.0 [20] as illustration for our concept. The concept, however, is independent of the language used. For instance, it is possible to model an Essential Flow Model using PM-Graphs [17] and use BPEL as implementation language [5].

* The authors contributed equally to this paper.

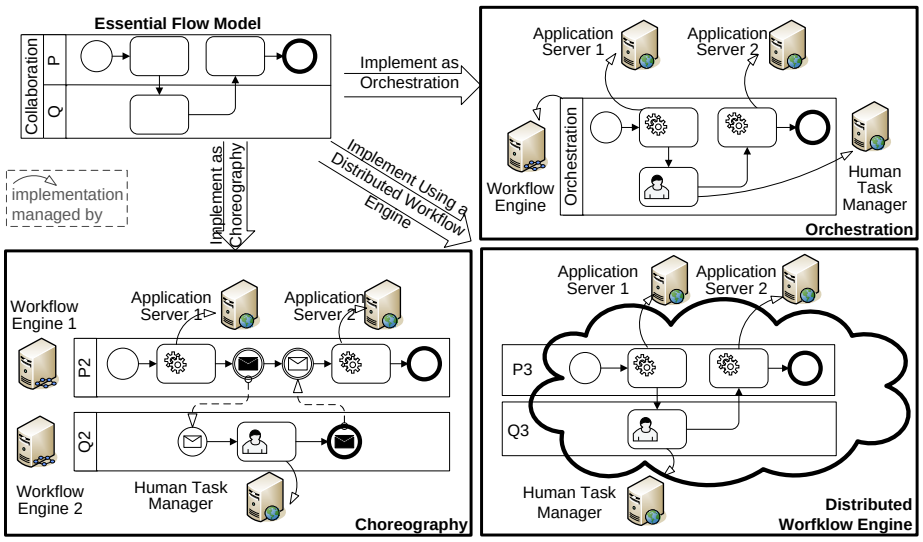


Figure 1. Idea of the Essential Flow Modeling Approach

The remainder of the paper starts with an overview on the Essential Flow Model in Sect. 2. There, an example model is provided. The different implementation possibilities are illustrated using this example in Sect. 3. Related work is presented in Sect. 4. Finally, we conclude and present an outlook on future work in Sect. 5.

2 Designing an Essential Flow Model

An Essential Flow Model (EFM) captures the essential flows in a business process. In this paper, an EFM is a restricted BPMN process. For modeling control flow, we allow tasks, sequence flow, data-based gateways, one none start event, and one none end event only. The process may only be started by one none start event. We allow one pool only, which may contain multiple lanes. Each lane in the pool is interpreted as distinct entity having the *responsibility* for the activities contained in its lane. In future work, we intend to extend EFMs with other constructs, such as IT system assignments or data access rules. These extensions are out of scope of this paper.

An EFM is designed by humans to agree on a collaboration. In this paper, we assume that such a model has been created by business experts. In other scenarios, an EFM might be created out of existing interacting processes. For instance, a BPMN collaboration could be transformed to a BPMN process and thus forming an EFM. Such a merging procedure is part of our future work.

Figure 2 depicts an example scenario of an EFM. In this scenario a train manufacturer wants to develop a new rail car prototype. In the first activity the requirements of the rail car prototype are determined (e. g., the number of seats and toilets in the car). Based on these requirements the chassis of the rail car has to be designed by an engineer of the train manufacturer. After the design is completed, the wheels and the axes for the

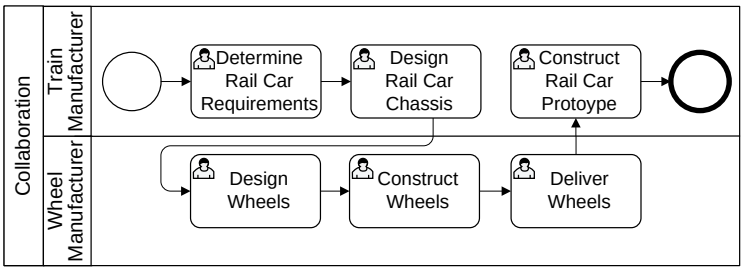


Figure 2. Example Scenario Modeled as Essential Flow Model

rail car are ordered from another organizational unit. This wheel manufacturing unit has to design the wheels in a way such that they fit into the chassis design. Thus, they have to use the chassis design as foundation. After the wheels are designed, prototypes are constructed and delivered to the train manufacturer, which is then able to construct a prototype of the rail car.

In the scenario, each activity has been configured to be a user task. It is also possible to configure each task as other type or postpone this decision to the implementation phase.

3 EFM Implementation Approaches

After the EFM was modeled, it has to be transformed to an executable process model. The first step is to create disjoint sets of the activities. Each set defines the activities of one workflow. In the following, we use swimlanes to define these sets. In case all activities should be executed in one workflow, one has to decide on using a standard workflow engine or a distributed workflow engine. In case the activities are distributed in more than one set, the choreography approach has to be taken. In each approach, the generated process models are abstract process models. That means, the process models are not executable by themselves, but have to be enriched with information needed for execution. This includes typing of variables, adding tasks for data transformation, and binding to concrete services.

Implementation as Orchestration The EFM is implemented on a single workflow engine, i.ė., the activities of each participant are executed on the same engine. The activity implementations of human activities can be performed by one or several human task managers (this depends on the binding information). Tasks which have not been typed, have to be typed. That means, one has to decide whether to use a user task, a service task, or another specific task type for implementing a task. In case a task has already assigned a task type, this assignment may not be changed. Figure 3 presents the orchestration model for our scenario. The final orchestration is executed on a workflow engine of the Train Manufacturer. The human tasks of the Train Manufacturer are executed by its human task manager and the human tasks of the Wheel Manufacturer are

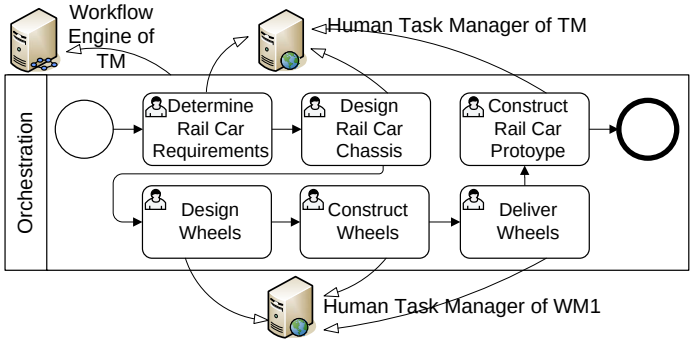


Figure 3. Train Prototype Scenario Implemented as Orchestration

executed by its own task manager, which is coordinated by the workflow engine of the Train Manufacturer.

Implementation as Choreography This approach splits the implementation of the EFM into one process model per participant. A workflow engine is assigned to one or more swimlanes. In our case, the Wheel Manufacturer activities are executed on a different workflow engine than the Train Manufacturer activities. Each EFM control flow dependency between activities assigned to different workflows is replaced by an intermediate message throw event, a message link, and a message catch event. The intermediate message throw event is connected to the source of the sequence flow to be replaced and the intermediate message catch event is connected to the target of the sequence flow. The first intermediate message catch event has to be replaced with a message start event. The other intermediate message events have to be connected to the preceding intermediate message throw events. The precedence relation is given by the precedence relation of the original EFM. It may be the case that this approach is too straight-forward for workflows where the control flow is split using gateways

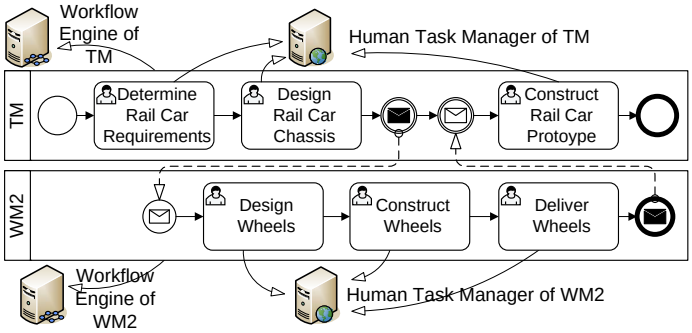


Figure 4. Train Prototype Scenario Implemented as Choreography

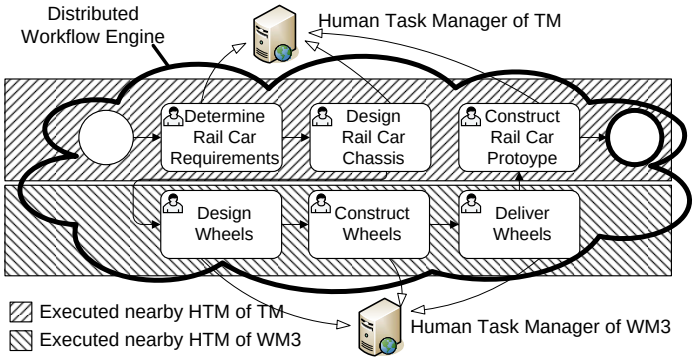


Figure 5. Train Prototype Scenario Implemented by a Distributed Workflow Engine

and where multiple intermediate message catch events without a local predecessor are generated. These aspects have been discussed by Khalaf and Leymann [9, 11] in the case of BPEL (see Sect. 4). A discussion of these aspects regarding BPMN should be tackled in future work. Figure 4 presents the final transformation result. The result forms a BPMN collaboration diagram, which is an interconnection choreography model [4]. The difference to the orchestration is that the activities of the Train Manufacturer are executed by its workflow engine and the activities of the Wheel Manufacturer are executed by its own workflow engine. The task manager of the Wheel Manufacturer is coordinated by its own the workflow engine.

Implementation using a Distributed Workflow Engine In a distributed workflow engine activities of the same workflow are executed on different nodes (e. g., a physical or virtual machine instance). For example, the distribution of the activities may be defined based on certain technical constraints or based on service level agreements. A technical constraint may be the overall decrease of the amount of data exchanged remotely between two activities by letting data-intensive activities on the same node. In our example the activities “Design Wheels” and “Design Train Chassis” might run on the same node as they exchange data as illustrated by Fig. 5. This means, that the EFM implementation may not be split by swimlanes (although this is still possible in this approach) but also by other criteria. Consequently, an appropriate fragmentation that meets those criteria has to be defined, this can be either done by a workflow designer or the fragmentation is automatically derived by the distributed workflow engine (e. g. based on the workload of their nodes). Based on the fragmentation the activities are distributed. Wutke et al. [18, 24] describe concrete runtime behavior and further concepts of distributed workflow engines.

4 Related Work

Khalaf et al. [9–12] discuss issues when doing a role-based decomposition of a single BPEL process into multiple BPEL processes. The main difficulties are (1) splitting

the control flow such that the split processes resemble the same order of activities, (2) correlating exchanged messages to the right the instances of each partner process, (3) keeping data consistent across partners, and (4) coordinating split scopes and split loops. The distribution of control flow among the split processes is solved by propagating the status of each link using messages [11]. Khalaf solves the correlation problem by using a globally unique correlation set for each process instance [9]. In case tasks read from and write to the same variable in the unsplit model and are placed in different partners in the split model, the data has to be kept consistent. A solution is to separate data flow from control flow [10]. In case of split scopes, the fault handling has to be coordinated: A fault occuring on one partner has to lead to a proper handling in the respective parts of the other part [12]. A transfer of these concepts to BPMN is part of our future work.

Koschmider et al. [16] discuss perspective-compliant business process design. One perspective is the view-based perspective. The EFM model may be regarded as one view on the collaboration, whereas the choreography model may be regarded as another view. When going from abstract process models to executable process models, the modeling perspective changes (analysis vs. execution). Koschmider et al. propose a tool supporting different perspectives by using process fragments. In our work, we do not focus on different views, but argue that the decision on the concrete connection between activities should be delayed to the implementation phase. The concrete mapping between the different views is left as future work.

Werth [23] presents a method, a metamodel, and a notation for "collaborative business processes". In his method, the designer has to distinguish between material flow, information flow, energy flow, and control flow between organizations. Werth leaves the implementation as future work. In our work, control flow is the only connection between tasks and we sketch a way from the model to an implementation.

Van der Aalst et al. [1] propose a method to describe a business process spanning multiple partners. They use open worfklow nets (oWFNs [19]) to model a "process-oriented contract which can be seen as the composition of the public views of all participating parties." [19]. The decision whether a place is an interface place is an integral part of the method. In our approach, this decision is deferred to the implementation phase. Decker et al. [5] describe a method for going from a choreography to executable processes. The starting point is a choreography, where the realization choice has been made in the starting model. The same applies for the methods proposed by Barros et al. [2], Dijkman and Dumas [6], Greiner et al. [8], and Werth [23].

Barros et al. [3] reason about possible interaction types between participants. In the current version of the EFMs, request-for-bid scenarios cannot be directly expressed. Thus, our current research is to investigate whether and how those scenarios have to be supported by EFMs.

Kiepuszewski et al. [13] reason about fundamental control flow structures in workflows. In contrast, our paper focused on the overall fundamental ingredients of a model where a workflow (possibly involving multiple parties) is modeled. Patig and Casanova-Brito [21] conducted a survey on general requirements of process modeling languages. Users mostly started modeling by capturing the interactions between departments. The survey did not distinguish between control flow and data flow.

Zimmermann et al. [25] present a concrete reusable architectural decisions framework for enterprise application development. Our decision on whether to use one process engine or a set of engines falls in the class RADM-C, where conceptional decisions have to be tackled.

We assumed that the modeling starts by defining an EFM model. Another start might be existing processes, which are merged into an EFM. Regarding merging, Steinmetz [22] shows how an interconnection choreography model can be generated out of multiple interacting processes. Kopp et al. [15] show how an interconnection choreography model can be converted to an interaction choreography model. This approach might also be used to merge multiple pools into a single pool, which is out of scope of this paper.

When generating process models, which need to be modified by a process modeler, the consistent and clear layout becomes important [14]. In this paper, we do not investigate on appropriate process visualization techniques and process layouting techniques [7], but leave their application as future work.

5 Conclusion and Outlook

We have presented the idea of an "Essential Flow Model". This model captures the most basic flows between activities required for communicating the model between the involved modelers and with the persons responsible for the implementation of the model. We have presented three different approaches to implement an EFM. Each mapping from an EFM to a skeleton for each implementation approach has been explained using an example. Thus, our next step is to provide a formal presentation of the transformation.

The presented EFM is very basic. In industrial settings, the participants have to agree to use certain IT systems or agree on data access rules. We are going to integrate these issues in a refined version of the EFM. The result is a concrete description of the ingredients of an EFM. In addition, we assumed that an EFM is created from scratch by humans. An EFM might be created out of an existing BPMN collaboration. This conversion is part of our future work.

Acknowledgments This work is partially funded by the projects ALLOW (`http://www.allow-project.eu/`), COMPAS (`http://www.compas-ict.eu`), and MASTER (`http://www.master-fp7.eu/`). They all are part of the EU 7th Framework Programme (contract no. FP7-213339, FP7-215175, and FP7-216917).

References

1. Aalst, W.M.P.v.d., Lohmann, N., Massuthe, P., Stahl, C., Wolf, K.: Multiparty Contracts: Agreeing and Implementing Interorganizational Processes. Comput. J. 53(1), 90–106 (2008)
2. Barros, A., Decker, G., Dumas, M.: Multi-staged and Multi-viewpoint Service Choreography Modelling. In: SEMSOA (2007)
3. Barros, A., Dumas, M., ter Hofstede, A.: Service Interaction Patterns. In: BPM. Springer (2005)
4. Decker, G., Kopp, O., Barros, A.: An Introduction to Service Choreographies. Information Technology 50(2), 122–127 (Feb 2008)
5. Decker, G., Kopp, O., Leymann, F., Weske, M.: Interacting services: From specification to execution. Data & Knowledge Engineering 68(10), 946–972 (Apr 2009)

6. Dijkman, R., Dumas, M.: Service-oriented Design: A Multi-viewpoint Approach. International Journal of Cooperative Information Systems 13(4), 337–368 (2004)
7. Effinger, P., Jogsch, N., Seiz, S.: On a study of layout aesthetics for business process models using BPMN. In: Second International Workshop on Business Process Modeling Notation. Springer (2010)
8. Greiner, U., Lippe, S., Kahl, T., Ziemann, J., Jkel, F.W.: Designing and Implementing Cross-Organizational Business Processes - Description and Application of a Modelling Framework, pp. 137–147. Springer (2007)
9. Khalaf, R.: Supporting business process fragmentation while maintaining operational semantics: a BPEL perspective. Doctoral thesis, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany (2008)
10. Khalaf, R., Kopp, O., Leymann, F.: Maintaining Data Dependencies Across BPEL Process Fragments. International Journal of Cooperative Information Systems (IJCIS) 17(3), 259–282 (September 2008)
11. Khalaf, R., Leymann, F.: Role-based Decomposition of Business Processes using BPEL. In: ICWS 2006. IEEE Computer Society (2006)
12. Khalaf, R., Leymann, F.: Coordination for Fragmented Loops and Scopes in a Distributed Business Process. In: BPM. Springer (2010)
13. Kiepuszewski, B., ter Hofstede, A., van der Aalst, W.: Fundamentals of control flow in workflows. Acta Informatica 39(3), 143–209 (2003)
14. Kitzmann, I., Konig, C., Lubke, D., Singer, L.: A Simple Algorithm for Automatic Layout of BPMN Processes. E-Commerce Technology 0, 391–398 (2009)
15. Kopp, O., Leymann, F., Wu, F.: Mapping interconnection choreography models to interaction choreography models. In: ZEUS (2010)
16. Koschmider, A., Habryn, F., Gottschalk, F.: Real Support for Perspective-Compliant Business Process Design. In: Aalst, W., Mylopoulos, J., Sadeh, N.M., Shaw, M.J., Szyperski, C., Ardagna, D., Mecella, M., Yang, J. (eds.) Business Process Management Workshops. Springer (2009)
17. Leymann, F., Roller, D.: Production Workflow – Concepts and Techniques. Prentice Hall PTR (2000)
18. Martin, D., Wutke, D., Leymann, F.: A Novel Approach to Decentralized Workflow Enactment. In: EDOC. IEEE Computer Society (2008)
19. Massuthe, P., Reisig, W., Schmidt, K.: An Operating Guideline Approach to the SOA. Annals of Mathematics, Computing & Teleinformatics 1(3), 35–43 (2005)
20. Object Management Group (OMG): Business Process Model and Notation (BPMN) Version 2.0 (2011), OMG Document Number: formal/2011-01-03
21. Patig, S., Casanova-Brito, V.: Requirements of Process Modeling Languages – Results from an Empirical Investigation. In: Wirtschaftsinformatik (2011)
22. Steinmetz, T.: Generierung einer BPEL4Chor-Beschreibung aus BPEL-Prozessen. Student Thesis: University of Stuttgart, Institute of Architecture of Application Systems (2007)
23. Werth, D.: Modellierung unternehmensübergreifender Geschäftsprozesse – Modelle, Notation und Vorgehen fr Geschäftsprozesse. Salzwasser Verlag (2007)
24. Wutke, D., Martin, D., Leymann, F.: Tuplespace-based Infrastructure for Decentralized Enactment of BPEL Processes. In: Wirtschaftsinformatik. OCG, Vienna, Austria (2009)
25. Zimmermann, O., Zdun, U., Gschwind, T., Leymann, F.: Combining Pattern Languages and Reusable Architectural Decision Models into a Comprehensive and Comprehensible Design Method. In: WICSA (2008)

Towards deciding policy violation during service discovery

Jan Sürmeli

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

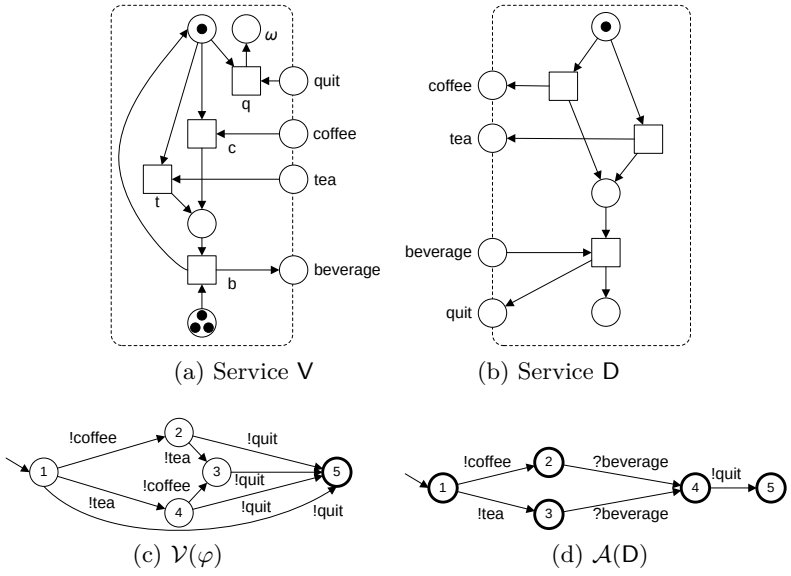
Abstract. In a service-oriented architecture, a *provider* publishes its service in a service repository. A *requester* approaches a *broker* which returns a service S matching the requester's service R. Then, S and R are coupled. The provider of S may require a specific relation between the expenses and rewards for an execution of S, summarized in a *policy* φ. The control flow of S may contain both internal and external decisions: By sending messages, R may trigger a certain execution path. Based on models of S and R, the broker may decide if R *violates* φ before coupling. If so, the broker may not couple S and R. In this paper, we provide a formal framework to model policies, and introduce a decision procedure for policy violation based on open net models of the services.

1 Setting and problem

We understand a *service* as a component with an inner control flow and an interface to exchange messages asynchronously with other services. Thereby, it provides a certain functionality which may be used by other services. A *provider* publishes its service in a repository. A *requester* approaches a *broker* for accessing a previously published service. The provider earns a reward for providing its service. This reward may manifest as a usage fee, or a provision from the repository owner, or from any third party. Usually, a provider desires some beneficial relation between this reward and the expenses for providing its service. As an example, a provider might want the expenses to be covered by the reward. We specify such requirements as *policies*. A partner either *violates* a policy or not. The provider aims at its service being coupled only with non-violating partners. Both reward and expenses may have fixed and variable components. This is a quite usual problem in economics and solutions for this problem are known for a long time. However, in our case, we encounter another difficulty: We consider *stateful* services. A stateful service has its own control flow which is influenced by internal and external decisions. External decisions are made through asynchronous message exchange. Therefore, reward and expenses for providing a service vary from requester to requester.

As a running example, consider a vending machine which sells coffee and tea, modeled as an *open net* [1] in Fig. 1(a). In its initial state, it waits for one of

three messages: Either an order for coffee, an order for tea, or a quit message. To receive an order it executes the respective transition c or t. Subsequently, it serves the beverage by executing b. A quit message may be consumed by executing q, resulting in a final state ω. The machine may serve up to three beverages, as indicated by the three tokens in the place in the bottom. The provider of the vending machine may have fixed expenses of 10 units for providing its service and variable expenses for each served beverage depending on the type: 20 units for coffee and 10 units for tea. As a reward, the provider collects a fixed amount of 5 units and additionally 25 units per served beverage. Assume the provider desires the expenses to be fully covered by the reward, specified in a policy φ_{V}. We find that a customer ordering at least one beverage is a good customer, whereas a customer ordering nothing and simply quitting is not. However, asynchronous message exchange induces a subtle problem: A customer ordering a beverage and then sending the quit message before receiving the beverage is a bad customer: The vending machine might receive the quit message first. A simple partner for V is shown in Fig. 1(b): D orders either a tea or a coffee, receives the beverage, and sends a quit message. Obviously, D does not violate φ_{V}.



(a) Service V (b) Service D

(c) $\mathcal{V}(\varphi)$ (d) $\mathcal{A}(\mathsf{D})$

Fig. 1. Services V and D, modeled as open nets, a finite representation $\mathcal{V}(\varphi)$ of all φ-violating partners of V, and a finite automaton $\mathcal{A}(\mathsf{D})$ characterizing D.

There exists work on *quality of service (QoS)*, e.g. [2] and pricing of services, e.g. [3], focusing on non functional properties of *stateless* services. Such services do not have their own control flow. Therefore, for each dimension, for example *costs*, a value is given. Research is centered on finding composites of many services

which reach a specific goal in the least expensive way, e.g. [4]. For stateful services, there exists work on over-approximating the costs incurring while running the service with a given partner service or a set of partner services described as constraints [5]. This analysis gives the service provider an idea on the QoS of its service. However, it does not provide a sufficient basis to decide if a partner violates the provider's policy or not. Likewise, [6,7] compute the costs for running a business process. In our setting, the closed system is not known beforehand. Instead, we consider open systems.

We sketch our solution. Upon publishing, the provider states its requirements in form of a *policy* φ. Upon requesting, the broker decides whether the requester *violates* the policy or not. This procedure is similar to that of Operating Guidelines [8] which allows the broker to decide policy violation: We use an automaton characterizing partner behavior which may lead to policy violation.

We illustrate our approach on the running example: Figure 1(c) shows automaton $\mathcal{V}(\varphi_{\mathsf{V}})$ which finitely represents violating partner behavior. For instance, the sequence !tea!quit (read: *send* tea, *send* quit) may result in expenses that are not covered by the reward and therefore *violates* φ_{V}. By comparing $\mathcal{V}(\varphi_{\mathsf{V}})$ with an abstract model of a requester, the broker may decide policy violation. For example, automaton $\mathcal{A}(\mathsf{D})$ in Fig. 1(d) is an abstract model of open net D in Fig. 1(b). Comparing $\mathcal{V}(\varphi)$ and $\mathcal{A}(\mathsf{D})$, we find no common path to a final state. Therefore, D does not violate φ_{V}. As a consequence, the broker may return V.

The rest of the paper is structured as follows: Section 2 shortly recalls basic formal concepts, especially open nets. We introduce a framework for policies in Sect. 3. We sketch our approach to compute a finite representation of violating behavior in Sect. 4. Finally, we conclude our paper and present ideas for future work in Sect. 5.

2 Basic notions

As usual, $\mathbb{Z}$ denotes the set of all *integers*. We write Σ^* for the set of all *finite sequences* over an alphabet Σ. For $\sigma \in \Sigma^*$, we write $\sigma(i)$ for the *i-th character* in σ. We denote the *restriction* of σ to $\Sigma' \subseteq \Sigma$ with $\sigma_{|\Sigma'}$. We recall the basic notions of *Petri nets*: A *Petri net* is a tuple $N = \langle P, T, F, m_0 \rangle$ of *places* P, *transitions* T, *arcs* F and *initial marking* m_0. We denote the set of all markings of N with $\mathcal{M}_N$. We denote the *preset* and *postset* of $x \in P \cup T$ with ${}^\bullet x$ and $x^\bullet$, respectively. We canonically extend these notions to sets of net elements by union. We call a sequence $\sigma \in T^*$ firing sequence if the transitions in σ may be fired subsequently starting in m_0. We write $beh(N, m)$ for the set of all firing sequences resulting in a marking m.

Open nets are Petri nets with an interface declaration and a set of final markings: We define an *open net* as a tuple $N = \langle P, T, F, m_0, I, O, \Omega \rangle$ where $\langle P, T, F, m_0 \rangle$ forms a Petri net, I, O are disjoint subsets of P with ${}^\bullet I \cup O^\bullet = \emptyset$, called *input* and *output places*, respectively, and $\Omega \subseteq \mathcal{M}_N$ is a set of *final markings*. We call the Petri net $ip(N) = \langle P', T, F', m_0' \rangle$ the *inner process* of N where $P' = P \setminus (I \cup O)$, $F' = F \cap ((P' \times T) \cup (T \times P'))$, and $m_0'(p) = m_0(p)$

for all $p \in P'$. We call two open nets N_1, N_2 *partners* if their inner processes are component-wise disjoint, $I_1 = O_2$, and $O_1 = I_2$. We compose two partners N_1, N_2 by accordingly merging the interface places, yielding $N_1 \oplus N_2$.

Example 1. Figure 1(a) shows an open net V with input places quit, coffee and tea, and output place beverage. The set Ω_V of final markings cannot be seen from the figure. We define $\Omega_\mathsf{V} = \{m \mid m \in \mathcal{M}_\mathsf{V} \wedge m(\omega) > 0\}$. The open net D in Fig. 1(b) and V are partners.

3 A formal framework for policies

A *policy* specifies the allowed behavior of a provided service N in composition with an arbitrary partner Q. The main building blocks of a policy are *cost functions* and *constraints*. For the following definitions, we assume a given open net N with transitions T_N.

Cost functions. There are different approaches to define cost functions based on behavior. The most general is to define a cost function as a mapping from transition sequences to some value domain. Throughout this paper, we use the set of integers $\mathbb{Z}$ for this purpose. In our approach, we specify the costs for executing a *single* transition after having executed a (finite) *history*. This covers varying costs for executing a single transition based on the knowledge which transitions have been fired. A cyclic service usually has infinitely many and arbitary long (finite) runs. To ease up analysis, we encode histories into *hash values*.

A hashing from a set A into a set B is a function $f : A \to B$. Usually, the idea is that the elements of B, called hash values, are more lightweight than those of A. Thus, a hash function may be used for efficient table lookups and the like. Typically, a hash function is required to fulfill a number of properties ensuring its usability. In this paper, we use *finite histories* as input: a finite sequence of transitions. We define a *history hashing* h as a hashing from T_N^* into some set H having two properties: *Continuity* and *Finiteness*.

Definition 1 (Continuity, finiteness, history hashing). *We define two properties for functions $h : T_N^* \to H$:*

1. Continuity. *Let $\sigma, \sigma', \sigma'' \in T_N^*$. If $h(\sigma) = h(\sigma')$, then $h(\sigma\sigma'') = h(\sigma'\sigma'')$.*
2. Finiteness. *H is finite.*

We call h a history hashing *if h has both properties* continuity *and* finiteness.

Intuitively, *continuity* demands that, given two histories with the same hash value, each equal continuation of the two results in the same hash value again. *Finiteness* restricts history hashings to finite sets of hash values. We elaborate on the value of these properties for analysis in Sect. 4. For the definitions of this section, we assume a given history hashing h into a set of hash values H.

As mentioned above, we define *cost functions* for executing single transitions based on the hash values. Thus, the domain of such a function is the cross product of the set of transitions and the set of hash values. Induction yields the semantics of the cost function: The costs for a complete transition sequence.

Definition 2 (Cost functions). *We call a function* $f : T_N \times H \to \mathbb{Z}$ cost function. *We define the* semantics *of* f *as the function* $\|f\| : T_N^* \to \mathbb{Z}$ *with*

- $\|f\|(\epsilon) = 0$, *and*
- $\|f\|(\sigma t) = f(t, h(\sigma)) + \|f\|(\sigma)$ *if* $t \in T_N, \sigma \in T_N^*$.

Example 2. We define the cost functions mentioned in Sect. 1 for open net V in Fig. 1(a). For both functions, we need to know whether a coffee or a tea order has been received last. We cover those two cases by the hash values coffee and tea. For totality, we introduce a third hash value, other. We define a hash function $\mathsf{h} : T_\mathsf{V}^* \to \mathsf{H}$ with $T_\mathsf{V} = \{\mathsf{c}, \mathsf{t}, \mathsf{b}, \mathsf{q}\}$ and $\mathsf{H} = \{\mathsf{coffee}, \mathsf{tea}, \mathsf{other}\}$. We define the sets of occurrences of c and t in $\sigma \in T_\mathsf{V}^*$: $\mathsf{OC}_\sigma = \{i \mid \sigma(i) = \mathsf{c}\}$ and $\mathsf{OT}_\sigma = \{i \mid \sigma(i) = \mathsf{t}\}$.

$$\mathsf{h}(\sigma) = \begin{cases} \mathsf{other} & \text{if } \mathsf{OC}_\sigma = \mathsf{OT}_\sigma = \emptyset, \\ \mathsf{coffee} & \text{if } \mathsf{OC}_\sigma \neq \emptyset \wedge (\mathsf{OT}_\sigma \neq \emptyset \Rightarrow \max(\mathsf{OC}_\sigma) > \max(\mathsf{OT}_\sigma)), \\ \mathsf{tea} & \text{otherwise.} \end{cases}$$

Based on this hashing, we define the cost functions f and g over T_V with hashing h to specify the *expenses* and the *reward* for the provider to execute V:

- $\forall t \in T_\mathsf{V} \setminus \{\mathsf{b}\}, a \in \mathsf{H} : \mathsf{f}(t, a) = \mathsf{g}(t, a) = 0$,
- $\mathsf{f}(\mathsf{b}, \mathsf{coffee}) = 20$, $\mathsf{f}(\mathsf{b}, \mathsf{tea}) = 10$, $\mathsf{f}(\mathsf{b}, \mathsf{other}) = 0$, and
- $\mathsf{g}(\mathsf{b}, \mathsf{coffee}) = \mathsf{g}(\mathsf{b}, \mathsf{tea}) = 25$, $\mathsf{g}(\mathsf{b}, \mathsf{other}) = 0$.

The choice of the history hashing determines the class of cost functions that may be build. We propose to make use of *deterministic finite automata (DFA)*. A history may be interpreted as a word. Using a history σ as input for a DFA, the resulting state q may be understood as a hash value for σ. Such a history hashing obviously satisfies *continuity* as a DFA is deterministic and total. Additionally, since its set of states is finite, *finiteness* holds. Utilizing such a hashing, we can express any cost function where conditions consist of checking membership of the history in regular languages.

Constraints. We introduce *constraints* as restrictions on behavior by specifying *conditional bounds* for cost functions. As conditions, we use markings. As bounds, we use integer intervals. Intuitively, a transition sequence meeting the condition *satisfies* a constraint if its costs are inside given bounds. If the transition sequence results in a different marking or cannot be fired, it trivially satisfies the constraint.

Definition 3 (Constraints). *We call a pair* $p = \langle m, \tau \rangle$ constraint *over a set* G *of cost functions, iff* $m \in \mathcal{M}_{ip(N)}$ *is a marking of the inner process of* N *and* τ *maps each cost function* f *to an integer interval.*

A transition sequence $\sigma \in T_N^*$ satisfies $p = \langle m, \tau \rangle$, *written* $\sigma \models p$, *iff* $\sigma \in beh(ip(N), m) \Rightarrow \forall f \in G : \|f\|(\sigma) \in \tau(f)$.

Example 3. We model the constraint informally described in Sect. 1: Upon reaching a final marking, all expenses should be covered by the reward. We define this as one constraint p_m per final marking $m \in \Omega_{\mathsf{V}}$. Each p_m is defined over cost function $(\mathsf{g} - \mathsf{f})$ as defined in Example 2. At first glance, the acceptance interval for this cost function is $[0, \infty)$. However, we did not model the fix costs yet. We thus shift the acceptance interval by the difference of the fix costs yielding $\mathsf{i} = [5, \infty)$. We inspect some example firing sequences and decide constraint satisfaction for each. The firing sequence cb trivially satisfies each p_m since it does not result in a final marking. The firing sequences $\sigma = \mathsf{cbq}$ and $\sigma' = \mathsf{q}$ result in a final marking. While σ satisfies each p_m, σ' does not: $(\mathsf{g} - \mathsf{f})(\sigma) = 5 \in \mathsf{i}$, $(\mathsf{g} - \mathsf{f})(\sigma') = 0 \notin \mathsf{i}$.

Policies. A *policy* is basically a collection of constraints over a given set of cost functions. Policy violation requires a change of the viewpoint: Policies are defined over the behavior of a fixed open net N. However, policy violation is not a property of N but of a partner Q of N. By sending messages to N, Q may influence the control flow of N. Intuitively, Q *violates* a policy φ, if it sends messages, such that N *may* choose a firing sequence which does not satisfy all constraints in φ. Formally, we define policy satisfaction for a transition sequence of N and based thereon policy violation.

