

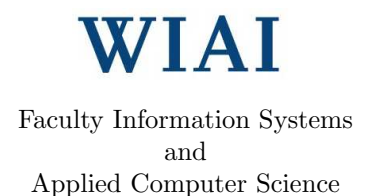
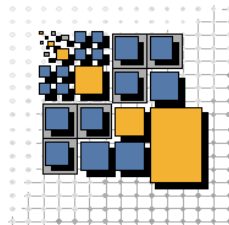
Andreas Schönberger, Oliver Kopp,
Niels Lohmann (eds.)

Services and their Composition

4th Central European Workshop on Services and their Composition
4. Zentral-europäischer Workshop über Services und ihre Komposition
ZEUS 2012, Bamberg, February 23.-24. 2012

Post-Workshop Proceedings

ZEUS 2012 is kindly supported by:



Editors:

Andreas Schönberger
University of Bamberg, Distributed Systems Group,
Feldkirchenstr. 21, 96052 Bamberg, Germany
andreas.schoenberger@uni-bamberg.de

Oliver Kopp
University of Stuttgart, Institute of Architecture of Application Systems
Universitätsstraße 38, 70569 Stuttgart, Germany
oliver.kopp@iaas.uni-stuttgart.de

Niels Lohmann
University of Rostock, Institute of Computer Science
18051 Rostock, Germany
niels.lohmann@informatik.uni-rostock.de

Copyright © 2012 for the individual papers by the papers' authors. Copying permitted for private and academic purposes. Re-publication of material from this volume requires permission by the copyright owners.

Program Committee

Rafael Accorsi	University of Freiburg
Daniel Beimborn	University of Bamberg
Gero Decker	signavio
Daniel Eichhorn	Karlsruhe Institute of Technology
Dirk Fahland	TU Eindhoven
Christian Gierds	Humboldt-University of Berlin
Christian Huemer	TU Vienna
Meiko Jensen	Ruhr-University Bochum
Nils Joachim	University of Bamberg
Oliver Kopp	University of Stuttgart
Philipp Leitner	TU Vienna
Niels Lohmann	University of Rostock
Christoph M. Pflügler	University of Augsburg
Stephan Reiff-Marganiec	University of Leicester
Andreas Schönberger	University of Bamberg
Jan Sürmeli	Humboldt-University of Berlin
Robert Warschofsky	Hasso Plattner Institute Potsdam
Matthias Weidlich	Technion, Israel
Marco Zapletal	TU Vienna

Additional Reviewers

Uwe Breitenbücher	University of Stuttgart
Robert Engel	TU Vienna
Andreas Lehmann	University of Rostock
Dieter Mayrhofer	TU Vienna
Richard Müller	Humboldt-University of Berlin
Christian Pichler	TU Vienna
Robert Prüfer	Humboldt-University of Berlin
Steve Strauch	University of Stuttgart
Jan Sürmeli	Humboldt-University of Berlin
Tobias Unger	University of Stuttgart
Christoph Wagner	Humboldt-University of Berlin
Sebastian Wagner	University of Stuttgart

Table of Contents

Service Adaptation and Synthesis

Toward Deciding the Existence of Adaptable Services	1
<i>Christian Gierds</i>	
Cost-minimal Adapters for Services	9
<i>Jan Sürmeli</i>	
Partner Synthesis for Data-Dependent Services	17
<i>Christoph Wagner</i>	

Service Composition

HarmonICS - a Tool for Composing Medical Services	25
<i>Dariusz Doliwa, Wojciech Horzelski, Mariusz Jarocki, Artur Niewiadomski, Wojciech Penczek, Agata Polrola, and Jaroslaw Skaruz</i>	
Automated Composition of Timed Services by Planning as Model Checking	34
<i>Daniel Stöhr and Sabine Glesner</i>	
Best Service Synthesis in the Weighted Roman Model	42
<i>Diego Calvanese and Ario Santoso</i>	

Service Language Characteristics

Choreographies in BPMN 2.0: New Challenges and Open Questions	50
<i>Mario Cortes-Cornax, Sophie Dupuy-Chessa, and Dominique Rieu</i>	
Building Orchestrations in B2Bi - The Case of BPEL 2.0 and BPMN 2.0	58
<i>Jörg Lenhard and Guido Wirtz</i>	
A Survey on Approaches for Timed Services	66
<i>Kristian Duske and Richard Müller</i>	

Process Analysis

Weak Conformance of Process Models with respect to Data Objects	74
<i>Andreas Meyer, Artem Polyvyanyy, and Mathias Weske</i>	
Towards Verification of Process Merge Patterns with Allen's Interval Algebra	81
<i>Sebastian Wagner, Oliver Kopp, and Frank Leymann</i>	

Towards Automatic Generation of Process Architectures for Process Collections	89
<i>Rami-Habib Eid-Sabbagh</i>	
Workflow Management	
Towards Process Evaluation in Non-Automated Process Execution Environments	97
<i>Nico Herzberg, Matthias Kunze, and Andreas Rogge-Solti</i>	
Towards a Human Task Management Reference Model	104
<i>Daniel Schulte</i>	
Contextsensitive Online Adaption of Workflows	112
<i>Johannes Kretschmar and Clemens Beckstein</i>	
Service Design	
Six Strategies for Building High Performance SOA Applications	120
<i>Uwe Breitenbücher, Oliver Kopp, Frank Leymann, Michael Reiter, Dieter Roller, and Tobias Unger</i>	
Guided Control Flow Unfolding for Workflow Graphs Using Value Range Information	128
<i>Thomas Heinze, Wolfram Amme, Simon Moser, and Kai Gebhardt</i>	
Model Support for Confidential Service-Oriented Business Processes	136
<i>Andreas Lehmann and Niels Lohmann</i>	
Index of Authors	144

Toward Deciding the Existence of Adaptable Services

Christian Gierds

Humboldt-Universität zu Berlin, Department of Computer Science,
Unter den Linden 6, 10099 Berlin, gierds@informatik.hu-berlin.de

Abstract. *Service adaptation* allows two services to interact properly using a mediator or adapter. In service discovery one question is whether an *adaptable service* exists for a given service, i. e. whether a service exists which can be interacted with properly by using an adapter. We look at a setting where this question boils down to deciding *distributed controllability*, and we present an idea for changing an algorithm for controller synthesis which answers this question.

1 Motivation

In recent years the idea of *service adaptation* gained momentum within the scientific community. Services already are a wide-spread paradigm also used in industry. Thus the pool of independently developed services is huge. As services are made to be coupled, the question of *service discovery* (viz. does a service exist that my service can interact with) plays an important role in Service-Oriented Architectures [6]. However, as a service might be developed without knowledge about existence of potential partners, it is likely that service discovery fails because of incompatibilities. In this case an *adapter* [8] might overcome these incompatibilities. As indicated in Fig. 1a, the adapter acts as mediator between two services S_1 and S_2 ensuring *semantically correct processing of messages* and *proper termination* of the services. In case service discovery fails, i. e. there does not exist any service for direct interaction with, *the question concerning service discovery now can be extended to the adapter setting*.

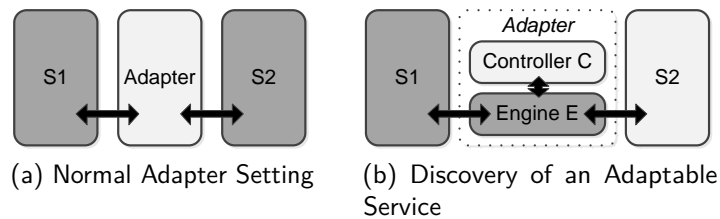


Fig. 1. Two different perspectives (dark-gray means given, light-gray means wanted)

Existence of an Adaptable Service: Is there another service and an adapter, such that my service can communicate correctly with this other service using the adapter?

This question actually is not easy to answer. It appears, that we may simply apply an algorithm for service selection in order to pick S_2 and an adapter. Service adaptation proposes to *create*, not to select a mediating service though. As it mainly works on a semantical rather than a functional level adapters are not meant to be stored and be publicly available. Thus the question above would suggest trying to create an adapter for each candidate S_2 .

Setting. Many newer approaches [1–4] recommend to first describe the semantical constraints of an adapter and then to ensure also behavioral correctness (viz. proper termination by coordinated transformation of messages). Figure 1b shows one possible solution [4]: Let the services S_1 and S_2 be given, as well as a set of *message transformation rules* describing valid transformations of messages. These transformations are implemented in an *engine* service E ensuring the semantically correct exchange of messages. If we find a controller C triggering transformations as they are actually needed, then the composition of $E \oplus C$ is an adapter.

For this approach we may assume to have S_1 and at least the transformation rules and thus the engine E given, when looking for a service S_2 . Checking the *Existence of an Adaptable Service* then asks for a service S_2 for which a controller C exists, such that the composition $S_1 \oplus E \oplus C \oplus S_2$ properly terminates.

Contribution. We provide an algorithmic idea to abstract from C and then ask for the existence of an S_2 using existing work on partner synthesis. As to the best of our knowledge the question for the *Existence of an Adaptable Service* has not been answered so far. The idea proposed in this paper is a first step in solving the problem.

The ultimate goal is the *characterization of all adaptable services*. Then, given a candidate in a repository, we could decide, whether an adapter can be computed for this candidate or not. However, this problem will remain for future work. So far, we simply want to be able to check, whether it is possible to find such a candidate, or if no such service exists as the transformation rules are not sufficient for adaptation.

In the following we first introduce service adaptation on a formal level (Sec. 2). Then we briefly discuss the problem of distributed controllability (Sec. 3)—our main question relates to this problem—before giving an idea for solving the problem in the special case of adapters (Sec. 4). Finally we give an outlook (Sec. 5) on how to extend the solution to partially solve the more general case of *distributed controllability*.

2 Service Adaptation

In the last couple of years many approaches [1–3] (among others) emerged for the adaptation of independently developed services. Many of these approaches

describe the semantical level of an adapter independently of its control structure. The semantical level is described by *message transformation rules* defining the transformation of a message in order to meet semantical constraints imposed by its content. Typical transformations range from simple renaming to the combination of several message into one message (or vice-versa), or the creation/deletion of protocol message (e. g., acknowledgments).

We use previous work [4] by colleagues and myself based on Petri nets to formally describe the setting (as shown in Fig. 1b). Using Petri nets gives some advantages: they allow to easily describe distributed models, and message transformation rules can be directly translated into a net structure.¹ We use the typical definition of a *Petri net* $N = (P, T, F, m_0)$ with finite sets of places P , transitions T , a flow relation $F \subseteq (P \times T) \cup (T \times P)$, and an initial marking m_0 . A marking $m : P \rightarrow \mathbb{N}$ assigns to every place a number of tokens. The firing semantics are as usual: a transition $t \in T$ is enabled in a marking m , when all places in the preset are marked appropriately ($(p, t) \in F \Rightarrow m(p) > 0$), and t may fire only, if it is enabled and thus changes the marking to $m'(p) = m(p) - F(p, t) + F(t, p) \forall p \in P$.

An *open net* additionally uses transition labels to express communication via some channel c : a transition may asynchronously send a message ($!c$), receive a message ($?c$)—thus restricting firing if c contains no message—, or it may synchronize ($\#c$). Further we provide a set Ω of *final markings*, indicating in which state a service is allowed to terminate.

The approach for adapter synthesis then can be summarized as follows: let us assume to have given service models S_1 and S_2 (as open nets) as well as message transformation rules, which can be canonically translated into an open net E (the *engine*). Each transition of E is synchronously connected to a controller port, thus allowing full control about the application of transformation rules. We then use existing algorithms for controller synthesis for constructing a controller and thus an adapter, if any exists. The controller’s role is to ensure proper termination as transformation rules may not be applied arbitrarily. In the following *proper termination* is used synonymously to *weak termination* (viz. it is always possible to reach a final state).

Figure 2 shows our (technical) running example. We use the typical graphical notation for Petri nets. Additionally the dashed line shows the boundary of one open net, places indicating a proper final state are depicted using a double line. Communication labels are written inside a transition (omitting the synchronous labels used for communication in the adapter between the lower engine and the upper controller part). Service S_1 (Fig. 2a) receives an initial message ($?e$), waits for an external choice (either $?b_1$ or $?b_2$), sends an appropriate answer (either $!t$ or $!c$), and returns to its initial state. Service S_2 (Fig. 2c) simply sends a message ($!m$) and waits for a reply ($?k$); or it may decide to terminate. Within the engine part of the adapter (Fig. 2b) we see the communication with S_1 and S_2 as well as the transformation rules (r_1 to r_5). In more detail the rules are: $r_1 : m \mapsto e$

¹ N. B.: The whole theory could be canonically translated into state machines. However we decided to use Petri nets.

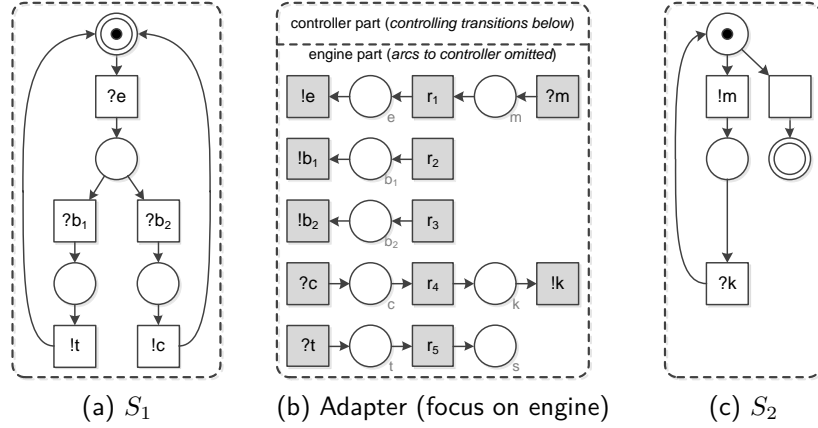


Fig. 2. Running example: Adapter for two services S_1 and S_2

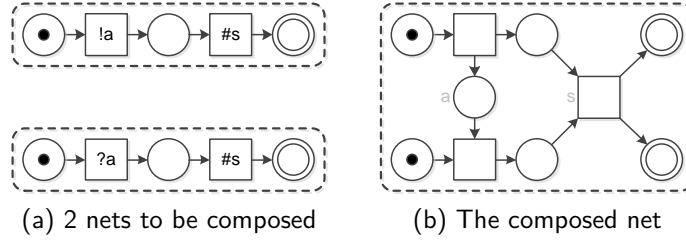


Fig. 3. Composition of open nets

(rename m to e); $r_2 : \mapsto b_1$ (create b_1); $r_3 : \mapsto b_2$ (create b_2); $r_4 : c \mapsto k$ (rename c to k); and $r_5 : t \mapsto s$ (rename t to s). As we can see, rules canonically translate into Petri net transitions, where the channel names translate into places. *Please note*, that there are additional arcs between the engine and controller part that we omitted for sake of simplicity.

Composition of two open nets A and B is realized by introducing a buffer place p_c for each asynchronous channel c and adding an arc (t, p_c) for each transition t with label $!c$, an arc (p_c, t) for each transition t with label $?c$, and each pair of transitions with the same synchronous label $\#c$ is merged while preserving their individual presets and postsets (Figure 3 shows an example).

We now want to change the perspective in order to check the *Existence of an Adaptable Service*: let us assume we only know about service S_1 and some transformation rules. *Does any service S_2 exist, such that S_1 and S_2 are adaptable given the transformation rules?* Regarding Fig. 1a we have given services S_1 and E and are looking for services S_2 and C (where the latter is of minor importance). Nevertheless in our setting we are actually looking for *two* services that are not

allowed to communicate with each other directly—as S_2 and C only communicate with the engine E —but that we can match during construction.

Let us rephrase the problem a bit: *given $S_1 \oplus E$, we are looking for a service S_2 , such that we can be sure, an appropriate C also exists.*

3 The Problem of Distributed Controllability

The problem we want to solve in the adapter setting—checking for the *Existence of an Adaptable Service*—relates to the problem of *distributed controllability* [7]. Given a service A ($S_1 \oplus E$) with two distinguished ports for communication, do two services B_1 (service S_2) and B_2 (controller C) exist, such that the composition of $B_1 \oplus A \oplus B_2$ describes a proper system. In this setting, B_1 communicates with A over one port, and B_2 over the other. However, B_1 and B_2 are not allowed to directly communicate with each other.

The described problem is known to be solvable for acyclic services [7]; however, for cyclic services there exists strong suspicion, that the problem is in fact undecidable.²

Although checking the *Existence of an Adaptable Service* thus relates to a problem suspected to be undecidable, we present an idea for tackling this problem in the special case of adapters. This is possible as the engine of an adapter has a very special structure—every transitions within the engine is under control. The decisions of any controller are very determined, which helps us in predicting a controllers decisions. Thus even in case of cyclic services we can actually decide, whether an adaptable service exists.

In the next section we exploit this fact by actually foretelling some of the controllers decisions and then checking for the *Existence of an Adaptable Service* using existing algorithms for partner synthesis.

4 Existence of Adaptable Services

In this section we shortly sketch the idea for answering the question, whether a service S_2 exists for a given service S_1 and a set of transformation rules, such that S_1 and S_2 are adaptable with respect to the transformation rules.

First of all we have to fix an interface for S_2 . Otherwise it is not clear, which messages used within the transformation rules are actually meant to be sent or received—which actually is not always clear from the rules. However, in many cases the rules suggest a certain direction and we leave it to future work to find heuristics for allowing more flexibility in choosing the interface.

When trying to find S_2 we have to take care of certain points: first, building a central controller (one serving both ports) can be misleading. There exists S_1 and E which have a central controller, but no distributed one (e. g., when S_2 would need to react on messages exchanged between E and C). An algorithm working on the central controller must be aware of this. Second, as we are

² Personal communication with Karsten Wolf. Unpublished result.

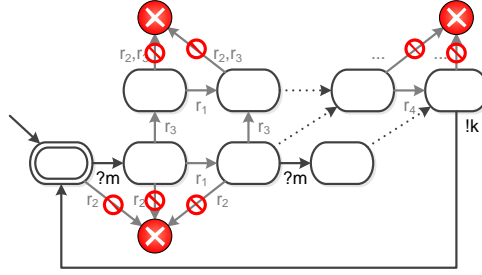


Fig. 4. Automaton derived from $S_1 \oplus E$ with branches leading to deadlocks removed

mainly interested in S_2 we could abstract from the interface (i.e. remove all communication) between engine and controller. This would respect somehow the independence of S_2 and controller C . However, this does not take into account the possibility for design-time coordination of S_2 and C , thus failing in finding any S_2 in many cases, when actually one exists.

The approach we want to follow is a mixture of both ideas: abstract from the communication between engine and controller, but use information about transitions being part of the engine to decide, whether bad states could be avoided by an yet unknown controller.

The algorithm [5] for constructing a controller in the adapter setting presented above is based on communicating automata which can be canonically derived from the reachability graph of $S_1 \oplus E$. In our setting there might be many states, where avoiding a deadlock or livelock seems impossible for S_2 alone. For a transition of the engine a controller can however decide *not* to fire it if it leads unavoidably to bad states.

Our algorithm for finding an adaptable service S_2 (without generating a corresponding controller part C) can be summarized as follows: Let us have given a service S_1 and an engine E representing message transformation rules. Translate the reachability graph of $S_1 \oplus E$ into a communicating automaton. If there is a transition with a controller label—a label for communication between engine E and controller C —which results in a state, where reaching a bad state is unavoidable for any S_2 , then remove this transition and all states becoming unreachable. If no such further transitions exist, remove all controller labels (making corresponding transitions communicating with the controller part of an adapter internal) and run the algorithm for partner synthesis.

This way we exploit the possibility to rely on a correct decision of C in case it is necessary. As we are only removing transitions where reaching a final state cannot be enforced anymore—neither by S_2 or any controller C —communication between engine and controller is not determined and a level of uncertainty remains, which S_2 has to cope with (viz. S_2 is still not aware of communication between E and C).

Running example: Let us consider the example in Fig. 2. The transition r_3 (creating b_2) should fire once for every m received, but never a second time. As we know that r_3 is part of the engine, we know that a C can decide to fire r_3 one time (as a final state is reachable in the example), but never a second time, when no further m was received (as a final state might not be reachable anymore in the example). Thus we remove all transitions related to the second firing of r_3 . We can see the (partial) result of this operation in Fig. 4. Arcs and labels related to engine transitions are gray, an unavoidable deadlock is indicated by a cross, the removal of arcs by prohibition signs. When we start partner synthesis on the artifact depicted in Fig. 4, then we get as result a service corresponding to the net initially depicted in Fig. 2c. Thus we are able to compute a witness to show that S_1 is adaptable provided the given transformation rules.

5 Conclusion and Outlook

The use of adapters extends the setting of Service-Oriented Architectures by some challenging questions. In the case of service discovery we may not only be interested a (compatible) partner service, but also a partner service usable through an adapter would serve our purposes. Thus the question arises if any *adaptable service does exist*. In case we regard an adapter as a union of semantical constraints and control flow we have provided an algorithmic idea to answer this question. We have omitted a proof for the correctness of this approach. Surely it highly relies on the very special structure of the engine (unique communication labeling, controller always has complete knowledge about the state of the engine, thus decisions are determined, etc.).

If we want to lift this approach to decide distributed controllability in a more general setting, certain pitfalls are ahead that do not allow to directly apply the idea on arbitrary services. Some major issues are transitions with equal communication labels, a higher degree of uncertainty, and asynchronous communication labels on both ports (asynchronous communication normally needs to be restricted due to undecidability results, what the algorithm is not yet aware of).

Nevertheless we want to refine the algorithm in a way, that if we abstract an *arbitrary* two-port service S from one port and find a controlling service for the second port, then only because S is distributed controllable. For the adapter setting we want to show on a formal level, that the algorithm actually decides the problem. Thus if $S_1 \oplus E$ is distributed controllable, then the algorithm finds some S_2 .

Further, we would like to characterize *all* adaptable service. This would allow us to actually do Service Discovery more efficiently without synthesizing an adapter for each candidate service S_2 . We could simply *match* S_2 against the characterization.

References

1. Benatallah, B., Casati, F., Grigori, D., Motahari Nezhad, H.R., Toumani, F.: Developing adapters for web services integration. In: CAiSE. pp. 415–429 (2005)
2. Brogi, A., Popescu, R.: Automated generation of BPEL adapters. In: ICSOC. pp. 27–39 (2006)
3. Dumas, M., Spork, M., Wang, K.: Adapt or perish: Algebra and visual notation for service interface adaptation. In: Business Process Management. pp. 65–80 (2006)
4. Gierds, C., Mooij, A.J., Wolf, K.: Reducing adapter synthesis to controller synthesis. *IEEE Transactions on Services Computing* 99(PrePrints) (2010)
5. Lohmann, N., Wolf, K.: Compact representations and efficient algorithms for operating guidelines. *Fundam. Inform.* 108(1-2), 43–62 (2011)
6. Papazoglou, M.P.: *Web Services: Principles and Technology*. Pearson - Prentice Hall, Essex (Jul 2007)
7. Wolf, K.: Does my service have partners? *T. Petri Nets and Other Models of Concurrency* 2, 152–171 (2009)
8. Yellin, D.M., Strom, R.E.: Protocol specifications and component adaptors. *ACM Trans. Program. Lang. Syst.* 19(2), 292–333 (1997)

Cost-minimal adapters for services

Jan Sürmeli

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

Abstract The composition of compatible services is central in service orientation. Adapters resolve incompatibilities between services. Adapter synthesis generates an adapter A for two given services N_1 and N_2 . Generally, there exist several different adapters for N_1 and N_2 . In this paper, we suggest a framework to express preference between those adapters. Additionally, we sketch the synthesis of a cost-minimal adapter.

1 Introduction

We understand a *service* [6] as a component with an *inner process* and an *interface* for message exchange with other services. The actions of the inner process may be linked with the interface, declaring which actions *receive* and *send* which type of messages. A central concept of service orientation is the *composition* of services. Clearly, it is only feasible to compose *compatible* services. There exist four core aspects [5]: Syntactical, behavioral, semantical, and non-functional. In this paper, we concentrate on *weak termination*, a behavioral compatibility notion, similar to soundness [10] in business processes. A set of services is compatible w.r.t. weak termination, iff the composition of its elements is free of deadlocks and livelocks. An *adapter* [12] solves the problem that two services N_1 and N_2 are incompatible by mediating between them. An adapter A is a service, such that the set N_1 , N_2 , and A of services is compatible. The idea of *adapter synthesis* is to automatically generate an adapter for two services. Generally, there are different adapters for two services N_1 and N_2 . In this paper, we propose a framework to express *preference* between such adapters by means of *cost models* and *cost functions*. We discuss two cost models. For one cost model, we sketch the synthesis of a *cost-minimal adapter*.