Definition 4 (Policies). *We define a* policy *as a tuple $\varphi = \langle N, h, G, C\rangle$ where G is a set of cost functions and C is a set of constraints over G. A transition sequence $\sigma \in T_N^*$* satisfies *φ, written $\sigma \models \varphi$, iff $\forall p \in C : \sigma \models p$. A partner Q of N* violates *φ iff there exists a firing sequence σ of $N \oplus Q$, such that $\sigma_{|T_N} \not\models \varphi$.*

Example 4. We combine the open net V from Fig. 1(a), the hashing h from Example 2, the cost functions f, g from Example 2, and the constraints p_m ($m \in \Omega_{\mathsf{V}}$) from Example 3 to *policy* $\varphi_{\mathsf{V}} = \langle \mathsf{V}, \mathsf{h}, \{(\mathsf{g} - \mathsf{f})\}, \{\mathsf{p}_m \mid m \in \Omega_{\mathsf{V}}\}\rangle$. We find $\mathsf{cbq} \models \varphi_{\mathsf{V}}$ and $\mathsf{q} \not\models \varphi_{\mathsf{V}}$. Consider open net D from Fig. 1(b) as a partner for V. We answer the question if D violates φ_{V} or not: There are two firing sequences σ, σ' of $\mathsf{V} \oplus \mathsf{D}$ which result in a final marking of V: $\sigma_{|T_{\mathsf{V}}} = \mathsf{cbq}$, and $\sigma'_{|T_{\mathsf{V}}} = \mathsf{tbq}$.

To decide whether D violates φ_{V}, we need to decide $\sigma_{|T_{\mathsf{V}}} \models \varphi_{\mathsf{V}} \wedge \sigma'_{|T_{\mathsf{V}}} \models \varphi_{\mathsf{V}}$ which boils down to deciding $\forall m \in \Omega_{\mathsf{V}} : \sigma_{|T_{\mathsf{V}}} \models \mathsf{p}_m \wedge \sigma'_{|T_{\mathsf{V}}} \models \mathsf{p}_m$. D does not violate φ_{V} because $\{(\mathsf{g} - \mathsf{f})(\sigma_{|T_{\mathsf{V}}}), (\mathsf{g} - \mathsf{f})(\sigma'_{|T_{\mathsf{V}}})\} = \{5, 15\} \subseteq [5, \infty)$.

4 Toward deciding policy violation

In this section, let N be an open net and $\varphi = \langle N, h, G, C\rangle$ be a policy. Our approach follows three steps: (1) Compute the *φ-state space* $\mathcal{S}(\varphi)$. (2) Finitely represent all φ-violating behavior, yielding $\mathcal{V}(\varphi)$. (3) Decide policy violation utilizing $\mathcal{V}(\varphi)$.

The φ-state space. The state space of a system is usually a directed graph where each vertex represents a state of the system and each edge stands for a transition

from the source state to the target state. In case of a Petri net, a state is a marking and an edge is a transition. Many properties may be decided on the state space by exploring the set of all states or their order.

Our property of interest is the following: Which firing sequences satisfy the given policy? The problem is that even a finite state space generally may represent infinitely many firing sequence due to cyclic behavior. This can be easily overcome if it is possible to transform the property into a state property, i.e. if it is sufficient to inspect a state to conclude if the firing sequences resulting in this state have the property or not.

We intend to do the same trick for our property of interest. In a first step, we enrich states with the so far incurred costs for each cost function. However, this is not sufficient: Let σ, σ' be firing sequences resulting in the same costs and marking. Let σ'' be a transition sequence, such that $\sigma\sigma''$ and $\sigma'\sigma''$ are firing sequences again. Then, $\sigma\sigma''$ and $\sigma'\sigma''$ do not necessarily result in the same costs again.

Example 5. Consider the firing sequences $\sigma_1 = \mathsf{cbc}$ and $\sigma_2 = \mathsf{cbt}$ of the inner process of open net V from Fig. 1(a). According to cost function f from Example 2, both σ_1 and σ_2 result in the same state: Firing yields obviously the same marking and the same costs of 20 units. However, continuing with b, we find that $f(\mathsf{cbcb}) = 40 \neq f(\mathsf{cbtb}) = 30$.

We thus add the hash value of the history. Formally, we define a φ-state as a triple $q = \langle m, x, \tau_q \rangle$ consisting of a marking m, a hash value x, and a mapping $\tau_q : G \to \mathbb{Z}$.

Example 6. We continue Example 6. According to the history hashing h from Example 2, the hash values for σ_1 and σ_2 are different: $\mathsf{h}(\sigma_1) = \mathsf{coffee} \neq \mathsf{h}(\sigma_1) = \mathsf{tea}$. We can distinguish the resulting states of σ_1 and σ_2 by their hash values.

Given a φ-state q, it is trivial to decide if the firing sequences resulting in q satisfy the policy or not: We check membership of the current values given by τ_q with the intervals given by τ for each constraint $\langle m, \tau \rangle$. Additionally, if the *φ-state space* $\mathcal{S}(\varphi)$ is finite, we may compute it with a depth first search, thereby exploiting property *Continuity*. Property *Finiteness* ensures that the set S of reachable states is finite iff $\{\langle m, \tau_q \rangle \mid \langle m, x, \tau_q \rangle \in S\}$ is. Given $\mathcal{S}(\varphi)$, we may compute $\mathcal{V}(\varphi)$, similarly as in [8]. So far, we do not have a solution for infinite φ-state spaces.

Deciding φ-violation. We finitely represent the *φ-violating partner behavior* as a finite automaton $\mathcal{V}(\varphi)$. Thereby, a word represents partner behavior: $?a!b?c$ stands for receiving a, followed by sending b and receiving c. We can represent the set of traces of the inner process of any open net Q as a finite automaton $\mathcal{A}(Q)$, if it is finite state. By setting the set of final states to the complete state set, policy violation may be decided by comparing the languages of $\mathcal{V}(\varphi)$ and $\mathcal{A}(Q)$. If their intersection is non-empty, Q *violates* φ.

Example 7. Figure 1(c) shows $\mathcal{V}(\varphi_{\mathsf{V}})$ of φ_{V} from Example 4, Fig. 1(d) shows $\mathcal{A}(\mathsf{D})$ of D from Fig. 1(b). $\mathcal{V}(\varphi_{\mathsf{V}})$ and $\mathcal{A}(\mathsf{D})$ do not share an accepting run, φ_{V} is not violated.

Similarly, we believe that $\mathcal{V}(\varphi)$ may be used as a *constraint automaton* as introduced in [9] to compute policy-aware operating guidelines. The result may then be used to decide policy violation and behavioral compatibility in one step.

5 Conclusion and future work

We provided a formal framework to specify policies φ, describing acceptable behavior of a partner based on cost functions and constraints. We explained how a φ-state space may be computed and processed if it is finite. We sketched a decision procedure based on this representation. In the future, we aim at solving the problem that the φ-state space is not necessarily finite. We intend to apply techniques similar to the coverability graph for Petri nets. We plan to extend our proof of concept implementation to evaluate the practical usability of our approach with a case study.

References

1. Kindler, E.: A compositional partial order semantics for Petri net components. In: ATPN'97. Volume 1248 of LNCS. (1997) 235–252
2. Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-aware middleware for web services composition. IEEE Trans. Software Eng. **30**(5) (2004) 311–327
3. Ding, W.: Services pricing through business value modeling and analysis. In: IEEE SCC. (2007) 380–386
4. Schuller, D., Miede, A., Eckert, J., Lampe, U., Papageorgiou, A., Steinmetz, R.: Qos-based optimization of service compositions for complex workflows. In Maglio, P.P., Weske, M., Yang, J., Fantinato, M., eds.: ICSOC. Volume 6470 of Lecture Notes in Computer Science. (2010) 641–648
5. Gierds, C., Sürmeli, J.: Estimating costs of a service. In Gierds, C., Sürmeli, J., eds.: Proceedings of the 2nd Central-European Workshop on Services and their Composition, ZEUS 2010, Berlin, Germany, February 25–26, 2010. Volume 563 of CEUR Workshop Proceedings., CEUR-WS.org (2010) 121–128
6. Sampath, P., Wirsing, M.: Computing the cost of business processes. In: UNISCON. (2009) 178–183
7. Magnani, M., Montesi, D.: BPMN: How much does it cost? An incremental approach. In: BPM. (2007) 80–87
8. Wolf, K.: Does my service have partners? LNCS ToPNoC **5460**(II) (March 2009) 152–171 Special Issue on Concurrency in Process-Aware Information Systems.
9. Lohmann, N., Massuthe, P., Wolf, K.: Behavioral constraints for services. In Alonso, G., Dadam, P., Rosemann, M., eds.: BPM. Volume 4714 of Lecture Notes in Computer Science., Springer (2007) 271–287

Dienstgüte-basierte Service-Selektion für Zustandsbehaftete Services

Dieter Schuller[1] and Jan Sürmeli[2]

[1] Multimedia Communications Lab (KOM),
Technische Universität Darmstadt, Germany
Dieter.Schuller@KOM.tu-darmstadt.de
[2] Institut für Informatik,
Humboldt-Universität zu Berlin, Germany
suermeli@informatik.hu-berlin.de

Zusammenfassung In Serviceorientierten Architekturen können Geschäftsprozesse durch Komposition von lose gekoppelten Services realisiert werden. Sind entsprechende Services auf Service-Marktplätzen vorhanden, kann ein Service-Konsument zwischen Services wählen, die die von ihm benötigte Funktionalität bereitstellen, basierend auf deren Dienstgüte (engl.: Quality of Service – QoS). Diese QoS-basierte Service-Selektion wird in der Literatur zumeist für Zustandslose Services durchgeführt. Zustandsbehaftete Services können, je nach tatsächlichem Ausführungspfad, verschiedene Ausprägungen in ihren QoS-Attributen annehmen, was in verwandten Arbeiten bisher nicht berücksichtigt wird. In der vorliegenden Arbeit werden Ansätze skizziert, mit denen die QoS-basierte Service-Selektion auch für Zustandsbehaftete Services durchgeführt werden kann.

1 Einleitung

In hoch kompetitiven Märkten, in denen die agierenden Unternehmen ähnliche Produkte und Services anbieten (wie bspw. in der Finanzindustrie), ist es erforderlich, dass die Geschäftsprozesse effizient ausgeführt werden. Zudem müssen Unternehmen in solchen Märkten in der Lage sein, ihre Geschäftsprozesse dynamisch und flexibel an sich ändernde, marktgetriebene Rahmenbedingungen anzupassen. Diese Flexibilität können Unternehmen durch die Umsetzung einer Serviceorientierten Architektur (SOA) erreichen, in der lose gekoppelte Services – mit einer mehr oder weniger grob granularen Funktionalität (vgl. [1]) – für die Realisation der Unternehmenseigenen Geschäftsprozesse eingesetzt werden. In diesem Zusammenhang – um agile Geschäftsprozesse zu ermöglichen und sie zu unterstützen – wird das SOA Paradigma häufig empfohlen [2].

Um die Realisation der Geschäftsprozesse möglichst effizient zu gestalten, sollten diejenigen Services eingesetzt werden, die sowohl die notwendigen Funktionalitäten bereitstellen als auch den qualitativen Anforderungen der Unternehmen genügen. Diese Services müssen dabei nicht notwendigerweise ausschließlich im eigenen Unternehmen vorhanden sein. Sind (aus funktionaler Sicht) entsprechende Services auf Service Marktplätzen verfügbar – wie in der Vision des

Internet of Services postuliert – können Unternehmen zwischen Services (die die benötigte Funktionalität bereitstellen) basierend auf deren Qualität (engl.: Quality of Service – QoS) wählen. Dieses Service-Selektions-Problem (SSP) wurde in der Literatur bereits in vielen Arbeiten (bspw. in [3–5]) adressiert, jedoch lediglich für Zustandslose Services gelöst. Bei Zustandsbehafteten Services sind die zur Laufzeit aufgerufenen Operationen innerhalb der Services im Vorhinein unbekannt, sodass zur Planungszeit (Zeitraum, in dem das SSP gelöst wird) nicht mit Sicherheit gesagt werden kann, welche QoS-Attribute die einzelnen Services zur Laufzeit haben werden. Dieses Problem wird in der vorliegenden Arbeit adressiert.

Im folgenden Abschnitt 2 wird die vorliegende Arbeit von verwandten Arbeiten abgegrenzt. In Abschnitt 3 wird das SSP (für Zustandslose Services) vorgestellt, für dessen Lösung feste (eindeutige) QoS-Werte erforderlich sind. Die besonderen Eigenschaften für Zustandsbehaftete Services werden anschließend in Abschnitt 4 beschrieben. Darauf aufbauend skizziert Abschnitt 5 mögliche Ansätze für das SSP mit Zustandsbehafteten Services. Abschnitt 6 dient der Zusammenfassung des vorliegenden Beitrags sowie der Vorstellung von Ideen für zukünftige Arbeiten.

2 Verwandte Arbeiten

Das SSP (für Zustandslose Services) wurde in der Literatur bereits von vielen Autoren adressiert. In einigen Arbeiten (bspw. in [3, 6, 7]) werden heuristische Lösungsansätze vorgeschlagen. Ansätze, die auf eine optimale Lösung des SSP abzielen, werden in [8,9] beschrieben. Um das SSP für komplexe Workflowpatterns zu lösen, wird bei diesen Arbeiten für jeden möglichen (sequenziellen) Ausführungspfad eine Lösung mithilfe von Standardmethoden (wie Branch & Bound) aus dem Bereich des Operations Research [10] erstellt. Dies umfasst auch Zyklische Strukturen, sodass solche Strukturen zunächst offen gelegt werden müssen. Insofern ist Kenntnis und Berücksichtigung aller möglichen Ausführungspfade, deren Anzahl mit jeder zusätzlichen Verzweigung exponentiell ansteigt, für die Berechnung einer optimalen Lösung Voraussetzung.

Der vorliegende Ansatz zielt ebenfalls auf die Berechnung einer (nahezu) optimalen Lösung für das SSP ab. Die Kenntnis aller möglichen Ausführungspfade wird dabei jedoch nicht benötigt, wodurch sich der vorliegende Ansatz von verwandten Arbeiten in diesem Bereich abgrenzt. Des Weiteren können auch rekursive Verschachtelungen von Workflowpatterns berücksichtigt werden, was nach unserem Wissensstand in verwandten Arbeiten nicht adressiert wurde.

3 Service-Selektions-Problem

Bei der QoS-basierten Service-Selektion geht es darum, für einen abstrakten Workflow (bspw. in Business Process Modeling Notation – BPMN) konkrete Services zu finden, die die einzelnen Workflow-Schritte (d. h., die abstrakten Services) realisieren. Das Ergebnis der Service-Selektion stellt dabei einen Ausführungsplan

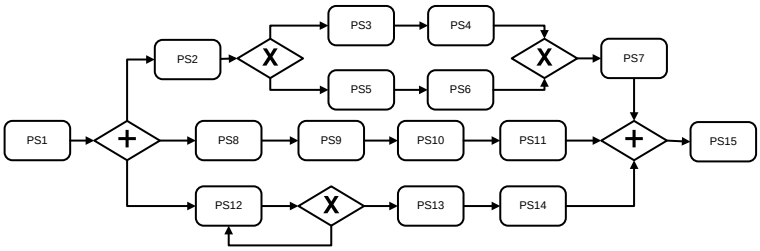


Abbildung 1: Beispiel für einen Workflow

dar, in dem eine Zuordnung von konkreten Services zu den einzelnen abstrakten Services vorgenommen wird.

Um ein solches SSP spezifizieren und lösen zu können, müssen die QoS-Attribute der infrage kommenden Services entsprechend ihrer Anordnung im Workflow aggregiert werden. In der vorliegenden Arbeit berücksichtigen wir dabei die Workflowpatterns *Sequenz*, *AND-Split/-Join*, *XOR-Split/-Join*, die in [12] beschrieben sind, sowie *Simple Loop* (vgl. [13]). Ein Beispiel für einen solchen Workflow ist in Abbildung 1 gegeben. Die Workflowschritte sind in dieser Abbildung mit PS abgekürzt.

Die Menge der abstrakten Services bezeichnen wir mit I, $i \in I = \{1, ..., n\}$. Jedem abstrakten Service wird exakt ein konkreter Service $j \in J_i = \{1, ..., m_i\}$ zugeordnet. Dabei geben die Entscheidungsvariablen $x_{ij} \in \{0, 1\}$ wider, ob ein konkreter Service j einem abstrakten Service i zugeordnet ist. Als nichtfunktionale bzw. QoS-Parameter verwenden wir Ausführungszeit e (benötigte Zeit, um einen Service auszuführen), Kosten c (Kosten für die Invokation eines Services), Zuverlässigkeit r (Wahrscheinlichkeit, dass der Service erfolgreich ausgeführt wird), sowie Durchsatz d (Anzahl paralleler Service Invokationen). Mit diesen Parametern lassen sich die Aggregationstypen Summation, Multiplikation sowie Min/Max-Operator abdecken, sodass weitere QoS-Parameter, die zu diesen Aggregationstypen gehören, leicht eingefügt werden können. Bezüglich möglicher Verzweigungen definieren wir die Menge L der Pfade l als $l \in L = \{1, ..., l^{\#}\}$. D. h., l stellt die entsprechende Pfad-Nummer innerhalb einer Verzweigung dar. Die AND-Verzweigung nach $PS1$ erzeugt drei Pfade l, d. h. $L = \{1, 2, 3\}$. Die Menge der abstrakten Services innerhalb einer Verzweigung L wird mit $IW_L \subseteq I$ bezeichnet. $IW_l \subseteq IW_L$ stellt die Menge der abstrakten Services eines Pfads l dar. Das Ergebnis der Service-Selektion, d. h. die Menge der selektierten Services, wird durch S dargestellt. Gibt es bei dem betrachteten Workflow mehrere Ausführungsmöglichkeiten bzw. mehrere Ausführungspfade (wie bspw. bei dem XOR-Split nach $PS2$ in Abbildung 1), ist zur Planungszeit nicht bekannt, welcher der möglichen Pfade ausgeführt wird.

In einer Average-Case Analyse werden Wahrscheinlichkeiten p_l für mögliche Pfade l angenommen, die angeben, mit welcher Wahrscheinlichkeit (bei einem XOR-Split) ein bestimmter Pfad ausgeführt wird, und bei Berechnung einer

QoS	**Sequenz**	**AND-Split/-Join**	**XOR-Split/-Join**	**Loop**
e	$\sum_{i \in IS} \sum_{j \in J_i} e_{ij} x_{ij}$	$\max_{l \in L}(\sum_{i \in IW_l} \sum_{j \in J_i} e_{ij} x_{ij})$	$\sum_{l \in L} p_l \sum_{i \in IW_l} \sum_{j \in J_i} e_{ij} x_{ij}$	$\frac{1}{1-\rho_i} e_{ij}$
c	$\sum_{i \in IS} \sum_{j \in J_i} c_{ij} x_{ij}$	$\sum_{l \in L} \sum_{i \in IW_l} \sum_{j \in J_i} c_{ij} x_{ij}$	$\sum_{l \in L} p_l \sum_{i \in IW_l} \sum_{j \in J_i} c_{ij} x_{ij}$	$\frac{1}{1-\rho_i} c_{ij}$
r	$\prod_{i \in IS} \sum_{j \in J_i} r_{ij} x_{ij}$	$\prod_{l \in L} \prod_{i \in IW_l} \sum_{j \in J_i} r_{ij} x_{ij}$	$\sum_{l \in L} p_l \prod_{i \in IW_l} \sum_{j \in J_i} r_{ij} x_{ij}$	$\frac{(1-\rho_i) r_{ij}}{1-\rho_i r_{ij}}$
d	$\min_{i \in IS}(\sum_{j \in J_i} d_{ij} x_{ij})$	$\min_{l \in L}(\min_{i \in IW_l} (\sum_{j \in J_i} d_{ij} x_{ij}))$	$\sum_{l \in L} p_l \min_{i \in IW_l} (\sum_{j \in J_i} d_{ij} x_{ij})$	d_{ij}

Tabelle 1: Average-Case Aggregationsfunktionen

Lösung für das SSP berücksichtigt. Hierfür verwenden wir die Aggregationsfunktionen in Tabelle 1, die in unserer Arbeit in [14, 15] erläutert werden. Die Berücksichtigung von Loops ist ebenfalls dort beschrieben. Die berechnete Lösung (als Ergebnis der Optimierung) spiegelt dann den Mittelwert über alle Pfade wider. Insofern werden gegebene untere bzw. obere Schranken (Restriktionen für die QoS-Parameter) lediglich (rechnerisch) im Durchschnitt eingehalten. Die Durchführung einer solchen Average-Case-Analyse wird in unserer Arbeit in [14] vorgestellt und daher in der vorliegenden Arbeit nicht weiter beschrieben.

Bei einer Worst-Case-Analyse betrachtet man im Unterschied zu der Average-Case-Analyse für jeden QoS-Parameter den schlechtest möglichen Pfad – bspw. den Pfad mit der größten aggregierten Ausführungszeit oder mit der geringsten Zuverlässigkeit. Dies können durchaus unterschiedliche Pfade für die verschiedenen QoS-Parameter sein. Das Ergebnis einer solchen Service-Selektion für den Worst-Case wäre ein Ausführungsplan, der die QoS-Restiktionen für den gesamten Workflow auf keinen Fall verletzt. Bei einer Best-Case-Analyse wird analog der Worst-Case-Analyse für jeden QoS-Parameter der best mögliche Pfad bei der Optimierung berücksichtigt. Das bedeutet, dass für die jeweils anderen Pfade die Restriktionen nicht berücksichtigt und insofern (höchstwahrscheinlich) verletzt werden. Das Ergebnis einer solchen Service-Selektion führt zu einem Ausführungsplan, bei dem die Zielfunktion den best möglichen Wert im Vergleich zur Worst-Case und Avg-Case-Analyse aufweist, jedoch werden die Restriktionen möglicherweise in keinem der möglichen Ausführungspfade eingehalten. Bezüglich der in Tabelle 1 angegebenen Average-Case-Aggregationsfunktionen würden sich bei einer Worst-/Best-Case-Analyse lediglich die Funktionen für XOR-Split/-Join ändern – wie in [15] beschrieben – und es würden untere (Best-Case) bzw. obere (Worst-Case)-Schranken für die Anzahl an Iterationen bei Loops angenommen. Diese (geänderten) Funktionen werden dann für die Optimierung des SSPs herangezogen.

Unabhängig davon, welche dieser drei möglichen Analysen schlussendlich durchgeführt wird, würde bei der Optimierung des SSPs (wie in [14]) davon ausgegangen, dass für jeden QoS-Parameter eines Services genau ein Wert existiert. Werden jedoch Zustandsbehaftete Services betrachtet, kann diese Annahme nicht getroffen werden. Stattdessen stehen hier Wertebereiche für die einzelnen QoS Werte zur Verfügung, was im folgenden Abschnitt erläutert wird.

4 Zustandsbehaftete Services

Ein Zustandsbehafteter Service ist ein offenes System mit eigenem Kontrollfluss und eigenen Zuständen. Ein Zustand umfasst wie bei einem abgeschlossenen System den aktuellen Wert aller Systemgrößen. Der Kontrollfluss ändert den Zustand des Systems basierend auf dem aktuellen Zustand: Es werden Entscheidungen basierend auf Systemgrößen oder nichtdeterministisch getroffen. Zudem werden besagte Systemgrößen manipuliert. Der Unterschied zu einem abgeschlossenen System besteht darin, dass ein Service mit seiner Umwelt kommuniziert. Diese Umwelt kann wiederum aus mehreren Services bestehen. So kann beispielsweise ein Service zur Buchung einer Reise an einen Service für die Hotelsuche und einen Service zur Flugsuche gebunden sein. Die Ein- und Ausgabedaten eines Zustandsbehafteten Services sind also Ein- und Ausgabeströme von Nachrichten der Umwelt. Die Reaktion des Services auf eine bestimmte Nachricht seiner Umwelt ist stets abhängig von seinem inneren Zustand: So wird beispielsweise eine bestimmte Anfrage a erst genehmigt, wenn von der Umwelt vorher bereits Nachricht b gesendet wurde. An seinem inneren Zustand kann der Service erkennen, ob b bereits empfangen wurde oder nicht.

Da zur Planungszeit das genaue Verhalten der Umwelt nicht bekannt ist, also nicht vorausgesagt werden kann, welche Nachrichten genau an den Service gesendet werden, können wir auch nicht genau sagen, welcher Pfad des Kontrollflusses tatsächlich während der Ausführung gewählt wird. Daher ist auch zur Planungszeit nicht klar, welche Operationen wie häufig und in welcher Reihenfolge ausgeführt werden. Dies macht die Angabe von fixen QoS-Werten für einen Zustandsbehafteten Service in der Regel unmöglich.

Es ist jedoch möglich, Wertebereiche V für jeden einzelnen QoS-Parameter anzugeben (bspw. V_e, V_c, V_r, V_d), die jeden möglichen Pfad des Kontrollflusses abdecken. Die einfachste Variante dafür ist ein Intervall, das eine untere und eine obere Grenze für jeden QoS-Parameter angibt. In einigen Fällen ist es schwierig oder gar unmöglich, eine genaue obere oder untere Grenze für einen Parameter anzugeben. In diesem Falle muss entweder approximiert oder das fehlende Wissen über eine Schranke als offenes Intervall kodiert werden. Alternativ zu Intervallen können wir uns bspw. Paare aus Erwartungswert und Standardabweichung vorstellen. Das Finden solcher genauen oder approximierten Wertebereiche kann auf unterschiedliche Art und Weise geschehen. Eine Variante ist es, das Modell des Services zu analysieren [16]. Dabei wird jedes mögliche Verhalten der Umwelt in Betracht gezogen; das Ergebnis ist ein Intervall. Ohne weiteres Wissen über das bevorzugte Verhalten der Umgebung ist es nicht möglich, die Werte im Intervall zu gewichten. Eine weitere Variante wäre Monitoring, wobei alle Ausführungen des Services aufgezeichnet werden (vgl. [17]). Anhand dieser gespeicherten Informationen können möglicherweise Rückschlüsse auf nachfolgende Ausführungen getroffen werden. Die Kombination beider Verfahren würde die Gewichtung der Werte innerhalb des Intervalls erlauben.

5 Service-Selektion bei Zustandsbehafteten Services

Stehen keine eindeutigen QoS-Werte für die einzelnen Services zur Verfügung sondern Wertebereiche, gestaltet sich der Vergleich zweier Services bezüglich ihrer QoS schwieriger. Dies wird an folgendem Beispiel verdeutlicht: Kostet ein Service S1 bspw. fest 2ct pro Invokation und benötigt er dabei fest 3s (Ausführungszeit), so können diese Werte mit denjenigen von Service S2 verglichen werden, der bspw. fest 3ct kostet bei einer festen Ausführungszeit von 2s. Hinsichtlich Kosten wäre S1 besser als S2, hinsichtlich Ausführungszeit wäre S2 zu bevorzugen. Über die individuelle Präferenz des Nutzers hinsichtlich der verschiedenen QoS-Parameter kann dann entschieden werden, welcher Service ausgewählt werden soll. Sind jedoch Wertebereiche anstatt fester Werte für jeden QoS-Parameter gegeben, ist diese Entscheidung nicht mehr trivial. Zwei Services können ohne Weiteres nicht miteinander verglichen werden. Insofern kann auch kein Ranking für die Kandidaten von Services für eine abstrakte Aktivität erstellt werden. Für die Service-Selektion müssen Services jedoch miteinander verglichen werden können.

Im Folgenden werden daher Ansätze skizziert, um das SSP bei Vorliegen von Wertebereichen für einzelne QoS-Parameter zu lösen. Für jeden Ansatz wird eine Annahme bezüglich der Spezifikation des Wertebereichs getroffen. Für den jeweils erstellten Ausführungsplan können die QoS für den gesamten Workflow zwar nicht eindeutig bestimmt werden, sie haben allerdings je nach Verfahren bestimmte Eigenschaften.

5.1 Worst QoS – Worst Case

Für dieses Verfahren wird angenommen, dass der Wertebereich in Form eines geschlossenen Intervalls vorliegt. Für jeden Service wird für jeden QoS-Parameter der jeweils schlechteste Wert – Worst QoS – für das Optimierungsproblem herangezogen. Der schlechteste Wert ist für jeden QoS-Parameter entweder die linke oder rechte Grenze des Intervalls: $e := max(V_e)$, $c := max(V_c)$, $r := min(V_r)$, $d := min(V_d)$. Um hier tatsächlich eine Worst-Case Abschätzung zu erhalten, schlagen wir vor, eine Worst-Case-Analyse durchzuführen (vgl. Abschnitt 3). Ergebnis dieser Worst-Case-Analyse ist ein Ausführungsplan. Um nun dem Umstand Rechnung zu tragen, dass keine fixen QoS Werte sondern lediglich QoS Intervalle haben, werden in einem zweiten Schritt für die erhaltene Selektion von Services QoS-Intervalle VW_e, VW_c, VW_r, VW_d für den gesamten Workflow berechnet. Hierfür werden die unteren und oberen Schranken durch Aggregation der jeweils schlechtesten bzw. besten Werte der jeweiligen QoS-Intervalle (der selektierten Services) bestimmt. D. h., $min(VW_e) = Aggregate(min(V_{e_j})|\forall j \in S)$ und $max(VW_e) = Aggregate(max(V_{e_j})|\forall j \in S)$, wobei sich $Aggregate$ auf die Verwendung der entsprechenden Aggregationsfunktion in Tabelle 1 bezieht. Die Berechnung der entsprechenden Intervalle für die anderen QoS-Parameter erfolgt analog. Somit wird ein Ausführungsplan erstellt, für den die gegebenen QoS-Restiktionen stets eingehalten werden und für den die aggregierten QoS-Werte in den entsprechenden Intervallen $[min(VW), max(VW)]$ liegen.

Zusammenfassend sei an dieser Stelle noch mal erwähnt, dass für die Durchführung der Worst-Case-Analyse die untere/obere QoS Intervallgrenze der einzelnen Services herangezogen wird, um diese Services für die Optimierung miteinander vergleichen zu können, was (wie oben erwähnt) eine Voraussetzung zur Durchführung der Optimierung darstellt. Für die Berechnung der zu erwartenden QoS für den gesamten Workflow werden anschließend sowohl die oberen als auch unteren QoS Intervallgrenzen der ausgewählten Services berücksichtigt, um auf diese Weise das ganze QoS Intervall eines Services abdecken zu können.

5.2 Average QoS – Average Case

Ein zweiter möglicher Lösungsansatz funktioniert unter der Annahme, dass für jeden QoS-Parameter der Wertebereich in Form von Mittelwert und Standardabweichung vorliegt. Diese können bspw. durch Simulation oder Monitoring gefunden werden (vgl. Abschnitt 4). Es bietet sich die Durchführung einer Average-Case-Analyse an (vgl. Abschnitt 3), für die die besagten Mittelwerte der einzelnen QoS-Parameter herangezogen würden. Das Ergebnis wäre ein Ausführungsplan, der die Restriktionen im Durchschnitt erfüllt. Mithilfe der Standardabweichungen für die QoS-Werte der selektierten Services oder durch Simulation ließe sich ebenfalls ein Intervall für die QoS-Werte des gesamten Workflows gemäß des errechneten Ausführungsplans generieren.

5.3 Best QoS – Best Case

Analog zu Abschnitt 5.1 könnten für die Service-Selektion auch die jeweiligen besten Werte aus den QoS-Intervallen ($e := min(V_e)$, $c := min(V_c)$, $r := max(V_r)$, $d := max(V_d)$) herangezogen werden. Mit diesen würde eine Best-Case-Analyse durchgeführt. Basierend auf dem berechneten Ausführungsplans ließen sich analog zu Abschnitt 5.1 Intervalle für die QoS-Werte auf Workflow-Ebene bestimmen. Jedoch ist bei Durchführung einer Best-Case-Analyse sowie durch die Wahl der jeweils besten Werte für die QoS-Parameter die Wahrscheinlichkeit eher hoch, dass bei der Ausführung des auf diese Weise erstellten Ausführungsplans die QoS-Restriktionen verletzt werden.

6 Zusammenfassung und Ausblick

Das SSP wird in der Literatur in vielen wissenschaftlichen Arbeiten diskutiert. Nach unserem Wissensstand wurden hier jedoch stets Zustandslose Services berücksichtigt. In der vorliegenden Arbeit wurde das SSP mit Zustandsbehafteten Services adressiert. Die Herausforderung besteht dabei darin, dass für die infrage kommenden Services keine eindeutigen QoS-Werte existieren (vgl. Abschnitt 4). In unserer Arbeit in [16] haben wir diesbezüglich ein Verfahren entwickelt, mit dem sich Intervalle für die QoS-Werte approximieren lassen. Unter Verwendung dieser Intervalle wurden in Abschnitt 5 mögliche Ansätze für die Lösung des SSP mit Zustandsbehafteten Services skizziert. Unsere zukünftige Arbeit zielt auf die Umsetzung und Evaluation dieser Ansätze sowie auf deren Kombination ab.

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

Literatur

1. Krafzig, D., Banke, K., Slama, D.: Enterprise SOA: Service-Oriented Architecture Best Practices. Prentice Hall PTR, Upper Saddle River, NJ, USA (2004)
2. Papazoglou, M.P.: Service-Oriented Computing: Concepts, Characteristics and Directions. In: Web Information Systems Engineering. (2003) 3–12
3. Anselmi, J., Ardagna, D., Cremonesi, P.: A QoS-based Selection Approach of Autonomic Grid Services. In: International Conference on Service-oriented Computing. (2007) 1–8
4. Menascé, D.A., Casalicchio, E., Dubey, V.: A Heuristic Approach to optimal Service Selection in Service-oriented Architectures. In: Workshop on Software and Performance. (2008) 13–24
5. Huang, A.F.M., Lan, C.W., Yang, S.J.H.: An optimal QoS-based Web Service Selection Scheme. Information Sciences **179**(19) (2009) 3309–3322
6. Jaeger, M.C., Rojec-Goldmann, G.: SENECA-Simulation of Algorithms for Selection of Web Services for Composition. In: Technologies for E-Services. (2005) 84–97
7. Mabrouk, N.B., Georgantas, N., Issarny, V.: A semantic end-to-end QoS Model for dynamic Service oriented Environments. In: Proceedings of PESOS. (2009) 34–41
8. Ardagna, D., Pernici, B.: Adaptive Service Composition in Flexible Processes. IEEE Trans. Software Eng. **33**(6) (2007) 369–384
9. 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
10. Domschke, W., Drexl, A.: Einführung in Operations Research. Springer Verlag, Heidelberg (2007)
11. Van Der Aalst, W.M.P., Ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. Distributed Parallel Databases **14**(1) (2003) 5–51
12. Cardoso, J., Sheth, A.P., Miller, J.A., Arnold, J., Kochut, K.: QoS for Workflows and Web Service Processes. Journal of Web Semantics **1**(3) (2004) 281–308
13. Schuller, D., Miede, A., Eckert, J., Lampe, U., Papageorgiou, A., Steinmetz, R.: Qos-based optimization of service compositions for complex workflows. In: International Conference on Service Oriented Computing. (2010) 641–648
14. Schuller, D., Eckert, J., Miede, A., Schulte, S., Steinmetz, R.: QoS-Aware Service Composition for Complex Workflows. In: International Conference on Internet and Web Applications and Services. (2010) 333–338
15. Gierds, C., Sürmeli, J.: Estimating costs of a service. In: Central-European Workshop on Services and their Composition. (2010) 121–128
16. Repp, N., Schuller, D., Siebenhaar, M., Miede, A., Niemann, M., Steinmetz, R.: On distributed SLA Monitoring and Enforcement in Service-oriented Systems. International Journal On Advances in Systems and Measurements **2**(1) (2009) 33–43

Filtering Undesirable Service Substitution Behaviors using Filtering Guidelines

Jarungjit Parnjai

Humboldt-Universität zu Berlin, Institut für Informatik, Berlin, Germany.
parnjai@informatik.hu-berlin.de

Abstract. A service T can substitute a service S if T interacts with every partner of S in a deadlock-free manner. We introduce the notion of a *filtered service* of a service S with less or the same traces of external messaging behavior as S, yet can substitute S. We propose a finite representation of all *filtered* services of S called *filtering guidelines*. Given a service T that cannot substitute S, we can employ the filtering guidelines based techniques to construct a filtered service for S from T by *filtering* out certain undesirable behavior of T that are not described by the filtering guidelines.

1 Introduction

Service-orientation supports process evolution by considering a complex business process as a collaboration of several simpler, interacting services. Substituting one or more of these services by another may endanger the proper interaction in unexpected ways.

In this paper, we study the evolution of the business protocols [7] which specifies the external messaging behavior that are exchanged between stateful services. We consider the behavioral substitution criterion that is formalized by the notion of *accordance* [9]. A service T can substitute a service S under accordance if T can interact with every partner R (in every context) of S in a deadlock-free manner.

Whenever a service S changes due to various reasons, e.g. changes in regulations or the operational behavior of services, an incremental modification of an existing protocol requires a construction of new service T without the need of redefining T from scratch. Yet, constructing a new service T that can substitute a service S is a time-consuming and error-prone task, typically based on trial-and-error methods. The activity to construct a substitutable service requires a systematic support from formal approaches and tools that naturally optimize the time and effort to construct such a service.

Nevertheless, the languages and tools that are currently available on the market offer only limited support. The language WS-BPEL, for example, has rules (called profiles) allowing to transform a service S into a service T that can substitute S. These syntactical rules are usually restricted and their extensions regarding behavioral compatibility (e. g., in [1]) are still incomplete, and hence it

is not possible to construct every service that can substitute a given service S by means of transformation.

To systematically support such a construction, a finite representation of all services that can substitute a given service has been proposed in [6, 8]. Such a finite representation describes all services that can substitute a given service S under accordance, and therefore, this approach realizes several analysis and synthesis challenges of service substitution such as deciding and constructing a service that can substitute S under accordance.

Consider a service T that cannot substitute S under accordance, this means T is not describing by a finite representation proposed by [6, 8]. In many situations, a new service that is constructed from such a representation introduces either additional external message behaviors (e. g., a new order of sending message activities) or completely new behavior where none of behavior of T is preserved. Nevertheless, we may want to promote reusability and rapid development of services by *removing* undesirable behavior of T rather than introducing new behavior, and by doing so, we derive a new service T' from T such that T' can substitute S under accordance. Such a *filter* operation on service behaviors should allow the construction of a new service T', that can substitute S under accordance, from service T with optimal amount of time and effort.

In this paper, we extend the approach of [6, 8] by presenting *filtering guidelines* that realizes the *filter* operation for services. The *filtering guidelines* for service S describes all services that have less or the same traces of external messaging behavior as S, yet can substitute S under accordance. With our approach, one can construct a new *filtered service* using the filter guidelines to *filter out* the undesirable behavior that possibly introduces a deadlock when interacting with a deadlock-free partner of S. We ensure that the *filtered service* can substitute service S under accordance.

The remainder of this paper is organized as follows. Section 2 describes preliminary notions and related works. Section 3 defines filtering services and presents a finite representation of all filtering services. Finally, Section 4 concludes the paper and sketches possible extensions of the approach and future work.

2 Background and Related Works

A service consists of a control structure describing its *behavior* and an interface for asynchronous communication with other services. An interface is a set of (input and output) *channels*. We abstract from the syntax of service description languages and use *service automata* to model service behavior.

A *service automaton* [4] (or *service*, for short) consists of a finite set Q of states, an initial state $q_0 \in Q$, a set I of input channels, a set O of output channels (I and O are disjoint and do not contain internal message τ), a non-deterministic transition relation $\delta \subseteq Q \times (I \cup O \cup \{\tau\}) \times Q$, and a set Ω of finals states. A non-final state with no outgoing transition is a *deadlock*.

Example 1. Figure 1 shows the communication skeleton (or abstract processes in terms of BPEL) of three customer services S, T_1, and T_2 as service automata.

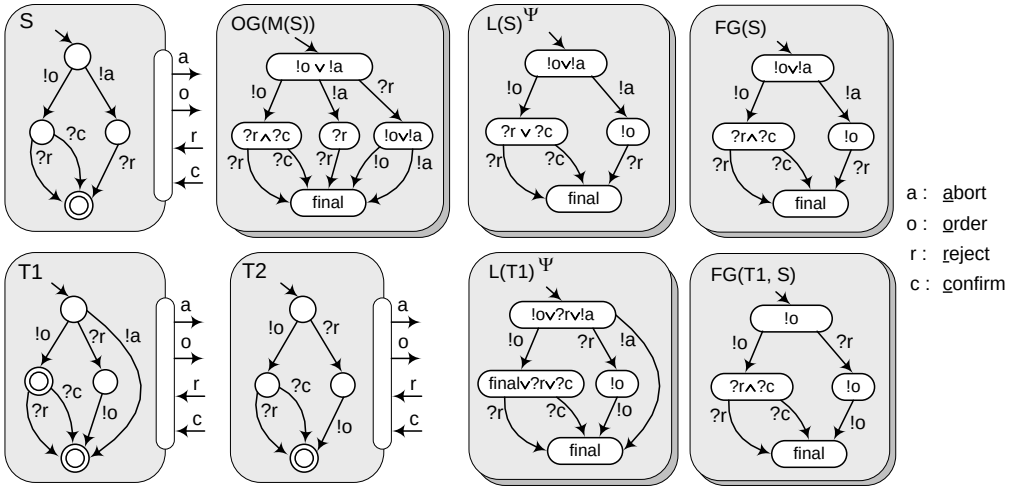


Fig. 1. Running examples

All customers have the same interface ($I = \{r, c\}$ and $O = \{a, o\}$) and can send an order (labeled $!o$), send an abort message (labeled $!a$), receive a reject message (labeled $?r$), or receive a confirm message (labeled $?c$). Each customer has different behavior described by its control flow. Final states are depicted with double circles.

We assume asynchronous communication between services with bounded and unordered message buffer (i. e., messages may overtake each other in a bounded channel). Two services R and S can be *composed* if they have compatible interface (input channels of R are the output channels of S, and vice versa). The composition yields a service with empty interface. Two services R and S interact properly if their composition is *deadlock-free* (every reachable non-final state has an outgoing transition). In that case, R is a *controller* (i. e., deadlock-free partner) of S. The set $\mathit{Controllers}(S)$ denotes the set of all controllers of S. The notion of a controller is symmetric; if R is a controller of S, then S is also a controller of R. If S has at least one controller, then S is *controllable*.

A service T can substitute a service S under *accordance* [9] if (1) S and T have the same interface (same input and output channels) and (2) every controller of S is a controller of T (i. e., $\mathit{Controllers}(S) \subseteq \mathit{Controllers}(T)$). The set $\mathit{Accord}(S)$ denotes the set of all services T that *accords with* S (i. e., T can substitute a service S under accordance).

Example 2. In Fig. 1, service T_1 cannot substitute service S under accordance. This is because T_1 terminates after sending abort message, but there is one deadlock-free partner of S (not shown in the examples) which, after receiving abort message, will send back a reject message. Obviously, T_1 cannot interact

deadlock-freely with this partner of S. However, T_2 can substitute S under accordance, because T_2 can interact deadlock-freely with every partner of S.

Each controllable service S has a *most-permissive controller* of S, denoted by $mp(S)$, which can exhibit all behaviors that any controller of S can exhibit (i. e., if $R \in \mathit{Controllers}(S)$, then $mp(S)$ must simulate R [10]). The set of all controllers of a controllable service S can be represented by the *operating guidelines* $OG(S)$ of S [4], denoted by $OG(S)$. The operating guidelines $OG(S)$ of S, is the Boolean annotated most-permissive controller $mp(S)^\phi$ of S, where each state q of $mp(S)$ of S is annotated with a Boolean formula $\phi(q)$. These formulas consist of conjunctions $\wedge$, disjunctions $\vee$, and atomic propositions $I \cup O \cup \{\tau, \mathit{final}\}$, indicating for a state whether certain outgoing edges are present and whether the state is final.

To determine whether a service R is a controller of S, we analyze whether R matches with $OG(S)$, denoted by $R \in \mathit{Match}(OG(S))$. Service R *matches* with $OG(S)$ if R has the same interface as $mp(S)$ and $mp(S)$ simulates R such that, for each pair of simulated states (q_R, q_m) the Boolean formula $\phi(q_m)$ at state q_m of $mp(S)$ is satisfied in the assignment β of state q_R of R. An assignment β is a Boolean function on $I \cup O \cup \{\tau, \mathit{final}\}$ that assigns x, for $x \in I \cup O \cup \{\tau\}$, to true if there exists an outgoing transition from q_R with labels x, and assigns *final* to true if q_R is a final state of R.

Proposition 1 ([4]). $\mathit{Controllers}(S) = \mathit{Match}(OG(S))$.

In addition to the operating guidelines of service S, the set of all controllers of a service S can also be represented by a single controller called *maximal controller* (or called *maximal strategy* in [8]) of S, denoted by $M(S)$. For each controlllable service S, every controller R of S can substitute $M(S)$ under accordance [5]. A maximal partner $M(S)$ for S has several useful applications to service substitution [6, 8] and one of them is to characterize the set $\mathit{Accord}(S)$ of all services that can substitute a service S under accordance.

To represent the set $\mathit{Accord}(S)$ of all services that can substitute a service S under accordance, we calculate the operating guidelines of a maximal controller $M(S)$ of S, denoted by $OG(M(S))$. To determine whether a service T accords with S (i. e., if $T \in \mathit{Accord}(S)$), we analyze whether T matches with $OG(M(S))$ [6, 8], denoted by $R \in \mathit{Match}(OG(M(S)))$.

Proposition 2 ([6, 8]). $\mathit{Accord}(S) = \mathit{Match}(OG(M(S))$.

Example 3. $OG(M(S))$ depicted in Fig. 1 represents the set $\mathit{Accord}(S)$ of all services that can substitute S under accordance. T_2 matches with $OG(M(S))$ because $OG(M(S))$ simulates T_2 and every state of T_2 fulfills an assignment of a Boolean formula at the respective state of $OG(M(S))$. However, T_1 does not match with $OG(M(S))$ because T_1 terminates after sending abort message $!a$; this means, the assignment at the state in T_1 after sending abort message $!a$ does not satisfy the formula $?r$ at the respective state of $OG(M(S))$. Therefore, $T_1 \notin \mathit{Match}(OG(M(S))$ and $T_2 \in \mathit{Match}(OG(M(S)))$. We conclude from Proposition 2 that T_2 can substitute S under accordance, but T_1 cannot do so.

3 Filtering Guidelines

Given service S, we propose *filtering guidelines* for S which can describe all services S' each has less or the same traces of external messaging behavior to S, yet can substitute S under accordance.

For a service S, a *trace* of S is a finite or infinite sequence of non-internal messages from the initial state of S. The set $\mathit{Traces}(S)$ denotes the set of all traces of S. A service T *refines* service S if every trace of T can be replayed in S (i. e., $\mathit{Traces}(T) \subseteq \mathit{Traces}(S)$). For the attentive reader, this relation is also known as trace refinement relation. The set $\mathit{Refine}(S)$ denotes the set of all services that refines S.

Example 4. In Fig. 1, $T_2 \in \mathit{Refine}(T_1)$ and $T_1, T_2 \notin \mathit{Refine}(S)$.

Definition 1. *A service T is a* filtered service *of a service S if T refines S and T accords with S. We define the set of all filtered services of S as $\mathit{Filter}(S) = \mathit{Refine}(S) \cap \mathit{Accord}(S)$.*

Observe that the set $\mathit{Filter}(S)$ is never an empty set, as it always includes S.

Our first goal is to construct a finite representation of $\mathit{Filter}(S)$. As suggested by Definition 1, the set $\mathit{Filter}(S)$ is an intersection of the two sets $\mathit{Refine}(S)$ and $\mathit{Accord}(S)$. Technically, if each set can be represented by a Boolean annotated service automaton [4, 9], we can employ the operating guidelines-based technique proposed by [9] to characterize the intersection of two given sets using the product of two Boolean annotated service automata.

For this purpose, we require two main ingredients: (1) a finite representation of $\mathit{Accord}(S)$ and (2) a finite representation of $\mathit{Refine}(S)$.

The first ingredient has been proposed by [6, 8] to compute a finite representation of $\mathit{Accord}(S)$ as the operating guidelines of a distinguished controller of S, called a maximal controller $M(S)$ of S (see also Proposition 2 in Section 2).

For the second ingredient, we require a finite representation of $\mathit{Refine}(S)$. Observe that service automaton S is also a natural candidate for our second ingredient. Nevertheless, we would like to employ the operating guidelines-based techniques from [4, 9] to characterize the intersection of two service sets using the product of Boolean annotated service automata. Therefore, we require both of our ingredients to be represented as Boolean annotated service automata.

To compute a finite representation of $\mathit{Refine}(S)$, we first define a *liberal service* of a service S as the deterministic service automaton $L(S)$ such that $L(S)$ has the same interface as S, the same set of traces as S, and simulates S.

Next, we define the annotation function ψ of $L(S)$ as a mapping ψ from the set of states of $L(S)$ to the Boolean formulas over literals $I_{L(S)} \cup O_{L(S)} \cup \{\tau, \mathit{final}\}$. A formula $\psi(q)$ at each state q of $L(S)$ is a disjunction $\vee$ of atomic propositions $x \in I_{L(S)} \cup O_{L(S)} \cup \{\tau, \mathit{final}\}$ where each atomic proposition indicates that either from state q there is an outgoing edge with label x (in case $x \in I_{L(S)} \cup O_{L(S)} \cup \{\tau\}$), or state q is a final state (in case $x = \mathit{final}$).

The following lemma shows that an annotated liberal service $L(S)^{\psi}$ characterizes the set $\mathit{Refine}(S)$.

Lemma 1. *Refine*$(S) = Match(L(S)^{\psi})$.

Proof. We will prove this lemma in two directions.

$\subseteq$: Suppose $T \in Refine(S)$; i. e., $Traces(T) \subseteq Traces(S)$ holds. Because $L(S)$ simulates S by construction, it follows that $L(S)$ also simulates T. Because the formula $\psi(q)$ at state q in $L(S)$ is the disjunction of all literals $x \in I \cup O \cup \{\tau\}$ for an outgoing transition from q with labels x, and of literal *final* if q is a final state in $L(S)$; the Boolean formula $\psi(q)$ at every state q of $L(S)$ is always satisfied in the assignment β of state q_T in T that is simulated by state q in $L(S)$. Thus, $T \in Match(L(S)^{\psi})$ holds.

$\supseteq$: Suppose $T \in Match(L(S)^{\psi})$; i. e., $L(S)$ simulates T and for each simulated pair (q_T, q_L) the state q_T satisfies the formula $\phi(q_L)$. Consider $\sigma \in Traces(T)$. We have $\sigma \in Traces(L(S))$ following from $L(S)$ simulates T. Because $Traces(S) = Traces(L(S))$ by construction, it follows that $\sigma \in Traces(S)$ holds and $Traces(T) \subseteq Traces(S)$ follows. Thus, $T \in Refine(S)$ holds. □

Example 5. Figure 1 shows two annotated liberal services $L(S)^{\psi}$ of S and $L(T_1)^{\psi}$ of T_1. We can see that $T_2 \in Match(L(T_1)^{\psi})$ but $T_2 \notin Match(L(S)^{\psi})$. We conclude from Lemma 1 that $T_2 \in Refine(T_1)$ but $T_2 \notin Refine(S)$.

Given the two ingredients, we employ the product of Boolean annotated service automata [9] to characterize the intersection of *Refine*(S) and *Accord*(S). The product of the two annotated service automata is an annotated service automaton that characterizes the intersection of all services that match with these two service automata. The product of annotated service automata assumes two input annotated service automata with the same interface. Technically, the product can be derived from the synchronous product of two annotated service automata where each state of the product is annotated by the conjunction $\wedge$ of two formulas that contribute to the synchronizing state.

Proposition 3 ([9]). *Let* $OG_{\otimes} = OG(S_1) \otimes OG(S_2)$ *be the product of two annotated service automata* $OG(S_1)$ *and* $OG(S_2)$*, Then* $Match(OG_{\otimes}) = Match(OG(S_1)) \cap Match(OG(S_2))$.

We refer to the product of annotated service automata $L(S)^{\psi}$ and $OG(M(S))$ as *filtering guidelines* $FG(S)$ for S, i. e., $FG(S) = L(S)^{\psi} \otimes OG(M(S))$. Theorem 1 shows that $FG(S)$ describes the set $Filter(S)$ of all filtered services of S.

Theorem 1. $Filter(S) = Match(FG(S))$.

Proof.
$Filter(S) = Refine(S) \cap Accord(S)$, [by definition]
$= Match(L(S)^{\psi}) \cap Match(OG(M(S)))$, [Lemma 1, Proposition 2]
$= Match(L(S)^{\psi} \otimes OG(M(S)))$, [Proposition 3]
$= Match(FG(S))$. [by definition] □

Example 6. Figure 1 shows filtering guidelines $FG(S)$ for S which describes all filtered services of S.

Given two services T and S with the same interface but T cannot substitute S, we generalize our filtering guidelines to describe all services T' that have less or the same traces of external messaging behavior to T, yet T' can substitute S under accordance. To this end, we first define a *filtered* service of T for S.

Definition 2. *A service T' is a* filtered *service of T for S if T' refines T and T' accords with S. We define the set of all filtered services of T for S as $Filter(T, S) = Refine(T) \cap Accord(S)$.*

Clearly, $Filter(T, S)$ is possibly an empty set, this means there exists no filtered service of T for S.

Next, we propose a procedure to decide if the set $Filter(T, S)$ of all filtered services of T for S is empty. In case it is not empty, we provide an artifact that represents the set $Filter(T, S)$.

The result from Theorem 1 suggests that we can employ the product of annotated service automata technique to characterize the intersection of $Refine(T)$ and $Accord(S)$. This means, we compute an annotated liberal service $L(T)^{\psi}$ of T as a finite representation of $Refine(T)$, the operating guidelines $OG(M(S))$ as a finite representation of $Accord(S)$, and then the product of the two service automata $L(T)^{\psi}$ and $OG(M(S))$ as a finite representation of $Filter(T, S)$. The product of the $L(S)^{\psi}$ and $OG(M(S))$ is called *filtering guidelines* $FG(T, S)$ of T for S, i. e., $FG(T, S) = L(T)^{\psi} \otimes OG(M(S))$.

Corollary 1. $Filter(T, S) = Match(FG(T, S))$.

To decide if there exists a filtered service of T for S, we check if $FG(T, S)$ is empty. An empty $FG(T, S)$ means that it is not possible to synthesize from T a service that both refines T and accords with S. In case of non-empty $FG(T, S)$, we can synthesize a filtered service T' from $FG(T, S)$ by removing all annotations from $FG(T, S)$. The underlying service automaton of $FG(T, S)$ without annotation represents a *most-liberal* filtered service where all undesirable behaviors have been removed only if it is necessary to do so.

Example 7. Figure 1 shows the filtering guidelines $FG(T_1, S)$ of T_1 for S. We see that T_2 matches with $FG(T_1, S)$; therefore, T_2 is a filtered service of T_1 for S (i. e., $T_2 \in Filter(T_1, S)$). However, T_1 does not match with $FG(T_1, S)$; therefore, T_1 itself is not a filtered service of T_1 for S (i. e., $T_1 \notin Filter(T_1, S)$). Given T_1 that cannot substitute S. We can use $FG(T_1, S)$ as guidelines to remove the sending abort message $!a$ from T_1 as it is not described by $FG(T_1, S)$. By doing so, we can derive T_2 that can substitute S from T_1.

We can also apply all existing techniques for operating guidelines to our filtering guidelines $Filter(S)$ and $Filter(T, S)$. For example, suppose we want to impose additional requirements on the service T' that is described by either $Filter(S)$ or $Filter(T, S)$. Then we can restrict our filtering guidelines to services that satisfy certain behavioral constraints [3], or perform certain activities [9].

In case it is not possible to synthesize a service that refines T and accords with S, we can employ the simulation-based graph edit distance approach from [2]

on the finite representation of $Accord(S)$ to compute edit actions that are needed for transforming T into T' that accords with T, possibly by means of adding new messaging behavior into T'. Though the simulation-based graph edit distance approach is applicable only for acyclic and deterministic services.

4 Conclusion and Future Work

We presented the notion of a *filtered service* which has less or the same traces of external messaging behavior as a given service, yet can substitute a given service under the substitution criterion called *accordance*. To describe all filtered services, we proposed a finite representation of all filtered services called *filtering guidelines*. The filtering guidelines realizes the *filter* operation by suggesting all possible construction of a filtered service by removing certain undesirable behaviors that are not described by the guidelines. We ensure that the *filtered service* can substitute a given service under accordance.

The idea of the filtering guidelines for services is related to [3] in the sense that all trace-refined services are expressed as behavioral constraints and all trace-refined services that are not substitutable services can be *filtered out*, yielding a customized operating guideline which represents the set of all filtered services.

It is further work to implement our filtering guidelines and obtain experimental results. We also plan to extend our approach to realize the filter operation for services with different input and output channels by blocking some channels of a given service in addition to its external messaging behavior.

References

1. König, D., Lohmann, N., Moser, S., Stahl, C., Wolf, K.: Extending the compatibility notion for abstract WS-BPEL processes. In: WWW 2008. pp. 785–794 (Apr 2008)
2. Lohmann, N.: Correcting deadlocking service choreographies using a simulation-based graph edit distance. In: BPM 2008. LNCS, vol. 5240, pp. 132–147. Springer (Sep 2008)
3. Lohmann, N., Massuthe, P., Wolf, K.: Behavioral constraints for services. In: BPM 2007. LNCS, vol. 4714, pp. 271–287. Springer (2007)
4. Lohmann, N., Massuthe, P., Wolf, K.: Operating guidelines for finite-state services. In: ICATPN 2007. LNCS, vol. 4546, pp. 321–341. Springer (2007)
5. Mooij, A.J., Voorhoeve, M.: Proof techniques for adapter generation. In: WS-FM 2008. LNCS, vol. 5387, pp. 207–223. Springer (2009)
6. Mooij, A., Parnjai, J., Stahl, C., Voorhoeve, M.: Constructing substitutable services using operating guidelines and maximal controllers. In: WS-FM 2010. To appear (2010)
7. Papazoglou, M.P.: The challenges of service evolution. In: CAiSE 2008. LNCS, vol. 5074, pp. 1–15. Springer (2008)
8. Parnjai, J., Stahl, C., Wolf, K.: A finite representation of all substitutable services and its applications. In: ZEUS 2009. pp. 8–14. CEUR vol. 438 (Mar 2009)
9. Stahl, C., Massuthe, P., Bretschneider, J.: Deciding Substitutability of Services with Operating Guidelines. ToPNoC II 2(5460), 172–191 (Mar 2009)
10. Wolf, K.: Does my service have partners? ToPNoC 5460(II), 152–171 (Mar 2009), special Issue on Concurrency in Process-Aware Information Systems

On BPMN Process Fragment Auto-Completion

Oliver Kopp, Frank Leymann, David Schumm, and Tobias Unger

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

Abstract. Process fragments provide reusable granules of business processes to enable process modeling based on existing knowledge. Current verification tools cannot deal with BPMN process fragments and support complete BPMN processes only. To enable verification for BPMN process fragments, we sketch how a single BPMN fragment can be completed to a BPMN process, where additional gateways and start events are added. ¡

1 Introduction

A process fragment is intended as a reusable granule for business process design: Parts, which are reoccurring in multiple processes do not have to be modeled from scratch, but stored in a process fragment library, where they can be managed and retrieved. The main characteristics of process fragments are (i) control links with either no source or no target (called fragment entries and fragment exits or dangling edges) and (ii) place holders for variability (called regions). Regarding publicly available tool support for BPMN process verification, there is a Petri net semantics for BPMN available [5] and model checkers for Petri Nets (e.g. LoLa [18]). These tools, however, lack support for handling BPMN process fragments. To enable existing tooling to handle BPMN process fragments, there are two possibilities: (i) modify the tooling to support BPMN process fragments and (ii) to provide a method to add a minimum set of elements to a BPMN process fragment to form a complete business process without any non-standard elements. This enables any tooling to exploit the reuse concept of process fragments. This paper reasons on the second approach. The objective is to generate a BPMN process, where for each task there exists a process execution path. In other words, the resulting BPMN process has to be relaxed sound [3]. We opted for this criterion as it provides "an adequate correctness understanding" [4].

We present our concept of BPMN process fragments in Sect. 2 and discuss the auto-completion issues in Sect. 3. Section 4 presents related work and Sect. 5 concludes and discusses future work.

2 BPMN Process Fragments

In this paper, we follow the fragment definition by Schumm et al. [20]. There, a process fragment is defined as "connected graph, however with significantly relaxed completeness and consistency criteria compared to an executable process graph.

[...] A process fragment has to consist of at least one activity and there must be a way to complete it to an executable process graph". To ease understanding, we adopted the Business Process Model and Notation (BPMN) standard for the graphical representation of process fragments in [19]. Figure 1 shows basic BPMN constructs: A task represents a work item which has to be performed in a process by a human being or executed by a service or program invocation. Sequence flows are used for connecting and gateways for forking and joining the control flow. We extended the BPMN notation by a shape representing a region and by an icon representing constraints which can be annotated to process elements such as tasks, regions, and gateways. The usage of annotations is explained in [20].

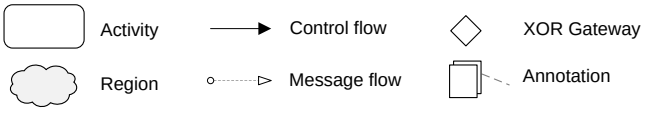


Fig. 1. Fragments in BPMN 2.0

Figure 2 presents an example BPMN fragment. The fragment models a part of a loan approval process. A form is checked for completeness. If it is not complete, the control flow has to leave the fragment. The fragment may also be started by a complete form (and thus the completeness check may be skipped). Using the complete form, the overall credit is assessed. One of the two fragment exits is taken dependent on the assessed risk.

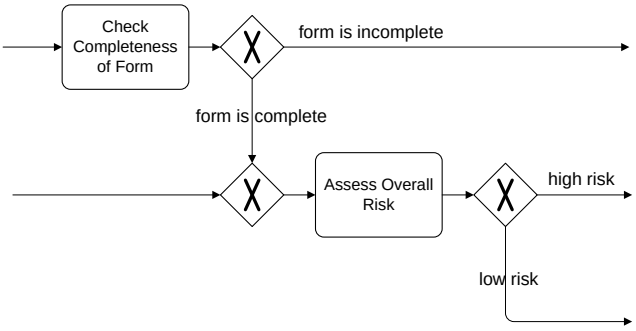


Fig. 2. Example BPMN fragment

3 Auto-completion of Process Fragments

Figure 3 presents a naive auto-completion: Each fragment entry is connected to a message start event. This approach works if mutually exclusive process entries

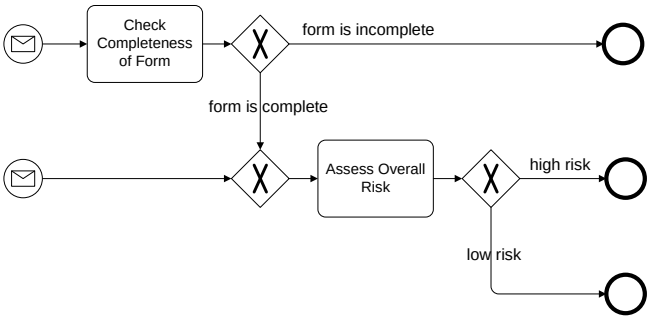


Fig. 3. Naive auto-completion

lead to a proper process execution. In case of the example, this approach is valid. In case the control flow of entries is joined via an AND join, this approach is not valid as the process execution will be stuck at the AND join as there will never be tokens on each incoming link.

Theorem 5.1 by Kiepuszewski et al. [9] shows that "every sound model with multiple end nodes can be transformed into an equivalent sound model with a unique end node" [16]. Polyvyanyy et al. [16] state that "The reverse technique can be applied to models with multiple start nodes". They, however, provide no proof for this claim. For an auto-completion for fragment entries we need such a technique. The approach by Kiepuszewski et al. requires the model to consist of AND and XOR joins only. Mendling et al. [15] showed that all OR joins in (acyclic) EPC process models can be converted to a process model containing only XOR and AND splits and joins.

We intend to apply the technique by Kiepuszewski et al. by using the following steps:

1. Merge all end nodes to a single end node.
2. Convert the BPMN model to a Petri net.
3. Reverse the edges in the process.
4. Apply the technique by Mendling et al. to eliminate all OR joins in the process model.
5. Apply the technique by Kiepuszewski et al.
6. Convert the Petri net back to a BPMN model.
7. Change the newly created end node to a message start event.
8. Reverse the edges again.
9. Undo the merging of all end nodes.

Regarding step 1, this merging requires that the end nodes are mutually exclusive. Figure 4 presents the auto-completed fragment.

It is not proven whether this approach is valid. The reversal of the edges might change properties of the Petri net [12]. For instance, the reversal of a free choice Petri net does not preserve the property of free choice.

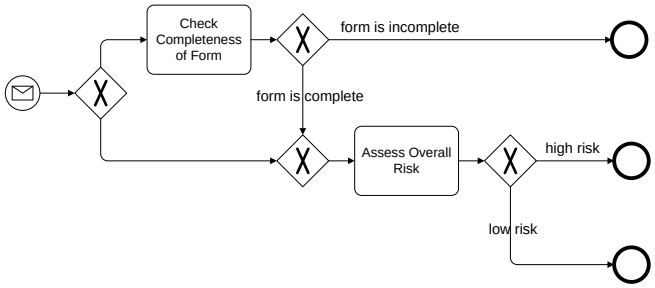


Fig. 4. Auto-completed fragment

We claim that it is *not* possible to automatically deduct concrete conditions on the sequence flows going out from the new root activity as we cannot guess the intention of the fragment designer. The fragment does not state how the fragment entries have to be reached and which conditions decide which fragment entry has to be taken. As current process verification abstracts from data, the concrete conditions are not necessary for the completion.

4 Related Work

Polyvyanyy et al. [16] showed how a subset of an unstructured BPMN models can be converted to a structured BPMN model. They require the model to contain activities, control links, and XOR and AND forks and joins only. The Refined Process Structure Tree (PST [21]) is used to (i) classify a process models which can be structured and (ii) structure the process model. A "sound and safe acyclic process model is inherently unstructured if its RPST has a rigid component for which the modular decomposition of its ordering relations contains a primitive [module]." [16]. This technique requires the process model having one entry and one exit. They rely on the technique presented by Kiepuszewski et al. [9], which we also do.

Other auto-completion approaches rely on existing business processes or process variants stored at a repository. During editing [2, 8, 11] or runtime [13], fragments matching the current process model/instance are selected. In our approach, however, we do not rely on existing process models and process variants, but do a purely syntactical extension. Syntactical autocompletion is offered by Mazanek and Minas in the case of BPMN [14]. The authors, however, rely on block-structured business process models. In this paper, we discussed arbitrary structured process models.

Generating adapters might also be a solution to auto-complete a process fragment. "An adapter is an artifact acting as mediator between services." [6]. A service here needs to be executable or at least model the behavior of the implementation. A fragment is not executable and the behavior of the fragment cannot be directly derived as the relation of the entries to each other is not directly specified by the fragment. In other words, a fragment does not state

if the entries are mutually exclusive or if all entries have to be taken for each instance of the fragment.

Gschwind et al. [7] presented a technique for process modeling by using patterns during the design of a process. One starts with a plain process and may only add activities if this inclusion conforms to a pattern. That implies that each intermediate process model is a structured process model. The fragment approach, however, enables creating and storing unstructured parts of processes.

Fragments as reusable parts have been investigated by Avrilionis et al. [1] in the case of Petri nets and by Rolland et al. [17] in the case of a process meta-model intended for artificial intelligence. Our work applies the idea of process fragments to modern business process modeling languages. Subprocesses are also a unit of reuse, but restrict the logical form to a single logical entry and a single logical exit and are not intended to be copied into a process, but used as separate process [10].

5 Conclusion and Outlook

In this paper, we reasoned on auto-completion of process fragments to either gain nearly-executable process models or nearly-complete fragments. A nearly complete fragment can be used as a part of a process model without the need of additional activities to correctly trigger the fragment entries. The concrete proposal is to apply the techniques by Kiepuszewski et al. [9] and Mendling et al. [15] to BPMN process models. This enables a proper auto-completion of fragments, where the types of the joins are AND, OR, and XOR. A proof of the presented technique is not provided in this paper and left as future work. In case arbitrary join conditions are used, the proposed approach will not work. Thus, future work is to classify arbitrary join conditions into (i) those which can be converted to XOR and AND gateways and into (ii) those which cannot.

We did not implement the proposed transformation. Thus, we did not proof whether BPMN process fragments can be really verified using a model checking tool. In case the auto-completed fragment is presented to the modeler of the fragment, the modeler might be surprised that the fragment looks differently than he has modeled it. This might lead to confusion. Therefore, we propose to show verification results on the original fragment and not on the auto-completed one.

In the discussion, we did not define what a valid fragment is and how to verify whether a fragment is valid. For instance, a fragment can be modeled, where an activity a is never reached. We would like to consider such a fragment as invalid fragment. As there is currently no criteria for valid fragments, our future work is to define such criteria.

Acknowledgments This work was supported by funds from the European Commission (contract no. 215175 for the FP7-ICT-2007-1 project COMPAS, `http://www.compas-ict.eu` and contract no. FP7-213339 for the FP7-ICT-2007-1 project ALLOW, `http://www.allow-project.eu/`).