2 Running example

As a running example, we introduce two simple incompatible services, which are depicted in Fig. 1 in a notion similar to BPMN: Boxes are tasks, diamonds are splits and merges. Initialization and termination are represented by circles and bold circles, respectively. The dashed line encapsulates a service. A rounded box on the dashed line is a port, consisting of input and output message types. We further explain syntax and semantics by describing the actual models.

Service N_1 has one port with the input message types ANSWER and CANCEL, and the output message types LOGIN, REQUEST, and DONE. Initially N_1 sends a LOGIN message. This is modeled as a task named !L, where ! stands for *sending*, and L stands for LOGIN – we abbreviate the names of the message types. Then, N_1 enters a loop. In each iteration, it sends an R (resembled by !R) and waits for an A or a C. If it receives a C, the loop is left. If it receives an A, it decides internally between starting another iteration, or leaving the loop. In the latter case, it sends a D to inform its environment that it is done sending requests. Upon leaving the loop, it terminates. Service N_2 serves its environment by receiving a H followed by a P. It then returns a S followed by receiving a G. Initially and after receiving a G, N_2 can receive a Q to terminate.

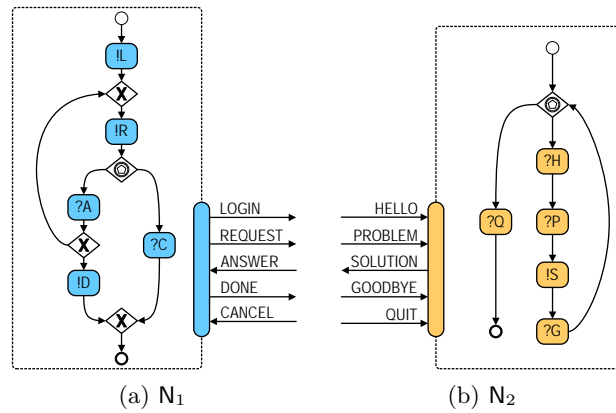
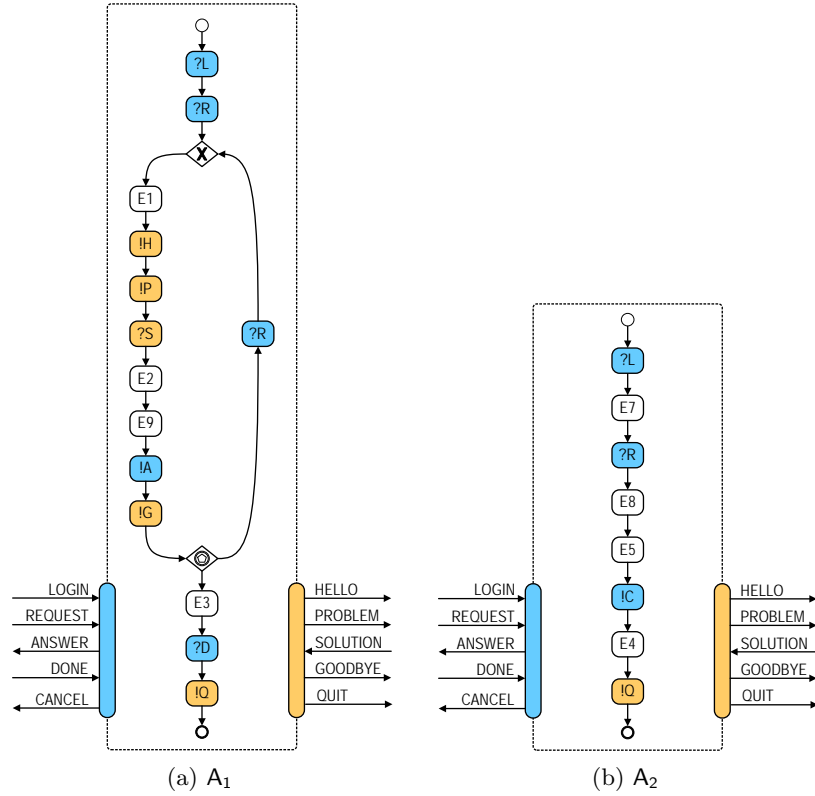


Figure 1: Running example: Two incompatible services N_1 and N_2

Obviously, N_1 and N_2 are incompatible. Even if one tries to map the interfaces as far as possible (i.e. $L \mapsto H$, $R \mapsto P$, $S \mapsto A$, $D \mapsto G$), the composition deadlocks: After one iteration of the loop, N_2 waits for another H, and N_1 waits for an A.

There exist different adapters for N_1 and N_2 . Figure 2 depicts two adapters A_1 and A_2 for our running example. Adapter A_1 sends the missing H in each loop iteration. Once N_1 decides to be done, it quits N_2 . In contrast to the previously studied models, A_1 executes tasks neither starting with ? nor !, e.g. E_1 . Such a task is internal, that is, neither sending nor receiving a message. The label E_1 refers to a message transformation in Fig. 3(a). We explain this in more detail in the next section. Adapter A_2 resolves the incompatibility trivially by sending a C to N_1 , and a Q to N_2 .

One might *prefer* one adapter over the other. For instance, A_1 could be preferable because it enforces both services to enter their main part. In contrast to that, A_2 simply quits both services, which does not seem very useful. However, one could also prefer A_2 over A_1 , because executing the main part of the services requires many costly message transformations.


 Figure 2: Running example: Two adapters A_1 and A_2

3 General synthesis approach

In this section, we propose a general approach to synthesize a cost-minimal adapter. We first explain the approach in [2] to synthesize an arbitrary adapter. Second, we explain how to extend this approach such that it yields a cost-minimal adapter.

We synthesize an adapter for N_1 and N_2 based on a given *specification of elementary activities* (SEA). Such an SEA contains all semantical activities an adapter may perform. An elementary activity is a rule of the form $x_1, \dots, x_m \mapsto y_1, \dots, y_n$, where $x_1, \dots, x_m, y_1, \dots, y_n$ are message types. For each message type x_i , a message of that type is consumed, for each message type y_j , a message of this type is produced. Message transformations are performed on internal buffers. That is, the adapter stores incoming and intermediate messages locally.

Continuing the running example, Fig. 3(a) shows an SEA $\{E_1, \dots, E_9\}$. We recognize all but one message type from the running example: Message type X is a temporary message to remember that rule E_2 has been applied. That is, that at least one S has been translated to an A . Using such intermediate messages is a

mechanism to cope with dependencies between rules. We identify which adapter uses which rules. Adapter A_1 (Fig. 2(a)) executes E_1 , E_2 , E_9 , and E_3 . Adapter A_2 (Fig. 2(b)) executes E_7 , E_8 , E_5 , and E_4 . Finally, the adapter A_{\min} (Fig. 3(b)) executes E_1 , E_2 , E_3 , E_9 , E_8 , E_6 , and E_4 .

One advantage of this approach is that adapter synthesis can be reduced to *partner synthesis*. Intuitively, partner synthesis [11] solves the problem to find a compatible service N_2 for a given service N_1 , called partner of N_1 . First, we create an *engine* E from the SEA. The engine has three ports, one for each N_1 and N_2 , and one for a *control service*. We compose N_1 , E , and N_2 . We synthesize a control service C . The composition of E and C serves as an adapter for N_1 and N_2 . We shortly discuss the limitations of this approach. It is possible to adapt more than just two services at once, but only by a central adapter. Additionally, this approach adapts single instances of each service. If one desires to adapt n instances of one service, it is required to treat them each as a different service, instead. This is obviously infeasible if the number of instances is variable and not known beforehand.

We propose to specify costs based on the SEA, because it contains all activities which are known before synthesis. We then follow the above approach and reduce the problem to synthesize a cost-minimal adapter to the problem to synthesize a cost-minimal partner.

4 Cost models and cost functions

In this section, we introduce two formal constructs: *Cost models* and *cost functions*. A cost model determines how costs are represented, aggregated, and compared. A cost function specifies the costs for executing a rule of the SEA Σ . These cost may resemble monetary costs, for calling a web service, or penalties. Given a cost model and a cost function, the costs of an adapter are well-defined.

Formally, a cost model $\mathcal{C} = [\mathcal{D}, \mathcal{S}, \leq]$ consists of a set \mathcal{D} , called *domain* of \mathcal{C} , a function $\mathcal{S} : \mathcal{D}^* \rightarrow \mathcal{D}$, called *sequence aggregator function* (SAF) of \mathcal{C} , and a partial order \leq on $2^{\mathcal{D}}$, called *set ordering relation* (SOR) of \mathcal{C} .

A *cost function* $\mathcal{F} : \Sigma \rightarrow \mathcal{D}$ specifies the cost of executing a single rule $a \in \Sigma$. We combine a cost function with a cost model to determine the costs of a sequence of rules. Let $\sigma = a_1 \dots a_n \in \Sigma^*$. We define the costs of σ as $\langle \mathcal{F}, \mathcal{C} \rangle(\sigma) := \mathcal{S}(\mathcal{F}(a_1) \dots \mathcal{F}(a_n))$. Let $L, L' \subseteq \Sigma^*$ be sets of sequences. We define $\langle \mathcal{F}, \mathcal{C} \rangle(L) := \{\langle \mathcal{F}, \mathcal{C} \rangle(\sigma) \mid \sigma \in L\}$. We define $L \leq^{\mathcal{F}} L'$ iff $\langle \mathcal{F}, \mathcal{C} \rangle(L) \leq \langle \mathcal{F}, \mathcal{C} \rangle(L')$.

Let N_1 and N_2 be incompatible services. The *costs of an adapter* A w.r.t. \mathcal{F} , \mathcal{C} , N_1 , and N_2 are then determined by inspecting the *terminating runs* of the composition of N_1 , N_2 , and A . A terminating run results in a common final state of all services. We define $\langle \mathcal{F}, \mathcal{C}, N, Q \rangle(A) := \{\langle \mathcal{F}, \mathcal{C} \rangle(\sigma|_{\Sigma}) \mid \sigma \text{ is a terminating run of } N \oplus A \oplus Q\}$, where $\sigma|_{\Sigma}$ denotes the restriction of σ to alphabet Σ . For two adapters A, A' , we define $A \leq^{\mathcal{F}, N, Q} A'$ if and only if $\langle \mathcal{F}, \mathcal{C}, N, Q \rangle(A) \leq^{\mathcal{F}} \langle \mathcal{F}, \mathcal{C}, N, Q \rangle(A')$.

A general limitation of this approach is the fact that costs for a sequence of rule executions are built from the costs of the executions of single rules. One may desire

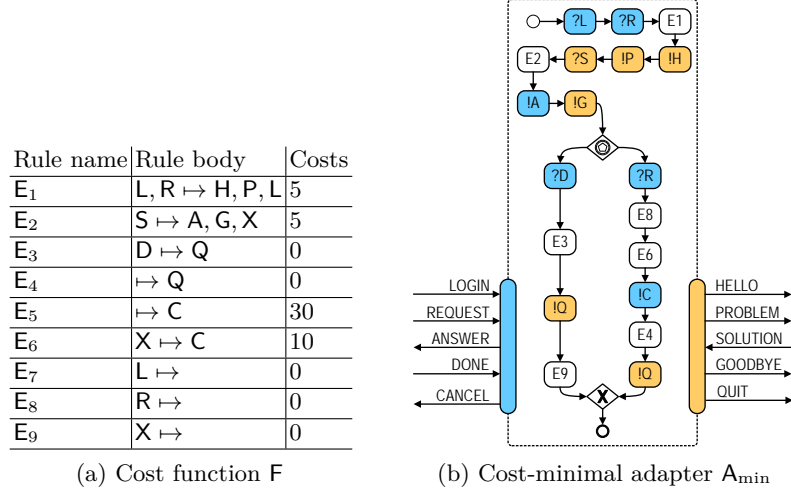


Figure 3: A cost function F , and a cost-minimal adapter A_{\min} w.r.t. cost function F , cost model \mathcal{T} , and services N_1 and N_2

to express dependencies, for instance, executing rule $E = x_1, \dots, x_m \mapsto y_1, \dots, y_n$ could be cheaper if a message X was received beforehand. This can be partly realized by intermediate messages. For this case, one would introduce three new rules $R = X \mapsto X, received_X$, $R' = received_X \mapsto$, and $E' = received_X, x_1, \dots, x_m \mapsto received_X, y_1, \dots, y_n$. One would apply the appropriate lower costs to E' . An open question is to find the class of dependencies one can express in this way. For example, it is not possible to declare that the costs for executing a rule are reduced by factor two each time.

In the remainder of this section, we discuss two cost models, and apply these cost models to our running example. For that purpose, we define a cost function F based on the SEA in Fig. 3(a). Most of the rules have costs of zero. However, E_1 and E_1 have costs of 5, because the message content has to be translated to a different format. Rule E_5 has a penalty of 30, to symbolize that we do not prefer to send a C to N_1 . Rule E_6 has a lower penalty, because it may only be applied after at least one request has been answered.

4.1 A cost model for worst case total costs

The idea of this cost model is to compute the total costs of each run, and select the supremum to compare two adapters. Total costs are determined by addition. We prefer an adapter A_1 over an adapter A_2 if the worst-case total costs of A_1 are lower than the worst case total costs of A_2 . In order to enable analysis, we choose the natural numbers as domain. The advantage is an inherent monotony. We define the cost model \mathcal{T} as the cost model with domain $\mathcal{D}_{\mathcal{T}} := \mathbb{N}_0$, the SAF $\mathcal{S}_{\mathcal{T}}(a_1 \dots a_n) := a_1 + \dots + a_n$, and SOR $X \leq_{\mathcal{T}} Y$ if and only if $\sup(X) \leq \sup(Y)$. Thereby, \mathbb{N}_0 denotes the set of natural numbers, $+$ denotes integer addition,

$\sup(X) \in \mathbb{N}_0 \cup \{\infty\}$ denotes the supremum (least upper bound) of X , and \leq denotes the natural order on $\mathbb{N}_0 \cup \{\infty\}$.

Comparing the adapters of our running example, we find: $A_1 \mapsto \{10, 20, 30, \dots\}$, $A_2 \mapsto \{30\}$, $A_{\min} \mapsto \{10, 20\}$. We prefer A_{\min} , because $20 \leq 30 \leq \infty$. We believe that it is obvious that A_{\min} is the cost-minimal adapter w.r.t. F , \mathcal{T} , N_1 and N_2 . It is impossible to adapt N_1 and N_2 with less costs, because N_1 decides whether it sends another request, or whether it is done.

4.2 A cost model for worst case average costs

The disadvantage of cost model \mathcal{T} is that all adapters with unbounded worst case total costs are equivalent. That is, we may not distinguish between them. However, there might be services N_1 and N_2 which are not adaptable by a service with bounded worst case costs. In this case, we would like to be able to distinguish two adapters A_1, A_2 with unbounded costs. Intuitively, we compare the costs for executing an additional rule in each adapter. We evaluate the average costs of each run (instead of the total costs), and use again the supremum to compare. We prefer an adapter A_1 over an adapter A_2 if the worst-case average costs of A_1 are lower than the worst case average costs of A_2 . Obviously, this cannot be done in the natural numbers anymore. Hence, we select the non-negative rational numbers as domain. We define the cost model \mathcal{A} as the cost model with domain $\mathcal{D}_{\mathcal{A}} := \mathbb{Q}_0^+$, the SAF $\mathcal{S}_{\mathcal{A}}(a_1 \dots a_n) := \frac{a_1 + \dots + a_n}{n}$, and SOR $X \leq_{\mathcal{A}} Y$ if and only if $\sup(X) \leq \sup(Y)$. Thereby, \mathbb{Q}_0^+ denotes the set of non-negative rational numbers, $+$ denotes rational number addition, $\sup(X) \in \mathbb{Q}_0^+ \cup \{\infty\}$ denotes the supremum (least upper bound) of X , and \leq denotes the natural order on $\mathbb{Q}_0^+ \cup \{\infty\}$.

Comparing the adapters of our running example, we find: $A_1 \mapsto \{\frac{10}{4}, \frac{20}{7}, \frac{30}{10}, \dots\}$, $A_2 \mapsto \{\frac{30}{4}\}$, $A_{\min} \mapsto \{\frac{10}{4}, \frac{20}{5}\}$. We prefer A_1 , because $3 \leq 4 \leq 7.5$.

5 Synthesis of cost-minimal adapters

For the cost model \mathcal{T} , we solved the problem to synthesize a cost-minimal partner in [9]. We use the same mechanism to synthesize a cost-minimal adapter: Two services N_1, N_2 , an engine E , and a cost function \mathcal{F} serve as input. The engine E may be computed from an SEA by the tool Marlene¹. We first compose N_1, N_2 , and E to the service $N = N_1 \oplus E \oplus N_2$. Then, we synthesize a cost-minimal partner for N w.r.t. \mathcal{F} .

We sketch the synthesis approach. A central concept of the synthesis is the *minimal budget* of a service N w.r.t. a cost function \mathcal{F} . That is, the unique costs of the cost-minimal partner. For \mathcal{T} , the minimal budget is either a natural number, or infinite.

In a first step, we find an upper bound b for the minimal budget with the following property: b is infinite iff the minimal budget is infinite. The upper

¹ Marlene is available at <http://service-technology.org/marlene> [Last accessed on 2012-14-02]

bound b is found by examining N together with its most-permissive partner. Intuitively, the most-permissive partner simulates each other partner of N . The most-permissive partner is computed with the tool Wendy [3]. If b is infinite, every partner of N is cost-minimal.

If b is finite, we find the cost-minimal partner by iteration: For a given budget k , it is possible to check whether there exists a partner with costs k . If such a partner exists, it can be synthesized. This is realized by first transforming N to a service $N_{\mathcal{F}}^k$, and then synthesizing a partner for $N_{\mathcal{F}}^k$. We find the lowest $k \leq b$, such that there exists a partner with costs k . This is implemented as a binary search. The resulting partner is by construction cost-minimal.

We implemented this procedure prototypically in Tara². For other cost models, we do not have a solution yet.

6 Related work

Seguel et al. [8] study the problem to create *minimal adapters* to resolve deadlock between services. Thereby, minimality is defined w.r.t. to the number of messages considered. The resulting adapter resolves deadlocks and is minimal w.r.t. messages. We study the more general criterion of weak termination. We showed that the minimal adapter is not necessarily cost-minimal. We reduced the synthesis of a cost-minimal adapter to the synthesis of a cost-minimal partner. Zeng et al. [13] use integer programming to find an optimal composite service. That is, an optimal composition of atomic tasks each implemented by a web service. The services do not communicate based on its state, whereas we consider stateful services. De Paoli et al. [4] propose a similar approach for WS-BPEL [1] processes. Both approaches work on well-structured services, whereas we support arbitrary services. The *CLAM framework*, introduced by Zengin et al. [14], combines several adaptation approaches of different layers in one tool to cope with different concerns. It remains open how our approach fits into such a framework. In [9], we presented our results utilizing the partner synthesis by Wolf [11] as a basis. Instead, one could build our approach on top on other synthesis approaches, for instance [7].

7 Conclusion

In this paper, we described the problem of cost-minimal adaptation. We suggested a framework to express preference between different adapters based on cost models and cost functions. We discussed two cost models: Worst-case total costs, and worst-case average costs. We sketched the synthesis procedure for worst-case total costs.

We structure our intended future work as follows: (1) Identification and classification of further cost models, (2) solving partner and thus adapter synthesis for other cost models than \mathcal{T} , (3) evaluating the complete approach. For (1) we

² Tara is available at <http://service-technology.org/tara> [Last accessed on 2012-14-02]

plan to further investigate the literature on formalisms which cope with costs in general, for instance, weighted automata. Additionally, we plan to consider results from decision theory. To tackle (2), we will start to develop an algorithm which decides whether one adapter is to be preferred over an other. Then, we extend this result to synthesis of a partner. Part (3) could be realized by a case study with our prototype on real world services.

References

1. Alves, A., et al.: Web Services Business Process Execution Language Version 2.0. OASIS Standard, 11 April 2007, OASIS (Apr 2007)
2. Gierds, C., Mooij, A.J., Wolf, K.: Reducing adapter synthesis to controller synthesis. *IEEE Transactions on Services Computing* 99(PrePrints) (2010)
3. 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> [Last accessed on 2012-14-02].
4. Paoli, F.D., Lulli, G., Maurino, A.: Design of quality-based composite web services. In: *ICSOC*. pp. 153–164 (2006)
5. Papazoglou, M.: What’s in a service? In: Oquendo, F. (ed.) *Software Architecture, Lecture Notes in Computer Science*, vol. 4758, pp. 11–28. Springer Berlin / Heidelberg (2007)
6. Papazoglou, M.P.: *Web Services: Principles and Technology*. Pearson - Prentice Hall, Essex (Jul 2007)
7. Pistore, M., Traverso, P., Bertoli, P., Marconi, A.: Automated synthesis of composite *bpel4ws* web services. In: *Proceedings of the IEEE International Conference on Web Services*. pp. 293–301. *ICWS '05*, IEEE Computer Society, Washington, DC, USA (2005)
8. Seguel, R., Eshuis, R., Grefen, P.: Constructing minimal protocol adaptors for service composition. In: *Proceedings of the 4th Workshop on Emerging Web Services Technology*. pp. 29–38. *WEWST '09*, ACM, New York, NY, USA (2009)
9. Sürmeli, J.: Synthesizing cost-minimal partners for services. *Informatik-Berichte 239*, Humboldt-Universität zu Berlin (2012), <http://u.hu-berlin.de/suermeli-techreport>, [Last accessed on 2012-14-02], in publication queue
10. Van Der Aalst, W.M.P.: The application of petri nets to workflow management. *The Journal of Circuits Systems and Computers* 8(1), 21–66 (1998)
11. Wolf, K.: Does my service have partners? *LNCS ToPNoC 5460(II)*, 152–171 (Mar 2009), special Issue on Concurrency in Process-Aware Information Systems
12. Yellin, D.M., Strom, R.E.: Protocol specifications and component adaptors. *ACM Trans. Program. Lang. Syst.* 19, 292–333 (March 1997)
13. Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-aware middleware for web services composition. *IEEE Trans. Software Eng.* 30(5), 311–327 (2004)
14. Zengin, A., Marconi, A., Pistore, M.: Clam: cross-layer adaptation manager for service-based applications. In: *Proceedings of the International Workshop on Quality Assurance for Service-Based Applications*. pp. 21–27. *QASBA '11*, ACM, New York, NY, USA (2011)

Partner Synthesis for Data-Dependent Services

Christoph Wagner

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

Abstract. A service is *controllable*, if there exists a service with which it can interact properly. We sketch an approach to decide controllability for a certain class of services. Controllability is decided by synthesizing a service that controls the given service. For a class of services which abstracts from data, the synthesis problem is already solved. In this paper, we present an approach for a class of services that deals with data explicitly.

1 Introduction

A service is designed with the goal that it can interact with another service. A service may interact properly with one service but not interact properly with another service. For example, two services may end up blocking each other, making any further interaction impossible. The possibility of the occurrence of an error depends on both interacting services and in general can not be attributed to one service alone. However, at the time of design of a service, the other services the service will be interacting with typically are not known in advance. Nevertheless, we can check a fundamental property called *controllability* [7] when considering one service in isolation. A service is called *controllable*, if there is at least one other service it can interact with properly. Controllability can be decided by *synthesizing* a service that interacts with the given service properly.

The synthesis problem is solved for the finite automaton based service model used in [3,7]. This service model abstracts from data, i. e. messages are not distinguished by their content. For a more realistic model which takes data into account, the synthesis problem is still open. The goal of our work is to solve the synthesis problem for a service model that includes data.

In Sect. 2, we introduce the concept of a *partner* of a service. Section 3 presents an approach to synthesize a partner for a High-Level Petri net based service model. Section 4 concludes our work.

2 Partners

We represent a service by an *open net* [3] (Fig. 1). An open net is a Petri net with distinguished *interface places* that represent channels for asynchronous message exchange. We use High-Level Petri nets [2] so that data can be represented by coloured *tokens*.