References

1. Avrilionis, D., Cunin, P., Fernström, C.: OPSIS: a View Mechanism for Software Processes which Supports their Evolution and Reuse. In: ICSE. IEEE (1993)
2. Born, M., Brelage, C., Markovic, I., Pfeiffer, D., Weber, I.: Auto-completion for Executable Business Process Models. In: Business Process Management Workshops. LNBIP, vol. 17. Springer (2008)
3. Dehnert, J., Rittgen, P.: Relaxed Soundness of Business Processes. In: Advanced Information Systems Engineering. LNCS, vol. 2068. Springer (2001)
4. Dehnert, J., Van Der Aalst, W.M.P.: Bridging the gap between business models and workflow specifications. International Journal of Cooperative Information Systems 13(3), 289–332 (2004)
5. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Information and Software Technology 50(12), 1281–1294 (2008)
6. Gierds, C.: Finding Cost-Efficient Adapters. In: AWPN. CEUR Workshop Proceedings, vol. 380. CEUR-WS.org (2008)
7. Gschwind, T., Koehler, J., Wong, J.: Applying Patterns during Business Process Modeling. In: BPM. LNCS, vol. 5240. Springer (2008)
8. Hornung, T., Koschmider, A., Oberweis, A.: Rule-based Autocompletion Of Business Process Models. In: CAiSE Forum 2007 (2007)
9. Kiepuszewski, B., ter Hofstede, A., van der Aalst, W.: Fundamentals of control flow in workflows. Acta Informatica 39(3), 143–209 (March 2003)
10. Kopp, O., Eberle, H., Leymann, F., Unger, T.: The Subprocess Spectrum. In: BPSC. LNI, vol. P-177. Gesellschaft für Informatik e. V. (2010)
11. Koschmider, A.: Ähnlichkeitsbasierte ModellierungsunterstÃijtzung fÃijr Geschäftsprozesse. Ph.D. thesis, Fakultät für Wirtschaftswissenschaften, Institut für Angewandte Informatik und Formale Beschreibungsverfahren (AIFB) (2007)
12. Lohmann, N.: Personal discussion at ZEUS 2011
13. Lu, R., Sadiq, S., Governatori, G.: On managing business processes variants. Data & Knowledge Engineering 68(7), 642 – 664 (2009)
14. Mazanek, S., Minas, M.: Business Process Models as a Showcase for Syntax-Based Assistance in Diagram Editors. In: Model Driven Engineering Languages and Systems. LNCS, vol. 5795. Springer (2009)
15. Mendling, J., van Dongen, B.F., van der Aalst:, W.M.P.: Getting Rid of OR-Joins and Multiple Start Events in Business Process Models. Enterprise Information Systems (EIS) 2(4), 403–419 (October 2008)
16. Polyvyanyy, A., García-Bañuelos, L., Dumas, M.: Structuring Acyclic Process Models. In: BPM. LNCS, vol. 6336. Springer Berlin / Heidelberg (2010)
17. Rolland, C., Prakash, N.: Reusable Process Chunks. In: DEXA. Springer (1993)
18. Schmidt, K.: LoLA: A Low Level Analyser. In: ICATPN. pp. 465–474 (2000)
19. Schumm, D., Karastoyanova, D., Leymann, F., Strauch, S.: Fragmento: Advanced Process Fragment Library. In: ISD. Springer (2010)
20. Schumm, D., et al.: Integrating Compliance into Business Processes: Process Fragments as Reusable Compliance Controls. In: MKWI (2010)
21. Vanhatalo, J., Völzer, H., Koehler, J.: The Refined Process Structure Tree. Data Knowl. Eng. 68(9), 793–818 (2009)

BPMN for Healthcare Processes

Richard Müller[1] and Andreas Rogge-Solti[2]

[1] Institut für Informatik, Humboldt-Universität zu Berlin, Germany
richard.mueller@informatik.hu-berlin.de
[2] Hasso Plattner Institute, University of Potsdam, Germany
andreas.rogge-solti@hpi.uni-potsdam.de

Abstract. The nature of healthcare processes in a multidisciplinary hospital is inherently complex. In this paper, we identify particular problems of modeling healthcare processes with the de-facto standard process modeling language BPMN. We discuss all possibilities of BPMN adressing these problems. Where plain BPMN fails to produce nice and easily comprehensible results, we propose a new approach: Encorporating role information in process models using the color attribute of tasks complementary to the usage of lanes.

1 Introduction

Recently, business process management (BPM) has become to be considered a valuable asset in the healthcare domain [14]. BPM heavily relies on process models to identify, review, validate, represent and communicate process knowledge [17]. Among the wide variety of process modeling languages, the Business Process Model and Notation 2.0 [10] (BPMN) can be considered a de-facto standard [3]. Nonetheless, the utilization of BPMN as modeling language in specific domains may prove to be difficult. The healthcare domain serves as a good example, since the nature of healthcare processes in a multi-disciplinary hospital is inherently complex [8].

In this paper, we identify particular problems of healthcare processes concerning roles and task assignment. These problems arose during process elicitation in a medical environment. We present and discuss possibilities of BPMN addressing these problems. Further, in the case that plain BPMN fails to produce nice and easily comprehensible results, we introduce a new and tailored approach: We propose to incorporate role information in process models using the color attribute of tasks as a complementary visualization to the usage of lanes. The rest of this paper is organized as follows. In Sect. 2, we introduce specific modeling requirements of the healthcare domain. Section 3 is devoted to the presentation and evaluation of existing and new approaches for handling them. Finally, Sect. 4 concludes the paper and gives directions for future work.

2 Requirements of Healthcare Processes

In the context of the SOAMED graduate school[1] we elicit healthcare processes at Charité SPZ[2]. Latter consists of five separate departments jointly working together to

[1] http://www.soamed.de [2] http://spz.charite.de

provide long-time care for disabled or chronically ill children. All departments have to synchronize their actions and knowledge, resulting in inherently complex processes. This complex setting creates specific requirements concerning roles, which need to be supported by a modeling language capturing the processes:

Many roles participate in one process. In the described setting many specialist roles, e.g., office staff, nurses, different kinds of doctors and therapists, work together to offer the patients a highly tailored and professional treatment.

Several specialists work together on a shared task. The most common example for this requirement is a surgery, where, besides the head surgeon, different assistants, nurses and other personnel work together to treat the patient.

A task can be alternatively performed by different roles. An example for this case is that a doctor may perform a task which is usually done by a nurse, i.e., taking blood of a patient.

A task can optionally involve additional roles. An example is a doctor who may request a specialist on demand for consultation-hours.

In this paper, we focus on the first two requirements, since the two latter ones can be seen as special form of a shared task. In the next section, we discuss BPMN and its capabilities of modeling many roles and shared tasks, as well as the issues arising.

3 BPMN for Healthcare Processes

BPMN by the OMG[3] is designed to be understandable by both business professionals and IT-specialists. The explicit design for non-technical users makes it a promising candidate for healthcare process modeling, where medical staff need to understand and discuss the process models. In his book, Silver [13] emphasizes the possibility to model different events and exceptions for routing a process. This matches with healthcare processes again, which tend to have many exceptions [7]. Furthermore, BPMN is an open and free standard, which enjoys broad tool support. As of writing, the official BPMN webpage[4] lists 73 implementations.

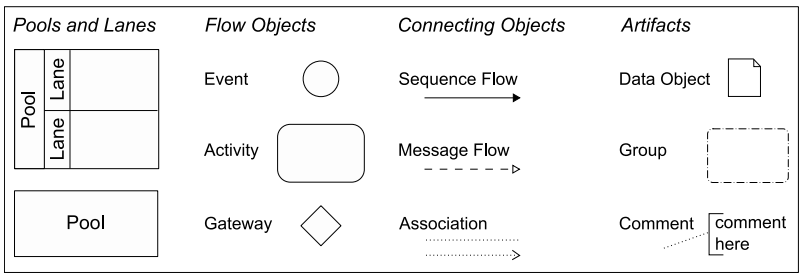


Fig. 1. Core BPMN graphical modeling elements.

[3] http://www.omg.org [4] http://www.bpmn.org

The core modeling elements of BPMN are depicted in Fig. 1. Pools and lanes are used to structure the process diagram and separate organizational units (lanes) and organizations (pools). There are three categories of flow objects: Events, activities, and gateways. Connecting objects set these flow objects in relation to each other. In a pool, a sequence flow indicates the order in which flow objects are performed. Message flows are used between pools to model communication with other organizations. Associations relate artifacts to other elements, and artifacts are either data objects, groups or comments.

Recently, version 2.0 of BPMN was released. Several important issues regarding execution semantics and interchange formats have been addressed by the new version, yet still some open issues remain. One of these open issue is the proper integration of role modeling concepts [10]. Note that the rudimentary concept of pools and lanes, which is generally used for that purpose, has no semantic meaning in BPMN. Several other deficiencies of BPMN have been addressed by the research community [1], e.g., its lack of support for resource allocation modeling, user interface modeling or data modeling.

Figure 2 shows one process fragment we elicited at Charité SPZ describing the preparation process for a difficult surgery. This example captures the requirements of many roles and shared tasks simultaneously: Seven different roles participate in the process (visualized by seven lanes), of whom four perform a shared task, i.e., the surgery indication. In the following, we discuss the capabilities of BPMN of modeling many roles and shared tasks, as well as the issues arising.

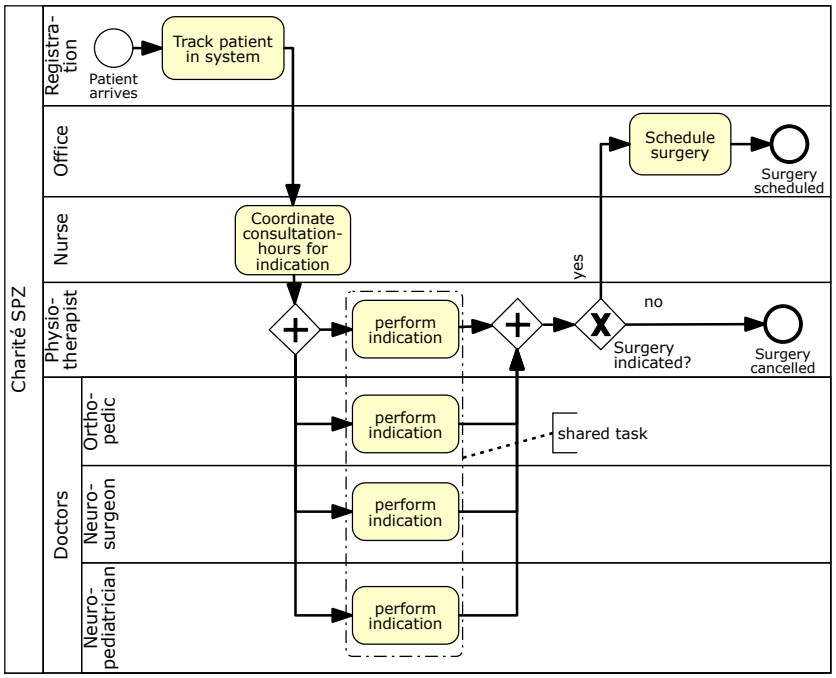


Fig. 2. Process of the preparation for a surgery at Charité SPZ.

3.1 Many Roles in BPMN

Of particular interest in this paper is the question: How does BPMN deal with role information of specific tasks? In plain BPMN, we distinguish two main approaches: The use of space as indicator for roles, or the use of annotations.

BPMN provides the notion of lanes for these roles in an organization. Thus, a diagram can be split into horizontal compartments containing the tasks assigned to a specific role. Role members can look at the diagram, search their lane and scan it for tasks they have to perform in the process. It is also fairly easy to identify handover of work, by looking at crossings of lane borders and control flows. Though these are nice features, the drawback of this visualization method is wasted space, particularly if each role has only one or proportionally few tasks in the process. Hence, lanes usually cause a disproportionate rise in diagram size while encoding only simple role information. In the worst case, a waterfall layout, the diagram size grows quadratically with the task count.

Another possibility to add role information is the use of an annotation for each task, similar to role information in EPCs [6]. The annotation may be a comment, an artifact or any kind of unique symbol. The drawback of annotations for each task is hindered readability, especially if many tasks are involved.

Both previously mentioned approaches suffer from specific drawbacks when dealing with many roles, either wasted space or hindered readability. Thus, we present a third approach for adding role information to tasks in BPMN: We use colored tasks instead of lanes in order to capture role information of a process in a more compact way. In contrast to the method proposed in [15], we do not encode levels of care, but roles in the colors of tasks. Since this approach alters the visual representation of the model, it can be considered a process view according to [12]. In the following, the patterns used to derive the different views on the process model are given in brackets.

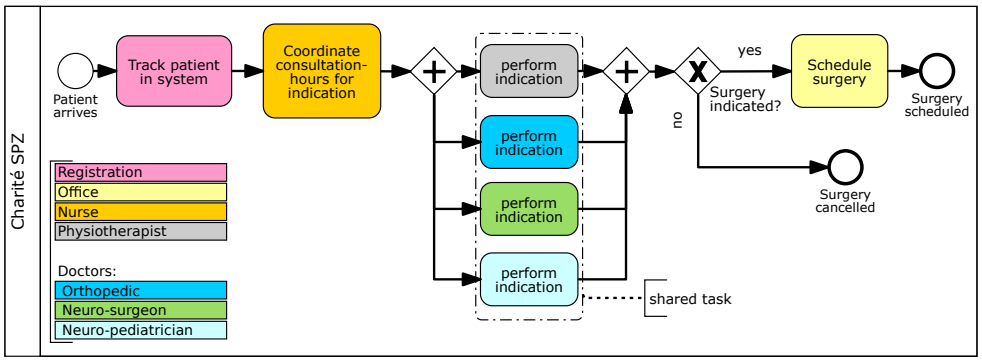


Fig. 3. Process of the preparation for a surgery at Charité SPZ with role information encoded in colors.

In BPMN, there is flexibility in the size, color, line style, and text positions of the defined graphical elements (unless specified otherwise). Among others, the specification

explicitly permits colored elements, and "the coloring *may* have specified semantics that extend the information conveyed by the element" ([10], p. 38). Hence, a first idea is to map each role in a process to a different color. After coloring each task (appearance pattern [12]) with the afore mapped color of its corresponding role, we can remove all potentially present lanes from the BPMN diagram without losing any role information (omission pattern [12]).

Figure 3 illustrates the effect of such a mapping on the size and readability of a BPMN diagram. The diagram in Fig. 2 depicts the same process as the one in Fig. 3. Yet, the former consists of seven different lanes representing different roles of the underlying surgery preparation process. Although from minor complexity, the diagram is large and unnecessarily hard to read. Figure 3 encodes the same role information in colors instead of lanes. The corresponding mapping is given as a comment in the diagram (insertion pattern [12]). Note that by using colors instead of lanes, the resulting diagram is both smaller in size and easier to read.

3.2 Shared Tasks in BPMN

BPMN does not support explicit modeling of shared tasks [18]. However, as workarounds, different methods have been proposed to capture this behavior in BPMN:

To draw the shared task on the border between two lanes. This approach is not applicable for more than two roles participating in a shared task. Besides, it is not standard-conform, as a task needs to be associated to one lane only [10].

To create a new lane for the team working on the shared task. This approach breaks the convenience of scanning a single lane for all the tasks assigned to a role. Another drawback is that the diagram size grows further with each new combination of roles sharing a task.

To have the shared task only in the responsible role's lane. Although this solution has no drawbacks on diagram size, it causes a quite important loss of information. Role related information is completely left out for supporting roles in a shared task.

To annotate role information in associated comments. Similarly to EPCs [6] it is possible to create comments for each task containing the names of the associated roles. This approach is also used in [11], where the authors propose to use comments in YAWL [2] diagrams with one primary resource/role and optional additional resources attached to a shared task. This approach scales reasonably well for multiple roles and resources. The tradeoff is, that in order to identify all tasks a role participates in, all attached comments need to be parsed sequentially.

To have a copy of the shared task in each lane and group them. This approach scales reasonably well for multiple lanes, but an additional overhead of parallelizing the control flow and adding groups around the shared tasks spanning over different lanes is introduced. It is the most promising workaround for this situation, because it is standard-compliant, no information is lost, no additional lanes are introduced and only little additional diagram space is used for the gateways splitting and joining the control flow around the shared task.

All the presented solutions above have minor or major drawbacks. Fortunately, encoding role information in colors instead of lanes enables us to model shared tasks

as well: We allow tasks to be colored with more than one color, meaning more than one role participating in that task. See Fig. 4 for an example. The process depicted contains the same information as the one in Fig. 3, but unnecessary artifacts as the group, the comment and additional control flow nodes could be eliminated. The aggregation pattern [12] is quite similar, we restrict the usage to tasks with the same label executed in parallel, though.

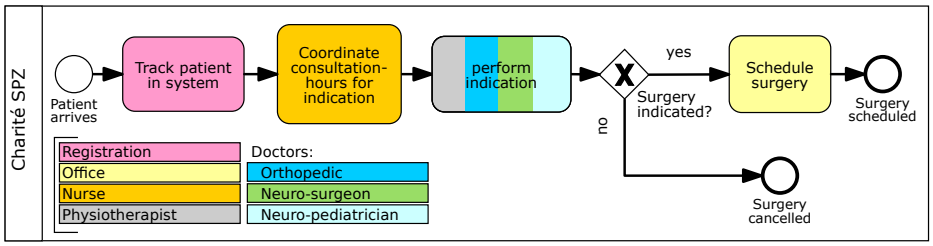


Fig. 4. Process of the preparation for a surgery at Charité SPZ making use of multiple colors for one task.

3.3 Evaluation of Colored Tasks

The colored approach retains all of the previously mentioned advantages of the currently most promising workaround for shared tasks, i.e., commented groups: It is standard-compliant to BPMN 2.0 and there is no information lost. Furthermore, no additional diagram space for lanes, control flow or groups is necessary. Beyond its aesthetic qualities, converging behavioral, neurophysiological and neuropsychological evidence suggest that color enhances the manner in which we perceive and recognize objects [16]. At the lower level, color segments the complex visual input into coherent regions. Thus, it should be rather easy to identify all tasks of a single role in the colored diagram. To sum up, the resulting colored diagram usually becomes more compact and easier to read as the uncolored one.

However, we are aware of some disadvantages of our proposal: Color-blind people may have problems differentiating the colors capturing role information, thus losing these information. Furthermore, for printed versions of the process model, a color printer is necessary. One idea for solving both problems is the usage of a pattern-based encoding instead of colors. This enables greyscale printing of diagrams and helps color-blind people to distinguish the roles. Another rather pragmatic solution would be the addition of a label to each colored part of a task. Unfortunately, this would add more elements to the diagram resulting in more cognitive load on the readers. Scalability of our method can become a problem, when too many roles exist in a diagram, or too many roles work together in a shared task. The ability to quickly distinguish many colors or patterns decreases with increasing number of colors and patterns. However, there is evidence that quality and understandability of a process model decreases with the size of the model [9].

This lead to a rule of thumb to create BPMN models which fit on a single printed page, and resort to subprocesses to specify further details when required. In the rare case that still too many roles exist after a reasonable hierarchical restructuring of a process model, we suggest the addition of labels to the view, or the usage of the original lane approach.

Finally, the use of colors in BPMN is currently non-normative. The semantics of colors may vary user to user or tool to tool, potentially leading to misinterpretations. Moreover, BPMN Diagram Interchange does not address or define the interchange of color information. But since the colored representation can be generated from the interchangeable models and is not limited to specific tools, these issues can be tackled by adding the capability to display the colored view to tools.

4 Conclusion and Future Work

We identified several role-related process modeling requirements of the healthcare domain. We argued that BPMN is suitable for this domain, even though there exist some deficits fulfilling these requirements. We discussed existing workarounds and presented the idea to incorporate role information in colors of tasks in BPMN models. Using this approach we gained more compact, yet still understandable process models that can capture the identified role requirements. Finally, we debated both advantages and disadvantages of our proposal.

As the approach proposed in this paper is still at idea stage, we have to figure out how to cope with the disadvantages mentioned in Sect. 3.3. However, some first ideas are presented there as well. Besides the discussed workarounds in BPMN, there exist other modeling languages which can cope with the specific requirements of many roles and shared tasks, e.g., Colored Petri Nets [5]. Future research includes comparing those with our solution and answering the question which modeling language fits best the healthcare domain.

There exist mature tools which support the modeling and analysis of BPMN models, e.g., Oryx[5]. For future works, we would like to implement colored BPMN in one of these tools. We imagine automated support for switching between the alternate visual representations of process models of either pools and lanes, task annotations, or color-encoded roles. In this paper we restricted the possible solutions for handling complex role requirements to stay in the BPMN standard. If we lift this restriction, the most appropriate solution for these problems would be to use a similar notation as in the choreography models [10], i.e., to add (a) colored partition(s) to a task labeled with the role(s). By supporting the notion of shared tasks, we introduced a new concept to BPMN: An 1:n assignment of a task. Future work includes the formalization of this concept. Finally, there is ongoing research on the topic of layout aesthetics for business process models [4]. Future work includes how the colored BPMN performs in terms of the layout catalogue in comparison to plain BPMN. This could require the definition of a concrete BPMN layout metric beforehand.

[5] `http://bpt.hpi.uni-potsdam.de/oryx`

References

1. Aagesen, G., Krogstie, J.: Analysis and design of business processes using BPMN. Handbook on Business Process Management 1 pp. 213–235 (2010)
2. van der Aalst, W.M., ter Hofstede, A.: YAWL: yet another workflow language. Information Systems 30(4), 245–275 (Jun 2005)
3. Allweyer, T.: BPMN 2.0–Business Process Model and Notation: Einführung in den Standard für die Geschäftsprozessmodellierung. BoD, Norderstedt (2009)
4. Effinger, P., Jogsch, N., Seiz, S.: On a Study of Layout Aesthetics for Business Process Models Using BPMN. In: Business Process Modeling Notation: Second International Workshop, Potsdam, Germany. Springer Verlag (2010)
5. Jensen, K., Kristensen, L.M.: Coloured Petri Nets - Modelling and Validation of Concurrent Systems. Springer (2009)
6. Keller, G., Nüttgens, M., Scheer, A.: Semantische Prozeßmodellierung auf der Grundlage "Ereignisgesteuerter Prozeßketten (EPK)". Inst. für Wirtschaftsinformatik (1992)
7. Lenz, R., Reichert, M.: IT support for healthcare processes – premises, challenges, perspectives. Data and Knowledge Engineering 61(1), 39–58 (2007)
8. Mans, R., Schonenberg, M., Song, M., Aalst, W., Bakker, P.: Application of Process Mining in Healthcare–A Case Study in a Dutch Hospital. Biomedical Engineering Systems and Technologies pp. 425–438 (2009)
9. Mendling, J., Strembeck, M.: Influence Factors of Understanding Business Process Models. Lecture Notes in Business Information Processing, vol. 7, pp. 142–153. Springer Berlin Heidelberg (2008)
10. OMG: Business Process Model and Notation (BPMN) – Version 2.0 (January 2011)
11. Ouyang, C., Wynn, M., Fidge, C., ter Hofstede, A., Kuhr, J.: Modelling complex resource requirements in Business Process Management Systems. ACIS 2010 Proceedings (2010)
12. Schumm, D., Leymann, F., Streule, A.: Process Viewing Patterns. In: Proceedings of the 14th IEEE International EDOC Conference, Vitória, Brazil. pp. 89–98. IEEE Computer Society (2010)
13. Silver, B.: BPMN Method and Style. Cody-Cassidy Press (2009)
14. Stefanelli, M.: Knowledge and Process Management in Health Care Organizations. Methods of Information in Medicine 43(5) (2004)
15. Svagård, I., Farshchian, B.: Using business process modelling to model integrated care processes: Experiences from a european project. Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living pp. 922–925 (2009)
16. Tanaka, J., Weiskopf, D., Williams, P.: The role of color in high-level vision. Trends in cognitive sciences 5(5), 211–215 (2001)
17. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer-Verlag New York Inc (2007)
18. White, S.A., Miers, D.: BPMN Modeling and Reference Guide: Understanding and Using BPMN. Future Strategies Inc. (2008)

Effiziente Abschätzung von Datenflussfehlern in strukturierten Geschäftsprozessen

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

Zusammenfassung. Neben dem Kontrollfluss von Geschäftsprozessen kann auch der Datenfluss Ursache einer fehlerhaften Prozessausführung sein, daher ist die Überprüfung eines Prozessmodells auf Datenflussfehler ebenfalls wesentlich. Wir schlagen in diesem Beitrag eine Methode zur Abschätzung von Datenflussfehlern für strukturierte Geschäftsprozesse vor. Auf Grundlage der durch eine Datenflussanalyse abgeleiteten Datenflussinformation geben wir Fehlermengen für mögliche und sichere Datenflussfehler eines Geschäftsprozesses an. Der Vorteil dieses Ansatzes besteht zum einen in der Effizienz der Analyse, andererseits aber auch in der Identifikation und Lokalisation von Fehlern in einem Schritt. Als Nachteil ergibt sich hingegen der Verlust absoluter Präzision.

1 Einführung

Neben der Verifikation von Geschäftsprozessen hinsichtlich Kontrollflussfehlern, wie Verklemmungen oder fehlender Synchronisation, ist die Analyse der Verwendung von Prozessdaten zur Gewährleistung einer fehlerfreien Prozessausführung von Interesse [5]. Typische Fehler in diesem Zusammenhang sind beispielsweise der lesende Zugriff auf noch uninitialisierte oder bereits gelöschte Daten, das parallele Schreiben und Lesen von Daten oder das Überschreiben ungelesener Daten. Enthält ein Prozessmodell auch Informationen zur Verwendung der Prozessdaten, in Form der durch Prozessaktivitäten geschriebenen, gelesenen und gelöschten Daten, kann eine Überprüfung auf derartige Datenflussfehler erfolgen.

Ein insbesonders für die Analyse des Datenflusses geeignetes Verfahren ist die *statische Datenflussanalyse* [2, 3]. Im Gegensatz zu Verifikationstechniken die auf einer vollständigen Modellprüfung beruhen erlaubt die Datenflussanalyse eine effiziente Ableitung von konservativer Datenflussinformation, verzichtet aber im Gegenzug auf exakte Ergebnisse. Auf diese Weise kann der exponentielle Verifikationsaufwand vermieden werden, der sich sonst bei einer präzisen Analyse ergibt. Der hohe Aufwand ist dabei auf die zur Überprüfung des Datenflusses notwendige Identifikation parallel ausführbarer Prozessaktivitäten zurückzuführen, die schon für strukturierte Prozesse und unter Ausschluß von Schleifen exponentielle Kosten verursachen kann [1]. Zusätzlich wird auch keine endliche Abstraktion für die in einem Geschäftsprozess auftretenden Daten benötigt, da lediglich der Datenfluss zwischen den Prozessaktivitäten berücksichtigt werden muss.

Fehlende Daten	Zugriff auf ein uninitialisiertes oder gelöschtes Datum
Redundante Daten	Schreiben eines Datums durch eine Prozessaktivität auf das im weiteren Verlauf nicht lesend zugegriffen wird
Überschriebene Daten	Überschreiben eines Datums durch eine Prozessaktivität auf das noch nicht lesend zugegriffen wurde
Inkonsistente Daten	Zugriff einer Prozessaktivität auf ein Datum und dazu paralleles Schreiben oder Löschen desselben Datums
Nicht gelöschte Daten	Fehlendes Löschen für ein geschriebenes Datum
Doppelt gelöschte Daten	Zweimaliges Löschen ein und desselben Datums
Zu spät gelöschte Daten	Letzter lesender Zugriff einer Prozessaktivität auf ein Datum ohne sich unmittelbar anschließendes Löschen

Tabelle 1. Datenflussfehler (Anti-Muster) nach [5]

Im vorliegenden Beitrag zeigen wir, wie das Verfahren der Datenflussanalyse zur Überprüfung eines strukturierten Geschäftsprozesses auf Datenflussfehler genutzt werden kann. In Abschnitt 2 wird dazu zunächst ein Überblick zu Datenflussfehlern und dem hier verwendeten Prozessmodell gegeben. Danach erfolgt in Abschnitt 3 eine Beschreibung der Bestimmung von fehlenden, gelöschten sowie definierenden Daten mit Hilfe einer *statischen Datenflussanalyse*. Auf Grundlage der so für einen Prozess effizient ableitbaren Datenflussinformationen können wir dann in Abschnitt 4 *Fehlermengen* einführen, die Abschätzungen zu den im Prozess enthaltenen Datenflussfehlern in Form *sicherer* und *möglicher* Fehler bilden. Schließlich wird der Beitrag in Abschnitt 5 kurz zusammengefasst.

2 Grundbegriffe

2.1 Datenflussfehler

In der Arbeit [5] wird eine Sammlung der in Geschäftsprozessen auftretenden Datenflussfehler beschrieben. Dabei werden mehrere Anti-Muster vorgestellt, die Schwachstellen hinsichtlich einer fehlerfreien Verwendung von Prozessdaten darstellen. Wir beziehen uns im Folgenden auf diese, in Tabelle 1 angegebenen, Anti-Muster, wenn wir von Datenflussfehlern sprechen. Im Gegensatz zu [5] wird hier für die Anti-Muster *Redundante Daten* und *Überschriebene Daten* keine Unterscheidung zwischen Fehlern, die immer auftreten, und solchen, die nur in bestimmten Ausführungsszenarien auftreten, vorgenommen. Stattdessen werden für alle Anti-Muster die Begriffe *sicherer* und *möglicher Fehler* definiert:

Definition 1 (Sicherer/Möglicher Datenflussfehler). *Ein sicherer Fehler tritt unabhängig vom zur Ausführungszeit tatsächlich gewählten Kontrollfluss eines Prozesses immer auf. Ein möglicher Fehler ist ein Kandidat für einen Fehler der in mindestens einer Prozessausführung auftreten kann.*

2.2 Prozessrepräsentation

Zur Durchführung unserer Methode benötigen wir ein Prozessmodell, dass die Wiedergabe des Datenflusses innerhalb eines Prozesses gestattet. Zu diesem Zweck werden *erweiterte Workflow-Graphen* genutzt, die gewöhnliche Workflow-Graphen [4] mit Datenflussannotationen versehen. Auf der rechten Seite von

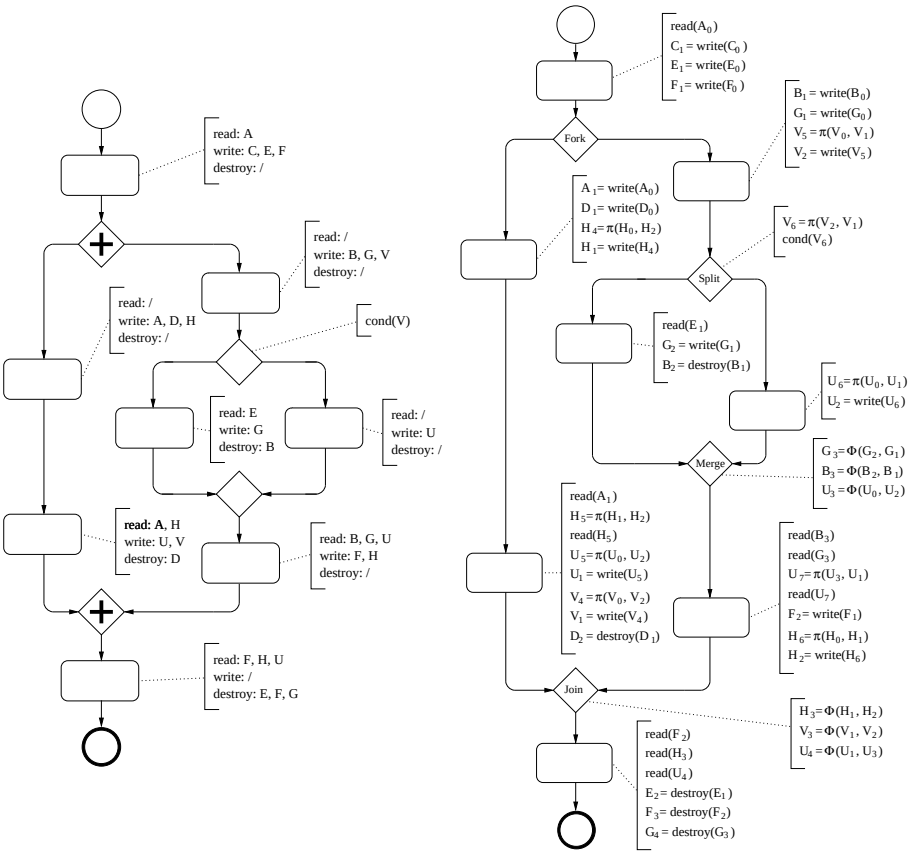


Abb. 1. Beispielprozess in BPMN-Notation (l.) und als Workflow-Graph (r.)

Abbildung 1 ist der erweiterte Workflow-Graph für den aus [5] übernommenen Beispielprozess dargestellt. Zum Vergleich bildet die linke Seite den Prozess auch in BPMN-Notation ab. In der BPMN-Darstellung sind die Prozessknoten mit Annotationen versehen, die in den Knoten gelesene, geschriebene oder gelöschte Daten in Form von Variablen (`A,B,...`) bezeichnen. Ferner wurde die bedingte Aufspaltung des Kontrollflusses mit der Verzweigungsbedingung versehen.

Im erweiterten Workflow-Graphen sind die Annotationen übernommen, nur dass diese nun im Format der *Concurrent Static Single Assignment Form (CSSA-Form)* [2] vorliegen. Zu diesem Zweck wurden die auf Daten operierenden Prozessaktivitäten auf vier Instruktionstypen abgebildet:

- $read(V_i)$ liest das Datum in Variable V_i,
- $cond(V_i)$ bestimmt den Wert einer Verzweigungsbedingung über Variable V_i,
- $V_i = write(V_j)$ überschreibt die alte Definition des Datums in Variable V_j mit einem neuen Datum und legt dieses in der Variablen V_i ab,
- $V_i = destroy(V_j)$ löscht das Datum in Variable V_j und setzt dadurch den Wert der Variablen V_i auf undefiniert.

Charakteristische Eigenschaft der *CSSA-Form* ist, dass jede Variable statisch einmal definiert ist, so dass für jede Variablendefinition durch die Instruktionen *write* oder *destroy* ein eigener Name eingeführt, und Variablenzugriffe entsprechend angepasst wurden (beispielsweise $G_1, \ldots, G_4$ für Variable `G`). Treffen mehrere Definitionen einer Variablen auf verschiedenen Pfaden des Kontrollflusses in einem Knoten zusammen, wurden spezielle Instruktionen mit wie folgt definierten *Φ-Funktionen* eingefügt, um die Definitionen zusammenzufassen:

Definition 2 (Φ-Funktion). *Eine Φ-Funktion für Variable V hat die Form $\Phi(V_1, \ldots, V_n)$, wobei die Operanden V_i den im Knoten der Funktion zusammenfließenden Definitionen von V entsprechen. Der Wert der Funktion ist der Operand V_i, der die zur Prozesslaufzeit tatsächlich, beziehungsweise als letztes, ausgeführte Instruktion mit einer Definition der Variablen V repräsentiert.*

Neben den Instruktionen mit *Φ-Funktionen* enthält die CSSA-Form weitere spezielle Instruktionen mit *π-Funktionen*, um Schreib-/Lese-Konflikte zwischen parallelen Prozessaktivitäten modellieren zu können:

Definition 3 (π-Funktion). *Eine π-Funktion für Variable V hat die Form $\pi(V_1, \ldots, V_n)$, wobei die Operanden V_i den im Knoten der Funktion konkurrierenden Definitionen von V entsprechen. Der Wert der Funktion ist der Operand V_i, der die zur Prozesslaufzeit letzte Definition von V repräsentiert.*

3 Datenflussinformation

Auf Grundlage der Repräsentation eines strukturierten Geschäftsprozesses durch erweiterte Workflow-Graphen können dann Informationen zum Datenfluss abgeleitet werden. Für die Bestimmung der sicheren und möglichen Fehler zu den in Tabelle 1 aufgeführten Fehlerarten werden Datenflussinformationen über die *fehlenden*, *gelöschten* und *definierenden Daten* des untersuchten Prozesses benötigt.

Als *fehlende Daten* werden Variablen bezeichnet, die uninitialisiert sind oder gelöscht wurden. Dabei kann unterschieden werden, ob eine Variable für mindestens ein Ausführungsszenario ein fehlendes Datum beschreibt, oder für alle möglichen Prozessausführungen. In unserem Beispiel aus Abbildung 1 entspricht die Variable A_0 immer einem fehlenden Datum, die Variable B_3 hingegen nur dann, falls die Instruktion $B_2 = destroy(B_1)$ ausgeführt wurde. Zur Darstellung der Datenflussinformation werden Wahrheitswerte genutzt, die angeben ob eine Variable ein fehlendes Datum beschreibt. Bedingt durch die Eigenschaft der CSSA-Form dass jede Variable statisch nur einmal definiert ist, können diese Werte den die Variablen definierenden Instruktionen zugewiesen werden:

Definition 4 (Fehlende Daten). *Für Instruktion s enthält $MISS^{MUST}(s)$ einen Wahrheitswert, der anzeigt ob die durch s definierte Variable auf allen Kontrollflusspfaden uninitialisiert/gelöscht ist und $MISS^{MAY}(s)$ einen Wert, ob die Variable auf mindestens einem Pfad uninitialisiert/gelöscht ist. Ist die Instruktion s nicht vorhanden, gilt $MISS^{MUST}(s) = MISS^{MAY}(s) = true$.*

Analog ergibt sich die Datenflussinformation zu *gelöschten Daten*, die angibt ob eine Variable im Prozess bereits gelöscht wurde:

Definition 5 (Gelöschte Daten). *Für Instruktion s enthält $DEL^{MUST}(s)$ einen Wahrheitswert, der anzeigt ob die durch s definierte Variable auf allen Kontrollflusspfaden gelöscht wurde und $DEL^{MAY}(s)$ einen Wahrheitswert, der anzeigt ob diese Variable auf mindestens einem Pfad gelöscht wurde. Ist die Instruktion s nicht vorhanden, gilt $DEL^{MUST}(s) = DEL^{MAY}(s) = false$.*

Die Datenflussinformation zu *definierenden Daten* entspricht hingegen der Menge von Daten, in Form von durch *write*-Instruktionen definierten Variablen, die den Wert einer Variablen in mindestens einer Prozessausführung festlegen (beispielsweise $\{H_1, H_2\}$ für Variable H_3 in Abbildung 1), oder in allen:

Definition 6 (Definierende Daten). *Für Instruktion s enthält die Menge $DATA^{MUST}(s)$ alle Daten, welche den Wert der durch s definierten Variablen auf allen Kontrollflusspfaden festlegen und die Menge $DATA^{MAY}(s)$ alle Daten, welche den Wert dieser Variablen auf mindestens einem Pfad festlegen. Ist die Instruktion s nicht vorhanden, gilt $DATA^{MUST}(s) = DATA^{MAY}(s) = \emptyset$.*

Um die so definierten Datenflussinformationen für einen Geschäftsprozess exakt bestimmen zu können, ist eine Analyse der parallel ausführbaren Prozessaktivitäten notwendig. Grundsätzlich ist eine solche Analyse Co-NP-schwer [1]. Daher verzichten wir auf die exakte Bestimmung und ermitteln stattdessen konservative Abschätzungen. Zu diesem Zweck wird das Verfahren der *statischen Datenflussanalyse* angewendet. Dieses erlaubt für die Charakterisierung eines Datenflussproblems durch ein System rekursiver Gleichungen eine Fixpunktlösung zu berechnen, die eine Abschätzung zur gesuchten Information bildet. Das Gleichungssystem zu den *definierenden Daten* ergibt sich beispielsweise wie folgt:

$$
\begin{array}{ll}
DATA^{MUST}(s) = \bigcap_{i\in\{1,\ldots,n\}} DATA^{MUST}(def(V_i)) & \text{für } s : V = \Phi(V_1, \ldots, V_n) \\
DATA^{MUST}(s) = \bigcap_{i\in\{1,\ldots,n\}} DATA^{MUST}(def(V_i)) & \text{für } s : V = \pi(V_1, \ldots, V_n) \\
DATA^{MAY}(s) = \bigcup_{i\in\{1,\ldots,n\}} DATA^{MAY}(def(V_i)) & \text{für } s : V = \Phi(V_1, \ldots, V_n) \\
DATA^{MAY}(s) = \bigcup_{i\in\{1,\ldots,n\}} DATA^{MAY}(def(V_i)) & \text{für } s : V = \pi(V_1, \ldots, V_n) \\
DATA^{MUST}(s) = DATA^{MAY}(s) = \{V_i\} & \text{für } s : V_i = write(V_j) \\
DATA^{MUST}(s) = DATA^{MAY}(s) = \emptyset & \text{sonst}
\end{array}
$$

Wie zu erkennen, werden darin jeder Instruktion s eines Prozesses Gleichungen $DATA^{MUST}(s)$ und $DATA^{MAY}(s)$ zugeordnet. Die Datenflussinformation für eine *write*-Instruktion s bildet gerade die Menge, die als einziges Element die durch s definierte Variable enthält. Für eine Instruktion mit Φ- oder π-Funktion ergeben sich die Mengen $DATA^{MUST}$ und $DATA^{MAY}$ als Schnitt beziehungsweise Vereinigung der Datenflussinformation zu den die Operanden definierenden Instruktionen (Instruktion $def(V_i)$ für Operand V_i). Da das Gleichungssystem über endlichen Mengen und monotonen Funktionen definiert ist, ist dessen Konvergenz sichergestellt. Für die Fixpunktbestimmung kann dann ein Algorithmus zur Datenflussanalyse auf CSSA-Form genutzt werden [2, 3], der diesen in höchstens quadratischer Zeit bezüglich der Anzahl von Prozessinstruktionen berechnet. Aufgrund des beschränkten Platzes wird hier auf die Angabe der Fixpunktgleichungen zu *fehlenden* und *gelöschten Daten* verzichtet.

Fehlerart	**Fehlermenge**
Fehlende Daten (sichere Fehler)	$\{\ s \mid (\ s:\ V_i = destroy(V_j)\ \vee\ s:\ read(V_j)$ $\vee\ s:\ cond(V_j)\ )\ \wedge\ MISS^{MUST}(def(V_j)) = true\ \}$
Fehlende Daten (mögliche Fehler)	$\{\ s \mid (\ s:\ V_i = destroy(V_j)\ \vee\ s:\ read(V_j)$ $\vee\ s:\ cond(V_j)\ )\ \wedge\ MISS^{MAY}(def(V_j)) = true\ \}$
Redundante oder überschriebene Daten (sichere Fehler)	$\{\ s \mid s:\ V_i = write(V_j)$ $\wedge\ V_i \notin (\ \bigcup_{s':\ read(V_k)} DATA^{MAY}(def(V_k))$ $\cup\ \bigcup_{s':\ cond(V_k)} DATA^{MAY}(def(V_k))\ )\ \}$
Redundante oder überschriebene Daten (mögliche Fehler)	$\{\ s \mid s:\ V_i = write(V_j)$ $\wedge\ V_i \notin (\ \bigcup_{\substack{s':\ read(V_k) \\ s'\ postdominiert\ s}} DATA^{MUST}(def(V_k))$ $\cup\ \bigcup_{\substack{s':\ cond(V_k) \\ s'\ postdominiert\ s}} DATA^{MUST}(def(V_k))\ )\ \}$
Inkonsistente Daten (mögliche Fehler)	$\{\ s \mid (\ s:\ V_i = destroy(V_j)\ \vee\ s:\ read(V_j)$ $\vee\ s:\ cond(V_j)\ \vee\ s:\ V_i = write(V_j)\ )$ $\wedge\ V_j\ ist\ definiert\ durch\ \pi{-}Funktion\ mit\ mehr$ $als\ einem\ Operanden\ \}$
Nicht gelöschte Daten (sichere Fehler)	$\{\ s \mid s:\ V_i = write(V_j)$ $\wedge\ V_i \notin (\ \bigcup_{s':\ V_l = destroy(V_k)} DATA^{MAY}(def(V_k))$ $\cup\ \bigcup_{s':\ V_l = write(V_k)} DATA^{MAY}(def(V_k))\ )\ \}$
Nicht gelöschte Daten (mögliche Fehler)	$\{\ s \mid s:\ V_i = write(V_j)$ $\wedge\ V_i \notin (\ \bigcup_{\substack{s':\ V_l = destroy(V_k) \\ s'\ postdominiert\ s}} DATA^{MAY}(def(V_k))$ $\cup\ \bigcup_{\substack{s':\ V_l = write(V_k) \\ s'\ postdominiert\ s}} DATA^{MAY}(def(V_k))\ )\ \}$
Doppelt gelöschte Daten (sichere Fehler)	$\{\ s \mid s:\ V_i = destroy(V_j)$ $\wedge\ DEL^{MUST}(def(V_j)) = true\ \}$
Doppelt gelöschte Daten (mögliche Fehler)	$\{\ s \mid s:\ V_i = destroy(V_j)$ $\wedge\ DEL^{MAY}(def(V_j)) = true\ \}$
Zu spät gelöschte Daten (mögliche Fehler)	$\{\ s \mid (\ s:\ read(V_i)\ \vee\ s:\ cond(V_i)\ )$ $\wedge\ \nexists\ s':\ V_j = destroy(V_i)\ in\ Basisblock\ von\ s$ $\wedge\ V_i \notin (\ \bigcup_{\substack{s'':\ read(V_k)\ \wedge\ s'' \neq s \\ s''\ postdominiert\ s}} DATA^{MUST}(def(V_k))$ $\cup\ \bigcup_{\substack{s'':\ cond(V_k)\ \wedge\ s'' \neq s \\ s''\ postdominiert\ s}} DATA^{MUST}(def(V_k))\ )\ \}$

Tabelle 2. Fehlermengen (s, s', s'' bezeichnen Instruktionen des Prozesses)

4 Abschätzung sicherer und möglicher Fehler

Nachdem Abschätzungen für die fehlenden, gelöschten und definierenden Daten für einen Prozess bestimmt wurden, können dessen *sichere* und *mögliche Datenflussfehler* abgeleitet werden. Zu diesem Zweck definieren wir für jeden der in Tabelle 1 aufgeführten Fehler zugehörige *Fehlermengen* (vergleiche Definition 1): Die *Menge sicherer Fehler* enthält Instruktionen, die den Fehler sicher und in allen Prozessausführungen aufweisen, ist also eine Teilmenge der tatsächlichen Fehler. Die *Menge möglicher Fehler* enthält Instruktionen, die den Fehler in einer Ausführung aufweisen können, ist also eine Obermenge der tatsächlichen Fehler. In Tabelle 2 sind die *Fehlermengen* dargestellt. Wie zu erkennen, konnten für alle Fehler, bis auf *Inkonsistente Daten* und *Zu spät gelöschte Daten*, sowohl die Menge der sicheren, als auch die Menge der möglichen Fehler angegeben werden.

Fehlerart	**sichere Fehler**	**mögliche Fehler**
Fehlende Daten	A_0	A_0, B_3, U_7, U_4
Redundante oder überschriebene Daten	C_1, F_1, D_1 davon redundant: C_1, D_1	$C_1, F_1, D_1, H_1, H_2, U_1, U_2, V_1, V_2, G_1,$ G_2, E_1, B_1 davon redundant: C_1, D_1, G_2, E_1, B_1
Inkonsistente Daten	/	$H_4, H_5, H_6, U_5, U_6, U_7, V_4, V_5, V_6$
Nicht gelöschte Daten	C_1, A_1	$C_1, A_1, H_1, H_2, U_1, U_2, V_1, V_2, B_1$
Doppelt gelöschte Daten	$\emptyset$	$\emptyset$
Zu spät gelöschte Daten	/	$H_3, H_5, A_1, B_3, E_1, G_3, U_4, U_7, V_6, A_0$

Tabelle 3. Abgeleitete Fehler für den Beispielprozess aus Abbildung 1

Die Menge *Fehlende Daten (sichere Fehler)* enthält Instruktionen s der Instruktionstypen $V_i = \mathit{destroy}(V_j)$, $\mathit{read}(V_j)$ und $\mathit{cond}(V_j)$, die für alle Kontrollflusspfade auf ein fehlendes Datum V_j zugreifen. Dazu wird überprüft, ob $MISS^{MUST}$ für die V_j definierende Instruktion $def(V_j)$ dem Wahrheitswert *true* entspricht. Die Menge *Fehlende Daten (mögliche Fehler)* umfasst Instruktionen, die für einen Pfad auf ein fehlendes Datum zugreifen können, und wurde analog über die Wahrheitswerte in $MISS^{MAY}$ definiert. Auf gleiche Weise ergeben sich die Fehlermengen *Doppelt gelöschte Daten*, nur das diese *destroy*-Instruktionen enthalten und als Datenflussinformation DEL^{MUST}, DEL^{MAY} genutzt wird.

Die Fehlermenge *Redundante oder überschriebene Daten (sichere Fehler)* umfasst *write*-Instruktionen, die Daten schreiben auf die im Prozess nie lesend, also durch Instruktionen *read* oder *cond* zugegriffen wird. Zu diesem Zweck wird die Menge aller im Prozess auf mindestens einem Kontrollflusspfad gelesenen Daten bestimmt, als Vereinigung der Mengen $DATA^{MAY}(def(V_k))$ für alle durch Instruktionen $s' : \mathit{read}(V_k)$ und $s' : \mathit{cond}(V_k)$ gelesenen Variablen V_k. Ist eine durch Instruktion $s : V_i = \mathit{write}(V_j)$ definierte Variable V_i kein Element dieser Menge, wird nie lesend auf V_i zugegriffen und die Instruktion erfüllt den Fehler. Die Menge *Redundante oder überschriebene Daten (mögliche Fehler)* ergibt sich analog, nur dass nun überprüft wird, ob eine durch Instruktion $s : V_i = \mathit{write}(V_j)$ definierte Variable V_i nicht auf allen Kontrollflusspfaden gelesen wird, unter Ausnutzung der Datenflussinformation $DATA^{MUST}$ und der Postdominanz-Relation.

In [5] wird zusätzlich eine Unterscheidung zwischen *redundanten* und *überschriebenen Daten* vorgenommen. Um auch eine solche Unterscheidung durchzuführen, kann die Menge $\bigcup_{s:\ V_l = \mathit{write}(V_k)} DATA^{MAY}(def(V_k))$ aller auf mindestens einem Kontrollflusspfad überschriebenen Variablen bestimmt werden. Ist die durch eine *write*-Instruktion definierte Variable kein Element dieser Menge, wird sie in keinem Fall überschrieben und muss daher redundant sein.

Die Menge *Inkonsistente Daten (mögliche Fehler)* umfasst Instruktionen, die auf ein Datum zugreifen, auf das auch eine parallel ausgeführte Instruktion schreibend oder löschend zugreift. Da solche Schreib-/Lese-Konflikte im erweiterten Workflow-Graphen bereits mittels π-Funktionen gekennzeichnet sind, ergibt sich die Fehlermenge als Menge aller Instruktionen, die auf eine durch π-Funktion mit mehr als einem Operanden definierte Variable zugreifen. In ähnlicher Weise zu den hier näher erläuterten Fehlermengen ergeben sich dann auch die Mengen zu den Datenflussfehlern *Nicht gelöschte Daten* und *Zu spät gelöschte Daten*.

Eine auf diese Weise durchgeführte Abschätzung für die Datenflussfehler im Beispielprozess aus Abbildung 1 ist in Tabelle 3 dargestellt. Aus Gründen der Übersichtlichkeit wurden nicht Instruktionen, sondern die zugehörigen Variablen angegeben. So enthalten die Mengen zum Fehler *Fehlende Daten* Variablen, die fehlende Daten beschreiben und auf die durch eine Instruktion zugegriffen wird, und die Mengen zum Fehler *Nicht gelöschte Daten* Variablen, die durch eine Instruktion geschrieben aber später nicht gelöscht werden. Die Fehlermengen beschreiben offenbar recht gut die im Prozess enthaltenen Fehler. Die Mengen sicherer Fehler repräsentieren so nur tatsächlich immer im Prozess auftretende Fehler. Die Mengen möglicher Fehler repräsentieren nahezu nur Fehler, die für mindestens eine Prozessausführung auch tatsächlich auftreten. Lediglich bei U_4 in der Fehlermenge *Fehlende Daten* und G_2 in der Fehlermenge *Redundante oder überschriebene Daten* handelt es sich um keine tatsächlichen Fehler.

5 Zusammenfassung

In der vorliegenden Arbeit haben wir eine Methode vorgestellt, die es für strukturierte Geschäftsprozesse erlaubt, Abschätzungen für Datenflussfehler effizient zu bestimmen. Zu diesem Zweck werden erweiterte Workflow-Graphen als Prozessmodell genutzt, die die Durchführung einer Datenflussanalyse begünstigen. Basierend auf den durch Datenflussanalyse ableitbaren Informationen zu fehlenden, gelöschten und definierenden Daten konnten dann Fehlermengen für sichere und mögliche Datenflussfehler angegeben werden. Die Methode bietet neben ihrer Effizienz den weiteren Vorteil, dass alle in einem Prozess enthaltenen Datenflussfehler in einem Schritt abgeschätzt werden können. Im Gegensatz dazu liefert ein Ansatz auf Grundlage einer Modellprüfung immer nur einen Fehler als Gegenbeispiel zur untersuchten Eigenschaft. Zukünftige Arbeiten sollen die praktische Relevanz dieser Vorteile anhand von Fallstudien weiter untersuchen.

Literatur

[1] Callahan, David ; Subhlok, Jaspal: Static Analysis of Low-level Synchronization. In: *ACM SIGPLAN Notices* 24 (1989), Nr. 1, S. 100–111

[2] Lee, Jaejin ; Midkiff, Samuel P. ; Padua, David A.: Concurrent Static Single Assignment Form and Constant Propagation for Explicitly Parallel Programs. In: *Languages and Compilers for Parallel Computing, 10th International Workshop, LCPC'97, Minneapolis, Minnesota, USA, August 7-9, 1997, Proceedings*, Springer, 1998 (LNCS 1366), S. 114–130

[3] Moser, Simon ; Martens, Axel ; Görlach, Katharina ; Amme, Wolfram ; Godlinski, Artur: Advanced Verification of Distributed WS-BPEL Business Processes Incorporating CSSA-based Data Flow Analysis. In: *2007 IEEE International Conference on Services Computing (SCC 2007), 9-13 July 2007, Salt Lake City, Utah, USA*, IEEE Computer Society Press, 2007, S. 98–105

[4] Sadiq, Wasim ; Orlowska, Maria E.: Analyzing Process Models Using Graph Reduction Techniques. In: *Information Systems* 25 (2000), Nr. 2, S. 117–134

[5] Trčka, Nikola ; van der Aalst, Wil M. ; Sidorova, Natalia: Data-Flow Anti-patterns: Discovering Data-Flow Errors in Workflows. In: *Advanced Information Systems Engineering, 21st International Conference, CAiSE 2009, Amsterdam, The Netherlands, June 8-12, 2009, Proceedings*, Springer, 2009 (LNCS 5565), S. 425–439

Service-Komposition von Reiseprozessen mittels Graphtransformation

Jörg Daubert[1], Erwin Aitenbichler[2], Stephan Borgert[2]

[1]Fachbereich Informatik, Technische Universität Darmstadt
[2]Telecooperation Group, Fachbereich Informatik, Technische Universität Darmstadt
{daubert|erwin}@informatik.tu-darmstadt.de,
borgert@tk.informatik.tu-darmstadt.de

Zusammenfassung In dieser Arbeit wird ein dezentrales Verfahren zur Planung von Reiseprozessen vorgestellt. Transportdienstleister bieten ihre Dienste über einen Service-Marktplatz an und können mit Hilfe der Unified Service Description Language (USDL) effektiv vorselektiert werden. Der Reiseprozess wird durch schrittweise Verfeinerung und Graphtransformation erstellt. Auf diese Transformationen können Dienste direkt Einfluss nehmen. Das macht unser Verfahren im Gegensatz zu zentralen Planungsansätzen flexibel, offen und erweiterbar.

1 Einleitung

In dieser Arbeit wird ein neues, dezentrales Verfahren zur intermodalen Reiseplanung vorgestellt, das auf aktuellen Internet-of-Services (IoS) Technologien [7] basiert. Transportdienstleister können beliebige Modalitäten (Flug, Bahn, Bus, Taxi, ...) anbieten und stellen diese über Softwaredienste (Services) bereit, welche für Kunden über einen Marktplatz ansprechbar sind.

Dabei müssen prinzipiell die folgenden Probleme gelöst werden: *Finden von Diensten*, *Routing*, und *Scheduling*. Zunächst müssen die Modalitäten und Dienstanbieter ausgewählt werden, die in Frage kommen. Danach befasst sich *Routing* mit dem Finden der optimalen Route zwischen zwei Stopps. Anbieter, die auf Basis eines Fahrplanes operieren, schränken die verfügbaren Abfahrts- und Ankunftszeiten ein. Unter Berücksichtigung dieser Constraints befasst sich *Scheduling* mit der Erstellung eines optimalen Zeitplanes.

Existierende Ansätze lassen sich im Wesentlichen in zwei Kategorien einteilen: Einerseits existieren Systeme mit guten Lösungen für das Routing- und Scheduling-Problem, die allerdings auf zentral gespeicherten Modellen basieren. Das stellt aber eine in der Praxis kaum realisierbare Idealvorstellung dar, da Transportdienstleister die Hoheit über ihre Daten (u.a. gerichtlich) verteidigen. Eine aktive Einbeziehung in die Planung ermöglicht außerdem die bessere Nutzung von domänenspezifischem Wissen. Andererseits existieren SOA-basierte Systeme, die meist Dienste nur auf Grund ihrer technischen Schnittstellen auswählen. Dies ist ineffizient, da im Service Discovery eine wesentlich bessere Vorselektion von Diensten erreicht werden kann.

Im Folgenden stellen wir unseren Ansatz zur intermodalen Reiseplanung vor, der es erlaubt, das Navigations- und Scheduling-Problem in offenen Service-Märkten zu lösen.

Der Rest des Papers ist wie folgt aufgebaut. In Abschnitt 2 werden zunächst verwandte Arbeiten diskutiert. In Abschnitt 3 wird die Systemarchitektur vorgestellt, danach die Planungsmethode in Abschnitt 4. Abschnitt 5 beschreibt die Implementierung. Der Artikel schließt mit der Zusammenfassung in Abschnitt 6.

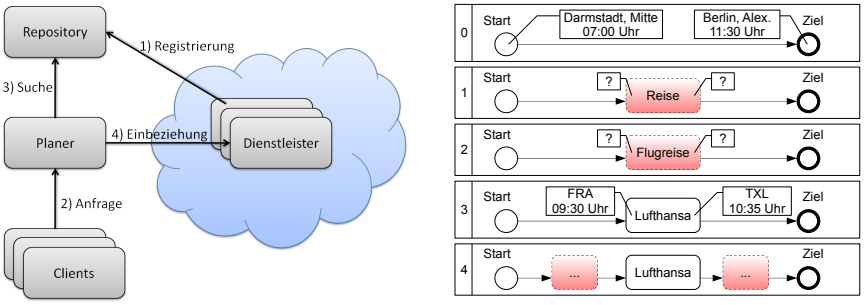
2 Verwandte Arbeiten

Graphenbasierte Modellierung mit mehreren Modalitäten wird in [8] vorgeschlagen. Hierbei werden jeweils eigene Kanten für jede Modalität verwendet. Zum Bestimmen von Routen wird eine an SQL angelehnte Abfragesprache vorgeschlagen. Der Ansatz adressiert primär das Routingproblem, berücksichtigt aber Abfahrtszeiten nur eingeschränkt. Im Rahmen des *iTransIT*-Frameworks [13] wird ein gemeinsames Datenformat für Modalitäten beschrieben, das *Common Data Model*. Es dient als Abstraktionsschicht, die über Geo-Datenbanken gelegt wird. Ein Reiseplaner ist durch den *Smart Traveler Information Service (STIS)* [9] realisiert. Für die Routenberechnung werden einzelne, logische Subgraphen für jede Transportmodalität verwendet. Jedoch wählt der Benutzer zuerst eine Modalität aus, danach wird eine Route auf dem entsprechenden Graphen mittels eines Kürzeste-Wege-Algorithmus gesucht, verbleibende "Lücken" werden dann mit weiteren Modalitäten geschlossen. STIS adressiert ebenfalls primär das Routing- und nicht das Schedulingproblem.

Ontologiebasierte Modellierung wird in [12] und [16] beschrieben. In [12] wird eine Reise als eine Reihe von geordneten *stop points* zwischen Start und Ziel modelliert. Lösungen werden mittels einer inferenzbasierten Ontology-Engine ermittelt, die zusätzlich auf Geo-Datenbanken zugreift. Unterschiedliche *journey patterns* können verwendet werden, um z.B. Routen mit "wenig Fußweg" oder mit "überdachten stop points" zu finden. In [16] wird eine Datenmodellierung mit dem Protégé-Werkzeug und eine Auswertung mit Hilfe des Jena-Frameworks vorgenommen. Die vorgestellte Evaluation ist mit nur 25 Elementen sehr klein.

Andere Ansätze wie [4] setzen auf **Constraint Programming**. Eine Reise besteht aus *tasks*, welche zu *templates* (etwa "tip", "fly") zusammengefasst werden können (Abstraktion). Je ein Teil der Reise wird herausgegriffen, Alternativen verglichen und dem Benutzer zur Auswahl gegeben. Das Constraint-Netzwerk umfasst alle Nebenbedingungen und Berechnungen, durch die Templates wird die Komplexität übersichtlich gehalten. Die Daten stammen von einer Reihe Agenten, etwa Wrapper und Screenscraper für Fahrplanauskünfte. Einerseits muss der Nutzer hier bei jedem Schritt aktiv werden und eine Wahl treffen, andererseits sind dem Aufbau einer Reise durch statische Templates enge Grenzen in der Flexibilität gesetzt [5].

SOA-basierte Systeme werden in [14] und [6] vorgestellt. Der in [14] vorgestellte Dienst ermittelt die günstigste Reise zwischen zwei Städten mittels eines Service-Mashups. Aufgrund beschränkter Granularität und kaum berück-



(a) Systemarchitektur (b) Schrittweise Prozessverfeinerung

sichtigter zeitlicher Nebenbedingungen eignet sich dieses Vorgehen kaum für ein allgemeineres Reiseproblem. Ähnlich ist *Self-Serv* [6] mit dem *Complete Travel Planning Service*, einer P2P-basierten Methode zur Web-Service Orchestrierung. Anhand eines State-Charts wird ein Reiseprozess erzeugt und nur auf Basis der Schnittstellen werden passende Services (etwa Flugbuchung) gewählt.

3 Architektur und Dienstbeschreibung

Der hier vorgestellte Ansatz zur automatischen Reiseplanung basiert auf IoS-Technologien und der Service-Beschreibungssprache USDL, die im Rahmen des Theseus/TEXO-Projekts [7] entwickelt wurden. Ziel von USDL (Unified Service Description Language) [15] ist es, eine umfassende Beschreibung zu schaffen, mit welcher zukünftig Dienstleistungen auf IoS-Marktplätzen angeboten und gefunden werden können. Eine wesentliche Neuerung von USDL ist der Einbezug nicht-technischer Eigenschaften von Diensten (“business”, “operational”). Somit können Ort und Zeit der Diensterbringung, sowie weitere nicht-funktionale Eigenschaften beschrieben werden.

Die Architektur ist in Abbildung 1a dargestellt und unterscheidet vier Arten von Teilnehmern: *Service Repository*, *Planer*, *Dienstleister* und *Clients*. Der Ablauf gestaltet sich wie folgt: Reisedienstleister beschreiben ihre Dienste mit USDL, also an welchen Orten diese in Anspruch genommen werden können, sowie Webservices zur Planung, und hinterlegen diese im Repository (1). Schickt ein Client eine Reiseplanungsanfrage an den Planer (2), erzeugt dieser mittels Graphtransformation einen Reiseprozess und fragt dabei die zu verwendenden Dienste am Repository ab (3). Dienste können dann vom Planer aktiv mit einbezogen werden (4).

4 Planung von Reiseprozessen

Das Ergebnis der Planung ist ein Reiseprozess, der detailliert beschreibt, wie der Nutzer vom Startort zum Zielort reisen kann. Dieser Prozess könnte später von einer Assistenzanwendung auf einem mobilen Gerät ausgeführt werden und dem Benutzer Navigationsanweisungen geben. Zur Modellierung, Darstellung und

Ausführung von Reiseprozessen verwenden wir Methoden aus dem Geschäftsprozessmanagement (BPM).

Das Prozessmodell basiert auf der Sprache PASS (Parallel Activities Specification Scheme) [11]. Unser Verfahren könnte ebenfalls zusammen mit anderen Sprachen, wie z.B. BPEL, angewendet werden. PASS erfüllt allerdings alle unsere Anforderungen und es kann viel an unnötiger Komplexität vermieden werden. Im Weiteren wurde in einer früheren Arbeit eine Engine entwickelt, die PASS-Prozesse auf mobilen Geräten ausführen kann [2]. Damit können Anwendungen zur mobilen Navigationsunterstützung des Benutzers erstellt werden.

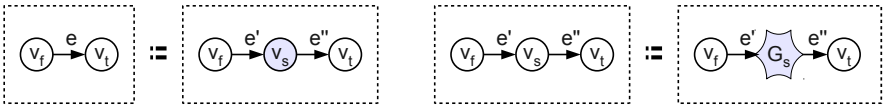
Aus der Sicht des Planers betrachten wir den Prozess zunächst abstrakt als Graphen $G = (V, E)$. Die Knoten V in diesem Graphen sind Aktivitäten, die durch unterschiedliche Dienstleistungen erbracht werden, oder Pseudoknoten, wie Start, Ziel, Split, Join, etc. Insbesondere entsprechen die Knoten also nicht räumlichen Orten, wie oftmals in Wegfindungsproblemen verwendet, sondern vielmehr Diensten in einem Prozess. Die Kanten E beschreiben mögliche Übergänge, also die zeitliche Abfolge von Aktivitäten.

Knoten sind mit Kontextinformationen attributiert, insbesondere sind dies Ort und Zeit. Diese Attribute existieren zweimal pro Knoten, nämlich für den geplanten Beginn der Aktion, sowie dem Ende. Weiterhin können alle Nicht-Pseudoknoten mit einer in USDL beschriebenen Dienstleistung versehen werden. Verwendet man einen Graphen mit ausgezeichnetem Start- und Zielknoten als Reiseprozess, dann ergibt sich mit jedem linearisierten Pfad zwischen Start und Ziel ein Reiseplan, der angibt, wann und wo welche Dienstleistung genutzt werden soll. Durch parallele Teilpfade lassen sich mehrere Alternativen ausdrücken, etwa dass ein Bus oder alternativ wenige Minuten später eine Straßenbahn verwendet werden kann. Durch einen derart gestalteten Graphen können auch komplexe Anfragen ausgedrückt werden, indem weitere Knoten für einen Hotelaufenthalt oder gewünschte Zwischenaufenthalte in den Ausgangsgraphen eingefügt werden.

Die Durchführung einer Reiseplanung erfolgt durch Anwendung einer Reihe von Regeln zur Transformation des Prozessgraphen. Die Kernidee dabei ist, mit einem sehr einfachen Graphen zu beginnen und diesen schrittweise zu verfeinern, also die Reise auszugestalten (Abbildung 1b). Der Graph wird mit dem Java-Framework *Graph Rewrite Library (GRL)* [1] bearbeitet. Graphsuchen und -ersetzungen werden dabei in der Sprache RDL (Rule Description Language) formuliert. Eine einfache Produktionsregel lautet z.B. wie folgt:

```
P() :- |F:Node,e:Edge,T:Node| :- F-e->T & T.startTime!=null
    := |S:Node,f:Edge| S=new Node(), f=new Edge(), F-e->S-f->T;
```

Die linke Seite der Regel (LHS) beschreibt das Muster, das im Graphen gefunden werden soll. In diesem Beispiel wird nach Belegungen der Variablen `F`, `e` und `T` gesucht, die zwei Bedingungen erfüllen: Der Pfadausdruck verlangt, dass `F` und `T` direkt durch die Kante `e` verbunden sind. Die folgende Bedingung überprüft, ob das `startTime`-Attribut von `T` gesetzt ist. Die rechte Seite der Regel (RHS) beschreibt die Transformation. Hier wird ein neuer Knoten `S` und eine neue Kante `f` eingefügt. Da GRL auch den Aufruf von Java-Methoden unterstützt,



(a) Einfügen (b) Substitution

Abbildung 1: Graphtransformationen

können Transformationen auch alternativ in Java implementiert werden. Das ist für komplexe Ersetzungen oft hilfreich.

Zu Beginn besteht der Graph nur aus dem Start- und Zielknoten, sowie einer Kante dazwischen. Zur schrittweisen Verfeinerung des Reiseprozesses dienen nun die folgenden drei Grundkonzepte: *Einfügen*, *Substitution* und *Adaption*.

Einfügen: Beim Einfügen wird im Repository ein Dienst gesucht, der die Transportlücke zwischen v_f (from) und v_t (to) möglichst gut schließt. Hierbei werden Ortsinformationen aus USDL-Beschreibungen ausgewertet. Weitere Kriterien, wie die aktuelle Komplexität des Graphen, sind möglich. Auch Vorlieben des Nutzers sind denkbar. Der neue Knoten v_s repräsentiert dann diesen Dienst. Kante e wird durch zwei neue Kanten e' und e'' ersetzt, deren summierte verbleibende Transportlücke kürzer als die von e sein muss. Mehrere Alternativen (parallele Pfade) sind ebenfalls möglich. Abbildung 1a zeigt diese Transformation.

Substitution: Im Falle der Substitution wird ein Knoten v_s, dessen USDL-Beschreibung einen Webservice zur Substitution umfasst, durch einen Subgraphen G_s ersetzt. Damit kann v_s an einen anderen Dienst delegieren, z.B. kann der abstrakte Knoten *Flugreise* durch den konkreten Anbieter *Lufthansa* ersetzt werden, oder G_s kann als Template für einen komplexen Subgraphen dienen. Der Anbieter *Lufthansa* kann etwa beschreiben, dass dieser Teil der Reise aus Check-in, Gepäckaufgabe, Boarding, Flug, ... besteht. Ein Dienstleister kann somit selbst die Transformation bestimmen, und damit den Reiseprozess entscheidend beeinflussen. Der Ansatz wird damit auch offen hinsichtlich beliebiger, neuer Transportmodalitäten. Dieses Vorgehen wird in Abbildung 1b illustriert.

Adaption: Die Adaption ist die schwächste Form der Transformation und wirkt sich nur auf die Attributierung eines Knotens aus. Auch hier wird ein durch die USDL-Beschreibung gegebener Webservice abgefragt. Sinnbildlich kann man diesen als Fahrplanauskunft betrachten. Dieses Konzept erlaubt die Handhabung von fahrplanbasierten und nicht-fahrplanbasierten Transportdiensten. Bei fahrplanbasierten Diensten erfolgt eine Anpassung an die Zeiten. Vor der Adaption könnte man etwa von einer unbestimmten Busfahrt sprechen, danach von einer festen Verbindung mit Haltestellen und Fahrzeiten. Nicht-fahrplanbasierte Dienste, wie eine Taxifahrt, stehen zu jeder Zeit zur Verfügung. Deshalb wäre es nicht möglich, diese Zeiten statisch in der USDL-Beschreibung zu hinterlegen.

Insgesamt wurden basierend auf diesen Konzepten 12 verschiedene Transformationsregeln entwickelt. Für Adaption und Substitution existieren mehrere Regeln um Optimierungsziele abzudecken, etwa ob von der Ankunftszeit bevor-

zugt zurückgeplant wird oder ob die Abfahrtszeit maßgeblich ist. Weitere Regeln der Adaption können etwaige Wartezeiten minimieren. Pruning-Regeln dienen zum Entfernen von schlechten Alternativen. Da Services direkt Transformationsregeln für den Prozess mit einbringen können, sind außerdem Validierungsschritte notwendig, um die Terminierung des Transformationsprozesses und korrekte Prozesse sicherzustellen.

Alle Regeln liegen in einer Prioritätsreihenfolge vor. Pruning hat eine hohe Priorität, Einfügen aus dem Repository sollte dagegen nur durchgeführt werden, falls eine Transportlücke nicht anderweitig geschlossen werden kann. Der Algorithmus führt eine Reihe von Transformationsphasen bestehend aus Suche und Transformation durch. Jede Phase beginnt mit der Suche. Dabei wird immer mit der Regel höchster Priorität begonnen. Trifft die Bedingung der Regel (LHS) auf dem Graphen an keiner Stelle zu, wird mit der jeweils nächsten Regel fortgesetzt. Trifft keine Regel zu, endet der Algorithmus. Nach der Suche wird die Transformation der Regel auf alle Treffer angewendet und die aktuelle Phase endet [10].