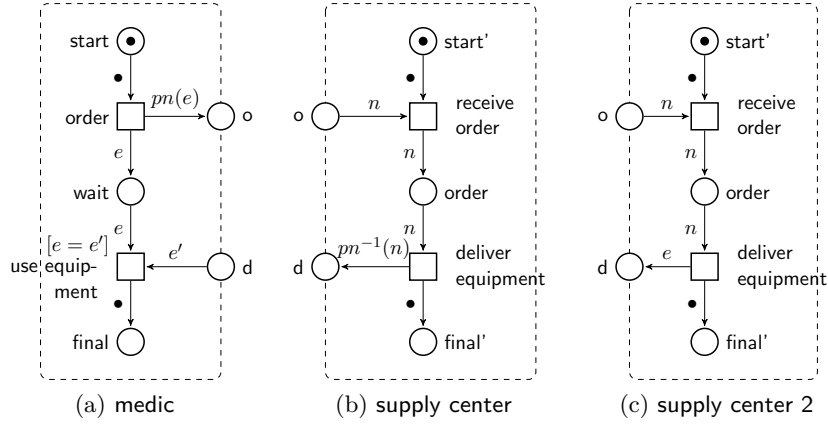


Fig. 1: Some open nets

The following example illustrates the interaction of a medic with a medical supply center from which the medic orders medical equipment. Both the medic and the supply center are services represented by the open nets in Fig. 1a and Fig. 1b. The composition ($\text{medic} \oplus \text{supply center}$) of the open nets *medic* and *supply center* is the Petri net we obtain by fusing the interface places *o* and *d*. The medic orders medical equipment by *firing* the transition *order*. Thereby the token \bullet is removed from place *start*. The variable e is assigned the equipment the medic orders, e. g., a syringe. Therefore, a token with value *syringe* is produced on place *wait*. The term $pn(e)$ evaluates to the product number of the syringe. A token with that value is produced on the output place *o*. This corresponds to sending a message to *supply center*.

The medic waits for an incoming delivery on place *d*. *supply center* receives the order message from *o* by firing *receive order* and n is assigned the product number of *syringe*. The transition *deliver equipment* takes the product number from place *order* and produces the equipment corresponding to that number (in this case *syringe*) on interface place *d*. Since the tokens on *wait* and *d* are equal, the *guard* $e = e'$ of transition *use equipment* is fulfilled. Therefore *use equipment* can fire and ($\text{medic} \oplus \text{supply center}$) reaches its *final marking*, i. e. the marking in which both places *final* and *final'* are marked with the token \bullet .

Two open nets N_1 and N_2 are called *partners*, if from each reachable marking of $N_1 \oplus N_2$ a final marking of $N_1 \oplus N_2$ is reachable. The *medic* and *supply center* are partners. The *medic* and *supply center 2* are not partners: When *deliver equipment* fires, any arbitrary equipment may be assigned to e which in general does not correspond to the product number assigned to n . Therefore, there is a marking reachable in ($\text{medic} \oplus \text{supply center}$) with a token with value *syringe* on *wait* and a token with another value (e. g. *stethoscope*) on *d*. This marking is a *deadlock*, because $e = e'$ is not satisfied and *use equipment* can not fire. On acyclic open nets, the reachability of a final marking is equivalent to deadlock freedom.

An open net N_1 is called *controllable*, if it has a least one partner. *medic* is controllable. In this example, we assumed that function pn has an inverse pn^{-1} . However, if we replace pn by a function pn' which is not bijective, *medic* becomes uncontrollable. In that case, a supply center can not infer the equipment the *medic* is waiting for from the product number. Therefore, it is not possible to guarantee that $e = e'$ is always satisfied and the final marking will be reached.

In the next section, we sketch an approach to synthesize a partner for a specific class of open nets.

3 Partner Synthesis

In this section, we sketch a partner synthesis algorithm for a given open net N . We consider a class of Petri nets where the domain of the variables and the colours of the tokens is infinitely large. The guards are denoted in a subset of first order logic that contains boolean algebra and quantifiers. For computational reasons, we assume that this subset is decidable (like e. g. Presburger arithmetic [5]). We also assume that N is acyclic and the number of tokens on each place is at most one. Therefore, the length of each path in the state space of N is bounded by some number k .

Due to infinitely many colours, the state space of N is infinitely large. Therefore the synthesis algorithm for finite state services given in [7] can not be applied to our service model. Nevertheless, conceptually, our synthesis algorithm follows a similar approach. Therefore, we briefly outline the approach used for finite state services: First, an *over-approximation* of the partner of N that will be synthesized, i. e., a service that is guaranteed to contain a partner as a sub-graph, is generated. Then certain states are removed from the over-approximation iteratively. The iteration is repeated as long as the composition of the given service and the over-approximation contains a deadlock. Eventually, two cases may occur: 1. The composition is deadlock-free. Then the remaining sub-graph of the over-approximation is a partner. 2. Every state has been removed. Then N is not controllable.

Now we give an overview of our synthesis algorithm. The details will be explained later by example. First, we construct an over-approximation S_0 of the partners of N . S_0 is a prefix of depth k of an infinite tree-like open net U we call *universal environment*. Then we iteratively add guards to S_0 . Adding a guard corresponds to removing states from the over-approximation in the finite case. The iteration is repeated until no deadlock is reachable in the composition of N and S_0 . Each time a guard is added, some of the deadlocks of the composition become unreachable. Each iteration step may also introduce new deadlocks which will then be eliminated in the next iteration step. If (and only if) N is controllable, the composition will eventually be deadlock free and the modified S_0 is a partner.

Now we show the derivation of a partner of *medic* from Fig. 1a. We assume that the colours of *medic* are integers and pn is the bijective function with $pn(x) = x + 1$. Therefore, *medic* can be expressed with Presburger arithmetic.

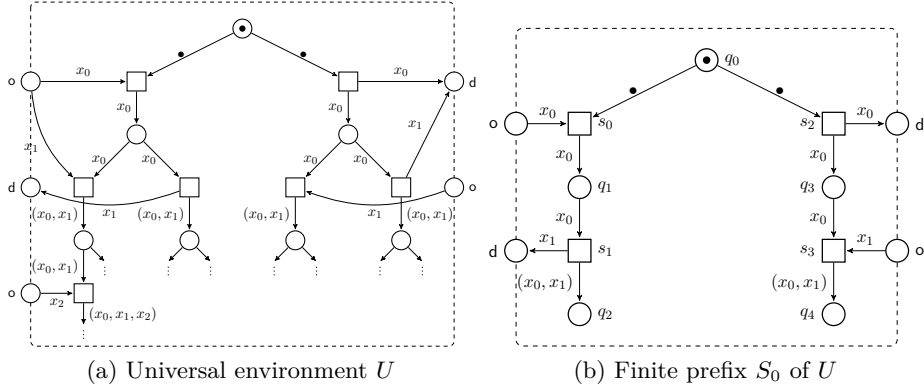


Fig. 2: Universal environment and its prefix. Interface places o and d are depicted multiple times to improve readability.

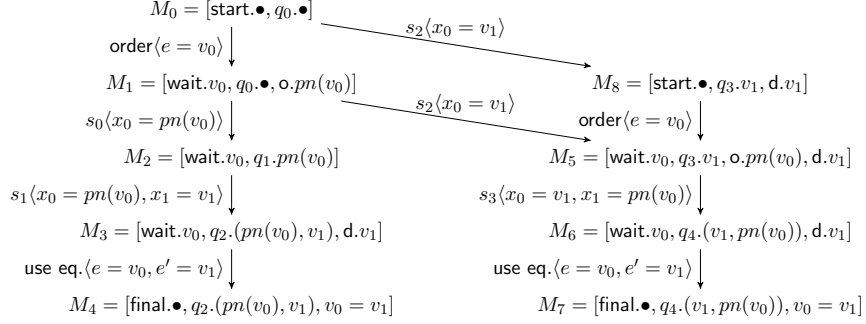
Fig. 2a shows the universal environment of *medic*. U is an infinite open net that has the inverse interface of N . U has a regular tree-like structure and can send and receive any (possibly infinite) sequence of messages. Therefore, U is an over-approximation of every partner of N . U stores every message sent or received from N . Each of the variables x_0, x_1, \dots corresponds to the value of a message. These variables will be used in the guards which we will add later on. Since U is infinitely large, it is not suited for computational methods. Due to the acyclicity of N , the number of messages N can send and receive is bounded by a number k . Therefore, the prefix of U of depth k is still an over-approximation of the partners of N .

In the example, k is 2. In this particular case, we can even remove the branches of U which send or receive two messages on the same interface place without destroying the over-approximation property. This is possible because *medic* sends or receives at most one message on each interface place. Thus, we get the prefix S_0 of U in Fig. 2b.

Now we iteratively derive a guard for each transition of S_0 . The transitions are processed in bottom-up order. Thereby, we obtain a sequence S_0, S_1, S_2, \dots of open nets with successively smaller reachability graphs.

Intuitively, a guard forbids a transition s of S_0 to fire in a certain firing mode if there is the possibility to reach a deadlock after s has fired in that mode. That way, deadlocks are successively removed from the composition.

Since a transition has infinitely many firing modes, we need a syntactical representation of all firing modes that may not lead to a deadlock. We derive the guard predicate that is assigned to each transition using a technique outlined in [6]. The technique is based on the *symbolic reachability graph* (SRG) of a High-Level Petri net. The symbolic reachability graph is a compact representation of the reachability graph that allows to represent a possibly infinite set of markings by a *symbolic marking*. Fig. 3 shows the SRG of the composition $\text{medic} \oplus S_0$. In a symbolic marking M , every value is represented by a term. Attached to M is a


 Fig. 3: Symbolic reachability graph of $\text{medic} \oplus S_0$

condition $COND(M)$ which restricts the set of valid assignments to the variables that occur in M . A marking m is reachable if and only if there is a symbolic marking M that evaluates to m for an assignment that satisfies $COND(M)$. Technically, during the construction of the SRG, $COND(M)$ is formed by the conjunction of the effects of every guard on a path to M . Each edge of the SRG is inscribed by a transition t and a *symbolic firing mode*. A symbolic firing mode of t assigns a term to each variable of t .

In our example, the integer that is chosen non-deterministically by `order` for e is represented by the variable v_0 in marking M_1 . Analogously, the integer sent by s_1 on d is represented by v_1 . The effect of the guard $e = e'$ of `use equipment` is represented by the condition $v_0 = v_1$ of the markings M_4 and M_7 . Every other symbolic marking has the condition *true*.

The method we use to derive the guard of a transition s of S_0 is inspired by Dijkstra's predicate transformer semantics [1]. The guard predicate can be regarded as the *weakest pre-condition* so that after firing of the transition s a specific post-condition holds. Here, the post-condition describes the assignments for which no subsequent symbolic marking evaluates to a deadlock. For each symbolic marking M we define a predicate $DF(M)$ that describes for which assignments of the variables v_0, v_1, \dots of the SRG M does not evaluate to a deadlock. $DF(M)$ is formed by the disjunction of the conditions of the successors of M . The SRG and the DF predicates are recalculated in each iteration step. We denote the iteration step by a subscript.

M_4 and M_7 are final markings. Therefore, $DF_0(M_4) \equiv DF_0(M_7) \equiv \text{true}$. Since M_3 has a successor marking only for those assignments of v_0 and v_1 with $v_0 = v_1$, we get $DF_0(M_3) \equiv v_0 = v_1$. Analogously, $DF_0(M_6) \equiv v_0 = v_1$. For every other symbolic marking M we get $DF_0(M) \equiv \text{true}$.

From every predicate $DF(M)$, we derive a predicate $DF'(M)$ which expresses the condition that M does not evaluate to a deadlock in terms of the variables used by the transition s of S_0 that precedes M in the SRG (s is unique because S_0 is a tree). By assigning $DF(M)$ as a guard to s , every evaluation of M which is a deadlock becomes unreachable. The relationship between the variables x_0, x_1, \dots

of S_0 and the variables v_0, v_1, \dots of the SRG is established by the the symbolic firing mode of s .

In the example, M_3 is reachable via exactly one path of the SRG. The last transition of S_0 on this path is s_1 with the symbolic firing mode $\langle x_0 = pn(v_0), x_1 = v_1 \rangle$. As stated above, $DF_0(M_3) \equiv v_0 = v_1$. With $x_0 = pn(v_0)$, $x_1 = v_1$ and the assumption that pn is injective follows that $v_0 = v_1$ is equivalent to $pn^{-1}(x_0) = x_1$. Formally, we express this transformation by universal quantification of v_0, v_1 :

$$\begin{aligned} DF'_0(M_3) &\equiv \forall v_0, v_1 : x_0 = pn(v_0) \wedge x_1 = v_1 \implies v_0 = v_1 \\ &\equiv x_1 = pn^{-1}(x_0) \end{aligned}$$

In words: $DF'_0(M_3)$ describes all assignments of x_0, x_1 for which the post-condition $v_0 = v_1$ is guaranteed to hold after the firing of s_1 in mode $\langle x_0 = pn(v_0), x_1 = v_1 \rangle$, regardless of which integers might have been non-deterministically chosen for v_0, v_1 .

The general form of a DF' predicate (for the special case that there is only one path in the SRG to M) is

$$\forall v_0, \dots, v_j : COND(M) \wedge x_0 = T_0 \wedge \dots \wedge x_n = T_n \implies DF(M)$$

where $\langle x_0 = T_0, \dots, x_n = T_n \rangle$ is the symbolic firing mode of the last transition s of S_0 on the path to M .

For M_4 , we obtain $DF'(M_4) \equiv true$. We add the conjunction $DF'(M_3) \wedge DF'(M_4)$ as a guard to the transition s_1 that precedes both M_3 and M_4 in the SRG. After adding this guard $x_1 = pn^{-1}(x_0)$ to s_1 , every deadlock in which q_2 is marked becomes unreachable.

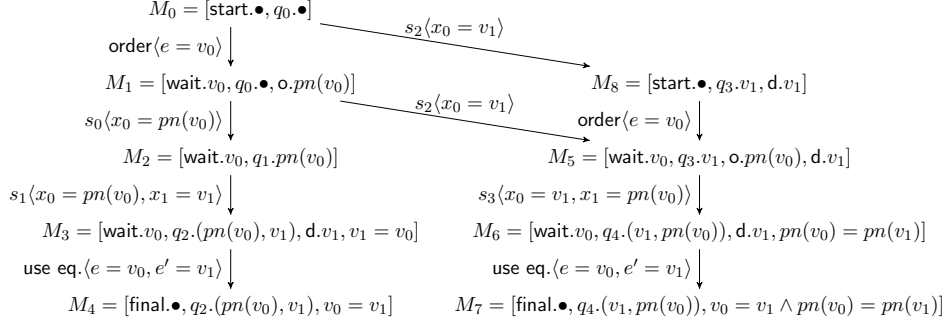
We repeat this procedure for s_3 . M_6 is reachable via two paths of the SRG (which differ only insignificantly). The last transition of S_0 on both paths is s_3 with firing mode $\langle x_0 = v_1, x_1 = pn(v_0) \rangle$. Analogously, we get

$$\begin{aligned} DF'_0(M_6) &\equiv \forall v_0, v_1 : x_0 = v_1 \wedge x_1 = pn(v_0) \implies v_0 = v_1 \\ &\equiv x_1 = pn(x_0) \end{aligned}$$

In a more general case, we may get a different predicate for each path on which a marking M is reachable. Then $DF'(M)$ is the conjunction of these predicates. With $DF'_0(M_7) \equiv true$ we get the conjunction $DF'_0(M_6) \wedge DF'_0(M_7) \equiv x_1 = pn(x_0)$. After assigning $x_1 = pn(x_0)$ to s_3 as a guard, no deadlock is reachable in which q_4 is marked.

Let S_1 be the open net derived from S_0 by adding the two guards to s_1 and s_3 . These guards introduce new deadlocks in which q_2 and q_4 are not marked, e. g. $[\text{wait}.0, q_3.3, \text{o}.1, \text{d}.3]$ is a deadlock in $\text{medic} \oplus S_1$ but not a deadlock of $\text{medic} \oplus S_0$. These new deadlocks will become unreachable in the next iteration step. In the SRG of $\text{medic} \oplus S_1$ (Fig. 4), M_3 has the condition $v_0 = v_1$ and M_6 has the condition $pn(v_0) = pn(v_1)$ due to the guards that were added. Therefore we obtain the predicates

$$\begin{aligned} DF'_1(M_2) &\equiv \forall v_0 : x_0 = pn(v_0) \implies \exists v_1 : v_0 = v_1 \equiv true \\ DF'_1(M_5) &\equiv \forall v_0, v_1 : x_0 = v_1 \implies pn(v_0) = pn(v_1) \equiv false \end{aligned}$$


 Fig. 4: Symbolic reachability graph of $\text{medic} \oplus S_1$.

Variable v_1 is existentially quantified, because v_1 is not yet defined in M_2 but will be created and chosen appropriately by s_1 in the step from M_2 to M_3 . Please note that existential quantifiers may only appear as a part of a DF' predicate.

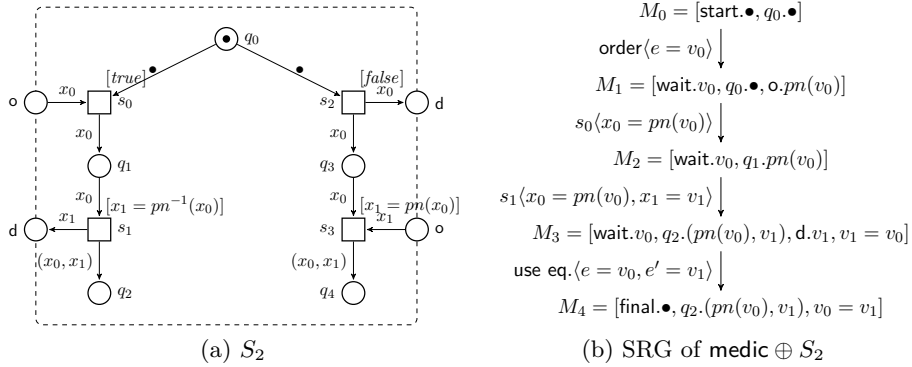


Fig. 5: Last iteration step of the partner synthesis

Eventually, by assigning the predicate $DF'_1(M_2) \equiv \text{true}$ to s_0 and $DF'_1(M_5) \equiv \text{false}$ to s_2 , we obtain the open net S_2 shown in Fig. 5a. By repeating our calculation on the SRG of $\text{medic} \oplus S_2$ (Fig.5b), we obtain $DF_2(M_0) \equiv \text{true}$. This indicates that no reachable marking in which q_0 is marked is a deadlock. Therefore, every deadlock has been eliminated from $\text{medic} \oplus S_2$ and S_2 is a partner of medic by definition. Please note that S_2 is very similar to N_2 from Fig. 1b. In general, the last open net S_i of the sequence is a partner of N iff for every symbolic marking M in which the root place q_0 is marked the predicate $DF_i(M)$ is fulfilled for every assignment that fulfils $COND_i(M)$.

4 Conclusion

We sketched an algorithm to synthesize a partner of a service represented by a High-Level Petri net. Currently, our approach is limited to acyclic services. We show a systematic approach to derive the relations between the values of incoming and outgoing messages that a service has to adhere to in order to be a partner of the given service. Since relations are denoted in first-order logic which is undecidable in the general case, we rely on an oracle to decide controllability. For a decidable theory like presburger arithmetic [5], controllability can be effectively computed. The technical details of the approach are not yet fully worked out.

Lohmann et al. [4] sketch a different approach to synthesize a partner for a High-Level Petri net based service. They use a symbolic representation for markings similar to ours. Their work focuses on the construction of the structure of the partner and only briefly discusses the derivation of the predicates. They give an ad-hoc explanation of the construction of the predicates for a particular example but do not describe a general derivation method. In particular, values that are non-deterministically chosen by the service are not treated.

In contrast to their approach, our approach does not consider structural aspects of the partner synthesis at all. All information concerning the behaviour of the partner is encoded by the guards. The structure of the partner is chosen in a generic way. However, our approach can handle values that are non-deterministically chosen by the service.

In our future work we aim at extending our approach to cyclic services. In this scenario it may be advantageous to choose a specific structure for the synthesized partner. E. g. case distinctions for specific values should result in a distinct node for each case. With a structure reflecting the behaviour of the service, it will be easier to identify isomorphic branches of the structure and nodes that can be combined without changing the behaviour. In order to ensure termination of the synthesis algorithm in the cyclic case, it is necessary to guarantee that there will be only finitely many nodes after the nodes have been combined.

References

1. Dijkstra, E.W.: A Discipline of Programming. Prentice Hall, Inc. (October 1976)
2. Jensen, K., Kristensen, L.M.: Coloured Petri Nets - Modelling and Validation of Concurrent Systems. Springer (2009)
3. Lohmann, N., Massuthe, P., Wolf, K.: Operating guidelines for finite-state services. In: ICATPN 2007, Siedlce, Poland, June 25-29, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4546, pp. 321–341. Springer-Verlag (2007)
4. Lohmann, N., Wolf, K.: Data under control. In: AWPN 2011 (Sep 2011)
5. Presburger, M.: Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In: Comptes Rendus du Premier Congrès des Mathématiciens des Pays Slaves. Warsaw (1929)
6. Wagner, C.: A data-centric approach to deadlock elimination in business processes. In: ZEUS 2011, Karlsruhe, Germany. CEUR-WS.org (2011)
7. Wolf, K.: Does my service have partners? LNCS ToPNoC 5460(II), 152–171 (Mar 2009), special Issue on Concurrency in Process-Aware Information Systems

HarmonICS - a Tool for Composing Medical Services ^{*}

Dariusz Doliwa¹, Wojciech Horzelski¹, Mariusz Jarocki¹, Artur Niewiadomski²,
Wojciech Penczek^{2,3}, Agata Półrola¹, and Jarosław Skaruz²

¹ University of Łódź, FMCS, Banacha 22, 90-238 Łódź, Poland
{doliwa, horzel, jarocki, polrola}@math.uni.lodz.pl

² Siedlce University of Natural Sciences and Humanities, ICS,
3-go Maja 54, 08-110 Siedlce, Poland
{artur, jskaruz}@ii.uph.edu.pl

³ Institute of Computer Science, PAS, Ordona 21, 01-237 Warsaw, Poland
penczek@ipipan.waw.pl

Abstract. The paper presents the tool HarmonICS designed for automated composition of medical services and implementing our approach to description and composition of web services. HarmonICS enables arranging sequences of services to satisfy a user's request specified by a query. The query language is rich enough to express requirements on the timing and the ordering of services used.

1 Introduction and Related Work

We present a new tool for automated composition of web services (WS) related to the medical domain. The tool implements our original approach [7] to WS composition, based on introducing a uniform semantic description of services, an object model for the problem, and applying a multi-phase composition supported by model checking methods. The planning process aims at satisfying a user's goal, specified in a declarative language, which enables not only to express features of the objects, but also requirements on the timing and ordering of services occurring in the plan.

The WS composition problem is a very important subject of research for which many various solutions exist. The simplest ones are based on explicit state space search algorithms [16], while more advanced ones employ a graph-based planning [5], logic programming [14], an AI planning [13, 11], model checking methods [10, 12], and genetic algorithms [3]. Vitvar et al. [17] proposed a solution based on WSMO/WSML [15] formalisms. While the fundamental ideas of their concepts seem similar to ours, it is important to mention that our approach is simpler and thus much easier to implement.

Our considerations follow that of Ambroszkiewicz [1], which provides a specification of an automatic composition system based on a multi-phase composition and uniform semantic descriptions of services. However, several extensions like enriched descriptions of services or a hierarchic organisation of services and objects they operate on, have been additionally designed. Doing all that, we keep the semantic base as simple as possible, which aims at enabling a translation of the WS composition problem to a problem solvable by means of efficient methods and tools from other domains.

^{*} Partly supported by National Science Centre under the grant No. 2011/01/B/ST6/01477.

The first “general” implementation of our approach (system Planics) was described by Doliwa et al. [8]. The tool Harmonics to be presented here is, on one hand, an extension of Planics due to incorporating new theoretical solutions, while on the other hand it is its specialization to a particular domain. In addition to the SAT-based planning method inherited from Planics, Harmonics offers also a new specialized SMT-based solution. The SMT-based concrete planner has been developed in response to insufficient performance of the previous solution in some particular cases. The bottleneck was a translation of TADDDPA¹ to SAT in the presence of a large number of conditions imposed on the variables, especially these “expensive” ones, e.g., using modulo operator.

In addition to introducing the SMT-based planner, our contribution consists in developing several extensions of the underlying formalisms, which are discussed in the next section together with the theoretical background of our approach. The rest of the paper is organized as follows. Sec. 3 introduces the main features of our solution, and gives an overview of the system implementation. Finally, a summary and a comparison between Harmonics and Planics are provided in Sec. 4.