Beispiel: Der Nutzer möchte von Darmstadt Mitte ab 07:00 Uhr nach Berlin Alexanderplatz (möglichst bis 11:30 Uhr) reisen. Der Planer konstruiert aus dieser Anfrage einen Graphen mit zwei attributierten Knoten: Start (07:00 Uhr, Darmstadt) und Ziel (11:30 Uhr, Berlin Alex.). Eine Kante zwischen beiden Knoten symbolisiert die Abfolge zwischen den Knoten, also den Reisewunsch, und somit die Aufgabestellung (Abbildung 1b, Schritt 0). Beide Knoten haben kein USDL-Attribut, daher scheiden Adaption und Substitution aus, es verbleibt das Einfügen. Entsprechend der Ortsangaben sowie des geringen Umfangs des Graphen liefert das Repository einen allgemeinen Dienst zurück, hier als Beispiel der Reise-Dienst der Lufthansa in Form einer USDL-Beschreibung. Daraus wird ein neuer Knoten (mit USDL-Attribut) erstellt und eingefügt (Schritt 1). Der neue Knoten besitzt noch keine Kontextattribute (Ort & Zeit), ist aber nach der USDL-Beschreibung substituierbar und wird daher in der nächsten Phase transformiert. Ein per USDL beschriebener Webservice des Reisedienstes wird mit den umrahmenden Kontextinformationen (von Start und Ziel) aufgerufen. Dieser wählt, hier anhand der Distanz, Fliegen als sinnvollste Reise-Modalität und liefert den Flugreise-Dienst zurück (Schritt 2). Da dem neuen Dienst ebenfalls noch Kontextinformationen fehlen, und kein Webservice zur Substitution enthalten ist, wird in Phase 3 eine Adaption durchgeführt und ein Webservice der Flugreise aufrufen. Der Service sucht nach entsprechenden Flughäfen und Flügen, hier Flug LH176 um 9:30 Uhr von Flughafen Frankfurt nach Berlin Tegel, und liefert diesen als Kontextinformationen zurück (Schritt 3). Hier wird auch deutlich, dass durch simultane Wahl von Orten (wie Flughäfen) und Zeiten Routing- sowie Scheduling kombiniert betrachtet werden. Der Dienst wählt, ähnlich der Verbindungssuche der Deutschen Bahn, Haltestellen, Verkehrsmittel sowie Abfahrtszeiten, um die Gesamtreisedauer zu minimieren, und nutzt dazu das umfangreiche domänenspezifische Wissen des Dienstleisters. Ein großer Teil der Transportlücke ist jetzt geschlossen. Es verbleiben kleinere Lücken, die in weiteren Phasen analog geschlossen werden. Natürlich können bei der Substi-

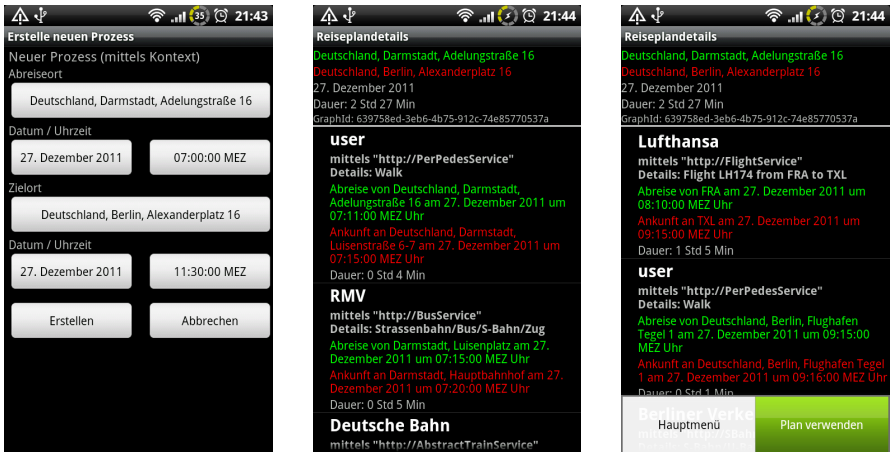
tution auch Graphen mit alternativen Dienstleistungen zurückgeliefert werden, etwa Flug- sowie Bahnreise als auch bereits mit Kontextinformationen (Flüge) versehene alternative Flugreise-Dienste. Mit der Adaption sind auch nachträgliche Änderungen möglich, beispielsweise ein späterer Flug aufgrund langer Anreise.

5 Implementierung

Im Rahmen der Arbeit wurde ein USDL-basiertes Service-Repository auf Basis von PostgreSQL und PostGIS entwickelt. Die Ortsinformationen werden aus den USDL-Beschreibungen extrahiert und können bei der Servicesuche verwendet werden. Der Zugriff auf das Repository erfolgt über SOAP/HTTP. Für ein realitätsnahes Szenario wurden eine Reihe von Diensten entwickelt, darunter ein Flug-Dienst auf Basis eines Crawlers für Lufthansa-Webseiten, Dummy-Services mit etlichen Haltestellen und zufälligen Verbindungen für die Deutsche Bahn, den RMV, die Berliner Verkehrsbetriebe, sowie ein Fußgängerservice.

Ein exemplarischer Client wurde als Android App realisiert (Abbildung 2). Eine Reise von Darmstadt nach Berlin wurde als Szenario zur Abdeckung der Dienste verwendet, hier kommen die Modalitäten zu Fuß, Bus, Zug, Flugzeug und S-Bahn kombiniert zum Einsatz. Die Kommunikation zwischen Planer und Client wurde mit der Kommunikations-Middleware MundoCore [3] implementiert.

Nach ersten Tests korreliert die Laufzeit einer Reiseplanung mit dem Umfang der Aufgabestellung. Der Haupteinflussfaktor sind die Zugriffe des Planers auf Webservices der Reisedienstleister, einschließlich einer Anfrage an das Repository sind dies maximal 3 Aufrufe für jeden Knoten. Das Beispielszenario umfasst final 8 echte Knoten, involviert 5 verschiedene Teilnehmer und wurde mit 17 Transformationen erstellt.



(a) Zieleingabe (b) Reiseplan (Teil 1) (c) Reiseplan (Teil 2)

Abbildung 2: Screenshots der Android App

6 Zusammenfassung

Eine Reiseplanung auf dieser Basis kann Dienstleister aktiv in die Planung mit einbeziehen, auf deren Fahrplanauskünfte und Buchungssysteme zurückgreifen und somit ideale, intermodale Reisepläne erstellen. Weiterhin wird im Rahmen der Verbreitung von Smartphones und Assistenzdiensten der Weg zu einer ineinandergreifenden Reiseunterstützung eröffnet.

Danksagung. Diese Arbeit wurde unterstützt durch das Theseus-Programm, gefördert durch das Bundesministerium für Wirtschaft und Technologie (Kennziffer: 01MQ07012).

Literatur

1. Aitenbichler, E.: Entwurf und Implementierung eines programmierten Graphersetzungssystems in Java. Master's thesis, Johannes Kepler Universität Linz (2000)
2. Aitenbichler, E., Borgert, S., Mühlhäuser, M.: Distributed Execution of S-BPM Business Processes. In: S-BPM ONE 2010 - The Subjectoriented BPM Conference. Springer (2011)
3. Aitenbichler, E., Kangasharju, J., Mühlhäuser, M.: MundoCore: A Light-weight Infrastructure for Pervasive Computing. Pervasive and Mobile Computing (2007)
4. Ambite, J., Barish, G., et al.: Getting from here to there: Interactive planning and agent execution for optimizing travel. In: Proc. of AAAI. pp. 862–869 (2002)
5. Arpinar, I.B., Zhang, R., Aleman-meza, B., Maduko, A.: Ontology-driven web services composition platform. In: Proc. of IEEE International Conference on e-Commerce Technology. pp. 6–9 (2004)
6. Benatallah, B., Sheng, Q.Z., Dumas, M.: The self-serv environment for web services composition. IEEE Internet Computing 7(1), 40–48 (January 2003)
7. BMWi: TEXO - Business Webs in the Internet of Services., http://www.theseus-programm.de/anwendungsszenarien/texo/default.aspx, Stand: 12.10.2010
8. Booth, J., Sistla, P., Wolfson, O., Cruz, I.: A data model for trip planning in multimodal transportation systems. In: Proc. of the EDBT. pp. 994–1005. ACM (2009)
9. Brennan, S., Meier, R.: STIS: Smart travel planning across multiple modes of transportation. In: Proc. of ITSC. pp. 666–671. IEEE (2007)
10. Daubert, J.: Service-Komposition von Reiseprozessen mittels Graphtransformation. Master's thesis, TU Darmstadt (2011)
11. Fleischmann, A.: Distributed Systems: Software Design and Implementation. Springer (1994)
12. Houda, M., Khemaja, M., Oliveira, K., Abed, M.: A public transportation ontology to support user travel planning. In: Proc. of RCIS. pp. 127–136. IEEE (2010)
13. Meier, R., Harrington, A., Cahill, V.: A framework for integrating existing and novel intelligent transportation systems. In: Proc. of ITSC. pp. 154–159. IEEE (2005)
14. Navabpour, S., Ghoraie, L., Malayeri, A., Chen, J., Lu, J.: An Intelligent Traveling Service Based on SOA. In: Proc. of Services. pp. 191–198. IEEE (2008)
15. SAP Research: USDL Specifications. http://www.internet-of-services.com/
16. Wang, J., Ding, Z., Jiang, C.: An Ontology-based Public Transport Query System. In: Proc. of Semantics, Knowledge and Grid (SKG). p. 62. IEEE (2007)

m^3 – A Behavioral Similarity Metric for Business Processes

Matthias Kunze, Matthias Weidlich, and Mathias Weske
{matthias.kunze, matthias.weidlich, mathias.weske}@hpi.uni-potsdam.de

Hasso Plattner Institute at the University of Potsdam
Prof.-Dr.-Helmert-Strasse 2-3, 14482 Potsdam

Abstract. With the increasing uptake of business process management, companies maintain large scale process repositories consisting of hundreds or thousands of process models. So far, discovery within these repositories is limited to free text search or folder navigation. In a separate stream of research, similarity measures were introduced to get a better understanding of the relationships between process models. Unfortunately, calculating such similarity is complex, so that these techniques cannot be used in large process model repositories, where they would be most valuable.
To overcome this issue, we introduce the m^3-metric, which is based on behavioral profiles that provide an abstraction on the detailed behavior of processes. This metric can be computed efficiently and enables tree based similarity search within large process model repositories.

1 Introduction

In recent years we saw large business process model collections grow in many organizations, whereas the effective management of such repositories requires efficient capabilities to find process models among hundreds or thousands of candidate models. The question of similarity between process models has been thoroughly studied. Still, existing approaches do not scale well in computation complexity, nor do they address transitivity, which is essential for efficient similarity search.

Similarity metrics provide such a property and significantly increase search performance, as we showed for process model structures, i.e., the graph edit distance, in [7]. In this paper we address behavioral aspects of processes and present the m^3-metric: A metric based on behavioral profiles that provides a similarity ranking of process models relative to a given query model and can be employed in metric similarity search methods, cf. [14]. Behavioral profiles focus on ordering relations between pairs of activities in a process model. While this notion abstracts from the actual behavior of a process, it is computed efficiently [11]. Approaches that take the complete state space of a process into account, in turn, suffer from exponential complexity due to the state space explosion problem.

The remainder of this work is structured as follows: In Section 2 we present previous work related to the topic of process model similarity and searching, while

Section 3 introduces formal concepts for the m^3-metric. In Section 4 we show how the aforementioned metric is constructed from behavioral profile relations and present its rationale by means of an illustrative example, before we conclude this work and give an outlook on future studies in Section 5.

2 Related Work

Similarity of process models has been addressed from various angles. An overview of linguistic, structural, and behavioral measures used for similarity search of process models can be found in [4]. Measures for structural similarity, e.g., the one based on the graph edit distance [3], neglect common behavior expressed in a different syntax when comparing process models. Modeling a loop with a loop activity in BPMN or with a control flow cycle would, therefore, impact on structural similarity of process models in a negative manner. Measures for behavioral similarity are insensitive to such syntactical differences. They may be based directly on the sets of possible traces of process models, e.g., by computing the intersection of traces of two models. In order to get a more fine-granular measure, an n-gram representation of the sets of traces may be used to judge on similarity [12]. Other approaches advocate the application of causal footprints to approximate the behavior and to measure similarity of process models [10]. Still, these approaches are computationally hard, so that recent techniques aim at a multi-step approach that narrows the search space in a step-wise manner [13]. We avoid such problems as behavioral profiles are computed efficiently for a broad class of process models. A behavioral abstraction close to the behavioral profile has been applied for matching BPEL process definitions [5]. However, the approach is restricted to BPEL processes and transitivity aspects of the proposed measures are not discussed.

In traditional databases, data is generally made up of simple structures and attribute data—indexing techniques have been very successfully elaborated on and implemented. However, for complex data, such as process models, these techniques are not applicable, because no intrinsic ordering exists among data objects and mapping them to simple values, i.e., hashing, is not meaningful. Similarity search addresses this field where nothing but pairwise distances between data objects can be measured [14]. This concept requires the distance—or dissimilarity—of two objects to be a proper metric, and thus to provide transitivity. By that, it becomes possible to predict or at least constrain the distance of a pair of data objects, if one knows the respective pairwise distances of these data objects to a third one. Several indexing techniques have been developed [2,6]. However, the above process model similarity measures have not been shown to provide proper metrics.

3 Background

This section introduces the background of our work in terms of the characteristics of a distance metric, a formal model, and the concept of a behavioral profile.

3.1 Distance Metric

To efficiently[1] search within a space of given objects, it is necessary to partition that space and exclude some of the partitions from exhaustive search. Partitioning is relatively easy for objects whose features can be mapped to vectors, i.e., in coordinate spaces. However, such a representation cannot be generally assumed, in particular for process behavior or graph structures, cf. [7]. However, in metric spaces—a generalization of coordinate spaces—nothing but a distance with certain properties is required to partition the space, the notion of such a distance is a metric [14].

Definition 1 (Distance Metric). *A metric space is a pair* $\mathcal{S} = (\mathcal{D}, d)$ *where* $\mathcal{D}$ *is the domain of objects and* $d : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$ *is a* metric*, i.e., a distance function with the following properties:*

- *symmetry:* $\forall o_i, o_j \in \mathcal{D} : d(o_i, o_j) = d(o_j, o_i)$
- *nonnegativity:* $\forall o_i, o_j \in \mathcal{D}, o_i \neq o_j : d(o_i, o_j) > 0 \bigwedge \forall o_i \in \mathcal{D} : d(o_i, o_i) = 0$
- *triangle inequality:* $\forall o_i, o_j, o_k \in \mathcal{D} : d(o_i, o_k) \leq d(o_i, o_j) + d(o_j, o_k)$

Particularly, the triangle inequality states that every pair of distances between three objects is larger than the remaining. This allows deriving minimum and maximum bounds for the distance of two points, if their respective distances to another point are given, and thus partitioning the search space.

3.2 Process Models

A process model is based on a graph containing activity nodes and control nodes. It captures the commonalities of most process description languages.

Definition 2 (Process Model). *A* process model *is a tuple* $P = (A, s, C, N, F, T)$ *where:*

- A *is a finite non-empty set of activity nodes,*
- C *is a finite set of control nodes,*
- $N = A \cup C$ *is a finite set of nodes with* $A \cap C = \emptyset$*,*
- $F \subseteq N \times N$ *is the flow relation, such that* (N, F) *is a connected graph,*
- $\bullet n = \{n' \in N | (n', n) \in F\}$ *and* $n\bullet = \{n' \in N | (n, n') \in F\}$ *denote direct predecessors and successors, we require* $\forall\ a \in A : |\bullet a| \leq 1 \wedge |a \bullet| \leq 1$*,*
- $s \in A$ *is the only start node, such that* $\bullet s = \emptyset$ *and* $\forall\ n \in N : s\ F^*\ a$ *with* F^* *as the reflexive transitive closure of* F*,*
- $T : C \to \{and, xor\}$ *associates each control node with a type.*

We assume trace semantics for process models. The behavior of a process model $P = (A, s, C, N, F, T)$ is a set of *traces* $\mathcal{T}_P$. It comprises a set of lists of the form $\sigma = \langle s, a_1, \ldots, a_n \rangle$ with $n > 0$, $n \in \mathbb{N}$, $a_i \in A$ for all $0 < i \leq n$, which represent the execution order of activities. These traces follow on common Petri net-based formalizations [9].

[1] An efficient algorithm is one that avoids examining every point in the set.

3.3 Behavioral Profiles

A behavioral profile captures behavioral characteristics of a process model by three relations between pairs of activity nodes. These relations are based on the notion of *weak order*. Two activities of a process model are in weak order, if there exists a trace in which one activity occurs after the other.

Definition 3 (Weak Order Relation). *Let $P = (A, s, C, N, F, T)$ be a process model and $\mathcal{T}_P$ its set of traces. The* weak order relation $\succ_P \subseteq A \times A$ *contains all pairs (x, y), such that there is a trace $\sigma = n_1, \ldots, n_m$ in $\mathcal{T}_P$ with $j \in \{1, \ldots, m-1\}$ and $j < k \leq m$ for which holds $n_j = x$ and $n_k = y$.*

Based on the weak order relation, the behavioral profile is defined as follows.

Definition 4 (Behavioral Profile). *Let $P = (A, s, C, N, F, T)$ be a process model. A pair $(x, y) \in A \times A$ is in one of the following relations:*
- *The* strict order relation $\rightsquigarrow_P$*, if $x \succ_P y$ and $y \not\succ_P x$.*
- *The* exclusiveness relation $+_P$*, if $x \not\succ_P y$ and $y \not\succ_P x$.*
- *The* interleaving order relation $||_P$*, if $x \succ_P y$ and $y \succ_P x$.*

The set $\mathcal{B}_P = \{\rightsquigarrow_P, +_P, ||_P\}$ of all three relations is the behavioral profile *of P.*

We illustrate the relations of the behavioral profile for the BPMN model in Fig. 1. It holds $A \rightsquigarrow D$ as both activities are ordered if they occur together in a trace and $B||C$ due to the concurrent execution of both activities. An activity is either exclusive to itself (e.g., $A + A$ in Fig. 1) or in interleaving order to itself. Further details on behavioral profiles can be found in [11], which also shows how a behavioral profile of a process model is computed in polynomial time to the size of the model under the assumption of soundness. Soundness is a correctness criterion that guarantees the absence of behavioral anomalies [1].

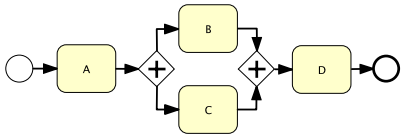


Fig. 1. Example BPMN model a

4 Construction of the m³-Metric

We assume two process models P and Q to be similar, if they expose a common share of behavior, i.e., they have a common set of activities that yield equal behavioral profiles: $(\rightsquigarrow_P \cap \rightsquigarrow_Q) \cup (+_P \cap +_Q) \cup (||_P \cap ||_Q) \neq \emptyset$. The larger this overlap of behavioral profiles is, the more similar two process models are. We quantify this overlap by means of the established Jaccard similarity coefficient for the similarity of two sets: $sim(A, B) = \frac{|A \cap B|}{|A \cup B|}$. If two sets of behavioral profile relations consist of the same pairs, they are equal, i.e., their similarity is 1. If two behavioral profile relations have no common pairs, their similarity coefficient is 0. From the relations of the behavioral profile we propose three individual similarity coefficients:

Exclusiveness Similarity captures the amount of exclusiveness, i.e., pairs of activities that must not occur together, shared by the two models, $s_{+}(P,Q) = \frac{|+_P \cap +_Q|}{|+_P \cup +_Q|}$.

Strict Order Similarity quantifies to which degree two processes expose an overlap in their order dependencies for pairs of activities, $s_{\rightsquigarrow}(P,Q) = \frac{|\rightsquigarrow_P \cap \rightsquigarrow_Q|}{|\rightsquigarrow_P \cup \rightsquigarrow_Q|}$.

Interleaving Order Similarity accounts for the observation that parallel execution of activities covers also sequential execution of the same activities in any order, i.e., activities that are executed in parallel can also be executed in a certain sequence and the according traces are therefore considered similar. $s_{||}(P,Q) = \frac{1}{2} \cdot \left(\frac{|(\rightsquigarrow_P \cup ||_P) \cap \rightsquigarrow_Q|}{|\rightsquigarrow_P \cup ||_P \cup \rightsquigarrow_Q|} + \frac{|\rightsquigarrow_P \cap (\rightsquigarrow_Q \cup ||_Q)|}{|\rightsquigarrow_P \cup \rightsquigarrow_Q \cup ||_Q|} \right)$.

A distance metric expresses a dissimilarity of two objects. Analogously, there exists a set distance that is constructed from the Jaccard similarity coefficient which has been proven to be a metric [8]: $d(A,B) = 1 - sim(A,B)$. Thus, each of the aforementioned similarity measures translates into a single distance metric. Through weighted summation of these three single metrics, we can compose them into one (thus the name m^3-metric). This composition preserves the properties of a metric.

Definition 5 (m^3-metric). *Let P and Q be two process models and $s_{+}(P,Q)$, $s_{\rightsquigarrow}(P,Q)$, and $s_{||}(P,Q)$ the similarity metrics based on behavioral profiles. Then, the m^3-metric is defined as*

$$m^3(P,Q) = 1 - \sum_i w_i \cdot s_i(P,Q)$$

with $i \in \{+, \rightsquigarrow, ||\}$ and weighting factors $w_i \in (0,1)$ such that $\sum_i w_i = 1$.

To illustrate this metric consider the sample processes, Fig. 1-3. The relations of the behavioral profile for these models are summarized in Table 1. We chose the following weights to demonstrate the m^3-metric: $w_{+} = 0.5, w_{\rightsquigarrow} = 0.3, w_{||} = 0.2$. Here, we understand exclusiveness as the strictest criterion and thus give it the highest weight to penalize violations thereof. Interleaving order offers the greatest flexibility and thus is considered the weakest criterion, which is why it receives the smallest weight.

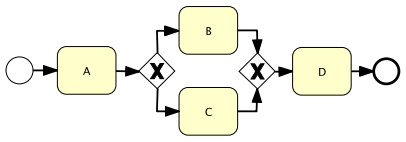


Fig. 2. Example BPMN model b

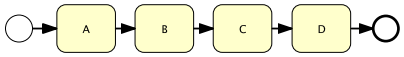


Fig. 3. Example BPMN model c

Table 1. Relations of the behavioral profile for the example process models

	Model a, Fig. 1	Model b, Fig. 2	Model c, Fig. 3
$+$	{(A,A), (B,B), (C,C), (D,D)}	{(A,A), (B,B), (B,C), (C,C), (D,D)}	{(A,A), (B,B), (C,C), (D,D)}
$\rightsquigarrow$	{(A,B), (A,C), (A,D), (B,D), (C,D)}	{(A,B), (A,C), (A,D), (B,D), (C,D)}	{(A,B), (A,C), (A,D), (B,C), (B,D), (C,D)}
$\|\|$	{(B,C)}	$\emptyset$	$\emptyset$

Building the metric space of behavioral profiles $\mathcal{M} = (\mathcal{B}, m^3)$ for the three example process models and computing the m^3-distances, we get $m^3(a, b) = 0.117$ and $m^3(b, c) = 0.183$. According to our metric, the behavioral distance between models a and b is smaller than the one between models b and c.

Since m^3 is a metric, it features the triangle inequality, which allows us to bound the distance of models a and c without actually computing it. Based on Def. 1, $|m^3(a, b) - m^3(b, c)| \leq m^3(a, c)$ (lower boundary) and $|m^3(a, b) + m^3(b, c)| \geq m^3(a, c)$ (upper boundary), i.e., $0.066 \leq m^3(a, c) \leq 0.3$. This approximation is confirmed by the actual computed value, which is $m^3(a, c) = 0.067$. The computed distances also comply with our perception of the behavioral similarity of the sample process models. Since possible traces of a cover the traces of c, due to the parallel branch, these two models are more similar (less distant) to each other than a and b, and b and c respectively.

5 Conclusion

Efficient similarity search requires a distance notion that obeys to certain properties: It must be a proper metric. We proposed a metric that allows comparing and searching process models with behavioral aspects in mind, based on the concept of behavioral profiles. These profiles are computed efficiently for a broad class of process models [11]. We explained that metric with a simple example.

The presented metric is our first attempt to investigate similarity of process models in terms of behavioral profiles. In future work, we shall address the metric, identify and rank further similarity coefficients, and construct a more sophisticated metric that is substantiated through exhaustive experiments, e.g., a regression analysis. The expressiveness of such a metric shall be compared to a reference model collection that has been evaluated by business process experts. Further, we will address the suitability of this improved metric in similarity search, as it is vital for a metric to be well discriminating in order to enable efficient searching with confident results.

References

1. W.M.P. van der Aalst. Workflow verification: Finding control-flow errors using petri-net-based techniques. In *BPM*, volume 1806 of *LNCS*, pages 161–183, 2000.
2. Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. Searching in Metric Spaces. *ACM Comput. Surv.*, 33(3):273–321, 2001.
3. Remco M. Dijkman, Marlon Dumas, and Luciano García-Bañuelos. Graph matching algorithms for business process model similarity search. In Umeshwar Dayal, Johann Eder, Jana Koehler, and Hajo A. Reijers, editors, *BPM*, volume 5701 of *Lecture Notes in Computer Science*, pages 48–63. Springer, 2009.
4. Marlon Dumas, Luciano García-Bañuelos, and Remco M. Dijkman. Similarity search of business process models. *IEEE Data Eng. Bull.*, 32(3):23–28, 2009.
5. Rik Eshuis and Paul W. P. J. Grefen. Structural matching of bpel processes. In *ECOWS*, pages 171–180. IEEE Computer Society, 2007.
6. Gisli R. Hjaltason and Hanan Samet. Index-driven similarity search in metric spaces (survey article). *ACM Trans. Database Syst.*, 28(4):517–580, 2003.
7. Matthias Kunze and Mathias Weske. Metric Trees for Efficient Similarity Search in Process Model Repositories. In *Proceedings of the 1st International Workshop on Process in the Large (IW-PL '10)*, Hoboken, NJ, September 2010.
8. Alan Lipkus. A Proof of the Triangle Inequality for the Tanimoto Distance. *Journal of Mathematical Chemistry*, 26:263–265, 1999. 10.1023/A:1019154432472.
9. Niels Lohmann, Eric Verbeek, and Remco M. Dijkman. Petri net transformations for business processes - a survey. *T. Petri Nets and Other Models of Concurrency*, 2:46–63, 2009.
10. Boudewijn van Dongen, Remco Dijkman, and Jan Mendling. Measuring Similarity between Business Process Models. In *Advanced Information Systems Engineering*, volume 5074 of *Lecture Notes in Computer Science*, pages 450–464. Springer Berlin / Heidelberg, 2008.
11. Matthias Weidlich, Jan Mendling, and Mathias Weske. Efficient consistency measurement based on behavioural profiles of process models. *IEEE Transactions on Software Engineering*, 2010. To appear.
12. Andreas Wombacher. Evaluation of technical measures for workflow similarity based on a pilot study. In Robert Meersman and Zahir Tari, editors, *OTM Conferences (1)*, volume 4275 of *Lecture Notes in Computer Science*, pages 255–272. Springer, 2006.
13. Zhiqiang Yan, Remco M. Dijkman, and Paul Grefen. Fast business process similarity search with feature-based similarity estimation. In Robert Meersman, Tharam S. Dillon, and Pilar Herrero, editors, *OTM Conferences (1)*, volume 6426 of *Lecture Notes in Computer Science*, pages 60–77. Springer, 2010.
14. Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity Search: The Metric Space Approach.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

Internal behavior reduction for partner synthesis

Niels Lohmann

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

Abstract. Communication is a unique feature of services and allows for reusing services in different compositions. To make a statement about the correctness of a service in isolation, *partner synthesis* is a proven technique. It overapproximates the service's behavior in any possible composition. Unfortunately, the complexity of partner synthesis is an order of magnitude higher than that of classical model checking techniques. This paper approaches this problem by tackling one source of complexity, namely *internal behavior* (also called silent or τ-transitions). By applying rules known from compositional verification, we reduce the internal behavior of a service while preserving its external behavior, viz. its communication protocol.

1 Introduction

Correctness plays an important role in service-oriented systems, as they increasingly realize business processes or other important infrastructures. Because failures of a single service may affect all other participants of the composition, thorough testing or verification is of paramount importance. In previous work [16], we argued that *partner synthesis* is not only an effective means to check the correctness of single services, but can also be used to synthesize communication skeletons, construct operating guidelines, generate test suites, realize interaction models, correct choreographies, configure business processes, or synthesize adapters. Further references can be found in a survey [10]. Thereby, a partner of a service is another service such that their composition is correct (e. g., deadlock-free, sound, or weakly terminating).

Conceptually, a partner is synthesized by first overapproximating the service's behavior in any possible composition and then removing undesired states yielding deadlocks or livelocks. Unfortunately, the complexity of the partner synthesis is exponential in the size of the service; that is, *both* in the number of states and the size of the interface. Even worse, the service's behavior itself can already suffer from the state space explosion problem [13], which makes the overall complexity of partner synthesis devastating.

As partner synthesis focuses on the external behavior of a service (i. e., its communication protocol), the internal behavior is only important when internal decisions are modeled. To this end, *this paper aims at reducing internal behavior of a service while preserving its external behavior.* As a result, we can construct partner services with reduced effort. We therefore sketch partner synthesis in the

next section. As one contribution, we also survey in Sect. 3 different approaches to leverage the complexity to classify our reduction. Section 4 provides the main contribution of this paper: We adjusted several state space reduction rules to reduce internal service behavior. To assess our approach, we implemented it and discuss in Sect. 5 some first experimental results and the impact of the reduction to the partner synthesis, before Sect. 6 concludes the paper.

2 Partner synthesis in a nutshell

We shall briefly sketch the partner synthesis approach of Wolf [16]. Given a service model, a partner cannot observe the service's state at runtime. Hence, the only information a partner can rely on are (1) the service model and its behavior (i. e., its state space), (2) its own actions from the past, and (3) the asynchronous messages it receives from the service or synchronizations with the service.

Consequently, a partner can only make a vague statement on the concrete state of a service, and can only guess a *set* of states (called *knowledge*) the service might be in. For instance, the initial knowledge of a partner consists of all states of the service it can reach without influence of the partner; that is, all states the service can reach by performing internal transitions (also called silent or τ-transitions) or by sending asynchronous messages to the partner. Each action of the partner results in changed knowledge. For instance, sending a message to the service may result in additionally enabled receiving transitions. After building all possible knowledges and removing "bad" knowledges; that is, knowledges which imply unwanted behavior of the composition (such as deadlocks or livelocks), the remaining graph (unless empty) can be used as a partner. The partner synthesis algorithm is implemented in two tools, Fiona [11] and Wendy [9], we shall discuss later.

3 Reduction techniques for partner synthesis

There are several aspects that yield in the high complexity of the partner synthesis approach. For some of theses aspects already exist approaches to leverage the associated complexity.

State space. On source of complexity is the size of the state space of the service. Each knowledge is a subset of the service's states. Since services often employ concurrency, already this state space may be exponentially in the size of the service model. This state explosion can be fought in different fashions, and these state space reduction approaches can be classified as follows.

One idea is to reduce the original model (e. g., the Petri net or WS-BPEL process) before the calculation of the state space. The most prominent example for such an *a priori reduction* are Petri net reduction rules [12]. Applied to service models, these rules already allow to remove some internal behavior. However, experiments with business process models [3] show that their effect does not hardly justifies the required calculation time.

Another idea is not to generate the complete state space, but only a smaller fragment of it. An example for such an *on-the-fly reduction* are partial order techniques, for instance CTL* preserving partial order reduction [4]. First experiments with such a technique implemented in the tool Fiona are promising.

Finally, state space reduction techniques can also be applied *a posteriori*; that is, after the full state space is built, but before the partner synthesis. Such reduction rules [5] were already employed to reduce a characterization of all livelock-freely interaction partners [17], but not for the partner synthesis itself. This shall be the contribution of this paper.

Knowledge. Once the state space is built, the number of knowledges is another source of complexity. Here, we face two problems. First, a lot of "bad" knowledges are generated, but later removed because they contain unwanted behavior. For instance, a service model may contain a deadlock which is only reached after following a certain communication protocol. To avoid such unnecessary calculation, static analysis techniques can be used to preprocess the state space and to avoiding the calculation of "bad" knowledges as early as possible [9].

Second, not all knowledges need to be calculated in case only the *existence* of a partner is relevant. Such a (possibly less permissive) partner is usually much smaller. Weinberg [14] presents several partner reduction rules which turn out to be very effective during the partner synthesis of industrial service models [9].

Representation. Finally, symbolic data structures such as binary decision diagrams [2] may help to represent the state space and the knowledges in a compact manner. Early experiments [6] show that this technique known from model checking is also very effective when applied during partner synthesis.

The presented reduction techniques are modular, although they cannot be arbitrarily mixed. For instance, the tool Fiona [11] generates the state space on the fly and implements partial order reduction techniques, partner reduction rules, and symbolic representation. A reimplementation of Fiona, Wendy [9], generates the complete state space a priori to perform static analysis to avoid the calculation of "bad" knowledges. It also implements partner reduction rules. A case study [9] demonstrates that Wendy clearly outperforms Fiona. As Wendy does not implement any state space reductions, we shall focus on this aspect in the remainder of the paper.

4 Reduction of internal behavior

As mentioned earlier, Petri net reduction techniques are not effective enough to fight the state space reduction. Furthermore, on-the-fly reduction techniques cannot be combined with the powerful preprocessing techniques that avoid the generation of "bad" knowledge. Consequently, we shall investigate how a posteriori reduction techniques can be combined with partner synthesis.

To reduce the complexity of partner synthesis, we follow one idea: *States of the service, that would always appear in the same knowledge, should be merged before*

(a) redundant parallel transitions (b) fusion of internal loops

(c) states linked by internal transition (1) (d) states linked by internal transition (2)

Fig. 1. Reduction rules

the actual partner synthesis. As sketched in Sect. 2, this reduction particularly affects internal transitions, because — by definition — knowledge consists of those states that can be reached without interaction with the partner. Our approach takes the state space of a service model as input and constructs a reduced state space such that both the original and the reduced state space yield the same generated partner.

To achieve this goal, we employed state space reduction rules from Juan et al. [5]. These rules were defined to preserve IOT failure equivalence [5], which is very closely related to the preservation of external behavior we are interested in. Figure 1 depicts four of these rules. Thereby, a dashed arc stands for an arbitrary number of transitions with an arbitrary label.

Redundant parallel transitions. As the environment cannot distinguish which transition was taken, and both transitions reach the same state s_2, one transition can be safely removed, see Fig. 1(a). Note that this rule is not restricted to τ labels, but can also be applied for arbitrary communication transitions.

Fusion of internal loops. The internal transitions between the two states s_1 and s_2 are not observable. When in either state, the other state remains reachable without influence of a partner. Consequently, both states can be merged to a new state s_{12}. The internal loop is replaced by a self-loop, see Fig. 1(b).

States linked by internal transition. s_1 and s_2 are linked by an internal transition. For each outgoing transition of s_1 exists an outgoing transition of s_2 with the same label that reaches the same state. As a result, s_2 does not restrict any behavior compared to s_1 and the states can be safely merged to a new state s_{12}, see Fig. 1(c).

Table 1. Experimental results: effect of reduction to internal behavior

service model	sizes before reduction states	trans.	τ trans.	sizes after reduction states	trans.	τ trans.	reduction time
Deliver goods	4,148	13,832	9,288	150	397	12	3 s
Car analysis	11,381	39,865	27,231	420	1,211	164	64 s
Identity card	14,569	71,332	66,500	25	37	0	108 s
Product order	14,990	50,193	34,159	504	1,458	135	104 s
SMTP protocol	26,667	110,065	80,137	23,381	99,304	70,646	2,101 s
Philosophers	92,206	427,312	113,023	19,683	98,415	0	7,236 s

The rules are defined on the behavior of a service model and may appear very technical when considered in isolation. Nevertheless, the first two rules may allow the application of other rules. As special case of the third rule is when s_1 has no outgoing transitions other than the internal transition, see Fig. 1(d). Only this restricted setting is covered by a Petri net reduction rule [12]. In general, state space reduction rules allow for more reduction, because they can be applied on a simpler model and do not need to take concurrency into account.

Juan et al. [5] present more rules, but we refrain from a discussion of all of them. For instance, several rules deal with initial states. Furthermore, no original rule was aware of final states which are important in the area of services to distinguish desired final states from deadlocks or to detect livelocks.