2 Theory behind Harmonics

Our approach to automated composition of WSs is based on introducing a unified semantics for functionalities offered by services. A service is understood as a function which transforms a set of data into another set of data. The sets of data, i.e., inputs and outputs of services, are described in terms taken from a “dictionary” of types, introduced by an appropriate *ontology*. Each ontology follows the standard object model with classes, objects as their instantiations, and attributes as their components. More precisely, both the services and the items they operate on are organized into a multiple inheritance hierarchy of types, the top of which is composed of the following classes: *Thing* of no attributes and its descendants: *Object*, *Service*, and *Trace*.

Below we explain the meaning of the branches rooted at the three descendant classes of *Thing* mentioned above. An example fragment of the ontology tree is presented in Fig. 1, where the solid arrows stand for the inheritance relation. We embed our explanation in the context of medical services considered in the paper. Therefore, we use the names from Fig. 1 as examples, but, in fact, all the nodes below *Object*, *Service*, and *Trace* are **domain-dependent**, and even for a “fixed” domain they can vary depending on the modelling assumed.

The branch of classes rooted at *Object* introduces “types of beings”, *Patient*, *Diagnosis*, *Therapy*, that are necessary to specify what the services operate on, together with the “features” of these beings expressed by their attributes. For example the class *Patient* has the attributes *First_name*, *Last_name*, *Address*, *Date_of_Birth*, *Diagnosis* etc. having a clear intuitive meaning.

The branch of classes rooted at *Service* introduces types of services - *Visit*, *Treatment*, *Registration*. The attributes of the class *Service*, inherited by all its descendants, are as follows : *in*, *out*, *inout*, *preCondition*, and *postCondition*. The first three of them are aimed at listing objects (classified by names and types, similarly to subprogram parameters) which, respectively, are required to execute the service (*in*), are produced by

¹ Timed Automata with Discrete Data and Parametric Assignments

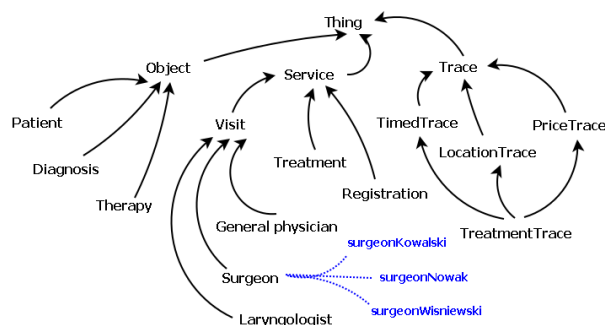


Fig. 1. A fragment of the ontology used by HarmonICS

the service (*out*), and are taken as an input and returned modified (*inout*). The aim of *preCondition* and *postCondition* is to specify respectively the conditions which should be satisfied by the “input” objects to have the service started and the conditions the “output” object satisfy after the service has been executed. For example we can express that the services of the type *Visit* modify an instance of *Patient* by placing $p:Patient$ in the *inout* list, and require a visit to result in a diagnosis by placing $isSet(p.Diagnosis)$ in the *postCondition*. The values of the attributes common for all the services of a given type are specified in a special instance of the corresponding class, called an *abstract service*. The *concrete services* of a given type (instances of the class representing this type) can introduce their own extensions to the attributes above. For example, the concrete service *GeriatricianSmith* of type *GeriatricianVisit* can require his patients to be older than 85 by extending the common *preCondition* by $p.Date_of_Birth < "1927-12-31"$. A more detailed description of the above elements of ontologies can be found in [7].

A new concept introduced to Harmonics is shown as the third branch (from the left) of the inheritance tree in Fig. 1, i.e., the class *Trace* and its descendants. The instances of the above classes, called *Traces*, are “virtual products” (not corresponding to real-world beings). The *out* list of each service contains exactly one element corresponding to a trace, e.g. $t:Trace$. The main motivation behind *Traces* is a need for dealing with imperative queries, when the user precisely points out to the types of services to be executed, just like in most of the considered medical scenarios. Moreover, *Traces* enable to associate the services types (and also their concrete instances) with attributes like price, duration, location or quality, without affecting the existing structure of the language.

The attributes of the class *Trace* are the following: *level*, *block*, *serviceType*, and *serviceName*. The first two of them aim at storing an information about a position of the service in the scenario generated, while the next two are used to identify the service executed. For example, if the service *GeriatricianSmith* is the first of the scenario, then the attributes of the trace t produced by this service are $t.level=0$, $t.block=0$, $t.serviceType="GeriatricianVisit"$, and $t.serviceName="GeriatricianSmith"$.

Traces enable to express certain requirements on sequences of services, both on the level of service descriptions and while specifying users goals. For example, *SurgeonVisit* can require seeing a general practitioner earlier by including $x:Trace$ in its *in* and $x.ServiceType="GPVisit"$ in its *preCondition*. The descendants of *Trace* can intro-

duce additional information. For example, a class *TimedTrace* with the attributes *start* and *stop* brings in time of the service execution. *PriceTrace* with the attribute *price* provides information about the service price, while *LocationTrace* with the attribute *location* introduces the information about the place where the service operates.

The user specifies its goal in the form of a *user query*, which defines what he “possesses” (the *initial world*) and what he “wants to possess” (the *effect world*), together with these of their features that are of his interest, using names of the classes from the branch rooted at *Object* and names of their attributes to this aim. For example, the user John Gold can specify that possessing “nothing”, he wants to possess the object *p:Patient* with *p.First_name="John"* and *p.Last_name="Gold"*, which means that he wants to become a patient. The goals can be also specified in terms of traces (i.e., names of the classes from the branch rooted at *Trace*). This enables to express that the user wants the scenario generated to contain a service of certain type (e.g., by specifying that the effect world should contain *t1:Trace* such that *t1.ServiceType="SurgeonVisit"*) or a service of a concrete provider (e.g., by extending the above requirement by adding *t1.ServiceName="SurgeonSmith"*). Traces enable also to require a given ordering of services in a plan (by the use of the *level* attributes), or a given ordering of groups of services (by the use of the attribute *block*) - for example, one can require the effect world to contain *t1,t2:Trace* such that *t1.ServiceType="GPVisit"*, *t2.ServiceType="SurgeonVisit"* and *t2.level<t1.level*, i.e., to see a GP after seeing a surgeon. Using other types of traces enables to influence the cost of services proposed, their time, location etc.

Our project follows the idea of separating two phases of the planning process. The first phase of searching for a sequence of services whose execution satisfies the user’s goal is called the **abstract planning**. It involves searching for sequences of **types** of services, which can transform the set of objects of the initial world into the set of objects of the effect world. The user’s query is redefined to discard all the expressions involving concrete values of the attributes. For example, *p.Last_name="Gold"* is replaced by the requirement that the corresponding attribute is assigned a value - *isSet(p.Last_name)*. The abstract planning process is based on the bounded backward search algorithm, which starts from the final world and matches abstract services (special instances of service classes described before), which are capable to produce a desired set of objects (with the appropriate attributes set), building this way a graph whose nodes are sets of objects, and the edges are labelled with service types. This “preliminary” phase enables to limit the number of concrete (real-world) services considered while creating the final scenario as only these of appropriate types will be taken into account. Obviously, in the case of queries involving traces the role of the abstract planning phase is limited. The user can specify fragments of the abstract plan “by hand”, using the appropriate attributes of traces. The next phase of the planning process, called the **concrete planning**, aims at finding a sequence of **instances of service types** (concrete services) corresponding to an abstract plan obtained from the previous phase. Contrary to the abstract planning, this phase takes into account all the requirements specified in the query, i.e., also these involving concrete values of attributes. The planning process exploits an SMT-based model checking procedure, which is discussed in the next section.

3 Main Features and Implementation of HarmonICS

HarmonICS is a scheduling system that has been implemented for the Rehabilitation and Cosmetology Centre (CRiK) in Poland. The centre offers various types of medical services for its direct clients as well as for other medical facilities. The definitions of needs and possibilities of satisfying them are specified by a relatively complicated semantics. Additionally, availability of certain resources in many cases can be determined only dynamically, by querying external independent data sources. Before implementing HarmonICS, due to the lack of IT solutions, the querying process was performed in an “unformalised” way, i.e., by phone or by e-mails. The knowledge obtained this way could not be processed automatically. The most important conclusions from the analysis of the functioning of CRiK, and from the users’ expectations are as follows:

- The main goal of the system is to make the scheduling of treatments easier and more convenient, and also to automate some internal procedures,
- The most common case is to schedule a series of treatments w.r.t. patient preferences and resources restrictions,
- The single steps of the whole process can be realized by various service providers cooperating with CRiK,
- Some aspects of the abstract and concrete planning processes should be significantly adapted to meet the specific CRiK requirements.

The implementation of HarmonICS, presented in the next part of this section, was designed to satisfy the above requirements.

The overall view of the HarmonICS components is presented in Fig. 2. The ontology designed for CRiK was discussed in Sec. 2. The main software components of the system are as follows: the Repository, the Graphical User Interface (GUI), and the Planner. The aim of the Repository is to store information about the available services and their types (according to the ontology). Currently, it is implemented on the top of jUDDI - a popular UDDI implementation. GUI is a GWT web application that enables the user

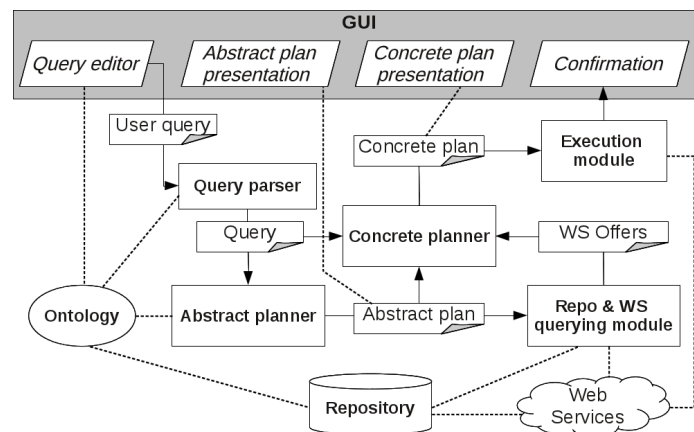


Fig. 2. The HarmonICS overview

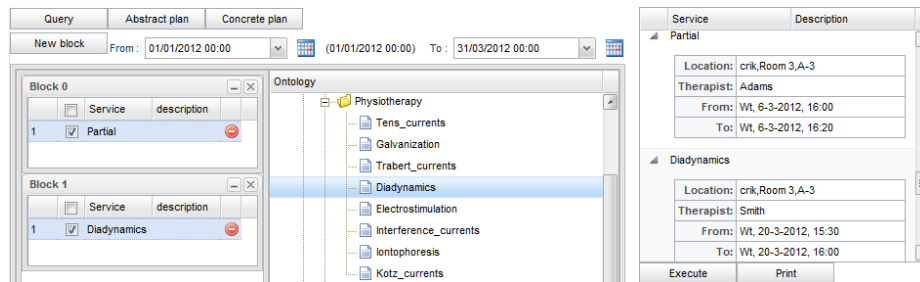


Fig. 3. On the left: the query editor, on the right: a fragment of a concrete plan

interaction with the system components. The Planner is a set of tools (represented by rectangles in the figure) for processing user queries (*Query parser*), creating plans (*Abstract and Concrete planners*), and interacting with the repository and with the web services (*Querying and Execution modules*). The rectangles with the right-bottom corner wrapped depicted in the figure correspond to the internal system objects. They are labels of the solid arrows which stand for a flow of objects. The dashed lines represent making use of some resource by a software component.

Let us now follow an example scenario, while giving more details concerning the implementation of individual components. First, using the *Query editor* (see Fig. 2 and Fig. 3), the user introduces a request, e.g., *I want to take a partial massage, once a week, for 10 weeks, and then a series of 5 diadynamics, every 2 days*. The user drags arbitrary services from the ontology tree at the right hand side and drops them to blocks of a plan at the left. The blocks are intended to enforce the order of an execution of the services. Each block of services is scheduled for execution when all of the services from the previous block have been completed. Putting some services in the same block means that they can be executed in an arbitrary order. Each of the services chosen can be parameterized by assigning a set of constraints, e.g., repeat conditions or specific requirements on the service date, time or location. The user should also specify an acceptable timing interval, providing the earliest start date and the latest end date of the whole sequence of services. The editor enables to hide the query language from the user offering a friendly and intuitive interface instead. The query of a formal syntax is produced in an automated way. For example, considering time interval from January, the 1st to the end of March of the current year, the formal query is as follows: *FROM null WHERE null TO repeat(t0:TreatmentTrace, 10, every 1 weeks), repeat(t1:TreatmentTrace, 5, every 2 days) WHERE _globalStart= "2012-01-01 00:00" and _globalStop= "2012-03-31 23:59" and t0.serviceType = "PartialMassage" and t0.block = 0 and t1.serviceType = "Diadynamics" and t1.block = 1*. As it is easy to see, the requirements on the service types and their ordering are expressed in terms of traces.

It should be mentioned also that the *repeat* statement is one of the novelties (comparing with [7, 8]) introduced to respond to the specificity of the domain, where the common case is to repeat some kind of treatment a number of times. Optionally, the repeat period can be given, just like in the example above. This construct makes editing of the query easier, as the user does not need to choose a service several times if he wants to repeat it. The user query is then processed by the *Query parser*, transformed to

the internal representation, and made available to the *Abstract planner* and the *Concrete planner* (see Fig. 2).

The *Abstract planner* uses the knowledge from the OWL ontology and the query (rebuilt by discarding the concrete values as described before) for generating abstract plans, which are visualized and presented as sequences of service types. The user is asked to choose one of them to be concretized. Due to the fact that specificity of the area implies the queries to have a more imperative nature than in a typical case (the users typically enumerate directly the services they want to use) the role of the abstract planning is not so fundamental. However, using the knowledge from the ontology can introduce to the plan services not required directly by the user. In our example, the abstract planner returns the sequence of service types: *Registration*, 10 occurrences of *PartialMassage*, and 5 occurrences of *Diadynamics*. The *Registration* service, although not required directly by the user, is necessary in the plan as it “produces” a *Patient*, required by all the treatment services but not existing in the initial world.

Next, basing on the abstract plan and the user query, the *Repo & WS querying module* (RQM for short) examines the repository for the registered web services realising the types of services from the abstract plan. In our example the repository will be asked: “Give addresses of all the services of the type *PartialMassage*, and of the type *Diadynamics*”. After getting an answer the RQM queries for offers the web services obtained (where by an offer we mean a service’s declaration to execute under certain conditions). In our example the services will be asked: “Give the dates and time, between 2012-01-01 00:00 and 2012-03-31 23:59, when the treatment procedure can be performed” (the query contains no other constraints than these on the time period to be considered).

The next step is to run the *Concrete planner*. Its input are as follows: the (original) query, the abstract plan chosen to be concretized, and the offers collected for this plan (a single offer corresponds to a possible realisation of a single step of the plan, i.e., executing one service of a given type). It is possible to run this planning using one of the two methods. The first one, inherited from Planics [8], is based on a satisfiability checking (SAT). The new one is realized by a translation to an instance of the SMT [2] problem. An SMT-solver checks satisfiability of the formula which is the conjunction of the disjunctions representing particular offers, and an expression encoding the conditions specified in the query (e.g., repeat period constraints) and resulting from the abstract plan (e.g., the order of services). If this SMT instance is satisfiable, then a sequence of concrete services, whose execution satisfies the user’s goal is decoded from the valuation returned by the solver. Going into more details, the attributes of the objects and the traces are encoded as SMT variables, and their values are mapped into natural numbers. For example, date-time values from our query are encoded as follows: the beginning of the considered period of time, the *_globalStart* value, is mapped to 0. All the date-time values are then related to the *_globalStart* value, according to a certain time scale. Currently the time scale is 5 minutes, which means that the value 10 represents the point in time 50 minutes after *_globalStart*. The SMT instance is encoded (using our original library) in SMT-LIB2 [4] format, which enables to use any compatible SMT-solver. In the current version we make use of the Z3 [6] solver.

In the case of typical queries, involving from a few to several dozens of services, and from several hundreds to about 20000 offers, the total time of computations can

Offers	Time [s]		Mem [MB]		Offers	Time [s]		Mem [MB]	
	plain	interval	plain	interval		plain	interval	plain	interval
5866	3.06	4.67	115.07	117.19	8281	2.38	5.18	90.13	91.31
10848	9.17	11.08	323.39	324.59	12697	5.09	8.92	187.42	188.48
13241	15.27	17.92	475.80	477.68	16561	8.33	19.60	289.37	291.95
17721	27.41	33.44	834.35	835.54	21253	12.54	19.21	448.32	449.90

Table 1. Time and memory consumption of the concrete planner. On the left - the plan of depth 16 for the scenario: *registration, 10 massages, and then 5 diadynamics*, on the right - the plan of depth 24 for the scenario: *registration and 23 bioptrons*. The columns headed *interval* contain the results with additional constraints on the repeat frequency.

vary from a few seconds to about 30 seconds. The concrete planning phase seems to be the most time- and memory-consuming element. Table 1 displays some statistics of our SMT-based solution. The columns headed *interval* of the left table contain the results for the query being the working example of this section.

The concrete plan computed is visualized (see Fig. 3) and presented to the user. If the user accepts it, the *Execution module* invokes the services. Again, the specificity of the domain makes things simpler: an execution of a service consists in scheduling an appointment only, so no execution engine is necessary. Obviously, always something unexpected can happen. At the moment we follow the simple transactional policy: when any step of the plan could not be successfully executed, we cancel all of the already scheduled appointments, and the user can repeat either the WS querying and concrete planning phases, or the whole planning procedure.

4 Final Remarks

Harmonics is a specialized implementation of the concept which can be applied to various domains, enabling to build an integration system for distributed services of a common characteristic (e.g., transport, accomodation, reservation in time). More generally, a similar system can be implemented in every domain in which we have to plan an access to some resources with an independent management and optimize the plan by customized quality measures.

Comparing Harmonics to its ancestor Planics [8], we can point out to an easier and more natural handling of relations between services thanks to the concept of *Traces*. Another advantage appears in translating semantics from different IOPR [9] services ontology - it is simpler and more natural. On the other hand, a modular architecture of the system allows to take advantage of a new and more efficient planning solution based on SMT-solvers. The efficiency follows not only from applying the SMT-based technique, but also from the extended role of the querying module - the concrete planner deals now only with these of the parameters whose exact values cannot be determined by querying concrete services. Moreover, the planning mechanism related to the time have been improved. A further contribution of Harmonics is in an extended language of queries, enabling to express more requirements occurring in practice. Its new elements are not only these which follow directly from introducing traces (like specifying requirements on ordering of services or their groups, or time or price of particular services), but also expressions enabling to require repetitions of services (the *repeat* statement) and summary constraints on the whole plan (e.g., *_globalStart*, *_globalStop*).

References

1. S. Ambroszkiewicz. *EnTish: An Approach to Service Description and Composition*. ISBN 83-910948-7-1, ICS PAS, Ordonia 21, 01-237 Warsaw, 2003.
2. A. Armando, J. Mantovani, and L. Platania. Bounded model checking of software using SMT solvers instead of SAT solvers. *Int. Journal on Software Tools for Technology Transfer*, 11(1):69–83, 2009.
3. S. Bahadori, S. Kafi, K. Zamani far, and M. R. Khayyambashi. Optimal web service composition using hybrid GA-Tabu search. *Journal of Theoretical and Applied Information Technology*, 9(1):10–15, 2005.
4. C. Barrett, A. Stump, and C. Tinelli. The SMT-LIB Standard: Version 2.0. In *Proc. of the 8th International Workshop on SMT*, 2010.
5. A. Blum and M. L. Furst. Fast planning through planning graph analysis. *Journal of Artificial Intelligence*, 90(1-2):281–300, 1997.
6. L. M. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *Proc. of TACAS'08*, volume 4963 of *LNCS*, pages 337–340. Springer-Verlag, 2008.
7. D. Doliwa, W. Horzelski, M. Jarocki, A. Niewiadomski, W. Penczek, A. Pórola, and M. Sreter. Web services composition - from ontology to plan by query. *Control & Cybernetics*, 40(2):315–336, 2011.
8. D. Doliwa, W. Horzelski, M. Jarocki, A. Niewiadomski, W. Penczek, A. Pórola, M. Sreter, and A. Zbrzezny. PlanICS - a web service composition toolset. *Fundamenta Informaticae*, 112(1):47–71, 2011.
9. A. Gómez-Pérez and J. Euzenat (Eds.). The semantic web: Research and applications. In *Proc. of the 2nd European Semantic Web Conference*, volume 3532 of *LNCS*. Springer, 2005.
10. S. Hoelldobler and H. P. Stoerr. Solving the entailment problem in the fluent calculus using binary decision diagrams. In *Proc. of the Workshop on Model Theoretic Approaches to Planning at AIPS2000*, pages 18–25, 2000.
11. J. Hoffmann, I. Weber, J. Scicluna, T. Kaczmarek, and A. Ankolekar. Combining scalability and expressivity in the automatic composition of semantic web services. In *Proc. of the 8th Int. Conf. on Web Engineering (ICWE'08)*, pages 98–107. IEEE Computer Society, 2008.
12. H. Kautz and B. Selman. Blackbox: A new approach to the application of theorem proving to problem solving. In *Working notes of the Workshop on Planning as Combinatorial Search, held in conjunction with AIPS-98*, 1998.
13. M. Klusch, A. Gerber, and M. Schmidt. Semantic web service composition planning with OWLS-XPlan. In *Proc. of the 1st Int. AAI Fall Symposium on Agents and the Semantic Web*, pages 55–62. AAAI Press, 2005.
14. S. R. Ponnekanti and A. Fox. SWORD: A developer toolkit for web service composition. In *Proc. of the 11st Int. World Wide Web Conference (WWW'02)*, 2002.
15. D. Roman, J. de Bruijn, A. Mocan, H. Lausen, J. Domingue, C. Bussler, and D. Fensel. WWW: WSMO, WSML, and WSMX in a nutshell. In *Proc. of the 1st Asian Semantic Web Conference (ASWC'06)*, volume 4185 of *LNCS*, pages 516–522. Springer-Verlag, 2006.
16. M. Sheshagiri, M. desJardins, and T. A. Finin. A planner for composing service described in DAML-S. In *Proc. of Workshop on Planning for Web Services, Int. Conf. on Automated Planning and Scheduling*, pages 28–35, 2003.
17. T. Vitvar, A. Mocan, M. Kerrigan, M. Zaremba, M. Zaremba, M. Moran, E. Cimpian, T. Haselwanter, and D. Fensel. Semantically-enabled service oriented architecture : concepts, technology and application. *Service Oriented Computing and Applications*, 1:129–154, 2007.

Automated Composition of Timed Services by Planning as Model Checking

Daniel Stöhr, Sabine Glesner

Technische Universität Berlin, Chair Software Engineering for Embedded Systems,
daniel.stoehr@tu-berlin.de, sabine.glesner@tu-berlin.de,
www.pes.tu-berlin.de

Abstract. Techniques of automated service composition can shorten development time by generating a concrete service composition out of a set of abstract composition requirements. However, no existing fully automated approach is able to deal with timed services and timed composition requirements. In this work, we propose an approach for the automated composition of timed services, represented as timed i/o automata, by adapting the AI planning method Planning as Model Checking. Thus, the concept of automated service composition can be used in domains with real-time requirements. As case study, we model a system where medical devices need synchronization during surgery.

Keywords: automated service composition, timed services, real-time, timed i/o automata, planning as model checking

1 Introduction

Designing controller programs coordinating distributed components in a safety- and time-critical environment, e.g., for synchronizing medical devices, is a very complex and time-consuming task. While keeping development time short, the software engineers have to assure that the overall system fulfills functional and safety-critical requirements. These opposites yield the need for automated and scalable tools supporting the development process. In our domain, such tools have to produce correct results and have to deal with nondeterminism and time as part of the service behavior. Speaking in terms of Service-oriented Architectures (SOAs), which are an uprising paradigm in those domains, the problem of designing a controller corresponds to the problem of finding a suitable orchestrator to compose a given set of services.

Therefore, we propose an approach for the automated composition of timed services including real-time properties as composition requirements. As to the authors' knowledge, no existing fully automated approach for service composition is able to deal with those requirements. To realize our approach, we describe the behavior of the services as *timed i/o automata* [VL92] and the orchestrator as an automaton handling the input and output actions of the original automata. Thus, we adapt the AI planning method *planning as model checking* [GT00] to

realize the automated composition process, by bringing real-time into the existing theory.

Moreover, as a part of future work, we want to implement a tool realizing the resulting composition algorithms. Such a tool can be used to shorten the development process for systems in our domain, because an initially correct controller model is generated where a hand-made model had to be created before. To discuss the requirements for our approach, we present a case study where medical devices have to be synchronized during surgery.

The rest of this paper is structured as follows. In Section 2 we shortly outline the concepts of automated service composition, timed i/o automata, and planning as model checking. They form the basis for our approach. Afterwards, in Section 3 we present our case study, our proposed approach and what extensions to the existing theory are required. In Section 4 we discuss related work. Finally, in Section 5, we conclude this paper and give an outlook on future work.

2 Background

In this section we introduce works upon which our approach is based. In Section 2.1, we shortly explain the concept of automated service composition and the decision for the underlying theory of our approach. Afterwards, in Section 2.2, we introduce timed i/o automata, used to formally represent the services to be composed. Finally, in Section 2.3 we present *planning as model checking*, which forms the basis for the composition process of our approach.

2.1 Automated Service Composition

In the context of our work, the term automated service composition denotes the process of generating an orchestrator for a set of services out of a set of composition requirements. An orchestrator is a central service within a service composition that communicates with the other services and directs messages between them in order to create the system behavior described through the requirements.

In [BP10] an exhaustive overview is given on automated composition approaches for web services. They present 27 approaches realizing different forms of automated service composition and compare them to each other with respect to a certain set of properties. In the following we outline the properties, which are relevant for our work.

Automation Describes the degree of automation, offered by an approach. We need a high degree of automation. Besides of a formal description of the composition requirements on a set of services, the user intervention shall be reduced to a minimum.

Nondeterminism An action may produce different nondeterministic outcomes. In our domain nondeterminism occurs, e.g., as device alarms.

Scalability Our approach shall deal with large and complex sets of services. Hence, we have to design our composition algorithms for high efficiency.

Correctness Compositions are guaranteed to be correct w.r.t. the composition requirements. Because our approach shall generate compositions for safety-critical domains, we have to assure the correctness of resulting compositions.

Based on the above-mentioned survey, we compared these properties to the different composition approaches. We decided that model checking based methods of automated service composition are most suitable to form the basis of our approach. These approaches are the only ones, fulfilling all these properties at once. However, none of the presented approaches includes real-time composition requirements.

2.2 Timed I/O Automata (TIOA)

To realize our approach, we need a suitable formalization to describe the behavior of the orchestrator and of the services to be composed. For that, we chose the theory of *timed i/o automata* (TIOA [VL92]). These are *finite automata*, whose actions are divided into input and output actions, so that we can describe the interface of our services. Moreover, we can express timed behavior over a set of clocks. Therefore, guards exist for transitions, and invariants for states.

In our case study (the synchronization of medical devices), we can model each device as a distinct automaton. Messages between connected devices are represented as input- and output-actions. With guards and invariants, we can express timed conditions on the interaction of our devices, e.g. when a device has to react within a certain timeframe.

An example for the graphical representation of TIOA is given in Section 3.1 where we present our case study.

2.3 Planning as Model Checking

In the following, we outline the AI planning method *planning as model checking* [GT00]. The underlying idea of this technique is to generate *plans* for a given *planning domain* by determining whether formulas are true in a model.

The planning domain is described through a model similar to *finite automata*. The planning problem is described in temporal logics, as a CTL formula, containing desired final states and constraints on the paths allowed in the planning domain. Solving the planning problem for a planning domain means finding paths leading from the initial state to the final states. Here, the problem is solved by lifting it to a model checking problem. The planning problem is expressed as a corresponding *kripke structure* and the plan is generated by checking whether suitable temporal logic formulas are true within it. For this purpose, an iterative algorithm checks paths in the structure against corresponding parts of the formulas.

The algorithm is based on *Binary Decision Diagrams* a data structure that can represent kripke structures as graphs representing boolean formulas (a common technique for solving model checking problems). The plan is iteratively built up as a BDD by comparing it to other BDDs (representing the domain and

requirements) and by performing transformations on it. It has been implemented within the *Model Based Planner* [MBP].

For our approach, we will translate the set of TIOA into a planning domain by building up the crossproduct. Thus, we transform the problem of automated composition into a planning problem.

3 Automated Service Composition with Real-Time Requirements

In the first part of this section we present a case study, demonstrating how our approach can be applied. In Section 3.2, we describe the workflow of our approach. In Section 3.3 we outline problems we have to tackle when extending the existing theory, by referring to our case study.

3.1 Case Study

We have investigated the requirements for our approach by creating a TIOA model of a use case where an x-ray device and an anesthesia machine ventilator need synchronization during surgery. The use case is described in [AGWL09], a work where the technical interoperability between medical devices is investigated. In this scenario, an x-ray image of a patient's chest had to be taken under general anesthesia. When the x-ray is performed, it must be ensured that the patient's lungs are empty in order to receive a clear image. Therefore, parameters of the anesthesia machine's ventilator are accessed by the controller, to trigger the x-ray exactly between two breaths. In Figure 1 we show a simplified version of the system model we created.

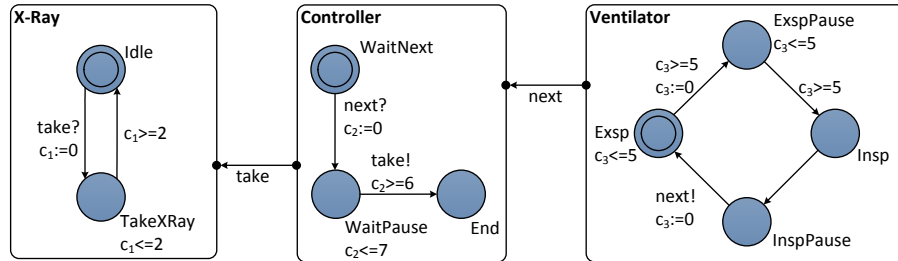


Fig. 1. synchronizing an x-ray and an anesthesia machine ventilator

The left automaton, **X-Ray**, describes the behavior of the x-ray device which, if triggered, needs 2 time units for taking an image. In its initial state, **Idle**, it awaits the reception of the input signal **take**. If the signal is triggered the automaton changes to the state **TakeXRay** and resets the clock **c1**. The guard of the transition back to **Idle** ensures that the x-ray stays in **TakeXRay** for at least 2 time units and the invariant ensures that the state is left after at most 2 time units.

The right automaton describes the ventilator machine which controls the breathing, i.e., the respiration cycle of the patient. In its' initial state **Exsp** the ventilator lets the patient breath out. The invariant and transition guard ensure, that the expiration phase lasts exactly 5 time units. When the state **ExspPause** is activated, the respiration pauses for 5 time units. Afterwards, the inspiration phase and pause take place, analogous to the expiration phase (here, we omitted the time constraints since they are not relevant for our use case). When **Exsp** is entered again, the signal **next** is emitted. That signal is the only possibility for other devices to synchronize with the ventilator.

Before describing the controller, we outline its behavior via CTL formulas (these will be the composition requirements for our approach). Firstly, we have a functional property (1) **AF TakeXRay** which says that the x-ray has to be performed somewhere during the controllers execution. Secondly, theres a safety property (2) **AG TakeXRay \rightarrow ExspPause** which describes that whenever the x-ray is exposed the ventilator has to be in the expiration pause mode.

Based on these requirements we can model the **Controller**. In its initial state **WaitNext** the controller waits for the signal **next** stating that a new respiration cycle begins. In **WaitPause** the controller is ready to trigger the x-ray (requirement 1) and waits for the right point in time to do so (requirement 2). The corresponding transition takes place after at least 6 time units. This is the point where the ventilator has entered **ExspPause** for sure. The invariant ensures that the exposure time of the x-ray does not overlap with the ventilator's inspiration phase.

3.2 The Approach

In this section we present our approach for realizing the automated composition of timed services. Therefore, we adapt the AI Planning method *planning as model checking* (Section 2.3). As language for our service models we have chosen TIOA (Section 2.2). We realize our approach by bringing real-time into the theory and by building a framework to make the theory compatible with Timed i/o Automata (inspired by the work discussed in Section 4). The workflow of our proposed approach is visualized in Figure 2.

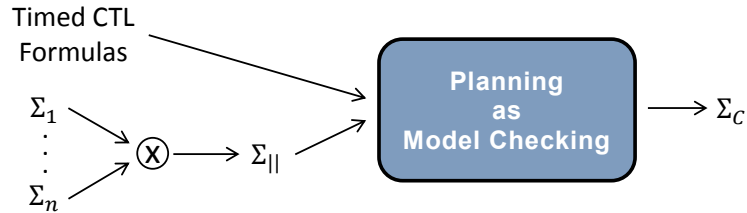


Fig. 2. Workflow of our approach for the automated composition of timed services

Initially, a set of TIOA $\Sigma_1, \dots, \Sigma_n$ describes the communicational behavior of our services, and a set of *Timed CTL formulas* [ACD93] describes functional and real-time composition requirements. In a first step, the parallel product $\Sigma_{||} = \Sigma_1 || \dots || \Sigma_n$ is built. In the sense of planning as model checking, $\Sigma_{||}$ leads to the *planning domain* and the formulas to the *planning problem*. In our example the crossproduct of the x-ray and ventilator automata are the planning domain and the CTL formulas (1) and (2) are the planning problem.

Afterwards, the algorithms of planning as model checking are applied to solve the planning problem by identifying all paths fulfilling our requirements. This gives us the *Control Automaton* Σ_C handling the inputs and outputs of the original automata, so that the required overall behavior of the composition is assured. By the means of service composition, Σ_C is an *orchestrator*.

Since the existing theory and implementation of planning as model checking can only deal with untimed domains and 'simple' CTL formulas, we have to make it capable of dealing with timed domains and Timed CTL formulas. We describe the problems, we await for the extensions, in the next section.

3.3 Extending the Existing Theory

In this section we sketch the problems we have to tackle for extending the existing planning theory, described in section 2.3. We identified three situations where timed behavior has to be considered. In the following, we enlist these properties and how they will affect the extensions.

Interaction of existing services Guards and Invariants can produce situations in which deadlocks may occur or where safety properties may be violated. In our example that would happen in the controller state `WaitPause` if the x-ray would be triggered too early or too late, so that condition (2) is violated. In our hand-implemented controller we solved this through additional guards and invariants.

To automate this step, the extended planning theory has to analyze the behavior of the existing services for those situations. We can achieve this by using an extended version of BDDs capable of representing timed automata (as used in the Rabbit Model Checker [Rab]). Here, several BDDs are used to represent the functional and timed behavior of an automaton separately. Therefore, we need adapt the BDD based operations of the planning algorithm.

Something has to happen in a specific moment This takes place, for instance, if we want the x-ray to be triggered exactly 6 time units after `next` has been received.

Here, we have to modify how the planning algorithms resolve single planning goals because those now depend on time constraints. Moreover, we have to find a way to express those requirements in Timed CTL.

Something has to happen iteratively within a specific interval This situation takes place, e.g., if we want an x-ray image to be taken every 100 time units. Here, too, CTL is not sufficient to express those requirements. For this situation we have to solve problems similar to the point above.

In the first part of this section we have presented a case study showing that time constraints occur, when medical devices have to be synchronized over a central controller. Since automated service composition can accelerate the development of those controllers and no composition approach is able to deal with time, we have proposed an approach for the automated composition of timed services. In the last part we outlined the problems we have to solve to realize our approach.

4 Related Work

In this section, we present works related to our approach. Firstly, we describe an already existing approach for automated service composition of web services that uses planning as model checking for the composition process (but does not include real-time requirements). Afterwards, we outline works bringing together (automated) service composition and real-time.

In [PTB05] a framework is described that uses planning as model checking to automatically generate a BPEL composition out of a given set of web services and composition requirements. The way how the planning theory was utilized to solve the composition problem served as an inspiration for our proposed approach. In contrast to our work, this tool cannot handle composition requirements expressing real-time properties of the services to be composed. However, real-time capabilities are one of the main characteristics of our proposed approach. Furthermore, this approach was designed for the domain of business processes and does not apply to our domain of controlling distributed devices in a safety-critical environment.

Most (if not all) works that try to bring together *non-automated* service composition and real-time consider time as measurement for communication latency between world-wide distributed services [MGY⁺10] or telephone servers [LL07]. In these cases, time is a part of the *Quality of Service* and helps choosing a proper service instance during the composition process. These works do not solve our problem because we need time as a part of the service's behavior itself.

As to the authors' knowledge, [KDM⁺09] is the only work where an approach for *automated* service composition with real-time requirements is realized. In contrast to our approach, this work offers a very low degree of automation, because the overall workflow of a BPEL composition has to exist before time requirements can be specified. Our approach, on the other hand, offers a very high degree of automation by generating the orchestrator from scratch.

5 Conclusion & Future Work

In this work we have proposed an approach for the automated composition of services with real-time capabilities. The domain for our approach is the generation of controller programs coordinating distributed services in a safety-critical environment out of a set of functional and safety-critical composition requirements. We presented a case study, where medical devices have to be synchronized, and have used *timed i/o automata* as a formal description of

the services' communicational behavior. To realize the automated composition process we currently adapt the AI planning method *planning as model checking* and have outlined the extensions we will have to bring in to make it capable of dealing with time.

In future work, we perform a larger case study than the one presented here, where we model devices used during a specific diagnostic method (a PET/CT scanner and an injection pump). We work on that case study in close cooperation with the Charité Berlin.

By using our proposed approach, the development time for the above-mentioned controller programs can be shortened because certain development steps can be performed automatically. Furthermore, the generated controller model is correct with respect to the composition requirements due to the use of model checking in the generation process. Thus, the iterative step of initially designing and refining a controller model by hand can be skipped and development time can be saved.

References

- ACD93. Alur, R.; Courcoubetis, C.; Dill, D.: Model-Checking in Dense Real-time. Information and Computation, vol.104, pp. 2-34, 1993.
- AGWL09. Arney, D.; Goldman, J. M.; Whitehead, S. F.; Lee, I.: Synchronizing an X-ray and Anesthesia Machine Ventilator: A Medical Device Interoperability Case Study. International Conference on Biomedical Electronics and Devices, pp. 52-60, 2009.
- BP10. Baryannis, G.; Plexousakis, D.: Automated Web Service Composition: State of the Art and Research Challenges. Technical Report - Foundation for Research & Technology - Hellas Institute of Computer Science, 2010.
- GT00. Giunchiglia, F.; Traverso, P.: Planning as Model Checking. Recent Advances in AI Planning, LNCS vol. 1809/2000, pp.1-20, Springer Berlin/Heidelberg, 2000.
- KDM⁺09. Kallel, S.; Charfi, A.; Dinkelaker, T.; Mezini, M.; Jmaiel, M.: Specifying and Monitoring Temporal Properties in Web Services Compositions. Seventh IEEE European Conference on Web Services, pp.148-157, IEEE press, 2009.
- LL07. Lin, L.; Lin, P.: Orchestration in Web Services and Real-Time Communications. Communications Magazine vol.45, no.7, pp.44-50, IEEE press, 2007.
- MBP. MBP: a Model Based Planner. <http://mbp.fbk.eu/>. Last visited: February 2012.
- MGY⁺10. Moussa, H.; Gao, T.; Yen, I.; Bastani, F.; Jeng, J.: Toward effective service composition for real-time SOA-based systems. Service Oriented Computing and Applications Vol.4, pp.17-31, Springer London, 2010.
- PTB05. Pistore, M.; Traverso, P.; Bertoli, P.: Automated Composition of Web Services by Planning in Asynchronous Domains. Artificial Intelligence vol. 174, pp.316-361, Elsevier Science Publishers, 2010.
- Rab. Rabbit and Cottbus Timed Automata. <http://www.sosy-lab.org/dbeyer/Rabbit/>. Last visited: February 2012.
- VL92. Vaandrager, F.; Lynch, N.: Action Transducers and Timed Automata. Proceedings CONCUR'92, LNCS vol.630, pp.436-455. Springer-Verlag, 1992.

Best Service Synthesis in the Weighted Roman Model

Diego Calvanese and Ario Santoso

KRDB Research Centre for Knowledge and Data
Free University of Bozen-Bolzano, Italy
{calvanese, santoso}@inf.unibz.it

Abstract. This paper presents an extension of a framework for synthesizing a composition of services, named Roman Model, such that it is able to model the best service composition synthesis problem. In such extension, which we call the Weighted Roman Model, the services are modeled as Weighted Transition Systems so that one can capture the cost of operations executed by a service. Within this setting, we can make a comparison among all possible compositions of the available services by considering the total cost of operation execution performed by each possible composition of services for each interaction between the service and the client. Besides defining the notion of best composition, we also propose an algorithm for synthesizing the best composition and show that it is sound and complete.

1 Introduction

Services are modular applications that can be described, published, located, invoked, and composed over a variety of networks (including the Internet): any piece of code and any application component deployed on a system can be wrapped and transformed into a network-available service, by using standard (XML-based) languages and protocols (e.g., WSDL, SOAP, etc.). One of the interesting aspects is that this wrapping allows each program to export a simplified description of itself, which abstracts from irrelevant programming details. The promise of Web services is to enable the composition of new distributed applications/solutions: when no available service can satisfy a client request, (parts of) available services can be composed and orchestrated in order to satisfy the request itself.

In reality, there could be several possible ways of composing the available services for satisfying the requested service. However, not all compositions of services can be considered as equally good. They might have different resource consumption (e.g., bandwidth, memory, etc). In this situation, one might be interested in finding the “best” composition among all possible ones.

In this paper, we consider the framework for service composition adopted in [2,5,13,10,3,4], sometimes referred to as the “Roman Model” [9]. In this work, we extend the Roman Model to the Weighted Roman Model in such a way that it is able to model the problem of synthesizing the best composition of services. Moreover, we also describe a sound and complete algorithm for synthesizing the best composition.

The rest of the paper is structured as follows. The next section explains the Roman Model and service composition in this setting. Section 3 presents the Weighted Roman

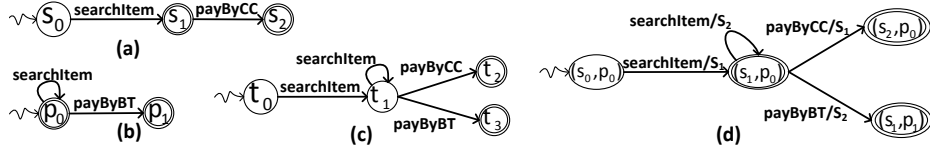


Fig. 1. (a) Service S_1 (b) Service S_2 (c) Target Service S_t (d) Possible composition of services S_1 and S_2 that simulates the target service in (c)

Model and defines the notion of best service composition. Section 4 presents the technique for synthesizing the best service composition within the Weighted Roman Model. Finally Section 5 concludes the paper.

2 Service Composition and The Roman Model

Services in the Roman Model (RM) represent software artifacts capable of performing operations. A service *interacts* with the client through the following steps: (i) it offers to its clients a choice of operations it can perform, (ii) based upon the service state; the client chooses one of the offered operations, and (iii) the service executes it, changing its state accordingly. Fig. 1 shows an example of services in the RM in the scenario of a simple online shopping system. Intuitively, in the service S_1 in Fig. 1(a), the interaction can be started at the initial state s_0 , where the service offers the “searchItem” operation (i.e., the “searchItem” operation is *executable* in this state). After executing this operation, the service’s state changes to s_1 . In s_1 the interaction can be terminated since it’s a final state, or the client can continue requesting the operation “payByCC” (i.e., to pay by credit card). Similarly, in S_2 after a finite sequence of item searches, the client can request a payment by bank transfer (“payByBT”). Formally, a *service* in RM is a transition system (TS) $\mathcal{S} = (S, \mathcal{O}, \delta, s_0, F)$, where: (i) S is the finite set of service’s *states*; (ii) \mathcal{O} is the set of possible *operations* that the service recognizes; (iii) $\delta \subseteq S \times \mathcal{O} \times S$ is the service’s *transition relation*, which accounts for its state changes; (iv) $s_0 \in S$ is the *initial state*; and (v) $F \subseteq S$ is the set of *final states*, i.e., those states where the interaction with the service can be legally terminated by the client. When $\langle s, o, s' \rangle \in \delta$, we say that *transition* $s \xrightarrow{o} s'$ is in \mathcal{S} . Given a state $s \in S$, if there exists a transition $s \xrightarrow{o} s'$ in \mathcal{S} , then operation o is said to be *executable* in s . A transition $s \xrightarrow{o} s'$ in \mathcal{S} denotes that s' is a possible successor state of s , when operation o is executed in s . In this work we consider only *deterministic* services, i.e., there are no two distinct transitions $s \xrightarrow{o} s'$ and $s \xrightarrow{o} s''$ with $s' \neq s''$. Such services are *fully controllable* by just selecting the operation to perform next.

A *community* $\mathcal{C} = \langle \mathcal{S}_1, \dots, \mathcal{S}_n \rangle$ of available services consists of n available services that share the same operations \mathcal{O} . A *target service* is a desired service that also shares the operations in \mathcal{O} . The *goal of the composition in the RM* is to maintain with the client the same, possibly infinite, interaction that he would have with the (virtual) target service, by suitably orchestrating the (concrete) available services. An *orchestrator* is a system component that is able to activate, stop, and resume any of the available services, and to instruct them to execute an operation among those executable in their current state. Essentially, the orchestrator, at each step, will consider the operation chosen by the client

(according to the target service) and delegate it to one of the services that can execute it, and so on, possibly at infinitum. The aim of the orchestrator is to maintain the interaction with the client, as if it was interacting with the target service, without ever failing to delegate an operation chosen by the client to one of the available services.

Formally, an orchestrator is a *function* from (i) the *history* of the whole system (which includes the state trajectories of all available services and the trace of the operations chosen by the client, and executed by the services), and (ii) the *operation* currently chosen by the client, to the index i of the service \mathcal{S}_i to which the operation has to be delegated. Intuitively, the orchestrator *realizes* a target service if and only if, at every step, given the current history of the system, it is able to delegate every operation executable by the target to one of the available services. Hence, the orchestrator controls the evolutions of the services' states in the community s.t. together they "mimic" the target service.

The goal of service composition is to synthesize an orchestrator that realizes the target service by exploiting available services. In [12,3], the problem has been tackled using a simulation-based approach. The idea is essentially checking if the target service is *simulated* by the *Community-TS*, which is the *asynchronous product* of the services in \mathcal{C} . Intuitively, it checks if the Community-TS supports all possible interactions that are supported by the target service (i.e., it checks if there is always a way to realize any interaction that is possible between the client and the target service).

Going back to the running example in Fig. 1, suppose the community consists of services \mathcal{S}_1 and \mathcal{S}_2 . Taking the asynchronous product we obtain the Community-TS and we can find a fragment of it that simulates the target service in Fig. 1(c). This fragment, which encodes the specification of the orchestrator, is shown in Fig. 1(d). It says how the requested operation can be delegated to the services in the community. For example the execution of operation "payByCC" is delegated to service \mathcal{S}_1 (denoted by the label "payByCC/ \mathcal{S}_1 " in the transition).

3 Best Service Composition and the Weighted Roman Model

An extension of the RM into the Weighted Roman Model (WRM) is partly inspired by the work on weighted automata [6]. The WRM framework aims at addressing the problem of best service composition synthesis. The target service in the WRM is represented using a transition system as in the RM, while the available services are represented using weighted transition systems (WTS). Intuitively, a WTS is a TS augmented with a semiring $\hat{S} = (\hat{C}, \hat{+}, \hat{\cdot}, \hat{0}, \hat{1})$ [8]. We use the semiring elements in the set \hat{C} to denote the cost of service's operation execution. Fig. 2 shows an example of services in the WRM. Intuitively, in service \mathcal{S}_1 (Fig. 2(a)), the cost of executing operation "searchItem" is 4. We use the semiring multiplication operator ($\hat{\cdot}$) for the aggregation of service's operations execution costs, and the semiring addition operator ($\hat{+}$) for comparing the costs. As a prominent example, consider the tropical semiring $\hat{T} = (\mathbb{Z} \cup \{\infty\}, \min, +, \infty, 0)$, whose elements are the integers together with positive infinity, whose multiplication operator is addition over integers, and whose addition operator is the minimum operator.

¹ A *semiring* is a structure $\hat{S} = (\hat{C}, \hat{+}, \hat{\cdot}, \hat{0}, \hat{1})$, where \hat{C} is a nonempty set closed under a binary, associative, and commutative *semiring addition*, $\hat{+}$, and a binary, associative *semiring multiplication* $\hat{\cdot}$, respectively with $\hat{0}$ and $\hat{1}$ as neutral elements, and where $\hat{\cdot}$ distributes over $\hat{+}$.