5 Experimental results: effect to partner synthesis

We implemented the reduction rules described in the prior section as a component of Wendy [9]. It takes a Petri net service model as input, calls LoLA [15] to generate a state space and then iteratively applies the reduction rules until a fixed point is reached; that is, no more rules can be applied. The output is a reduced state space which is then used during the partner synthesis. The integration of the reduction component is still in an early stage of development.

As a proof of concept, we analyzed several WS-BPEL services from a consulting company. Each process consists of around 40 WS-BPEL activities and models communication protocols and business processes of different industrial sectors. We translated the WS-BPEL processes into Petri nets using the compiler BPEL2oWFN implementing a feature-complete Petri net semantics [7]. Furthermore, the "Philosophers" service is an academic example.

Table 1 summarizes the results regarding the reduction: For most industrial models, nearly all internal transitions could be removed and the state space could be reduced dramatically. This is particularly important, because knowledges consist of subsets of these state spaces, so even a small reduction may have an exponential effect. The SMTP protocol shows, however, that the reduction is

Table 2. Experimental results: effect of reduction to partner synthesis

	synthesis without reduction			synthesis with reduction		
service model	knowledges	time	memory	knowledges	time	memory
Deliver goods	1,376	3 s	18 MB	1,376	0 s	3 MB
Car analysis	1,448	75 s	368 MB	1,176	2 s	13 MB
Identity card	1,536	88 s	427 MB	1,536	0 s	2 MB
Product order	57,996	299 s	1,467 MB	53,324	12 s	75 MB
SMTP protocol	13,456	210 s	249 MB	—[1]	—[1]	—[1]
Philosophers	481,646	4,098 s	6,078 MB	19,682	35 s	98 MB

not always effective. One reason might be that we have not implemented all applicable rules of Juan et al. [5] yet.

The partly devastating runtime can be explained by the prototypic status of the implementation. Nevertheless, the runtime of the reduction can be seen as a worthwhile investment, as shown by Table 2. We see that the synthesis times are usually much faster when the reduction is applied. Of course, we also need to take the reduction time into account. Nevertheless, only when analyzing the "Philosopher" model, the additional time does not pay off. Experiences from the implementation of Petri net reduction rules (i. e., parallel execution or index structures) may help to decrease the runtime by an order of magnitude.

More importantly, we can observe a dramatical reduction of around 95 % in the consumed memory. This allows us to synthesize partners for service models using a few megabytes rather than gigabytes.

6 Conclusion

Summary. In this paper, we discussed several sources of complexity of partner synthesis. We identified a large state space and in particular internal transitions as one reason partner synthesis might be intractable for larger service models. To tackle this problem, we presented a reduction technique that aims at reducing the internal behavior of service models. This technique is modular; that is, can be integrated in existing partner synthesis approaches. A prototypic integration into the partner synthesis tool Wendy [9] demonstrated principal effectiveness of the reduction. We observed a dramatic decrease in memory consumption which allowed us to apply partner synthesis to models we could not analyze before. This reduced memory reduction, however, is currently traded by a suboptimal runtime of the reduction.

The approach has another advantage: it is *compositional*. Suppose the state space of the net is too large to be calculated. As this calculation is a prerequisite for the synthesis algorithm, no partner could be computed. The rules, however, allow for a compositional approach. That is, we can (1) divide the net into parts, (2) apply the reduction rules to the state space of each part, and (3) compose

[1] We currently face a software bug when analyzing the reduced SMTP protocol model.

reduced state spaces. The interested reader is referred to [5] for a detailed discussion.

Lessons learnt. In retrospective, the results of this paper seem obvious and the approach straightforward. However, two questions were open in the run-up of this paper: First, little experimental results were published on the practical applicability of the reduction rules from Juan et al. [5] and their effectiveness to real-life service models. Second, the exact effect of the reduction to partner synthesis was unclear. In particular, we did not foresee that reduced internal behavior could have such a positive effect on the memory consumption. Also the fact that already four reduction rules have such an effect was unclear. The "Philosophers" model further showed that much fewer knowledges need to be calculated when synthesizing partners.

We would like to point out that only a prototypic implementation and access to realistic service models allowed us to perform experiments and to answer these questions. Thereby, the modular architecture of the partner synthesis tool Wendy facilitated the integration of the reduction rules to the partner synthesis. These experiences follow the observations we described in a recent survey [10]. Both the tool Wendy [9] and the experimental results are available via the Web site `http://service-technology.org/live` [8].

Future work. In future work, several open issues need to be approached. As already pointed out, we need to improve the efficiency of the rule application. From a conceptual point of view, a combination of the presented approach and partial order reduction techniques would be promising. Partial order reduction aims at avoiding the state space explosion by not enumerating all possible orders of transitions and intermediate states. This usually results in very small and also simpler structured state spaces. This in turn should boost the applicability of the reduction rules. Finally, a look at related rules [1] may allow for further reduction.

Acknowledgments. The author thanks Christian Stahl for his feedback on an earlier version of this paper and for pointing out the compositionality aspect.

References

1. Aalst, W.M.P.v.d., Lohmann, N., Massuthe, P., Stahl, C., Wolf, K.: From public views to private views — correctness-by-design for services. In: WS-FM 2007. pp. 139–153. LNCS 4937, Springer (2008)
2. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. IEEE Trans. Computers C-35(8), 677–691 (1986)
3. Fahland, D., Favre, C., Jobstmann, B., Koehler, J., Lohmann, N., Völzer, H., Wolf, K.: Instantaneous soundness checking of industrial business process models. In: BPM 2009. pp. 278–293. LNCS 5701, Springer (2009)
4. Gerth, R., Kuiper, R., Peled, D., Penczek, W.: A partial order approach to branching time logic model checking. Inf. Comput. 150(2), 132–152 (1999)

5. Juan, E.Y.T., Tsai, J.J.P., Murata, T.: Compositional verification of concurrent systems using Petri-net-based condensation rules. ACM Trans. Program. Lang. Syst. 20(5), 917–979 (1998)
6. Kaschner, K., Massuthe, P., Wolf, K.: Symbolic representation of operating guidelines for services. Petri Net Newsletter 72, 21–28 (2007)
7. Lohmann, N.: A feature-complete Petri net semantics for WS-BPEL 2.0. In: WS-FM 2007. pp. 77–91. LNCS 4937, Springer (2008)
8. Lohmann, N.: service-technology.org/live – replaying tool experiments in a Web browser. In: BPM Demos 2010. pp. 64–68. CEUR Workshop Proceedings 615, CEUR-WS.org (2010)
9. Lohmann, N., Weinberg, D.: Wendy: A tool to synthesize partners for services. In: PETRI NETS 2010. pp. 297–307. LNCS 6128, Springer (2010), tool available at `http://service-technology.org/wendy`.
10. Lohmann, N., Wolf, K.: How to implement a theory of correctness in the area of business processes and services. In: BPM 2010. pp. 61–77. LNCS 6336, Springer (2010)
11. Massuthe, P., Weinberg, D.: Fiona: A tool to analyze interacting open nets. In: AWPN 2008. pp. 99–104. CEUR Workshop Proceedings Vol. 380, CEUR-WS.org (2008), tool available at `http://service-technology.org/fiona`.
12. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77(4), 541–580 (1989)
13. Valmari, A.: The state explosion problem. In: Advanced Course on Petri Nets. pp. 429–528. LNCS 1491, Springer (1996)
14. Weinberg, D.: Efficient controllability analysis of open nets. In: WS-FM 2008. pp. 224–239. LNCS 5387, Springer (2009)
15. Wolf, K.: Generating Petri net state spaces. In: PETRI NETS 2007. pp. 29–42. LNCS 4546, Springer (2007), tool available at `http://service-technology.org/lola`.
16. Wolf, K.: Does my service have partners? LNCS ToPNoC 5460(II), 152–171 (2009)
17. Wolf, K., Stahl, C., Ott, J., Danitz, R.: Verifying livelock freedom in an SOA scenario. In: ACSD 2009. pp. 168–177. IEEE Computer Society (2009)

A Data-Centric Approach to Deadlock Elimination in Business Processes

Christoph Wagner

Institut für Informatik, Humboldt Universität zu Berlin,
Unter den Linden 6, 10099 Berlin, Germany
cwagner@informatik.hu-berlin.de

Abstract. In this paper, we sketch a data-centric approach to avoid deadlocks of a business process. If dependencies between data values are neglected or modelled incorrectly, this can lead to errors in the control flow of the business process. We address the problem of detecting deadlocks which are caused by the improper handling of data. We show by example, how these deadlocks can be detected by means of a symbolic reachability graph. Under certain conditions, we can derive the correct dependency between the involved data values. This allows to modify the business process in a way so that the detected deadlocks will not be reachable.

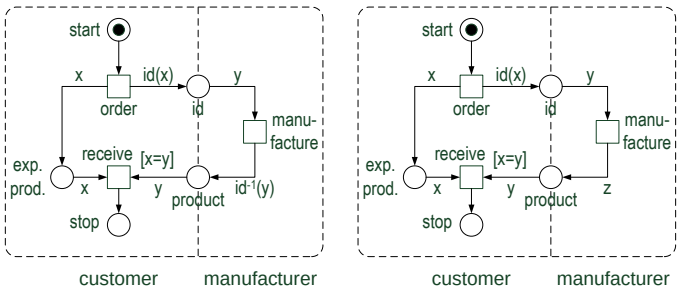
1 Background

The design of business processes is an error prone task. This motivates the use of formal verification to help a business process designer to avoid certain kinds of errors. Models of business processes can be transformed into more formal models like process algebra [4] or Petri nets [9,7]. A business process can also be designed as a Petri net directly (e. g. with CPN Tools [10]). These models can be checked for soundness and other properties by means of formal verification.

However, little attention has been paid to the influence of *data* on the correctness of a business process. Most formal models represent data only in a highly abstracted and imprecise form. Models that explicitly include data usually have a clear separation between the control flow part and the data part [3]. Often, the dependencies between data flow and control flow are not very complex, i. e. there is only a small set of values a data item can have. For many the properties in focus of recent research, the actual value of a data item is not important [12], [13] and primarily concerns the order in which read and write activities on variables are carried out. E. g., reading an uninitialized value is considered an error.

In this paper, we consider processes that are heavily influenced by data and might be unable to finish a task for some combinations of data values. Technically, this means that a *deadlock* is reachable under some conditions. Our goal is to find out those harmful combinations of data values and describe the relation the value must adhere to in order to avoid a deadlock (e. g. in the form of a function) and use this information to fix the process. Concerning this aspect, our approach is more general than the approach of [1] which does not deal with relations between data values.

We represent a business process by a *High-Level Petri net* [5]. A High-Level Petri net is an extension of a Petri net where places are typed, tokens have values of the respective type and arcs are inscribed with terms. When a transition fires, values are assigned to the variables appearing in the inscriptions of adjacent arcs. The evaluation of the inscriptions determines which values are produced and consumed. The terms are evaluated with a fixed interpretation (note that the Petri net is not a *schema* in the sense of [11]). We do not exploit restrictions on the Petri net's structure that a certain business process modelling language might impose. This allows us to handle more general models (e. g. as obtained from CPN Tools). We assume that the Petri net is bounded, acyclic and fulfils some technical requirements of minor importance which do not restrict the expressivity and will not be mentioned here. The restriction to acyclic nets can be relaxed as long as computational issues are neglected. We assume that the set of values used by the Petri net can be so large that an explicit enumeration would be computationally inefficient.



(a) Correct business process (b) Incorrect business process

Fig. 1: Business process consisting of a customer an a manufacturer, represented as a Petri net

Consider a business process formed by a customer and a manufacturer (Fig. 1a). The customer orders a product x from the manufacturer by telling the manufacturer the product's id (**order**). The manufacturer assembles the product associated with the id and returns the product to the customer (**manufacture**). Let us now assume that due to a design error in the manufacturer's internal workflow, the id y obtained from the customer is lost and replaced by the id of an arbitrary product z (Fig. 1b). Then, the customer will get a product different from the one he expects. In that case, no further action (**receive**) can be performed because the condition $x = y$ is not satisfied and we reach a *deadlock*. We call this a *conditional deadlock*, because it occurs only if the compared data values are not equal. Note that in a more complex scenario, a choice dependent on data may not always lead to a deadlock instantly but later on in the process. In that

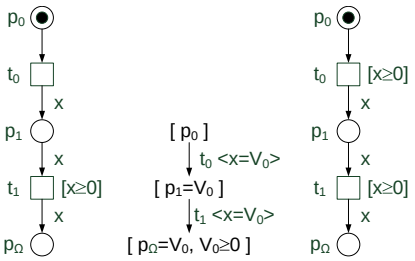
case, the deadlock condition has to be propagated backwards. This aspect will not be covered in this paper.

We introduce an approach to *detect* conditional deadlocks and to *derive* the dependency between values that must be used in order to *avoid* the deadlock. In Sect. 2, we illustrate by simple examples, how to identify deadlocks by means of a *symbolic reachability graph*. Section 3 shows by a more sophisticated example, how to derive the precise conditions under which an conditional deadlock can be avoided. It is not always possible to derive these conditions precisely. Section 4 shows an example that can not be corrected with our approach due to imprecise results. In Sect.5, we conclude our work.

2 Basic Idea

In this section, we show how to detect a deadlock by means of the *symbolic reachability graph* (SRG) of the Petri net. In the following examples, we assume that every place has the type integer except for some places that carry tokens (denoted as black dots) that do not have a value. A marking m of a Petri net N is considered a *deadlock*, if no transition is enabled and m is not contained in a set Ω of *final markings* of N.

In each of the following examples, we assume that the net is in a final marking exactly when the place p_Ω is marked. Consider the net N_1 in Fig. 2a. We can easily see that N_1 is not deadlock free and $M = \{[p_1 = n] | n \in \mathbb{Z}, n < 0\}$ is the set of deadlocks reachable in N_1. Obviously, by adding the guard $x \geq 0$ to t_0, we can ensure that t_1 will always be enabled and N_1 will eventually reach the final marking $[p_\Omega]$ (Fig. 2c).

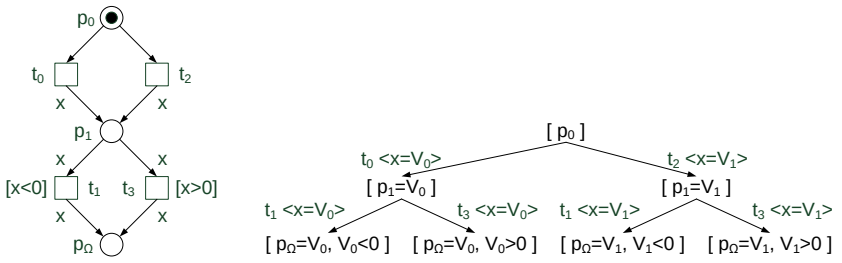


(a) N_1 (b) $SRG(N_1)$ (c) N_1'

Fig. 2: Net with a conditional deadlock and its correction

In a marking of the symbolic reachability graph, every value is represented by a term. Without going into technical details, we show how to construct the symbolic reachability graph of N_1. Starting from initial marking $[p_0]$, t_1 produces an arbitrary integer on p_1 (Fig. 2b). We represent this integer by a unique identifier V_0. Thus we get the marking $[p_1 = V_0]$. While formally V_0 is a constant, we treat V_0 as a variable: V_0 may later be *instantiated* by any value from $\mathbb{Z}$. Since t_1 is enabled only if V_0 is non-negative, we keep the condition in $V_0 \geq 0$ in the successor marking $[p_2 = V_0, V_0 \geq 0]$ of $[p_1 = V_0]$ (we obtain the condition by combining the firing mode of t_1 and the guard of t_1). We consider an instance of a marking of the symbolic reachability graph *valid*, if every condition denoted in the marking evaluates to *true*. Obviously, a marking of N_1 is reachable exactly if it is a valid instance of a marking of the symbolic reachability graph.

With the symbolic reachability graph, we can identify under which condition a marking instantiates to a deadlock. Here, $[p_1 = V_0]$ is a *conditional deadlock* for $\neg V_0 \geq 0$ since each instance of $[p_1 = V_0]$ has no successor for this condition. We now *enforce* that the condition $V_0 \geq 0$ holds in $[p_1 = V_0]$ by adding the guard $x \geq 0$ to the predecessor-transition t_0 of $[p_1 = V_0]$. We obtain a corrected version N_1' of N_1, which is deadlock free.



(a) N_2 (b) $SRG(N_2)$

Fig. 3: Net with a branching symbolic reachability graph

Fig. 3a shows a net N_2 which is not deadlock free and which has a branching symbolic reachability graph. Note that the values produced by t_0 and t_2 obtain different identifiers (although both branches behave symmetrically). $[p_1 = V_0]$ and $[p_1 = V_1]$ are conditional deadlocks of N_2. $[p_1 = V_0]$ is a deadlock for $\neg((V_0 < 0) \vee (V_0 > 0))$. Any successor of an instance of $[p_1 = V_0]$ belongs either to the branch with condition $V_0 < 0$ or the branch with condition $V_0 > 0$. We enforce the condition $(V_0 < 0) \vee (V_0 > 0)$ by adding the guard $(x < 0) \vee (x > 0)$ to t_0 (which is effectively equivalent to $x \neq 0$). By repeating this procedure for $[p_1 = V_1]$, we get the same guard for t_2 and obtain a deadlock free net.

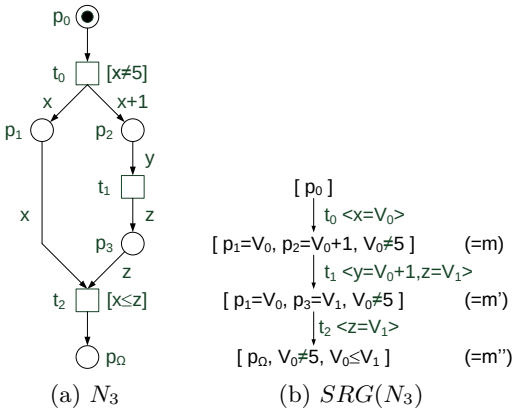
3 Derivation of a deadlock-preventing guard

In the previous section, we have shown that by adding an appropriate guard, we can prevent the reachability of deadlocks. Adding a guard or replacing a guard by a more restrictive one makes the net less *permissive*, that is the set of reachable markings gets smaller. Naturally, we want to prevent all deadlocks from being reachable. On the other hand, we do not want to prevent the reachability of more markings than necessary. This section addresses the issue of deriving a least restrictive guard. How to derive a guard (like $x > 0$) from a condition (like $V_0 > 0$) is not obvious if more than one variable is involved and functions are used in the arc inscriptions.

Without loss of generality, we assume that the symbolic reachability graph is a tree (if not, we can unfold it). Note that the symbolic reachability graph usually has an acyclic structure since names of value identifiers never repeat. In

the tree, we always modify the guard of the transition that directly precedes the deadlock. It should be mentioned here, that due to restrictions inherent to the modelled the business process (e. g. the dependency on of external events which can not be influenced), it might not be possible to modify that transition. In that case, we choose the first modifiable predecessor transition in the tree. As guard derivation is more involved in that case, it will not be shown here. Consider the net N_3 in Fig. 4a, which reaches a deadlock if the integer produced by t_1 on p_3 is greater than the integer produced by t_0 on p_1. An ad-hoc way to fix N_3 is to add the guard $z \geq y$ to t_1. Then, N_3 eventually reaches the final marking $[p_\Omega]$. However, the guard $z \geq y - 1$ would also ensure that N_3 reaches $[p_\Omega]$ but is less *restrictive* than $x \geq y$, since it evaluates to true for more assignments of x and y. The guard $y = 6 \vee z \geq y - 1$ is even less restrictive than $z \geq y - 1$.

We derive this guard from the symbolic reachability graph in Fig. 4b. m' is a deadlock for condition $\neg(V_0 \leq V_1)$ since every valid instance of m'' satisfies $V_0 \leq V_1$. We prevent the reachability of m' under condition $\neg(V_0 \leq V_1)$ by adding a guard to t_1. It is sufficient that the guard forbids the violation of $V_0 \leq V_1$ only for valid instances of m and m'. So we can assume that for a step $m \xrightarrow{a} m'$ (where $a = t_1\langle y = V_0 + 1, z = V_1\rangle$) with integers V_0 and V_1 given
(1) the condition $V_0 \neq 5$ already holds (precondition in m)
(2) y and z are bound to the valuations of $V_0 + 1$ and V_1 (firing mode of t_1).



(a) N_3 (b) $SRG(N_3)$

Fig. 4: A net that is less obvious to correct

This motivates the definition of the expression

$$\forall V_0, V_1 \in \mathbb{Z} : V_0 \neq 5 \wedge y = V_0 + 1 \wedge z = V_1 \implies V_0 \leq V_1$$

We call this expression *least restrictive $V_0 \leq V_1$-enforcing (for step $m \xrightarrow{a} m'$)*. Note that the more preconditions an expression has, the less restrictive it is. V_0 and V_1 are all-quantified because the condition $V_0 \leq V_1$ shall be enforced for *any* valid instance of m'. It is easy to see that this expression is indeed equivalent to $y = 6 \vee z \geq y - 1$. Since V_0, V_1 are uniquely determined by y and z, we can replace V_0 by $y - 1$ and V_1 by z, thus eliminating V_0, V_1 from the expression.

We go back to the business process introduced in Fig. 1a. The reader may believe that there is a deadlock $m' = [\text{exp. prod.} = V_0, \text{product} = V_1]$ for condition $\neg(V_0 = V_1)$ which is reachable from $m = [\text{exp. prod.} = V_0, id(V_0)]$ via transition **manufacture**. This leads to the expression $\forall V_0, V_1 \in \mathbb{Z} : y =$

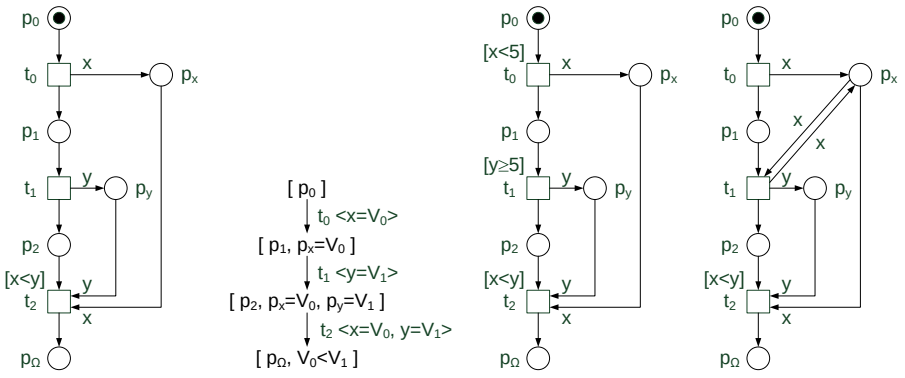
$id(V_0) \wedge z = V_1 \implies V_0 = V_1$, which is equivalent to $z = id^{-1}(y)$. Thus we have reconstructed the dependency between products and their id's and may correct the business process by replacing z by $id^{-1}(y)$.

In general, several successive deadlock elimination steps are necessary in order to obtain a deadlock free net, as every step may introduce new deadlocks. Our approach is similar to Dijkstra's method to derive the weakest precondition for which a given program terminates in a specified state [2]. However, our approach allows to derive a modification even for an intermediate step because a precondition and a postcondition are already given. Therefore, we may perform modifications in a local manner and do not have to start at the final markings. Conceptually, the approach is applicable even if no final marking is specified at all.

Note that modifying a transition may have non-local side-effects if the transition appears more than once in the tree. In that case, more markings are rendered unreachable than intended, leading to a suboptimal solution. As the next section shows that even without non-local side-effects, it is not always possible to get an optimal solution.

4 Uncorrectable net

Some nets can not be corrected using the expression derived in the last section. Consider the net N_4 in Fig. 5a. There is no *unique* way to avoid the deadlocks of



(a) N_4 (b) $SRG(N_4)$ (c) N_4' (d) N_4''

Fig. 5: A net and two possible corrections

N_4. For example, we get a deadlock free net N_4' (Fig. 5c) by adding the guard $x < 5$ to t_0 and $y \geq 5$ to t_1. However, instead of 5, we could have chosen any other integer. The symbolic reachability graph does not give us a hint on how to derive the general structure of the two guards. Here, the expression enforcing

$V_0 < V_1$ for $m \overset{t_1\langle y=V_1\rangle}{\rightarrow} m'$ (with $m = [p_2 = V_0, p_1], m' = [p_2 = V_0, p_3 = V_1]$) gives

$$\forall V_0, V_1 \in \mathbb{Z} : y = V_1 \implies V_0 < V_1$$

which evaluates to false for every $y \in \mathbb{Z}$. The expression is too restrictive due to a *lack of information*. The condition $V_0 \leq V_1$ that shall be enforced depends on both the values of V_0 and V_1, but t_1 has no access to the place p_x on which V_1 is stored. For the slightly different net N_4'' (Fig. 5d) in which t_1 has access to both values, an appropriate guard for t_1 can be derived: The expression enforcing $V_0 < V_1$ gives $\forall V_0, V_1 \in \mathbb{Z} : x = V_0 \wedge y = V_1 \implies V_0 < V_1$, which is equivalent to $x < y$.

A related phenomenon is known from controller synthesis. A controller forbids the supervised system to perform some actions in certain situations. The guards we add to a transition have an impact on the net comparable to a controller. If certain sets of states are indistinguishable for the controller, then there is no *unique* maximal permissive controller [6]. From the point of view of t_1, all markings that differ only on place p_x are indistinguishable.

5 Conclusion and Future work

The detection and correction of errors in a business process is a tedious task. Petri nets provide of formal foundation to apply formal verification on business processes. We have shown how to identify and avoid deadlocks of a High-Level Petri net by means of a symbolic reachability graph. As a byproduct, our approach allows to formulate the dependencies between data values that must hold in order to avoid a deadlock.

Our goal is to apply our approach in a distributed setting. Therefore, several problems have to be considered. Since one part of a business process usually does not have complete information about the state of the other parts, problems caused by a lack of information as described in Sect. 4 are more likely to occur. It is also less likely that a deadlock can be fixed locally. Especially in the presence of cycles, non-local side-effects occur inevitably.

In contrast to [8], which proposes a correction algorithm for services (but ignores the data issue), we can not add elements to the structure of the net. Having net N_4 from Fig. 5a in mind, this imposes a strong restriction on the applicability of our approach. We believe that our approach can still provide valuable hints for a business process designer. The designer may first design the business process model with the help of algorithms which do not take data into account but are more precise on the structure. After the general design of the business can be considered correct, our approach can be used to find small errors that occur only for very special combinations of values. In that case, the modification proposed by our approach might be precise enough to provide a reasonable correction. We also believe that our approach will produce useful results if the service that is modified has a very canonical structure (e. g., acyclic or even tree-like structure). We hope to gain valuable insights regarding the synthesis of a service that can communicate deadlock-freely with a given service.

References

1. Awad, A., Decker, G., Lohmann, N.: Diagnosing and repairing data anomalies in process models. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (eds.) Business Process Management Workshops, BPM 2009. Lecture Notes in Business Information Processing, vol. 43, pp. 5–16. Springer-Verlag (Mar 2010)
2. Dijkstra, E.W.: A Discipline of Programming. Prentice Hall, Inc. (October 1976)
3. Fan, S., Dou, W., Chen, J.: Dual Workflow Nets: Mixed Control/Data-Flow Representation for Workflow Modeling and Verification. pp. 433–444 (2007)
4. Ferrara, A.: Web services: a process algebra approach. In: Proceedings of the 2nd international conference on Service oriented computing. pp. 242–251. ICSOC '04, ACM, New York, NY, USA (2004)
5. Jensen, K.: Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use (Volume 1), EATCS Series, vol. 1. Springer Verlag (April 1992)
6. Kalyon, G., Le Gall, T., Marchand, H., Massart, T.: Control of Infinite Symbolic Transition Systems under Partial Observation. In: European Control Conference. Budapest Hungary (Aug 2009)
7. Lohmann, N.: A feature-complete Petri net semantics for WS-BPEL 2.0 and its compiler BPEL2oWFN. Informatik-Berichte 212, Humboldt-Universität zu Berlin (Aug 2007)
8. Lohmann, N.: Correcting deadlocking service choreographies using a simulation-based graph edit distance. In: BPM 2008. LNCS, Springer-Verlag (Sep 2008)
9. Lohmann, N., Verbeek, E., Dijkman, R.: Petri net transformations for business processes – A survey. In: Jensen, K., van der Aalst, W. (eds.) Transactions on Petri Nets and Other Models of Concurrency II, Lecture Notes in Computer Science, vol. 5460, pp. 46–63. Springer Berlin / Heidelberg (2009)
10. Ratzer, A.V., Wells, L., Lassen, H.M., Laursen, M., Qvortrup, J.F., Stissing, M.S., Westergaard, M., Christensen, S., Jensen, K.: CPN Tools for editing, simulating, and analysing coloured petri nets. In: van der Aalst, W.M.P., Best, E. (eds.) ICATPN. Lecture Notes in Computer Science, vol. 2679, pp. 450–462. Springer (2003)
11. Reisig, W.: On the Expressive Power of Petri Net Schemata. In: ICATPN 2005, Miami, USA. Proceedings. Lecture Notes in Computer Science, vol. 3536, pp. 349–364. Springer Verlag (May 2005)
12. Sun, S.X., Zhao, J.L., Nunamaker, J.F., Sheng, O.R.L.: Formulating the data-flow perspective for business process management. Information Systems Research 17(4), 374–391 (2006)
13. Trčka, N., Sidorova, N.: Data-flow anti-patterns: Discovering data-flow errors in workflows. In: CAiSE 2009. LNCS 5565. p. 425. Springer (2009)

Streamlining Pattern Support Assessment for Service Composition Languages

Jörg Lenhard, Andreas Schönberger, and Guido Wirtz

Distributed and Mobile Systems Group, University of Bamberg, Germany
{joerg.lenhard,andreas.schoenberger,guido.wirtz}@uni-bamberg.de

Abstract. Various process modeling formalisms have been leveraged to specify service compositions. For assessing the expressiveness of similar languages and for providing best practice knowledge, patterns have frequently been proposed. However, the pattern catalogs proposed do not all share and document the criteria that were used for assessing pattern support. Furthermore, the scaling of the support measure frequently is very coarse, only providing a basic level of selectivity. This paper proposes an approach that allows for measuring the pattern support for different catalogs in a uniform manner. The selectivity of the support measure is improved by using the edit distance for calculating its degree. The feasibility of the approach is shown by preliminary results of the analysis of selected patterns and orchestration languages.

Keywords: SOA, Pattern, Service Composition Language, Edit Distance, Orchestration

1 Introduction

A powerful property of *service-oriented architectures* (SOAs) is the service composition layer [9]. This layer covers the construction of composite services from other services which is often achieved by combining calls to existing services in a process-based manner. This essentially involves the definition of control- and data-flow dependencies between the different service invocations. Representations of process-based service composition languages are choreography and orchestration languages [10]. Traditional notions such as Turing-completeness are inappropriate for capturing the suitability of service composition languages. In the area of workflow systems, describing reasonable aspects of languages in the form of *patterns* and analyzing existing languages for their support for those patterns was proposed. This approach was initiated by the *workflow patterns initiative* [16] and was widely used by product vendors and scientific research from its start on. Today, many different pattern catalogs are available. However, a study that analyzes a language using multiple pattern catalogs faces several problems. Differences among the various publications according to what constitutes which level of support limit comparability. In fact, most authors use different notions of what counts as support and also do not document clearly what criteria need

to be fulfilled by a candidate solution to offer support for a pattern. This way, the degree of support determined sometimes seems to be based on personal bias.

The intent of the support measure is to describe how directly or easily a pattern can be implemented in a language using built-in constructs. It does generally *not* state whether or not a pattern can be implemented in a language at all. The degree of support states to what extent the user of a language is aided by the constructs directly available by or built into the language. Its scaling typically is trivalent (or in some cases such as [8, 18] quadrivalent) and distinguishes whether a solution provides *direct* (+), *partial* (+/-) or *no direct support* (-) for a pattern ([16], p. 50), based on the amount of constructs needed in a solution. Constructs are the core building-blocks of a language, such as a decision activity or a fork activity. Adjacent concepts, such as variables or correlation sets do generally not count as constructs. Usually, a candidate solution that uses only a single construct provides direct support. A combination of two constructs results in partial support and if more than two constructs are needed no direct support is provided. This trivalent degree can be too coarse. For example, consider the case where a pattern is directly supported in two languages by a single construct. In language A, the single construct can be used in a straight-forward manner and the solution to the pattern is complete. In language B, the single construct needs to be used and a complex configuration of the construct is necessary, consisting of, say, three changes to the default values of its attributes which may be interdependent on each other. Furthermore, the creation of a variable in the process model is also needed. Obviously, the solution in language A is more direct than the solution in language B. Nevertheless, they are equal concerning their degree of support.

This paper tackles these problems of comparability and selectivity by proposing a unified approach for determining the degree of support a given solution provides for a pattern. This approach is derived from the different methodologies used by the authors of relevant catalogs. It works in two steps:

1. For a given candidate solution, it is first determined whether it provides a *valid implementation* for a given pattern.
2. If so, the degree of support it provides is calculated. This calculation is done using an alternative scaling of the support measure, the *edit distance* based on high level change operations, to enhance the selectivity of the results.

The following section briefly describes relevant pattern catalogs and related analyses. Section 3 outlines the proposed approach, followed by preliminary results for the support of two orchestration languages for selected patterns in Sect. 4. Section 5 concludes.

2 Related Work

The workflow patterns initiative published several pattern catalogs, most notably the *control-flow patterns* [13, 16]. Other aspects of workflows are covered by the *data patterns* [12] and the *resource patterns* [11]. [14] also presents mechanisms

for *exception handling* in the form of a pattern catalog. The *service interaction patterns* [2] were the first pattern catalog which is specific for languages focusing on service-based processes and describe common interaction scenarios. [1] followed this catalog with a set of patterns that capture *correlation* mechanisms. [17, 18] consider patterns for dealing with *changes* to processes in *process-aware information systems*. Like the service interaction patterns, [15] describes functions that are common to business processes in the form of *activity patterns*. [3] addresses ways in which process instances can be created in the form of *process instantiation patterns*. Recently, also *time patterns* [7, 8] for analyzing the support for time constraints in a language have been proposed.

With the exception of the activity patterns, all these publications of pattern catalogs do also provide an analysis of the support of selected languages for the patterns of the catalog. They value the validity of possible solutions using criteria specific for the catalog. [1,3,7,8,13,16–18] do present a tri- or quadrivalent scaling of the support measure. [2, 14] simply state whether a pattern can be realized at all. The edit distance presented here relates to the *graph-edit distance* [5]. This edit distance is used in [17] based on editor operations for demonstrating the necessity for the support for adaptation patterns. Here, we use a more specific set of edit operations based on the structure of a language to measure the degree of pattern support.

Based on the pattern catalogs, there are also a variety of studies that perform additional analyses. In the area of orchestration languages, the *Web Services Business Process Execution Language* (BPEL) is analyzed in the context of the control-flow, data, resource, service interaction, correlation and process instantiation patterns (cf. above). [4, 19] also compare the expressiveness of BPEL to other Web Services composition languages. Both of the studies use several pattern catalogs. An alternative orchestration language, *Windows Workflow* (WF), is examined for its support of control-flow patterns in [21] to provide initial insights into its control-flow expressiveness and is compared to BPEL. This paper proposes a two-step approach to improve such analyses comprising multiple pattern catalogs which increases comparability and provides a higher level of selectivity.

3 Approach

The first step of the approach is to determine whether a given candidate solution forms a valid implementation of a given pattern. Only a solution that fulfills this minimal criterion is able to provide support for a pattern. The decision whether this is the case is based on the structure and components of a pattern which are similar for all pattern catalogs at hand, although not all of the catalogs contain all of the aspects discussed below. Five components of a pattern are essential for determining the validity of an implementation.