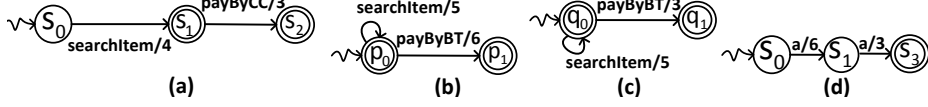


Fig. 2. Example of available services in the WRM: (a) Service \mathcal{S}_1 (b) Service \mathcal{S}_2 (c) Service \mathcal{S}_3 (d) Example of modeling cost dependencies inside a service in the WRM

In this setting, the total cost of service’s operations execution is aggregated by the addition operator (i.e., just the sum over each cost of operations execution) and then we can compare costs of service’s operations execution by using the minimum operator.

To make the comparison meaningful, we restrict the usage of semirings by adding the requirement that $c_1 \hat{+} c_2 = c_1$ or $c_1 \hat{+} c_2 = c_2$, where $c_1, c_2 \in \hat{C}$ (i.e., $\hat{+}$ is an operator for comparing two semiring elements). We call such semiring a *comparison semiring*. Given two semiring elements c_1 and c_2 in a comparison semiring, we say that c_1 is *better than* c_2 if $c_1 \hat{+} c_2 = c_1$, and vice-versa. Consider again the tropical semiring, where addition is the minimum operator. In this case c_1 is *better than* c_2 if $\min(c_1, c_2) = c_1$ (i.e., if c_1 is smaller than c_2).

The usage of a semiring in our framework gives us flexibility in defining the notion of “best”. For example, we can use the tropical semiring if we are interested in the minimum cost, while we can use the arctic semiring $\hat{A} = (\mathbb{Z} \cup \{-\infty\}, \max, +, -\infty, 0)$ if we are interested in the maximum cost. Moreover, it gives us flexibility in defining the domain of the cost, for example whether it ranges over integers, reals, positive integers, etc. Another example is where one models the situation where the cost of operation execution represents the probability of a service being successfully executed, in this case the cost might range from 0 to 1. However, to make explanations more intuitive, from now on we focus on the tropical semiring only.

Formally, a service in the WRM is a WTS $\mathcal{WS} = (S, \mathcal{O}, \hat{T}, \nu, s_0, F)$, where S, \mathcal{O}, s_0 , and F are as for a TS, $\hat{T} = (\mathbb{Z} \cup \{\infty\}, \min, +, \infty, 0)$ is the tropical semiring, and $\nu : S \times \mathcal{O} \times S \times \hat{C}$ is the service’s *transition relation*. When $\langle s, o, s', c \rangle \in \nu$, we say that *transition* $s \xrightarrow{o,c} s'$ is in \mathcal{S} . Intuitively, the semiring element c in the transition $s \xrightarrow{o,c} s'$, represents the cost of performing operation o in state s . In this case, the fact that available services are *deterministic* means that there are no two distinct transitions $s \xrightarrow{o,c} s'$ and $s \xrightarrow{o,c'} s''$ in \mathcal{S} such that $s' \neq s''$ or $c \neq c'$. The notion of community of available services in the WRM is similar to the one in the RM except that the services are represented by WTSs. As for simulation checking in the RM, we can construct a *Community-WTS* by taking the asynchronous product of all available services WTSs.

In the WRM, we can also model some scenarios in which the cost of executing an operation depends on the execution of another operation. Fig. 2(d) shows an example where the second execution of operation a is modeled as having a smaller cost than the first execution of a . In general, we could represent the fact that an operation a executed at some state where a has already been executed has smaller cost. However, there are many scenarios that cannot be captured easily, for example when the cost of executing an operation in a service depends on an operation execution by another service.

Now we introduce the notion of best composition in the WRM through our running example. Suppose the community of available services consists of the service $\mathcal{S}_1, \mathcal{S}_2$, and \mathcal{S}_3 in Fig. 2. The target service \mathcal{S}_t is still the one in Fig. 1(c). Two possible fragments

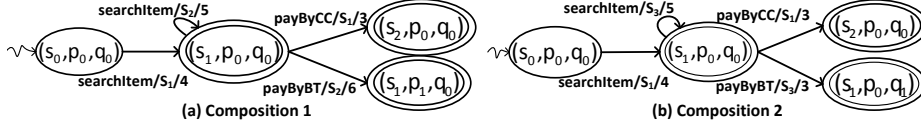


Fig. 3. Two possible composition of services S_1 , S_2 , and S_3 for the target service S_t in Fig. 1(c)

of the Community-WTS that simulate S_t (i.e., serve as a composition for S_t) are shown in Fig. 3. Intuitively, the first composition uses services S_1 and S_2 to “mimic” the target service S_t and the other one uses S_1 and S_3 . Considering the target service in Fig. 1(c), suppose the client requests to execute operation “searchItem” twice, followed by operation “payByBT”. Intuitively she searches for the item in the shop twice and then purchases it by using a bank transfer. Formally, this is represented by the path

$$\tau = t_0 \xrightarrow{\text{searchItem}} t_1 \xrightarrow{\text{searchItem}} t_1 \xrightarrow{\text{payByBT}} t_3$$

in target service S_t . Notice that τ starts at the “initial state” and ends at a “final state”. We call this an *accepting path*. To realize the request, an orchestrator must be able to delegate the execution of the requested operations to the available services in the community. An orchestrator D_1 based on Composition 1 in Fig. 3(a) delegates the first “searchItem” request to S_1 , the second one to S_2 , and the “payByBT” request to S_2 . Formally, this delegation corresponds to the following path in the Community-WTS:

$$\tau' = (s_0, p_0, q_0) \xrightarrow{\text{searchItem}/S_1/4} (s_1, p_0, q_0) \xrightarrow{\text{searchItem}/S_2/5} (s_1, p_0, q_0) \xrightarrow{\text{payByBT}/S_2/6} (s_1, p_1, q_0)$$

We call this a *realization path*. Since we use the tropical semiring, the *weight* of this realization path is just a summation of the weights of all operations along the realization path. In this case the weight of this realization path is 15. However, there might be more than one possible way to realize a certain sequence of operations request. In our examples, we might also delegate to S_3 the execution of the second “searchItem” and of the “payByBT” requests. Hence, there might be more than one corresponding realization in the Community-WTS. Each of them has its own weight. To compare them and find the best weight, since here we use the tropical semiring, we take the minimum among all of them. Hence, in this case we get the best weight as the minimum weight among all possible realizations. In our example one possible best weight is 12 (possibly obtained by doing the delegation based on Composition 2).

The goal of the best composition synthesis is to synthesize the *best orchestrator* D , which informally means that for all possible sequences of operations requested by the client (which correspond to accepting paths in the target service), we have that for all possible delegations of those operations execution to the available services done by D , the total cost of this execution is the best among any other possible delegation done by all possible orchestrators. Considering again the tropical semiring, intuitively we are interested in finding the best orchestrator that minimizes the total cost of the realization of all possible interactions between the target service and the client that started from the initial state and end at a final state. It is not immediate to gain the decidability of this problem, since once we have a loop in the target service, the client can make an infinite

number of different sequences of operations request. Hence we can't just enumerate all possible sequences of operations executions and check if a certain orchestrator can realize them all in the "best" way.

4 Best Composition Synthesis

Recall that given a Community-WTS \mathcal{WC} and the target service \mathcal{S}_t , it can be shown that an orchestrator D for the given \mathcal{WC} and \mathcal{S}_t corresponds to a certain fragment of the \mathcal{WC} that simulates \mathcal{S}_t . We call such fragment a *target service realization*. Intuitively, a target service realization encodes the specification for an orchestrator. Similarly, it can also be shown that a best orchestrator corresponds to a certain fragment of \mathcal{WC} , which we call a *best target service realization*. More formally, a best target service realization is a fragment \mathcal{SR} of \mathcal{WC} s.t. for all accepting paths τ in \mathcal{S}_t , the weight of each possible realization path of τ in \mathcal{SR} is the best. Intuitively, a best target service realization encodes the specification of a best orchestrator. Knowing this fact, we can reduce the problem of checking the existence of a best composition to the problem of checking the existence of the best target service realization. Moreover, a best orchestrator can be synthesized from a best target service realization, if it exists.

Before presenting the algorithm for checking the existence and synthesizing a best target service realization, we introduce some preliminary notions: (i) A *simple cycle path* is a cycle path that has no state repetition in it, except for the first and the last states, which coincide. In Fig. 1(c), the path $t_1 \xrightarrow{\text{searchItem}} t_1$ is a simple cycle while $t_1 \xrightarrow{\text{searchItem}} t_1 \xrightarrow{\text{searchItem}} t_1$ is not. (ii) A *k-bounded accepting path* is an accepting path where the length of each cycle path in it is less than or equal to k . In our example, for $k = 2$, the path $\pi' = t_0 \xrightarrow{\text{searchItem}} t_1 \xrightarrow{\text{searchItem}} t_1 \xrightarrow{\text{searchItem}} t_1 \xrightarrow{\text{payByCC}} t_2$ is a k -bounded accepting path while the path $\pi'' = t_0 \xrightarrow{\text{searchItem}} t_1 \xrightarrow{\text{searchItem}} t_1 \xrightarrow{\text{payByCC}} t_2$ is not.

The algorithm for checking the existence and synthesizing the target service best realization takes the target service and the community as input. We sketch it here briefly (1) For each fragment \mathcal{SR} of \mathcal{WC} , repeat the following steps. (2) Verify if \mathcal{SR} simulates the target service. (3) Verify if for each simple cycle path in the target service, we have that the weight of all its possible realization paths in \mathcal{SR} is the best among all its possible realization paths in \mathcal{WC} . (4) Verify if for each possible k -bounded accepting path in the target service, the weight of all its possible realization paths in \mathcal{SR} is the best among all its possible realization paths in \mathcal{WC} , where k is equal to the size of the community transition system. (5) If \mathcal{SR} fulfills the verification in Steps 2, 3, and 4, the algorithm returns "yes" and \mathcal{SR} is one possible target service best realization. Otherwise, go back to Step 1 to and continue the check with the next fragment. If none of the fragments fulfills the verification in Steps 2, 3, and 4 then the algorithm returns "no". It can be shown that the algorithm above is sound, complete, and always terminates, and that its complexity is double exponential in the combined size of the target service and the community of available services.

In our running example, suppose the target service is as in Fig. 1(c), and that for simplicity of explanation there are only two possible fragments of Community-WTS

as in Fig. 3 (Note: In this example we might have more than these two fragments). In the second step, it is easy to see that both fragments simulate the target service. For the third step, there is only one simple cycle, namely $t_1 \xrightarrow{\text{searchItem}} t_1$ and either in Fig. 3(a) or Fig. 3(b) there is only one possible realization path and both of them have the same weight. In our example, the one which has a realization path with different weight is only $t_0 \xrightarrow{\text{searchItem}} t_1 \xrightarrow{\text{payByBT}} t_3$. The corresponding realization path in Fig. 3(a) is $(s_0, p_0, q_0) \xrightarrow{\text{searchItem}/S_1/4} (s_1, p_0, q_0) \xrightarrow{\text{payByBT}/S_2/6} (s_1, p_1, q_0)$ with weight equal to 10, and the one in Fig. 3(b) is $(s_0, p_0, q_0) \xrightarrow{\text{searchItem}/S_1/4} (s_1, p_0, q_0) \xrightarrow{\text{payByBT}/S_3/3} (s_1, p_0, q_1)$ with weight equal to 7. Due to space limitation, we can't give the full illustration, but one can check that in this case for all of the possible k -bounded accepting paths in the target service, we have that the weight of each possible corresponding realization path in the fragment in Fig. 3(b) is the best, while this does not hold for the fragment in Fig. 3(a). Since the fragment in Fig. 3(b) satisfies the checks in Step 2, 3, and 4, the algorithm return “yes”, and this fragment is one possible target service best realization.

5 Related Work and Conclusions

In this work we have proposed a weighted extension of the Roman Model, named Weighted Roman Model. It enhances the Roman Model with the capability to model the cost of service's operation execution and allows one to address the problem of best composition synthesis. We have shown that the problem of checking the existence and synthesizing the best composition can be addressed by checking the existence and synthesizing the so called best target service realization (encoding the specification of the best orchestrator). Relying on this result, we proposed a sound and complete algorithm for checking the existence and synthesizing the best target service realization.

We provide here a brief overview of related work in the literature. In the SM4ALL project [4], the Roman Model is used as the underlying framework for establishing a collaboration of services, involving composition. The framework is applied to the real world scenario studied in the project, namely that of private homes for users with different abilities and needs. The Roman Model is adopted also in [7], which addresses an optimization problem in the area of service composition. However, it considers finding the best composition for an ad-hoc interaction, i.e., for a given sequence of requested operations. Instead, we consider here all possible sequences of requested operations, hence in general we do not know which might be the next operation requested by the client. Also, the use of a semiring gives more flexibility in defining the meaning of optimum cost. We mention also works where the quantitative aspect comes into play for measuring similarity between transition system-like structures. [14] presents a similarity measure on control flow graphs, which are formalized as labeled transition systems, that is based on a weighted variant of simulation. The work in [11] proposes a technique for matching statecharts that is motivated by model management in software engineering. However, in both works, the transition system-like structures do not contain quantitative information, as in our case.

One interesting further direction of our work is to model the situation where we consider the initial and final weights of a service. Intuitively, the initial weight can model the cost of initializing the service and the final weight the cost of terminating it. Another interesting direction is to analyze the intrinsic complexity of the best composition synthesis problem, and check whether our upper bounds can be improved. Fully taking into account data for verification and synthesis in the context of the Roman Model, or other service-based frameworks, is a very challenging task that has been tackled only recently, see, e.g., [1]. We are not aware of any work that addresses synthesis, fully taking into account data, even in an unweighted setting. It is a very interesting research direction, to tackle this problem, both for an unweighted and for a weighted setting.

References

1. B. Bagheri Hariri, D. Calvanese, G. De Giacomo, R. De Masellis, and P. Felli. Foundations of relational artifacts verification. In *Proc. of the 9th Int. Conference on Business Process Management (BPM 2011)*, volume 6896 of *LNCIS*, pages 379–395. Springer, 2011.
2. D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic service composition based on behavioural descriptions. *Int. J. of Cooperative Information Systems*, 14(4):333–376, 2005.
3. D. Calvanese, G. D. Giacomo, M. Lenzerini, M. Mecella, and F. Patrizi. Automatic service composition and synthesis: the Roman Model. *Bull. of the IEEE Computer Society Technical Committee on Data Engineering*, 31(3):18–22, 2008.
4. T. Catarci, F. Cincotti, M. Leoni, M. Mecella, and G. Santucci. Smart homes for all: Collaborating services in a for-all architecture for domotics. In *Collaborative Computing: Networking, Applications and Worksharing*. Springer Berlin Heidelberg, 2009.
5. G. De Giacomo and S. Sardina. Automatic synthesis of new behaviors from a library of available behaviors. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, 2007.
6. M. Droste, W. Kuich, and H. Vogler. *Handbook of Weighted Automata*. Monographs in Theoretical Computer Science. Springer, 2009.
7. C. E. Gerede, O. H. Ibarra, B. Ravikumar, and J. Su. Minimum-cost delegation in service composition. *Theoretical Computer Science*, 409(3):417–431, 2008.
8. J. Golan. *Semirings and Their Applications*. Kluwer Academic Publishers, 1999.
9. R. Hull. Web services composition: A story of models, automata, and logics. In *Proc. of the 3rd IEEE Int. Conf. on Web Services (ICWS 2005)*, 2005.
10. A. Muscholl and I. Walukiewicz. A lower bound on web services composition. *Logical Methods in Computer Science*, 4(2), 2008.
11. S. Nejati, M. Sabetzadeh, M. Chechik, S. Easterbrook, and P. Zave. Matching and merging of statecharts specifications. In *Proc. of the 29th Int. Conf. on Software Engineering (ICSE 2007)*, pages 54–64, 2007.
12. F. Patrizi. *Simulation-based Techniques for Automated Service Composition*. PhD thesis, SAPIENZA Università di Roma, Dipartimento di Informatica e Sistemistica, 2009.
13. S. Sardina, F. Patrizi, and G. De Giacomo. Behavior composition in the presence of failure. In *Proc. of the 11th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2008)*, 2008.
14. O. Sokolsky, S. Kannan, and I. Lee. Simulation-based graph similarity. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 3920 of *LNCIS*, pages 426–440. Springer, 2006.

Choreographies in BPMN 2.0: New Challenges and Open Questions

Mario Cortes-Cornax, Sophie Dupuy-Chessa, and Dominique Rieu

University of Grenoble, CNRS, LIG

{Mario.Cortes-Cornax, Sophie.Dupuy, Dominique.Rieu}@imag.fr,
<http://sigma.imag.fr/>

Abstract. The concept of choreography has emerged over the past years as a fundamental concept for capturing collaborative processes. The latest version of the Business Process Modeling Notation (BPMN 2.0) introduces the choreography diagram as a first-class citizen actor. After having evaluated BPMN 2.0 in a previous work, we discuss here the new challenges, future work and the open questions about the potential choreography standard language. We also describe the ameliorations that will be introduced in the evaluation framework.

Keywords: Choreography, Evaluation, BPMN 2.0, Quality Framework

1 Introduction

A choreography formalizes the way business *participants* coordinate their *interactions*. In a choreography, the focus is not on the work performed internally by each participant, but rather on the *exchange of information* (e.g. messages) between participants. Another way to look at choreography is to consider it as a type of *business contract* between two or more organizations.

Industry initiatives such as RosettaNet ¹ aim at standardizing business to business integration in a particular domain. However, these initiatives mostly turned to textual descriptions of the overall choreographies, centered in providing detailed message format descriptions [9]. W3C's efforts within the context of the *Web Service Choreography Description Language* proposal (WS-CDL [23]) did not achieve enough industry support and do not reach standardization. The WS-CDL's working work stopped the development of the language in July 2009. Previously, major lacks were detected in [2]. Over the past years, several research projects have proposed different languages for capturing choreographies such as *Lets's Dance* [24], *BPEL4Chor* [9] or *Multi-Agent Protocols (MAP)* [1]. However, these proposals remain far to be adopted by the industry. Popular languages as the *Message Sequence Charts* (MSC) [12] have also been used to capture cross-organizational interactions. But the latter is not rich enough to capture complex choreographies [8].

¹ <http://www.rosettanel.org/>

In early 2011 the OMG [18] released the latest version of the *Business Process Model and Notation* (BPMN version 2.0 [19]). Among other improvements, a choreography diagram is introduced. In previous versions of BPMN, the only way to represent choreographies was via *collaboration diagrams*. This new version allows modelers describing both choreography and collaboration approaches together or individually. Actually, a global view of interactions is represented in addition to the participants' view given by collaborations which enriches the expressiveness of the language [19].

In a previous work [7], we evaluate the adequacy of the constructs for choreography modeling introduced in BPMN 2.0. We also presented a catalogue of identified requirements that represents a clear overview of possible criteria for evaluating a choreography language as well as to better understand this increasingly used concept. After the evaluation, we detect some important drawbacks in the language.

The goal of this paper is to briefly resume the evaluation that we performed [7] and then discuss the major challenges and the research agenda to short out the problems detected. We also present several limitations that are identified in our evaluation framework, and the necessary improvements in order to complete it.

This paper is structured as follows. We resume our evaluation of choreographies in BPMN 2.0 in Section 2. A detailed discussion about major challenges and future work are presented in Section 3. Section 4 presents our research methodology. Finally, Section 5 concludes the paper.

2 The Evaluation of BPMN 2.0 for Choreographies

We based our evaluation of BPMN 2.0 on a semiotic quality framework proposed by Kogstie [14]. We extend it for the specific context of choreographies similarly to [17] for Business Processes. We look at three axes that are the *Domain Appropriateness (D)* (relates the language to the semantics of its domain), the *Comprehensibility Appropriateness (C)* (relates the language to the social actor) and the *Technical Actor Interpretation Appropriateness (T)* (relates the language to tools). In order to organize and categorize the identified choreography requirements, we placed the requirements in the different dimensions of the framework (Fig. 1). Most of this requirements were further refined in sub-requirements.

Domain requirements are mainly extracted from the *Service Interaction Patterns* [3] and from the service choreography requirements identified by Decker et al. in [9]. Looking at the refined notions of choreography presented in [21] that are *B2Bi Choreographies*, *Conceptual Choreographies* and *Service Choreographies* it could be argued that we are more focused in the two latter although we find many common requirements within the three of them. A detailed study about B2Bi requirements can be found in [20].

When analyzing *comprehensibility requirements* of the language, the major interest is given to the graphical notation principles described by Moody in [16]. We also analyzed other aspects such as the model and the meta-model

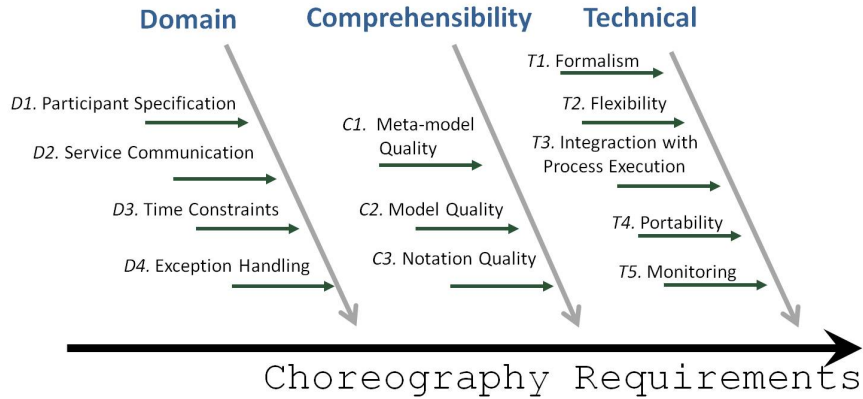


Fig. 1. The requirements axes extending the language quality framework

quality guided by researches as [22,10,4]. The necessity of taking into account comprehensibility aspects for a choreography language is already cited in [13]. *Technical requirements* were mostly induced by the analysis of previous choreography proposals. For further details about this evaluation, the reader can refer to [7].

3 Discussion about Future Work and Open Questions

3.1 Domain Requirements Analysis

Major Challenges. As we already mentioned, the domain requirements were mainly induced and based on the Service Interaction Patterns [3]. Lacks that will prevent BPMN 2.0 to support all the patterns were detected. Unlike *Participant Multiplicity* is supported in BPMN 2.0, *Message Multiplicity* (*Service Communication* sub-requirement), that is used to capture the definition of the number of messages sent from one (or more) participant(s) to other(s) is not supported. This lack will avoid fulfilling the so-called multi-transmission interaction patterns.

Another important detected problem is the weak support for *Reference Passing* (*Service Communication* sub-requirement) where participant A permits participant C to communicate with participant B by passing the reference of B to C. If the latter requirement is not supported, it will avoid fulfilling the so-called routing patterns. The major challenge here is to give support to all the interaction patterns. However one major issue for using BPMN 2.0 choreography is that its semantics are not well defined. The standard provides just an indicative idea of the semantics through local enforceability of different BPMN's choreography constructs and modeling situations. A preliminary work on clarifying the semantics should be done.

Future Work to Improve...

- ... **The Language.** Detecting major lacks within BPMN 2.0 for choreographies has been a first step in our work that might be completed by proposing an extension of the language. In [7], we suggest to recover the concept of channel introduced in WS-CDL to support reference passing. These channels could be explicitly captured with textual annotations in the diagram, following the principle of *Dual Coding* [16]. An extension of the concept of message to capture the *Message Multiplicity* is also suggested. These feature could be easily captured with a graphical construct in the diagram following the principle of *Semiotic Clarity* [16] that suggests one-to-one correspondence between symbols and semantic concepts. This will help avoiding ambiguities when defining it in a technical specification. However, these approach have to be matured and formalized.
- ... **The Evaluation.** A precise analysis of the support of the 13 patterns has to be considered as an important future work. The implementability of the patterns could be done analyzing separately the two possible ways of representing choreographies in BPMN 2.0 (i.e by means of collaboration diagrams and by the new choreography diagrams) but we could also think about evaluating BPMN 2.0 as a whole, considering that the two diagrams represent different views of choreography. Analyzing the service interaction patterns will give us a more precise idea of the limitations of BPMN 2.0 concerning choreographies. It will permit to perform an accurate comparison between the different choreography languages regarding the domain dimension.