Pattern description: The pattern description specifies the nature of the pattern and its *core aspects*. To provide a valid implementation, a candidate solution must cover all core aspects that are found in the pattern description,

as explicitly stated in [2]. This minimum component can be found in any pattern catalog.

Pattern context: The *context*, in some cases called *issues* [15], describes several assumptions or criteria about the environment in which a pattern operates. To provide a valid implementation, at most one of these criteria may not be met by a candidate solution. This pays tribute to the fact that the support for a pattern should still be calculated even if minor aspects cannot be covered. These constraints can be inferred from the evaluation criteria of [11–13]. As an example, the *Structured Synchronizing Merge* pattern requires the existence of preceding *Multi-Choice* construct in a context criterion ([13], pp. 17 - 19). Context criteria can be found in [2, 8, 11–13, 15, 16].

Execution traces: Closely related to the pattern context is the notion of *execution traces* ([8], p. 98). An execution trace defines the structure of all possible execution sequences of activities that are valid for a given pattern. Examples are mathematical expressions, used in [1, 8, 17], or graphical notations such as Petri Nets, used in [13, 15]. If a formalization for execution traces is present, a candidate solution must also conform to these traces which is explicitly stated in [8, 17].

Design choices: In most cases, the definition of a pattern is flexible to some extent. Certain aspects are left open to the choice of the implementer of a pattern, which are described as *design choices* ([8], p. 97). Each design choice denotes a list of alternative aspects one of which can be chosen when implementing a pattern. A combination of different aspects from the design choices attached to a pattern then forms a *pattern variant* ([8], p. 97). A candidate solution must implement at least one pattern variant (cf. [8, 18]), omitting at most one of the design choices of the variant. As an example, a solution for the *Durations* pattern that supports only maximum, but not minimum durations of activities still forms a valid implementation of the pattern ([8], p. 100). Design choices can be found in [2, 8, 15, 18].

Data types: A pattern might inherently depend on the availability of specific data types, such as dates or timestamps [1, 7, 8]. To provide support for a pattern, a corresponding data type must be available in a language. Additionally, if needed in a candidate solution, necessary operations for comparing or manipulating these types must be provided, as can be found in the evaluation contained in [7].

For a candidate solution that provides a valid implementation, the degree of support can be calculated. As shown in Sect. 1, the traditional trivalent degree can be too coarse. This situation can be improved by relying on an alternative measure. The problem of qualifying the effort needed to realize a pattern is basically a question of *distance* between processes. Say process X is a process stub without specific functionality and process Y is an extension of X that adds exactly the solution of a pattern. The less distant X is to Y, the less effort is needed to transform X into Y. So, the support for a pattern in a language can also be measured by computing the distance between two processes written in the language, where one of the processes extends the other one with

the implementation of a given pattern. Listing 1 outlines such a process stub for BPEL. It contains necessary import definitions and the definition of one `partnerLink` which is inevitable for a working process. The control-flow of the minimal process is formed by a `receive` activity that creates a new process instance and uses a `variable` as input embedded in a `sequence` activity. The pattern implementation then succeeds the `receive` activity.

Listing 1. Process stub for BPEL

```
<process>
   <import location="ProcessInterface.wsdl" />
   <partnerLinks>
       <partnerLink name="MyPartnerLink" myRole="patternRole" />
   </partnerLinks>
   <variables>
       <variable name="StartProcessInput"/>
   </variables>
   <sequence>
        <receive createInstance="yes" variable="StartProcessInput"
         partnerLink="MyPartnerLink" operation="StartProcess"/>
         <!--Pattern Implementation-->
   </sequence>
</process>
```

[20] presents several measures for computing the similarity between process models. A foundation for these similarity measures that seems very applicable for the problem at hand is the *edit distance*. This distance measures the smallest distance between two strings by calculating the minimum number of change operations, being substitutions, insertions or deletions of characters that are needed to transform one string into another. For the problem at hand, the basis are of course process models and not strings. The models to be compared are a process stub, as demonstrated in List. 1 and a process extending this stub with the implementation of a pattern. Counting substitutions of characters would make no sense here, as the distance in concepts and constructs would get lost in syntactical noise. For example a language could tend to have higher distances simply because its activities have longer names. Much more applicable in this case are high level changes to the structure of the process model, as opposed to changes of characters. The difference is that high level changes comprise larger structures and satisfy minimalistic semantical constraints. Examples are the insertion of an activity and the setting of its name, the insertion of a variable and the setting of its name and type or the setting of a target variable and expression in an assignment. A concrete example for BPEL would be the configuration of correlation for a `receive` activity. This involves the creation of a `correlations` and a `correlation` element, the setting of its name and potentially whether the correlation set should be initiated. Counting each syntactical modification, instead of the single high level operation *add correlation to receive*, adds noise to the final result. The intent of the edit distance here is after all not to capture differences in naming, but differences in concepts and constructs, because these differences better describe the effort needed by the user of a language. The edit distance can now be calculated by adding up the amount of such high level insertions, substitutions and deletions needed. Using the same set of high level changes

as basis for the edit distance in the assessment of different catalogs ensures comparability between the results. Generalizing the set of high level changes and making it applicable for different languages also provides comparability between the languages, even for different pattern catalogs.

Obviously, such edit operations can be facilitated by using a sophisticated integrated development environment. The aim of this study however, is to measure the support provided by a language and not by tools available for the language. The edit distance as discussed here abstracts from the availability of specific tools that facilitate edit operations. The same applies to the representation of the language [6]. The identification of constructs that add to the edit distance cannot easily be automated by relying on the syntactical elements of a representation format such as XML tags or state machine nodes.

4 Preliminary Results

Table 1 shows the results for an analysis of the support of the two orchestration languages WS-BPEL 2.0 and WF 4 for selected pattern groups of the control-flow [13, 16], the service interaction [2] and the time patterns [7, 8]. WF 4 represents

Table 1. Support of selected patterns. Edit distance is listed first, followed by trivalent scaling in parentheses.

Pattern	WF 4	BPEL 2.0
State-based Control-flow Patterns [13, 16]		
Deferred Choice	9 (+/-)	8 (+)
Interleaved Parallel Routing	- (-)	12 (+/-)
Milestone	11 (+/-)	11 (+/-)
Critical Section	9 (+/-)	11 (+/-)
Interleaved Routing	9 (+/-)	11 (+/-)
Multi-transmission Service Interaction Patterns [2]		
Multi Responses	71 (-)	90 (-)
Contingent Requests	28 (+)	38 (+)
Atomic Multicast Notification	40 (-)	- (-)
Recurrent Process Elements Time Patterns [7, 8]		
Cyclic Elements	12 (+/-)	- (-)
Periodicity	8 (+/-)	7 (-)

the Windows Workflow Foundation in revision 4 (`http://msdn.microsoft.com/en-us/netframework/aa663328.aspx`). The analysis is performed using the approach of the previous section. The process stub used for WF 4 is semantically identical to the one used for BPEL (cf. List. 1) and consists of a `Receive` activity that creates a new process instance embedded in a `Sequence` activity. There is currently no study that measures the degree of the pattern support of WS-BPEL 2.0 and WF 4 for these pattern catalogs. For comparison, we computed the degree of support using the edit distance and the trivalent measure. The edit distance is shown first followed by the trivalent measure in parentheses. A value of '-' for the edit distance means that no valid solution could be found in the scope of the

language. As opposed to this, a value of '-' for the trivalent measure means that either no valid solution could be found or that all possible valid solutions require the use of more than two constructs. The table shows that the edit distance allows for a better distinction. In several cases, both languages have the same degree of support with the traditional measure, while the edit distance unveils the differences.

As an example, the realization of the Deferred Choice pattern ([13], pp. 33/34) is outlined in List. 2. The pattern is realized using two `onMessage` activities embedded in a `pick` activity. The following steps are necessary to realize a valid implementation: (i) replace `receive` with `pick`; (ii) set `createInstance` attribute of `pick` to `yes`; (iii) create first `onMessage`; (iv) configure messaging properties of first `onMessage`, consisting of the setting of the `partnerLink`, `portType` and `operation`; (v) embed `empty` in first `onMessage` (an `onMessage` must contain a child activity); (vi - viii) create and configure the second `onMessage` similar to the first one. In summary, the edit distance of the solution amounts to eight.

Listing 2. Realization of Deferred Choice pattern in BPEL

```
<pick createInstance="yes">
    <onMessage partnerLink="MyPartnerLink" operation="Choice1">
        <empty />
    </onMessage>
    <onMessage partnerLink="MyPartnerLink" operation="Choice1" >
        <empty />
    </onMessage>
</pick>
```

5 Conclusion and Future Work

This work presents an improvement to the method of pattern-based analysis aiming at a higher degree of comparability between different pattern catalogs and a higher level of selectivity of the results. The comparability between pattern catalogs can be improved by using a unified approach for determining whether a given candidate solution provides a valid implementation of a pattern. The approach presented here essentially unites the methodologies used by other authors and insights gained during own analyses. The level of selectivity can be increased by using an alternative support measure, the edit distance based on high level changes. Preliminary results show the applicability of the approach. The next step is to test the approach in a complete study comparing several languages and multiple pattern catalogs which is ongoing work. Especially, languages with a focus on expressiveness such as YAWL might bear interesting results. Also, a better formalization of several pattern catalogs in terms of execution traces would be beneficial.

References

1. A. P. Barros, G. Decker, M. Dumas, and F. Weber. Correlation Patterns in Service-Oriented Architectures. In *FASE*, Braga, Portugal, March/April 2007.

2. A. P. Barros, M. Dumas, and A. H. M. ter Hofstede. Service Interaction Patterns. In *BPM*, pages 302–318, Nancy, France, September 2005.
3. G. Decker and J. Mendling. Process Instantiation. *DKE, Elsevier*, 68:777 – 792, 2009.
4. G. Decker, H. Overdick, and J. Zaha. On the Suitability of WS-CDL for Choreography Modeling. In *EMISA*, Hamburg, Germany, October 2006.
5. R. M. Dijkman, M. Dumas, and L. García-Bañuelos. Graph Matching Algorithms for Business Process Model Similarity Search. In *BPM*, Ulm, Germany, September 2009.
6. O. Kopp, D. Martin, D. Wutke, and F. Leymann. The Difference Between Graph-Based and Block-Structured Business Process Modelling Languages. *EMISAIJ, GI e.V.*, 4:3 – 13, 2009.
7. A. Lanz, B. Weber, and M. Reichert. Time Patterns in Process-aware Information Systems - A Pattern-based Analysis - Revised version. Technical report, University of Ulm, Germany, 2009.
8. A. Lanz, B. Weber, and M. Reichert. Workflow Time Patterns for Process-Aware Information Systems. In *BPMDS and EMMSAD in conjunction with CAiSE*, 2010.
9. M. P. Papazoglou and D. Georgakopoulos. Service-oriented Computing. *Communications of the ACM*, 46(10):24–28, October 2003.
10. C. Peltz. Web Services Orchestration and Choreography. *IEEE Computer*, 36(10):46–52, October 2003.
11. N. Russell, A. H. M. ter Hofstede, and D. Edmond. Workflow Resource Patterns: Identification, Representation and Tool Support. In *CAiSE*, pages 216–232, Porto, Portugal, June 2005. Springer.
12. N. Russell, A. H. M. ter Hofstede, D. Edmond, and W. M. P. van der Aalst. Workflow Data Patterns: Identification, Representation and Tool Support. In *ER*, Klagenfurt, Austria, October 2005. Springer.
13. N. Russell, A. H. M. ter Hofstede, W. M. P. van der Aalst, and N. Mulyar. Workflow Control-Flow Patterns: A Revised View. Technical report, BPM Center Report, 2006.
14. N. Russell, W. M. P. van der Aalst, and A. H. M. ter Hofstede. Workflow Exception Patterns. In *CAiSE*, pages 288–302, Luxembourg, Luxembourg, June 2006.
15. L. H. Thom, M. Reichert, and C. Iochpe. Activity Patterns in Process-aware Information Systems: Basic Concepts and Empirical Evidence. *IJBPIM*, 4(2):93–110, 2009.
16. W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distributed and Parallel Databases, Springer*, 14(1):5–51, 2003.
17. B. Weber, S. Rinderle, and M. Reichert. Change Support in Process-Aware Information Systems - A Pattern-Based Analysis. Technical report, University of Twente, 2007.
18. B. Weber, S. Rinderle, and M. Reichert. Change Patterns and Change Support Features - Enhancing Flexibility in Process-Aware Information Systems. *DKE, Elsevier*, 66:438–466, July 2008.
19. P. Wohed, W. M. P. van der Aalst, M. Dumas, and A. H. M. ter Hofstede. Analysis of Web Services Composition Languages: The Case of BPEL4WS. In *ER*, Chicago, Illinois, USA, October 2003.
20. A. Wombacher and C. Li. Alternative approaches for workflow similarity. In *IEEE SCC*, Miami, Florida, USA, July 2010.
21. M. Zapletal, W. M. P. van der Aalst, N. Russell, P. Liegl, and W. H. An Analysis of Windows Workflow's Control-Flow Expressiveness. In *ECOWS*, pages 200 – 209, Eindhoven, The Netherlands, November 2009.

Meta-Services als zusätzliche Beschreibungsdimension von Cloud-Services

Rainer Schmidt

HTW-Aalen
Anton-Huber-Straße 25
73430 Aalen

Abstract. Meta-Services sind ein Konzept zur Darstellung von Verwaltungsinteraktionen im Kontext von Cloud-Services.

Keywords: Cloud-Services, Meta-Services

1 Einleitung

Die Bereitstellung eines Service wird im Allgemeinen als die Bereitstellung einer bestimmten Funktionalität betrachtet, die Qualitätsparametern wie Zuverlässigkeit, Antwortzeit etc. erfüllen soll. Cloud-Services sind Services, die durch Cloud-Computing [1], [2] bereitgestellt werden. Cloud Services unterscheiden sich von Web-Services [3] durch die Bereitstellung von automatisierten Interaktionen [4] zur Unterstützung des gesamten Lebenszyklusses. Ein Beispiel ist eine Beschwerde eines Kunden. Diese hat zum Ziel, den Service wieder in den vom Kunden erwarteten Zustand zu versetzen, ist aber nicht Teil des eigentlichen Cloud-Service.

Die Verwaltungsinteraktionen zu einem Cloud-Service stellen eine vom Diensterbringer zusätzlich bereitzustellende Funktionalität dar, die zudem mit Qualitätsparameter erfüllen müssen. Sie sind nicht Bestandteil der Funktionalität des Cloud-Service sondern wirken auf den Cloud-Service ein. Beispielsweise sollte eine Beschwerde innerhalb einer bestimmten Zeit bearbeitet werden, was einem Service Level Agreement entspricht [5]. Eine Verwaltungsinteraktion bietet also eine bestimmte Funktionalität unter Einhaltung definierter Qualitätsparameter an.

Daher sollen die Verwaltungsinteraktionen selbst wieder als Services dargestellt werden. Diese Services unterscheiden sich vom Cloud-Service dadurch, dass sie nicht auf das Objekt des Cloud-Service einwirken, sondern auf diesen selbst. Es handelt sich also um einen Service, der einen Service als Objekt hat. Sie sollen daher als Meta-Services bezeichnet werden. Die Menge der Meta-Services ist nicht vorgegeben. So liegt es im Ermessen des Cloud-Service-Anbieters die Menge der von ihm angebotenen Meta-Services festzulegen. Dies kann auch auf der Grundlage von Marketingüberlegungen geschehen. Beispielsweise kann ein Basis-Cloud-Service mit nur wenigen Meta-Services angeboten werden, während ein höherwertiges Angebot zusätzliche Meta-Services enthält.

Ein erster Ansatz für eine Methode zur Bestimmung von Meta-Services ist die Analyse des Lebenszyklusses des Cloud-Service. Kandidaten für Meta-Services ergeben sich aus Veränderungen des Lebenszyklusses. So kann jede Veränderung des Zustands eines Cloud-Service als Meta-Service interpretiert werden. Wichtig ist dabei, die Betrachtung auch auf die Ausprägungsebene auszudehnen. So gibt es Meta-Services, die sich auf Ausprägungen des Service beziehen. Ein Beispiel ist eine Beschwerde, die sich auf die Qualität der Service-Erbringung im Einzelfall und nicht auf die Struktur des Service als solchen bezieht.

Meta-Services sind von den Cloud-Services sowohl unter Funktionalitäts- als auch unter Qualitätsaspekten unabhängig. Die Unabhängigkeit unter dem funktionalen Aspekt zweigt sich auf Typ- und Ausprägungsebene. Auf Typ-Ebene zeigt sich dies, indem ein und derselbe Meta-Service mehreren Cloud-Services zugeordnet sein kann. Beispielsweise kann ein Meta-Service zur Bearbeitung von Beschwerden eine Menge von Cloud-Services zugeordnet sein, die gänzlich unterschiedliche Aufgaben erfüllen. Gleichzeitig können verschiedenen Cloud-Services unterschiedliche Mengen von Meta-Services zugeordnet sein, um beispielsweise unterschiedlichen Kundenkreisen Rechnung zu tragen. Auch auf Ausprägungsebene wird die Unabhängigkeit von Cloud- und Meta-Services bei Kardinalitäts- und Zeitbeziehungen deutlich. So gibt es keine allgemeine Kardinalitätsbeziehungen zwischen einer Ausprägung eines Cloud-Service und eines Meta-Service. D.h. zur Ausprägung eines Cloud-Service kann es keine, eine oder mehrere Ausprägungen des Meta-Service geben. So kann es zur Ausprägung eines Cloud-Service keine, eine oder mehrere Beschwerden geben. Es gibt weiterhin keine allgemeingültige zeitliche Beziehung zwischen Ausprägungen von Services und Meta-Services. D.h. Ausprägungen von Meta-Services können vor, während oder nach den Ausprägungen des Service existieren. Ein Beispiel sind Verbesserungsvorschläge zu Services, die zu beliebigen Zeitpunkten vom Nutzer des Cloud-Service gemacht werden können. Auf Qualitätsebene wird die Unabhängigkeit dadurch deutlich, dass Meta-Services komplett unterschiedliche Qualitätseigenschaften wie der Cloud-Service haben können. Einem rund um die Uhr verfügbaren Cloud-Service kann ein nur für kurze Zeit verfügbarer Meta-Service zugeordnet sein.

2 Literatur

[1] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," 10-Jul-2009. [Online]. Available: http://csrc.nist.gov/groups/SNS/cloud-computing/. [Accessed: 14:17:52].

[2] M. Armbrust et al., "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50-58, 2010.

[3] M. P. Papazoglou, "Service-oriented computing: Concepts, characteristics and directions," in *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on*, 2003, pp. 3-12.

[4] M. Garschhammer et al., "Towards generic service management concepts a service model based approach," in *Integrated Network Management Proceedings, 2001 IEEE/IFIP International Symposium on*, 2002, pp. 719-732.

[5] M. Glinz, "On non-functional requirements," in *Requirements Engineering Conference, 2007. RE'07. 15th IEEE International*, 2007, pp. 21-26.

Building a Person-Centric Mashup System. CommunityMashup: A Service Oriented Approach.

Peter Lachenmaier[1], Florian Ott[1],

[1] Bundeswehr University Munich, Cooperation Systems Center Munich,
Werner-Heisenberg-Weg 39, 85577 Neubiberg, Germany
{Peter.Lachenmaier, Florian.Ott}@kooperationssysteme.de

Abstract. Based on the success of the Web 2.0, today's IT systems are continuously moving from a solely information-centric data perspective to a more person-centric model and are thereby becoming more social. In this paper we discuss the challenges within the redesign of established data models resulting from that shift. Our aim is to derive requirements for a flexible social (person-centric) data integration layer which enables us to aggregate data from several distributed services while retaining the assignment to the individual (personal) identities. In addition to these theoretical considerations we describe how a "CommunityMashup" could be developed, easily maintained and adapted to frequently changing APIs with a service-oriented approach.

Keywords: CSCW, Social Software, Mashup, CommunityMashup, Service Oriented Architecture, Model Driven Development

1 Motivation

A study of different commercial and open-source Enterprise 2.0 tools in 2009 showed that only two out of the seven systems supported tracking of user activities. But all of them enabled their users to trace content changes [1]. For example one of the most commonly used systems in that survey, Atlassian Confluence, did not support tracking users or their activities in 2008 (year of survey), but has recently (2010) switched to a more person-centric approach. Another study analyzed seven tools in 2006 and no more than two of them maintained "user centered" functionalities [2]. Compared to former CSCW[1] research (e.g. [3]) with the success of public Social Networking Services like e.g. Facebook particularly personal information has become more and more important. Atlassian and other global players like Microsoft have adjusted their strategy between 2007 and 2010 to a better support of community features [4]. This development can be put down to the peer-to-peer principle of the Web 2.0 where sharing and collaboration are the most important activities [5].

Beside increased significance of personal data today's IT services have become more modular and more open (in terms of accessibility to data) during the last decade. Hence we are facing a variety of different data sources and at the same time the wish to access (identical) data in different ways as for example with desktop applications,

[1] CSCW: Computer Supported Cooperative Work

websites or mobile devices. Already in 2005 Beale showed different systems supporting social interaction with smartphones and mobile consumption of content [6]. By now most Internet service platforms offer interfaces to access their data. But there is still no "standardized" way to access all data with all available devices.

Although there are other approaches dealing with data models for Social Software, e.g. the Semantically Interlinked Online Communities [7], most of the them still focus on linking content and are thereby not person-centric enough to fulfill all requirements of Social Software (e.g. handling awareness information). The requirements of person centrism, device independent access, modularity and easy adaptability are still not completely satisfied. So we derive the demand for a more adequate data model that can address the needs of a flexible integration service for Social Software.

As activities of other people / groups are becoming more important and the corresponding data will need to be consumable in various contexts with different devices we are using three representative application scenarios where the integration of person-centric data plays an important role. Adapted from these scenarios we will derive specific challenges and requirements for a data model and a technical solution.

Scenario 1: Elderly Interaction & Service Assistant (elisa)
For enabling elderly people to access awareness information from Social Networks without being a direct member of every online service, we currently construct a mobile application showing aggregated awareness information. The displayed information consists of data provided by several people of interest in different social services. In this application scenario information has to be delivered from several people through different services and networks to one single person adapted to his or her individual needs.

Scenario 2: CommunityMirrors
CommunityMirrors are large screens presenting information that is otherwise hidden in IT systems in (semi-) public places, like coffee corners, lobbies or beside the elevator as described e.g. in [8]. In this application scenario information from different sources is shown in an aggregated and unified way on the large screens. Data created by many people has to be delivered to many people without knowing their individual preferences in advance. The aggregation is driven by an organizational context.

Scenario 3: Decentral Federated Research Database (DFRD)
As third application scenario we use the decentral federated research database (DFRD) presented in [9]. Researchers are able to maintain their articles and projects in services of their choice and present them on several web sites by using aggregation mechanisms of the DFRD. Possible data targets are for example a private web site or an aggregated version together with works of other researchers on the university portal as well as presentations filtered for different research groups. In this setting data from the same origin has to be presented with different aggregation levels in various places.

Table 1. Scenario overview

Scenario	elisa	CommunityMirrors	DFRD
Goal	aggregation, filtering, personalization	aggregation, offline setting with synchronization	aggregation, individual / context-related presentation
Data sources	different online Social Networking Services (SNS)	Enterprise 2.0 services like wikis, blogs, microblogs or SNS	individual research services and project portals, Research SNS
Data targets	personalized presentation with mobile application	(semi-) public presentation in different social places	online presentations, e.g. personal or organizational websites
Data type	mainly awareness streams and events	organization specific content with corresponding awareness information	static information like research papers or projects, corresponding awareness information
Target device	mobile device, e. g. Tablet	rich client, large interactive screen	browser (web applications)

Table 1 shows an overview of the application scenarios. Based on this first overview we summarize the challenges during the conceptual design (Chapter 2) and then give a brief overview of the technical solution (Chapter 3). Chapter 4 outlines the mashup possibilities of the technical base. The paper is completed by a conclusion and an outlook to the further development plans of the "CommunityMashup" (Chapter 5).

2 Challenges

The term mashup gained more and more attention in the last years and is used in different meanings. An overview of different definitions is given in [10]. We are using the term mashup as a technical solution that combines data from more than one source, enriches it and provides a unified version for further usage. Based on the different application scenarios introduced in the previous chapter we derived the following four main challenges.

Heterogeneity of services
When two or more people are using different online services the heterogeneity of data formats and interfaces makes the data delivery from one person to another very difficult. Main challenges are the handling of differing data models, the availability of various formats and the use of frequently changing interfaces. In addition to that several authentication and authorization methods as well as questions concerning licenses and laws about how data may be reused or stored have to be considered.

Heterogeneity of clients
As ubiquitous devices in all thinkable shapes and colors have become more and more important during the last decade we are facing heterogeneity on the client side, too. Derived from the application scenarios (see Table 1) we distinguish between three different client classes that can consume the aggregated data. The main differences between these classes are the availability of computing power, memory and network connections.

1. Web applications are the most commonly discussed class in the context of mashup development. They are typically executed on a web server and can be accessed with web browsers.
2. Rich client applications execute most of the required calculations on the client device and therefore require a high amount of computing power and memory.
3. Applications for mobile devices have to deal with much less computing power and memory. Most of the available network connections have unreliable bandwidth.

While mobile and web applications need permanent network connections for data access there are certain settings for rich client applications without the availability of a continuous network connection. But there's still the need of permanent and fast access to required data. The data aggregation itself is independent of the application class. But the different classes require special interfaces to access the data.

Privacy and data protection

In the Web 2.0 era every user provides a huge amount of personal data. Thus, there is a demand for preventing third persons from seeing private information. For that reason most Social Networking Services offer the possibility to grant access only to specified persons or explicitly to external services. These mechanisms must not be bypassed by mashup solutions. So we are facing the requirement to integrate authentication and authorization mechanisms on the mashup side.

Aggregation of data

Another challenge is the combination of data coming from different sources. With the person-centric approach this mainly requires the decision if two profiles belong to the same human being. Social Software in general relies on users being able to manage their personal data by themselves, for example by providing links to their various profiles. This process can be supported but must not be restricted by any technological automatism, since it might not be sufficiently transparent for the user.

3 Technical Solution

To meet the challenges described above the most important requirement besides a person-centric approach is a highly configurable mashup system built with flexible service components reusing existing frameworks. Model driven development for components affected by data model changes is a major claim.

We propose an object-oriented solution to enable direct high-level access to and modification of aggregated data. The introduced application scenarios and the existence of flexible tool chains for application development are the main reasons for this approach. In addition to that generated application frameworks allowing easy development of similar community applications in future versions.

3.1 Person-centric data model

Figure 1 shows the core elements of the current version of the CommunityMashup data model. The main objects person, content and organization are derived from

existing models like SIOC[2] [7] or FOAF[3] [11]. The central element is the person, which can be grouped in organizations and can author or contribute to content. Organizations and contents can be modeled hierarchically (parent relation).

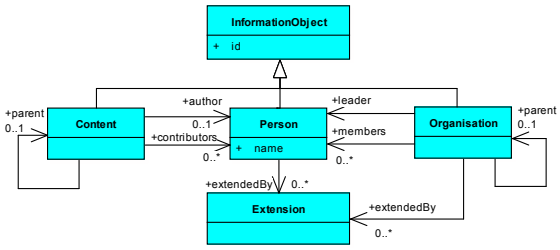


Fig. 1. Core elements of the CommunityMashup data model

In contrast to other models, one important design principle is to include less meta data but instead allow tagging and categorization of information. By this approach a lot of meta data coming from different data sources is transformed to categories and tags. Only identity preserving characteristics like e.g. the name of a person are modeled directly as attributes of the core objects.

Extensions for easier combination
A person is represented by one single object independently of how often he or she appears within the different sources. This means that persons as well as organizations only have one consolidated identity. With every new data source the aggregated information will just be extended. An extension object for the information coming from an additional source will be created, tagged and referenced in the dataset. This allows keeping track of the origin and reintegrating changes back into the source. For example specially tagged extension objects can easily handle complex relations between persons.

Model driven development enables data model evolution
There is no way to determine all facets of a data model in advance so that it fits all future needs. Therefore we need a way to make the adaption to model changes as easy as possible, especially without the need of manual changes to the applications based on the model. Using a continuous tool chain offers tracking of data model changes and the regeneration of application code as well as migration of existing data to the newer version. As we are using a central data model and transformation rules for data from external services, these model transformations can be adapted (semi-) automatically.

3.2 Service Oriented Approach

The CommunityMashup consists of smaller independent modules. Each of these modules acts as a service component and can be combined with others to a complex mashup system. This allows flexible integration of existing services and the distribution over different physical machines.

---

[2] SIOC: Semantically Interlinked Online Community
[3] FOAF: Friend of a Friend

The meta model shown in Figure 2 describes the possible system compositions. Basically a mashup system is composed of several sources. The mapping characterizes how the source data is assigned to the target data of the model. Every source has a configuration containing meta data, e. g. authentication parameters. Additionally there can be an explicit adapter used to transform the data from the external service to a representation according to the internal data mode. The data provided by a source is optionally filtered by a chain of filters, e. g. for privacy reasons. Furthermore every mashup itself can act as a source and thereby be reused in more complex compositions. In addition to that the mashup offers the possibility to persist data from every source as well as the internally "mashed-up" data in either a file or a database. Besides caching which is necessary in offline scenarios this can be used to satisfy performance or availability requirements.

Finally every source and every mashup will be an independent service component that can be deployed, enabled and disabled separately. This aspect is important in order to be able to create distributed mashup configurations. Also a direct execution of mashup components on the client device is possible. For example there could be a mashup component running along with its persistency component as a local cache. With this approach local components can be directly integrated in rich client applications. This enables high level access to data objects without the need to deal with data exchange. Together with a local persistency this allows seamless switching between offline and online data usage.

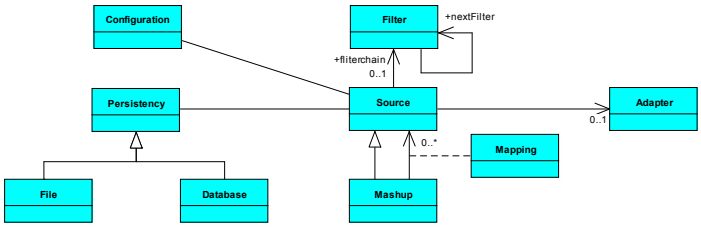


Fig. 2. CommunityMashup meta model

A graphical editor for the creation and change of mashup configurations based on the meta model is planned. Most parts of this editor can be generated with existing tools. Future versions of the editor should be able to interpret these configurations for an automatic deployment and execution of the individual mashup and all dependent components.

3.3 Solution Technologies

For the creation of the data model and the meta model we used the Eclipse Modeling Framework (EMF) [12]. EMF provides a tool chain for the generation of application code and a runtime engine containing persistency and serialization components as well as an integrated event mechanism that allows tracing of data changes. In addition to that there are many useful extensions for EMF like e.g. a tool named COPE[4] [13]

[4] COPE: Coupled Evolution

which helps to track changes of the data model and to migrate existing data to the newest version. This makes it possible to have a continuous tool chain fulfilling the model driven development approach. Because of the multitude of reusable service components we used the OSGi Service Platform with the Eclipse Equinox implementation [14] as service framework.

4 Mashup Possibilities

In the previous chapter we introduced the technical solution of the CommunityMashup. We are continuing with a brief overview of the possibilities of concrete applications based on this technical base.

One of the most important aspects is the discovery of communities that were formerly hidden and distributed over several networks. By combining data from the different networks connections of people will be visible, e. g. knowing the same person or liking the same content. Furthermore the CommunityMashup enables access to an aggregated profile of a person that contains all distributed information. The data model allows keeping track of the data origin, so that changes in the profile can be passed to the original source or delivered to all other profiles.

Many of the content-centric services, e. g. wikis, don't support person-centric awareness streams. With the integration of these services into the CommunityMashup there will be the possibility to automatically create activity streams or similar awareness information. This information can be used in applications based on the CommunityMashup and can be delivered to other source services integrated in the same mashup system. For example a new Wikipedia article can be automatically published as a status update in the Facebook activity stream of the author.

From a more technical point of view, the proposed CommunityMashup solution facilitates the development of applications based on this technical base. The implementation of the three introduced application scenarios (Chapter 1) can be managed without dealing with data integration questions by using a high level API and the integration of CommunityMashup components.

5 Conclusion and Outlook

Based on the motivation for a person-centric data model for Social Software we outlined challenges within the design and development of a flexible mashup solution and presented a person-centric data model. The use of model driven development can help to facilitate model changes. Application source code can be adapted without manual interaction. Furthermore existing data can be migrated without data loss. The CommunityMashup meta model shows the different components of the mashup solution and denotes the possible configuration options. Concerning the architecture of the CommunityMashup we gave a short overview and introduced the technologies that could be used for its implementation.

As stated in Chapter 2 we consider three target application classes: web applications, rich client applications and applications for mobile devices. Currently we are

working on the three usage scenarios of the CommunityMashup described in the motivation:

1. A solution for “best agers” (elderly people) to access awareness information from Social Networking Services with mobile devices.
2. Large semi-public wall-sized screens, our so-called “CommunityMirrors” as a rich client application.
3. A decentral federated research database as a web application with the possibility to present aggregated and filtered data on different web sites.

In all of these scenarios we try to enhance the presented data model and the mashup architecture itself in an iterative incremental way in order to be able to make empirical statements about e.g. the usefulness and efficiency in future examinations.

References

1. Büchner, T., Matthes, F., Neubert, C.: A concept and service based analysis of commercial and open source enterprise 2.0 tools. International Conference on Knowledge Management and Information Sharing (2009)
2. Rama, J., Bishop, J.: A survey and comparison of CSCW groupware applications. In: Proceedings of the 2006 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries - SAICSIT ’06, pp. 198--205, ACM Press (2006)
3. Rodden, T.: A survey of CSCW systems. In: Interacting with Computers 3, 3, pp. 319--353 (1991)
4. Harbridge, R.: SharePoint 2007 vs. SharePoint 2010 Comparison, http://www.rharbridge.com/?page_id=103
5. Ganesh, J., Padmanabhuni, S.: Web 2.0: conceptual framework and research directions. In: Proceedings of the 13th Americas Conference on Information Systems (AMCIS 2007), pp. 198--205 (2007)
6. Beale, R.: Supporting social interaction with smart phones. In: IEEE Pervasive Computing 4, 2, pp. 35--41 (2005)
7. Breslin, J., Decker, S.: SIOC: An approach to connect web-based communities. In: International Journal of Web Based Communities (IJWBC) 2, 2, pp. 133--142 (2006)
8. Koch, M., Ott, F., Richter, A.: Community Mirrors - Using Public Shared Displays to Move Information "Out of the Box". In: Supplementary Proceedings of the 11th European Conference on Computer Supported Cooperative Work (ECSCW), pp. 17--18 (2009)
9. Lachenmaier, P., Koch, M., Richter, A.: Supporting Open Research by making research activities visible, Proceedings of the Workshop on Academia 2.0, 11th European Conference on Computer-Supported Collaborative Work (ECSCW) (2009)
10 Hoyer, V., Fischer, M.: Market Overview of Enterprise Mashup Tools, Lecture Notes in Computer Science, Service-Oriented Computing – ICSOC 2008, vol. 5364, pp. 708--721 (2008)
11. The Friend of a Friend (FOAF) project, http://www.foaf-project.org/
12. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework. Addison-Wesley Professional (2008)
13. Herrmannsdoerfer, M., Benz, S., Juergens, E.: COPE-automating coupled evolution of metamodels and models. In: Lecture Notes in Computer Science, ECOOP 2009 – Object-Oriented Programming 5653/2009, pp. 52--76 (2009)
14. Wütherich, G., Nils, H., Berd, K., Lübken, M.: Die OSGi Service Platform. Dpunkt.verlag GmbH (2008)