3.2 Comprehensibility Requirements Analysis

Major Challenges. Regarding at comprehensibility requirements, the aim is to give more automation to our evaluation. Looking at the principles of graphical notation, a detailed analysis in the different principles is done. However, the great amount of graphical constructs difficult the work. Works like [11] where authors evaluate the cognitive effectiveness of BPMN 2.0's process models are a good reference to be applied to choreographies. If valuable metrics are defined, it will be much easier to compare the BPMN 2.0 features with other languages automatically. The challenge is to automate the evaluation of comprehensibility requirements.

We also detected a greater lack in the meta-model quality. A meta-model should be a useful tool for communication besides a technical description of a language. The way that meta-models are presented in BPMN 2.0 hinders the understanding of choreographies because they are presented in a very technical level. Therefore, another important challenge is to achieve a more comprehensible language.

We have also noted some underspecification and a lack of examples concerning choreographies that difficult an effective use of the language (e.g. the *ChoreographyLoopType*² construct).

² <http://www.omg.org/issues/bpmn2-rtf.open.html#Issue16554>

Future Work to Improve...

- ... **The Language.** In [6], the use of different levels of abstraction and the fact of clearly separate the structural and behavioral views in choreographies is recommended. This approach could also help to adapt graphical notation to different contexts similarly to Silver's proposal in [22] for business process models.
- ... **The Evaluation.** We consider essential to find concrete metrics to automate comprehensibility evaluation. Appropriate metrics could be found by conducting specific experimental studies for the different notation principles. In some cases as for example *Semantic Transparency* (visual representation appearance should suggest its meaning) it is difficult to find appropriate metrics that help evaluating this requirement. Although such evaluations provide valuable insights, they are time-consuming and only allow one to evaluate one or two specific aspects of a language (e.g. understandability or readability). It will also be interesting to work on indicators to better evaluate the meta-model readability and simplicity.
- ... **The Understanding of BPMN 2.0.** The standard should be illustrated to permit practitioners to easily know all the capabilities of the language. A set of examples, using the graphical constructors might be proposed. For example, the use of intermediate events attached to choreography activities are not clearly comprehensible as there are no examples in the standard. This might improve the language's pragmatic quality [15]. The introduction of abstraction layers similar to the ones proposed by Silver for business process models in [22] and the use of different views in the meta-model level will also help to understand the language in a more natural and progressive way.

3.3 Technical Requirements Analysis

Major Challenges. In the technical evaluation, the weakest point is concerning the underspecification of some requirements that leads to ambiguities in the evaluation. For example, terms such as *Formalism* or *Flexibility* lead to misunderstanding because there are not correctly defined.

It is also important to put forward the fact of having a completely new diagram integrated in the standard. This provoke that implementers had difficulties to support choreography conformance. Currently, there is an obvious preference besides process models and their execution rather than using the choreography approach. So we should still wait for implementers response to perform a detailed tool support analysis. The challenge is to find adequate requirements to guide proper tool support for choreographies.

Future Work to Improve...

- ... **The Evaluation.** An important limitation of our evaluation is the lack of technical requirements. To mitigate this lack, we turned to B2B integration requirements [20] and Rosetta Net project to complete this axis. Although

we do not target *B2Bi Choreographies*[21] but *Service Choreographies* and *Conceptual Choreographies* [21], many technical requirements are applicable to the different notions of choreographies. For example, we will have to introduce the *Message Formatting* requirement [9] as RosettaNet show that it is possible and fundamental to be considered. The detailed formatting of messages should be captured in the technical specification. However, different basic types of messages could be defined extending the notion of message. The analysis of orchestration requirements may also be helpful to infer critical choreography requirements.

We will also have to analyze carefully if all the the technical requirements are so well supported by BPMN 2.0 as currently considered. For example, in [13] authors argue that the choreography diagrams are tightly dependent on the technical configuration while we considered that the fact that choreographies do not need a technical configurations to be defined make them “flexible” and reusable.

4 Research Methodology

First, we identified the need of representing the choreography notion in a three-level multi-view approach to effectively bridge the Business-IT gap in [6,5]. These studies gave us an idea of the importance of abstraction levels and multi-views when managing choreographies. We gathered general requirements that should be supported by choreography languages basing our research on two main sources:

- Scientific studies dealing with choreography requirements such as [2,3,4,9].
- Choreography language proposals such as WS-CDL [23], Let’s Dance [24], BPEL4Chor [9] or MAP [1].

One of the most detailed prior evaluations of choreography definition languages is based on the *Service Interaction Patterns* [3], but these patterns only cover one perspective of the requirements for choreography definition languages. Accordingly, we complemented this patterns-based evaluation framework with other perspectives. Therefore, we categorized the choreography requirements with the three axes illustrated in Section 2 to evaluate *Domain*, *Comprehensibility* and *Technical* appropriateness for choreography languages. Special attention is given to graphical notation (*Comprehensibility* sub-requirement), since the graphical notation may be a key ingredient to bridge the gap between business world and technical specification.

Our goal now is to merge both works in a multi-leveled evaluation framework. It is obvious that we find different requirements depending on the level of abstraction that we are working on. For example, a graphical notation is essential in a higher level of abstraction (near the business world), while it might be less critical when a technical specification has to be implemented. On the other hand, message correlation is essential in a technical level while near the business level, it may not be essential to be captured. We want to analyze for each level of abstraction, what are the main requirements that have to be managed. Hence,

choreography requirements categorized in a three-leveled evaluation framework will be the foundation of a new service choreography language (sketched in [5]) or an extension proposal for choreographies in BPMN 2.0. It will also leads to a precise and useful guide for choreography language's evaluation.

5 Conclusion

We have summarized the evaluation carried out in [7] where we evaluated BPMN 2.0's constructs for choreographies using an extended quality framework. The major challenges are discussed and the main axis for future improvements are presented.

We conclude that in the domain dimension, important lacks such as *Reference Passing* and *Message Multiplicity* will prevent a fully support of all the requirements. A better evaluation of comprehensibility should be undertaken based on metrics or specific studies. The technical axis will be completed taking into account new requirements related to B2Bi requirements, industry initiatives as Rosetta Net, and orchestrations. However, our major efforts will be centered in *Service Choreographies* [21] and not B2B integration.

Having analyzing the necessity of defining the choreography notion in three different abstraction levels in previous works, we will propose a three-leveled evaluation framework keeping the *Domain*, *Comprehensibility* and *Technical* axes. A new choreography language or extensions of BPMN 2.0 for choreographies will be presented based on this updated framework.

References

1. Barker, A., Walton, C., Robertson, D.: Choreographing Web Services. *IEEE Transactions on Services Computing*, IEEE Computer Society **2**(2) (2009) 152–166
2. Barros, A., Dumas, M., Oaks, P.: A critical overview of the web services choreography description language. *BPTrends Newsletter* **3** (2005)
3. Barros, A., Dumas, M., ter Hofstede, A.: Service interaction patterns. *Business Process Management*, Springer (2005) 302–318
4. Barros, A., Decker, G., Dumas, M.: Multi-staged and multi-viewpoint service choreography modelling. Technical report (2006)
5. Cortes-Cornax, M.: Service choreographies through a graphical notation based on abstraction layers and viewpoints. *Research Challenges in Information Science (RCIS)*, 2011 Fifth International Conference on, IEEE (2011) 1–12
6. Cortes-Cornax, M., Dupuy-Chessa, S., Rieu, D.: Bridging the gap between business processes and service composition through service choreographies. *Engineering Methods in the Service-Oriented Context*, Springer (2011) 190–203
7. Cortes-Cornax, M., Dupuy-Chessa, S., Rieu, D., Dumas, M.: Evaluating choreographies in bpmn 2.0 using an extended quality framework. *Business Process Model and Notation*, Springer (2011) 103–117
8. Decker, G., Kopp, O., Barros, A.: An introduction to service choreographies. *Information Technology, Citiseer* **50**(2) (2008) 122–127

9. Decker, G., Kopp, O., Leymann, F., Weske, M.: Interacting services: from specification to execution. *Data & Knowledge Engineering, Elsevier* **68**(10) (2009) 946–972
10. Dijkman, R., Dumas, M.: Service-oriented design: A multi-viewpoint approach. *International Journal of Cooperative Information Systems* **13** (2004) 337–368
11. Genon, N., Heymans, P., Amyot, D.: Analysing the Cognitive Effectiveness of the BPMN 2.0 Visual Notation. *Software Language Engineering, Springer* (2011) 377–396
12. Harel, D., Thiagarajan, P.: Message sequence charts. *UML for Real, Springer* (2004) 77–105
13. Kopp, O., Leymann, F., Wagner, S.: Modeling Choreographies: BPMN 2.0 versus BPEL-based Approaches. In: *Enterprise Modelling and Information Systems Architectures -EMISA 2011, Gesellschaft für Informatik e.V. (GI)* (September 2011)
14. Krogstie, J., Sindre, G., Jørgensen, H.: Process models representing knowledge for action: a revised quality framework. *European Journal of Information Systems, Nature Publishing Group* **15**(1) (2006) 91–102
15. Lindland, O., Sindre, G., Solvberg, A.: Understanding quality in conceptual modeling. *Software, IEEE* **11**(2) (1994) 42–49
16. Moody, D.: The “Physics” of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. *IEEE Transactions on Software Engineering, IEEE Computer Society* (2009) 756–779
17. Nysetvold, A., Krogstie, J.: Assessing business process modeling languages using a generic quality framework. *Advanced topics in database research* **5** (2006) 79–93
18. OMG: Object management group. ”<http://www.omg.org/>” (1989)
19. OMG: Business process model and notation (bpmn 2.0). ”<http://www.omg.org/spec/BPMN/2.0/>” (2011)
20. Schönberger, A., Wilms, C., Wirtz, G.: A requirements analysis of integration business to business integration. Technical report, Otto-Friedrich- Universität Bamberg, *Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik* (2009)
21. Schönberger, A.: Do we need a refined choreography notion? ZEUS. *CEURWS.org* (2011)
22. Silver, B.: BPMN Method and Style: A levels-based methodology for BPM process modeling and improvement using BPMN 2.0. *Cody-Cassidy Press, US* (2009)
23. W3C: Web services choreography description language version 1.0 (ws-cdl) - w3c candidate recommendation (2005)
24. Zaha, J., Barros, A., Dumas, M., ter Hofstede, A.: Let’s dance: A language for service behavior modeling. *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, Springer* (2006) 145–162

Building Orchestrations in B2Bi – The Case of BPEL 2.0 and BPMN 2.0

Jörg Lenhard and Guido Wirtz

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

Abstract. Various approaches for service-oriented business-to-business integration (B2Bi) rely on a top-down development methodology. The starting point is a choreography model which is subsequently partitioned into multiple orchestrations. Most current approaches use the Web Services Business Process Execution Language (BPEL) for implementing the latter. At the same time, a plethora of other languages, such as Business Process Model and Notation (BPMN) 2.0 process diagrams, is available. As integration partners are free to select the orchestration language of their choice, it should be easy to integrate different orchestration languages with current choreography technology. Language transformation, starting from a suitable format, is a means to achieve this. In this paper, we assess BPEL 2.0 and BPMN 2.0 process diagrams for their suitability for this transformation in a services-based B2Bi setting using a requirements framework identified through a literature study.

Keywords: Orchestration, BPEL 2.0, BPMN 2.0, B2Bi

1 Introduction

Over the last years, the influence of service-orientation in the implementation of interorganizational processes has grown rapidly. Many approaches¹ for implementing such processes employ a combination of choreography and orchestration models [6, 19] to capture different viewpoints on an enterprise-crossing business process. Top-down approaches refine the first into a set of the latter which thereafter is implemented at each partner's side. A variety of languages has emerged, and is continuing to do so, on both levels of abstraction. Integration among the two types of languages and an easy translation from a choreography to a set of orchestrations is seen as a core issue [4].

Although BPEL [16] is widely used in these approaches for implementing orchestrations, it is facing rising competition by other languages, such as BPMN 2.0 process diagrams [17] or the Windows Workflow Foundation (WF) [3]. Consequently, an integration partner may choose to implement her orchestration not in BPEL, but in any other orchestration language, using the derived orchestration, which with current approaches mostly is a BPEL process, only as blueprint.

¹ Some examples, without claiming to be complete, are: [4, 8, 14, 18, 23, 30]

In this case, there is still a considerable gap between a BPEL process derived from a choreography and the final running orchestration. For instance, target languages may rely on a different (i.e., graph-based or block-structured [11]) execution model with a differing level of expressiveness. Bridging this gap manually, which is the current option, it is easy to introduce problems that hinder later execution. Behavioural incompatibilities that were eliminated using model checking techniques may be reintroduced and adjustments to the original interfaces and message definitions that arise from the limitations of the final execution platforms may become necessary. Here, an automatic transformation from an orchestration derived from a choreography model to orchestrations implemented in other languages can help. The idea is to not to compile choreography models to an executable artifact tailored to a specific execution platform in a single step, but to provide an artifact that can be automatically transformed to a variety of different types of languages and platforms, and also be executed directly. By automating this transformation step, it would be possible to:

1. Use model checking and related techniques [27] to ensure that no behavioural incompatibilities are introduced during the transformation.
2. Leverage the information of what execution platforms are ultimately used and will be communicating with one another to avoid pitfalls and communication problems known for these platforms² and perform optimizations such as the configuration of the most efficient communication binding known to exist among the platforms.

The starting point of this functionality is a suitable format to which choreography models are compiled and from which such a transformation is possible in the first place. The aim of this paper is to delimitate criteria (requirements) that are essential for such a format and which can be used to evaluate the suitability of a language, as well as to use these criteria to assess two languages that are natural candidates for this kind of task. BPEL could be seen as such a format and most researchers use it because of its status as de-facto standard; A more recent option is BPMN 2.0 process diagrams [13].

The requirements are identified through a literature study. Since the aforementioned format is to be used for model transformation and optimization in service-oriented top-down B2Bi, the requirements are derived from relevant sources in the B2Bi domain, service composition languages and transformation of process models that reside on the same level of abstraction³.

Furthermore, we assess the support of BPEL 2.0 and BPMN 2.0 process diagrams for the identified requirements and discuss the evaluation. The outcome suggests that BPMN 2.0 process diagrams are more suitable for this type of application.

² For example known problems among Java-based and .NET-based Web Services [25].

³ This is called horizontal transformation [15].

2 Related Work

Approaches for service composition using choreography and orchestration technology have attracted considerable interest. Most of these approaches (e.g. [4, 8, 14, 18, 23, 30]) rely on BPEL as the target language to which choreography models are compiled. Although being widely supported, BPEL is more and more rivaled by other languages [13], both based on standardization initiatives, such as BPMN 2.0 [17], or proprietary environments, such as WF [3]. Just as for BPEL, engines for executing orchestrations built in these languages are available.

There are many studies that explicitly specify and assess requirements for service composition languages, B2Bi, or model transformation with varying design goals. In the area of service composition, focus lies on choreography languages [4, 23]. In this paper, we center on orchestration languages instead, but take into account these studies, where requirements for choreographies and orchestrations intersect. The requirements defined in [4] concentrate on language expressiveness. [22, 23] on the other hand, take B2Bi-related requirements into account. General requirements for process languages and requirements related to horizontal transformation of these languages can be found in [7, 15, 29]. In this paper, we extract and unify the requirements from the preceding studies that are relevant to horizontal transformation of orchestration languages in B2Bi.

An assessment of BPEL 2.0 and BPMN 1.1 for parts of these requirements can be found in [4]. In [10], the requirements from [4] are used to assess BPMN 2.0 *collaboration* and *choreography diagrams*. Here instead, we concentrate on BPMN 2.0 *process diagrams* with the addition of **participants**, use an extended set of requirements and a different design goal; that is the assessment of orchestration instead of choreography capabilities.

3 Requirements for Orchestration Languages in B2Bi

Myriads of requirements could be taken into account when considering either B2Bi, service composition or language transformation and a vast amount of literature on these topics is available with varying design goals. We do not intent to start from scratch and therefore extract common requirements from several influential studies of recent years that did explicitly post such requirements and which match well our domain and design focus. Like in any literature study, this selection of sources is biased to some extent by our knowledge and we do not claim the completeness of the requirements listed here.

The requirements are sorted in four groups: (i) General requirements, (ii) B2Bi-related requirements (iii) interaction-related requirements and (iv) derived requirements. The final group does not originate from the requirement sources, but is derived in the context of this study.

I. General requirements:

R1 *Support for common control-flow structures*: An orchestration language must include a suitable amount of control-flow structures to allow for a direct implementation of domain relevant scenarios. This requirement is explicitly

stated in [7, 15, 22, 23]. Assessing languages for their support for control-flow patterns which describe such common structures can be used as benchmark for this requirement [28].

- R2 *Mechanism for hierarchical decomposition*: A key feature for dealing with the complexity of realistic orchestrations is a mechanism for hierarchical decomposition. The necessity of this feature is stated in [7, 15, 22, 23].
- R3 *Data handling mechanisms*: Just as for control-flow structures, appropriate mechanisms for defining, transferring, and manipulating data structures must be in place [4, 7, 15, 22, 23]. This requirement can be evaluated by assessing pattern support as well [20].
- R4 *Exception handling mechanisms*: Being executable, orchestrations must not only deal with best-case scenarios, but take into account erroneous circumstances that may arise during execution. This requirement is backed up by [4, 7, 22, 23]. It can also be assessed using pattern-based analysis [21].
- R5 *Extensibility*: An orchestration language should be extensible to allow for an easy adaptation and the introduction of new or modified constructs to support use cases with specific needs [7, 22, 23].

II. B2Bi-related requirements:

- R6 *Transactions*: An important primitive in enterprise computing is transactional integrity of interactions. For instance, the reliable transmission of business documents is crucial and a common means to this end are transactions. An orchestration language should provide mechanisms to denote transactional contexts during process execution [7, 22, 23].
- R7 *Quality of service (QoS)*: Several nonfunctional properties, esp. QoS parameters, are vital in B2Bi. These are authentication, message encryption and signatures, non-repudiation of message exchanges and time constraints. An orchestration language should provide explicit mechanisms to express these properties [4, 22, 23].
- R8 *Standards*: In the B2Bi setting, it is not possible to enforce technologies on different independent partners. A higher degree of interoperability is likely by relying on essential standards [22, 23].

III. Interaction-related requirements:

- R9 *Message correlation*: During execution, multiple orchestration instances run in parallel. To support a correct routing of messages by an engine, an orchestration language must provide mechanisms for message correlation [4, 22, 23]. As before, this aspect can be evaluated using patterns [1].
- R10 *Service selection and reference passing*: In realistic interaction scenarios, not all communicating parties may be known at design time. Instead, partner references are transferred in messages and are bound at run-time [4].
- R11 *Multi-lateral interaction*: Choreographies may consist of more than two partners. Consequently, orchestrations must be able to represent and communicate with multiple different parties [4].

IV. Derived requirements:

- R12 *Contract-first development and integration with choreography approaches*: It is a general engineering best practice to define interfaces or contracts

before implementing them. This is inevitable in a top-down development approach. Orchestration languages therefore must support contract-first development⁴. [4,23] specify that choreography languages must easily integrate with orchestration languages. Also the reverse is important: The applicability of an orchestration language in top-down approaches should be demonstrated.

R13 *Web Services and XML*: Choreographies should be technology-independent [4,23]. This does not apply to orchestrations, which need to be executable. Consequently, they should work with contemporary communication and integration technologies, most notably Web Services and SOAP. To allow for easy processing and transformation of orchestration models, languages should provide a XML serialization format [7,15,22,23].

4 Assessment of BPEL 2.0 and BPMN 2.0

In the following, BPEL 2.0 and BPMN 2.0 process diagrams are assessed for their support for the requirements. We state whether a requirement is supported *directly* (+), *partially* (+/-) or *not in a direct fashion* (-). This trivalent measure is relatively simplistic and subjective to a certain extent. Although enhanced alternatives do exist [12], it is used extensively [4,5,10,20,21,28,31]. For that reason and the space constraints of this paper, we use the above measure.

Assessment of BPEL 2.0: A detailed analysis of the control-flow capacity of BPEL can be found in [12]. BPEL 2.0 supports a range of control-flow structures and block-structured and graph-oriented control-flow definition. Given typical B2Bi use cases⁵ generally only require simple control-flow constructs [24], we consider this as evidence for the support of R1. BPEL 2.0 provides no explicit construct for hierarchical decomposition; that is, no direct notion of a subprocess⁶. It is possible to work around this requirement using nested scopes or Web Service invocation of another BPEL 2.0 process, which qualifies as partial support for R2. Compensation and try-catch constructs are present for exception handling and XML Schema and XPath 1.0 for data definition and manipulation. Although closer evaluations are only present for BPEL 1.1, we consider this as support for R3 and R4. BPEL 2.0 allows to extend the language with new engine-specific activities using `extensionActivity` and `extensionAssignOperation`, thereby fulfilling R5. As for the B2Bi requirements, BPEL 2.0 has no built-in mechanisms for scoping transactions, which must be implemented using additional standards such as WS-Coordination and WS-AtomicTransaction. Policies using these standards can be attached to operations at the WSDL-level. This policy-based approach

⁴ This may seem obvious. Nevertheless, there are languages, such as Windows Workflow in its current revision 4 [3], that do not support contract-first development.

⁵ Examples of such use cases are the *RosettaNet Implementation Guides*: <http://www.rosettanet.org/Support/ImplementingRosettaNetStandards/RosettaNetImplementationGuides/tabid/2985/Default.aspx>

⁶ Such a structure is introduced by the BPEL-SPE specification [9], a BPEL extension for subprocesses. However, this specification is not widely adopted and thus we limit ourselves to the BPEL specification [16] and related WS-standards in this evaluation.

enhances the flexibility and composability of the Web Services stack. However, in the case of B2Bi, it would be reasonable to insert the notion of transactions into the process itself⁷. BPEL 2.0 directly provides only *quasi-atomic* transactions through compensation [2]. Altogether, we consider it to provide only partial support for R6. The same applies to QoS requirements which cannot be represented directly, but with the help of additional standards, such as WS-ReliableMessaging and WS-Security. Moreover, BPEL's support for time constraints is fairly limited [12]. This results in partial support for R7. As BPEL is an OASIS standard, it fulfills R8. BPEL 2.0 uses key-based correlation with `correlationSets` that can be initialized and used by messaging activities, thus fulfilling R9. References can be passed and set via WS-Addressing `endpointReferences` which are first-class citizens of the specification. As this is only an implicit way of service selection, it is considered as partial support for R10 [4]. Multi-lateral interaction is possible using multiple `partnerLinks`, fulfilling R11. The applicability of BPEL in top-down approaches has been proven in multiple settings [8, 14, 18, 23, 30], fulfilling R12. Finally, the language is built on Web Services and provides a XML format (R13).

Assessment of BPMN 2.0: A discussion on control-flow pattern support is part of the BPMN 2.0 specification [17], granting support for R1. This is not the case for data or exception handling patterns, but here results from an older revision are applicable [21, 31], fulfilling R3 and R4. R2 is directly supported using `subProcesses` and `callActivities` which are considerably more powerful than BPEL `scopes` as they allow for a reuse of process definitions without having to resort to external Web Service invocation. BPMN 2.0 comes with an extension mechanism, based on `extension` and `extensionDefinition`, that can be used to define additional elements at the level of existing elements. For instance, by extending a `task` with additional attributes, it is possible to provide new functionality. This mechanism fulfills R5. A special type of `subProcess`, `transaction`, can be used to demark a transactional context in a process to provide a consistent outcome to the execution of a set of activities and allows for the configuration of the coordination protocol applied (typically WS-AtomicTransaction or WS-BusinessActivity). Additionally, `hazards` mark events in a `transaction` that enforce its immediate termination without compensation, but without terminating the complete process. Such *scope-termination* is not possible in BPEL. Altogether, this fulfills R6. Concerning QoS, BPMN processes are limited to simple time constraints in the same fashion as BPEL. Although it would be possible to annotate such configurations using `properties`, this only qualifies for partial support with respect to R7. BPMN is an OMG standard, satisfying R8. BPMN 2.0 supports key-based and, in contrast to BPEL 2.0, context-based correlation. This is sufficient for R9. `Participants` can be used to represent interaction partners of a process, thereby fulfilling R11. Service selection is no first-class member of the BPMN 2.0 specification. However, BPMN 2.0 allows to define `endPoints`, which may be comprised of WS-Addressing `endpointReferences`. It is possible to reference these `endPoints` in a `participant` and reassign them

⁷ For instance, this is also the strategy followed by [26]. There, policies are introduced at the level of `scopes` or `partnerLinks`, resulting in *coordinated scopes* / *partnerLinks*.

during process execution. This resembles the solution of BPEL. Therefore, we conclude that R10 is partially supported. The use of BPMN executable processes as orchestrations is just in its start. Nevertheless, in the BPMN environment, they are integrated into diagrams for modeling choreographies, so their applicability in a top-down development approach seems given (R12). Finally, BPMN 2.0 comes with a XML serialization format and in the context of messaging activities and tasks, Web Services are considered the default technology (R13).

5 Conclusion and Future Work

The results are summarized in Table 1. Both languages provide a strong degree of support for the requirements at hand, which is not surprising and the reason they were selected in the first place. Here, nuances in the support are of interest. As there is a more powerful mechanism for hierarchical decomposition and a concept for demarking transactions in BPMN 2.0 processes, they are considered as more suitable for the output of B2B-choreographies. To levitate existing deficiencies, it would be helpful to extend process elements with explicit notions for QoS. Also, the definition of mappings to further languages, such as WF, is promising.

Table 1. Assessment of Languages

Requirement	BPEL 2.0	BPMN 2.0
R1 Control-flow structures	+	+
R2 hierarchical decomposition	+/-	+
R3 Data handling	+	+
R4 Exception handling	+	+
R5 Extensibility	+	+
R6 Transactions	+/-	+
R7 QoS	+/-	+/-
R8 Standards	+	+
R9 Correlation	+	+
R10 Reference passing	+/-	+/-
R11 Multi-lateral interaction	+	+
R12 Choreography integration	+	+
R13 XML, Web Services	+	+

References

1. A. P. Barros, G. Decker, M. Dumas, and F. Weber. Correlation Patterns in Service-Oriented Architectures. In *FASE*, pages 245–259, Braga, Portugal, 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. B. Bukovics. *Pro WF: Windows Workflow in .NET 4*. Apress, June 2010. ISBN-13: 978-1-4302-2721-2.
4. G. Decker, O. Kopp, F. Leymann, and M. Weske. Interacting services: From specification to execution. *Data & Knowledge Engineering, Elsevier*, 68(10):946–972, 2009.
5. G. Decker, H. Overdick, and J. Zaha. On the Suitability of WS-CDL for Choreography Modeling. In *EMISA*, pages 21–33, Hamburg, Germany, October 2006.
6. R. Dijkman and M. Dumas. Service-oriented Design: A Multi-viewpoint Approach. *International Journal of Cooperative Information Systems*, 13:337–368, 2004.
7. S. I. Fernando, D. Creager, and A. Simpson. Towards Build-Time Interoperability of Workflow Definition Languages. In *SYNASC*, Washington DC, USA, 2007.
8. B. Hofreiter and C. Huemer. A model-driven top-down approach to inter-organizational systems: From global choreography models to executable BPEL. In *Join Conf CEC, EEE*, 2008.

9. IBM, SAP. *WS-BPEL Extension for Sub-Processes – BPEL-SPE*, September 2005.
10. O. Kopp, F. Leymann, and S. Wagner. Modeling Choreographies: BPMN 2.0 versus BPEL-based Approaches. In *EMISA*, 2011.
11. O. Kopp, D. Martin, D. Wutke, and F. Leymann. The Difference Between Graph-Based and Block-Structured Business Process Modelling Languages. *Enterprise Modelling and Information Systems Architecture, GI e.V.*, 4(1):3–13, 2009.
12. J. Lenhard, A. Schönberger, and G. Wirtz. Edit Distance-Based Pattern Support Assessment of Orchestration Languages. In *OTM Conferences*, Hersonissos, 2011.
13. F. Leymann. BPEL vs. BPMN 2.0: Should You Care? In *2nd International Workshop on BPMN*, 2010.
14. J. Mendling and M. Hafner. From WS-CDL choreography to BPEL process orchestration. *JEIM*, 21(5):525 – 542, 2008.
15. M. Murzek and G. Kramler. The Model Morphing Approach - Horizontal Transformations between Business Process Models. In *BIR*, pages 88 – 103, 2007.
16. OASIS. *Web Services Business Process Execution Language*, April 2007. v2.0.
17. OMG. *Business Process Model and Notation (BPMN) Version 2.0*, January 2011.
18. C. Ouyang, M. Dumas, A. H. M. ter Hofstede, and W. M. P. van der Aalst. From BPMN Process Models to BPEL Web Services. In *ICWS*, pages 285–292, 2006.
19. C. Peltz. Web Services Orchestration and Choreography. *IEEE Computer*, 36(10):46–52, October 2003.
20. 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*, LNCS, pages 353–368, Klagenfurt, Austria, October 2005. Springer, Heidelberg.
21. 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. Springer.
22. A. Schönberger, C. Wilms, and G. Wirtz. A Requirements Analysis of Business-to-Business Integration. Technical Report 83, Otto-Friedrich-Universität Bamberg, December 2009. ISSN 0937-3349.
23. A. Schönberger. The CHORCH B2Bi approach: Performing ebBP choreographies as distributed BPEL orchestrations. In *SC4B2B*, Miami, Florida, USA, July 2010.
24. A. Schönberger. Visualizing B2Bi choreographies. In *4th IEEE International Conference on Service-Oriented Computing and Applications*. IEEE, 2011.
25. S. Shetty and S. Vadivel. Interoperability issues seen in Web Services. *International Journal of Computer Science and Network Security*, 9(8):160 – 168, 2009.
26. S. Tai, R. Khalaf, and T. Mikalsen. Composition of Coordinated Web Services. In *ACM/IFIP/USENIX International Middleware Conference*, 2004.
27. W. M. P. van der Aalst, N. Lohmann, P. Massuthe, C. Stahl, and K. Wolf. From Public Views to Private Views - Correctness-by-Design for Services. In *Web Services and Formal Methods, Fourth International Workshop (WS-FM)*, Brisbane, 2007.
28. 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.
29. D. Vanderhaeghen, S. Zang, A. Hofer, and O. Adam. XMLbased Transformation of Business Process Models - Enabler for Collaborative Business Process Management. In *XML4BPM*, pages 81 – 94, 2005.
30. I. Weber, J. Haller, and J. Mülle. Automated Derivation of Executable Business Processes from Choreographies in Virtual Organisations. *IJBPM*, 3:85–95, 2008.
31. P. Wohed, W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede, and N. Russell. On the Suitability of BPMN for Business Process Modelling. In *Business Process Management*, pages 161–176, Vienna, Austria, September 2006.

A Survey on Approaches for Timed Services

Kristian Duske¹ and Richard Müller²

¹ Institut für Softwaretechnik und Theoretische Informatik
Technische Universität Berlin, Germany
kristian.duske@tu-berlin.de

² Institut für Informatik, Humboldt-Universität zu Berlin, Germany
richard.mueller@informatik.hu-berlin.de

Abstract. In the context of service-oriented computing, time has been extensively studied in literature. We present a survey on possible problem statements for timed services, and give an overview of state-of-the-art approaches. Thereby we identify which problems are already thoroughly researched and which problems warrant further research.

Keywords: SOC, timed services, behavior, quality of service, survey

1 Introduction

Service-oriented computing (SOC) [22] aims at building a complex system by composing less complex, loosely coupled building blocks called *services*. A service is an autonomous system providing its functionality via a well-defined interface. This interface is used to communicate with other services. Consequently, the *composition* of services into a new service is a key feature of SOC. Functional and non-functional *correctness* of a service composition is critical [27]. We define functional correctness in terms of the composition's *behavior*, for example deadlock freedom or weak termination. Non-functional properties refer to *Quality of Service* (QoS) dimensions like duration, response time, capacity, or reliability [7].

Time is an abstract concept which is crucial for many real-world systems like workflow systems [4], web services [3], or any kind of protocol [24]. The introduction of time to SOC affects both the behavior and the QoS of a composition [15,28]. On the one hand, timed constraints can limit the behavior of a composition. As an example, consider an airline booking service where a travel agency service may reserve a ticket for at most one hour. A travel agency service that always attempts to buy a ticket one day after reservation will always encounter a timeout. Hence, the behavior of the composition of these two services is limited to unsuccessful buying attempts. On the other hand, a composition may have to additionally satisfy timed requirements. For example, a travel agency service may prefer an airline booking service that takes less time to perform a reservation than other functionally equivalent airline booking services. In this paper, we investigate timed behavior and QoS of timed services, resp. of a timed service composition.

Given the importance and influence of time for services, there exist many different problem statements and approaches in the available literature. This makes it difficult to gather a common view on timed services. This survey elaborates on the problem

statements found in literature and gives an overview of state-of-the-art approaches. More precisely, the main contributions of this paper are as follows. The first step consists of defining a *problem space* — that is, a set of possible problem statements related to timed services. We present a problem space building upon five orthogonal problem dimensions in Sect. 2. In the second step, we *survey* a selection of existing approaches according to their corresponding problem statement in Sect. 3. Finally, we *evaluate* the state-of-the-art for every problem statement: “Is it valid and relevant?” “Is it an open problem?” “If it is solved, what is the quality of the applied approaches?”. Section 4 concludes the paper with a discussion of future work and a conclusion.

2 Problem space

Most approaches for timed services deal with a specific problem statement. Instead of classifying the approaches directly, we start by classifying the problem statements into a problem space. This way, we gain a systematic overview on timed services: While a classification of the approaches may deliver interesting results on its own, a classification of the problem statements helps identifying problem classes that require further research.

Our problem space is a systematic collection of time-related challenges that arise in SOC. Every point in the problem space represents a unique combination of certain problem *characteristics*. It is spanned by five orthogonal *dimensions*: criterion, lifecycle phase, time abstraction, timed constraint, and system (see Fig. 1 for an overview). The selected dimensions and their characteristics are tailored towards the properties of the problem statements that arise when timed services are considered.

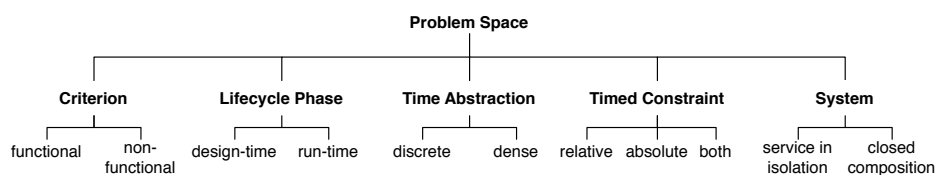


Fig. 1. Problem dimensions and their characteristics

In the following, we explain each dimension and its possible characteristics in more detail. For each, we give a description illustrated with some brief examples, explain its origin, and justify its relevance for timed services.

Criterion When dealing with timed services, two types of correctness criteria are relevant for a service-oriented system: functional and non-functional correctness. *Functional correctness* corresponds to the question whether the system works correctly — that is, the system’s behavior. It covers for example deadlock freedom, weak termination, or the satisfaction of a set of timed requirements. Functional correctness is a precondition for *non-functional correctness*, which corresponds to qualitative questions — that is, how well does the system work. Examples for non-functional correctness are Quality of Service criteria — that is, duration, response time, capacity (messages per time unit), reliability, or price.

Lifecycle phase A lifecycle is a structure consisting of distinguishable phases that is imposed on the development of a system. In this paper, we consider design-time and run-time as characteristics. *Design-time* is the phase where the services and compositions are defined and implemented. Here we consider preconceived compositions, models and static service definitions. *Run-time* is the phase during which the system is executed. At run-time, we may have different information at our disposal, leading to new problems like monitoring, run-time adaption, instance migration, or re-configuration.

The aforementioned problem dimensions — criterion and lifecycle phase — are ubiquitous and inevitable, and concern any kind of information system [25,10].

Time abstraction There are two ways of introducing time into a system, distinguishable by a different resolution. *Discrete time* uses a domain with countably many time values, while *dense time* uses a domain with uncountably many time values. A problem which is decidable in discrete time may become undecidable when regarded with dense time.

Timed constraint We have two combinable ways of expressing timed constraints. Firstly, timed constraints can be *relative* to some event like the occurrence of an action or the entry of a state. Secondly, timed constraints can be *absolute* — that is, referring to a commonly known point in time.

The problem dimensions time abstraction and timed constraint are inherent to any kind of information system dealing with a notion of time. However, the last dimension — system — is a direct result of our service-oriented setting.

System In SOC we distinguish between *open* and *closed* systems — that is, between a service in isolation and a closed composition. Both systems elicit fundamentally different questions. For a service in isolation, questions like well-formedness or controllability arise. By contrast, questioning for example deadlock freedom is meaningful for a closed composition only.

The combination of all problem characteristics yields 48 different classes. In the next section, we present the survey and classify all examined approaches for timed services according the defined problem space.

3 Survey

We conduct the survey following a three-step approach by Levy and Ellis [20]: Firstly, we query Google Scholar¹, Mendeley², DBLP³, and IEEE Xplore⁴ by keywords (“service”, “soc”, “time”, “timing”, “realtime”, “timed constraints”, “timed requirements”, “quality of service”), and conduct both backward and forward search on the found literature. This yields a total of 55 papers, 26 of which we find relevant to this survey. Secondly, we analyze the approaches related to timed services. Finally, we classify them according to their problem statements; see Table 1 for the result.

¹ <http://scholar.google.de/>

² <http://www.mendeley.com/>

³ <http://dblp.uni-trier.de/>

⁴ <http://ieeexplore.ieee.org>

We now give a brief overview of the surveyed approaches, which we group into approaches focusing on functional correctness and approaches dealing with non-functional correctness. Later, we summarize our findings.

3.1 Functional correctness

Most of the surveyed approaches focus on a service or a composition specified in BPEL. The general approach consists of providing BPEL with a formal semantics for verification purposes.

Mateescu et al. [21] propose a translation of BPEL to discrete-time labelled transition systems which handle activity durations and timeouts. Timed safety and liveness properties are analyzed using model checking tools.

Kazhamiakin et al. [19] use web service timed transition systems (WSTTS) for the analysis and verification of timing aspects of BPEL4WS compositions. WSTTS are a variant of timed automata tailored towards web services. All time-related constructs of BPEL4WS incl. absolute timeouts are supported. Timed requirements can be expressed using a discrete subset of duration calculus. Additionally, the authors present an algorithm to compute extremal bounds of the execution duration of a composition.

Guermouche et al. [14] propose the FIACRE verification language as their underlying formalism. It supports a rich set of timed constraints (activity durations, message delays and timeouts) which is further separated into local and global constraints. Due to this distinction, the authors can formulate a well-formedness criterion for isolated services and a compatibility criterion for service compositions. Furthermore, their approach supports rich timed requirements using a timed leads to operator.

Similarly, Fares et al. [13] capture both the behavioral and the timing aspects of all BPEL 2.0 constructs by mapping them to FIACRE. Timed requirements can be formulated as LTL formulas and are verified against a service composition using the ToolBox Tina.

Kallel et al. [18] employ XTUS-automata for the specification and verification of relative and absolute timed constraints. The authors propose to use existing model checking tools to verify functional correctness criteria such as deadlock freedom. In addition, they present a translation of the formal specifications to AO4BPEL aspects which enforce the temporal constraints at run-time.

The approach proposed by Song et al. [26] aims to verify timed requirements for a BPEL composition with the goal of identifying other services suitable for composition at run-time. It is based on a mapping to Time Petri-nets and an algorithm to compute the extremal bounds of the time interval between two transitions. The approach only supports the specification of simple timed requirements on a time interval between transitions.

Haddad et al. [15,16] treat functional correctness of isolated services. They introduce an algorithm that either generates a correct interaction controller for a given BPEL specification or detects whether the specification is ambiguous. The absence of ambiguity can be regarded as a correctness criterion for isolated services.

Benatallah et al. [2] describe an extension for business protocols with timeouts on the states of a protocol. Based on these timed business protocols, they formulate the notions of time-dependent compatibility and replaceability.

Ponge et al. [23,24] extend this approach by introducing richer timed constraints and fine-grained classes of time dependent compatibility and replaceability properties. These classes are characterized by a set of operators that manipulate and analyze timed protocols. A mapping from timed protocols to a special class of timed automata allows the authors to derive decidability results for these operators. These results come at the cost of requiring deterministic service behavior.

The approach presented by Berardi et al. [3] employs timed finite state automata to represent two types of relative timed constraints (timeouts and durations). Again, deterministic behavior is a requirement for the services.

Čaušević et al. [9] extend the resource-aware, timed hierarchical language REMES for behavioral service modeling. They focus on service capacity and time-to-serve as timed requirements. The correctness of a service composition can be verified by employing Dijkstra's and Scholten's strongest postcondition semantics. The approach is limited to synchronous communication between the services.

Zahoor et al. [29] introduce a declarative approach for modeling web services based on event calculus. Given the composition design with a timed properties representation, an event calculus reasoner can be used to compute a solution satisfying associated timed properties. The approach supports synchronous and asynchronous communication.

de Alfaro et al. [12] present an approach to check the compatibility of timed interfaces. A timed interface is a specification of the input assumptions and output guarantees (incl. timing) of a component. The authors develop a well-formedness and a compatibility criterion for such timed interfaces and present algorithms to decide these properties.

Based on this work, Henzinger et al. [17] present an interface algebra for real-time components. Here, an interface specifies guaranteed task latencies depending on assumptions about task arrival rates and allocated resource capacities. Interface compatibility can be checked on partial designs. An interface is comparable to a stateless service.

3.2 Non-functional correctness

In the area of non-functional correctness, there are several approaches that deal with the computation and optimization of the QoS of a service composition at runtime. Cardoso et al. [7] propose a predictive QoS model that allows the computation of the QoS of a workflow from the QoS values of the tasks. Task QoS values are updated at runtime by monitoring the execution of the workflow, and the approach uses probability estimates for workflow transitions during the computation of the overall QoS. Many of the other approaches in the area cite this model or propose a similar one. Zhao et al. [31] also present a QoS model that allows the computation of overall QoS of a service choreography specified in the Chor language.

Based on the assumption that several functionally equivalent services exist for each activity of the workflow, the QoS of a service composition can be optimized by computing an optimal selection of participating services. Zeng et al., Canfora et al. and Aggarwal et al. all propose similar approaches which only differ in the used QoS model and optimization method [1,6,30]. Canfora et al. [5] present a method to handle expected QoS violations by replacing services during the execution of a service composition. Cardoso et al. [8] propose an approach to include QoS values to select services that do not violate the QoS requirements of a composition.

3.3 Findings

Table 1 presents the classification of the surveyed approaches according to their respective problem statements. Some approaches show up multiple times because they attack more than one problem class. Notice that any approach which supports dense time naturally also supports discrete time. However, in such cases a discrete time approach may exist which has a lower computational complexity. Hence, we do not list dense time approaches in the row of discrete time approaches. Only three approaches deal with both absolute and relative timed constraints [13,18,19]. Since every other approach is limited to relative constraints, we omit the timed constraint dimension in Table 1.

		functional		non-functional	
		open	closed	open	closed
discrete	design-time	[21]	[3], [23], [2], [29], [19]		
	run-time		[29]		
dense	design-time	[16], [15], [12], [14], [17]	[12], [24], [14], [18], [9], [13]	[7], [31]	[7], [31]
	run-time		[26]	[30], [6], [1], [5], [8]	[18], [30], [6], [1], [5], [8], [11]

Table 1. Classification of the surveyed approaches

There are two clusters of approaches. Each cluster represents a problem which has been treated by several authors with different formalisms. The largest cluster of approaches deals with ensuring functional correctness of a service composition at design-time. Most of the approaches in this cluster use dense time. Naturally, the approaches vary in their expressiveness, but this is outside the scope of this paper. It should also be noted that some approaches consider deterministic services only [2,3,12,23,24].

The second significant cluster is in the area of ensuring non-functional correctness at run-time. Most authors propose similar approaches to optimize the overall QoS of a workflow by computing an optimal selection of services. The underlying assumption is that for each activity of the workflow there exist several functionally equivalent services. They solve the resulting optimization problem with methods like linear programming [1,30] or genetic algorithms [5,6]. Other approaches in this cluster deal with monitoring temporal QoS constraints [18] or semantic service composition [8].

There are several sparsely populated spots in the problem space. Only two approaches deal with functional correctness at run-time [26,29], both with limited expressiveness. This may be due to the fact that the computational complexity of all approaches that attack functional correctness is so high that their application at run-time is not feasible in realistic scenarios [26].

Another area where we found little to no existing research is the problem of verifying non-functional correctness at design-time. It can be argued that reasoning about non-functional correctness criteria at design time is not very useful because QoS attributes of service must be constantly updated and monitored at run-time and thus an optimal selection of services for a composition can only be computed at run-time.

4 Discussion and Conclusion

In this paper, we survey approaches for timed services. We define a problem space with five dimensions and classify the surveyed approaches according to this problem space. Thereby, we identify which problems are already thoroughly researched and which problems warrant further research.

It could be debated that our problem space is inaccurate and that we should include more dimensions. Indeed, it would be interesting to differentiate the surveyed approaches further by their chosen formalism(s) or verification techniques. Nonetheless, such features are not related to the problem space and should therefore be analyzed separately. In any case, this could not be included in this paper due to space limitations.

Currently, we distinguish only between design-time and run-time. These phases of a traditional software lifecycle may be insufficient to describe the lifecycle of a service-oriented system. We are going to investigate whether the lookup and composition phase should be separated from the run-time phase.

We identify two areas in the domain of timed services which warrant further research: functional correctness at run-time and non-functional correctness at design-time. We intend to focus on the former problem class in the future. Hence, our future work is divided into three areas. Firstly, we will extend this survey by including more approaches and by classifying the approaches by their features — that is, their formalism and verification technique. Secondly, we will research correctness criteria for isolated timed services. We are particularly interested in the properties of well-formedness and controllability of non-deterministic, asynchronously communicating timed services. Thirdly, we plan to investigate methods for verifying functional correctness at run-time, specifically during the composition phase we mentioned above. Due to the dynamic nature of service-oriented systems, it is not sufficient to ensure functional correctness at design time. However, the existing approaches cannot be easily employed at run-time because of their computational complexity.

References

1. Aggarwal, R., Verma, K., Miller, J., Milnor, W.: Constraint Driven Web Service Composition in METEOR-S. In: IEEE International Conference on Services Computing (2004)
2. Benatallah, B., Casati, F., Ponge, J., Toumani, F.: On Temporal Abstractions of Web Service Protocols. In: Proceedings of the CAiSE Forum (2005)
3. Berardi, D., De Rosa, F., De Santis, L., Mecelia, M.: Finite state automata as conceptual model for e-services. Contract (2003)
4. Bettini, C., Wang, X., Jajodia, S.: Temporal reasoning in workflow systems. Distributed and Parallel Databases 11(3) (2002)
5. Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: QoS-Aware Replanning of Composite Web Services. IEEE (2005)
6. Canfora, G., Penta, M.D., Esposito, R., Villani, M.L.: An approach for QoS-aware service composition based on genetic algorithms. Genetic And Evolutionary Computation Conference (2005)
7. Cardoso, J., Miller, J., Sheth, A., Arnold, J.: Modeling Quality of Service for Workflows and Web Service Processes. Tech. rep. (2004)

8. Cardoso, J., Sheth, A.: Semantic E-Workflow Composition. *Journal of Intelligent Information Systems* (2003)
9. Causevic, A., Seceleanu, C., Pettersson, P.: Modeling and Reasoning about Service Behaviors and Their Compositions. *Lecture Notes in Computer Science* (2010)
10. Chung, L., do Prado Leite, J.: On non-functional requirements in software engineering. *Conceptual modeling: Foundations and applications* (2009)
11. Cristian, F., Fetzer, C.: The timed asynchronous distributed system model. *Parallel and Distributed Systems, IEEE Transactions on* (1999)
12. De Alfaro, L., Henzinger, T.A., Stoelinga, M.: Timed Interfaces. *International Conference on Embedded Software* (2002)
13. Fares, E., Bodeveix, J.P., Filali, M.: Verification of Timed BPEL 2.0 Models. *Tech. rep.* (2011)
14. Guermouche, N., Zilio, S.D.: Formal Requirement Verification for Timed Choreographies. *Tech. rep.* (2011)
15. Haddad, S., Melliti, T., Moreaux, P.: Modelling web services interoperability. *International Conference on Enterprise Information Systems* (2004)
16. Haddad, S., Moreaux, P., Rampacek, S.: Client Synthesis for Web Services by way of a Timed Semantics. *International Conference on Enterprise Information Systems* (2006)
17. Henzinger, T.A., Matic, S.: An Interface Algebra for Real-Time Components. *IEEE RealTime and Embedded Technology and Applications Symposium* (2006)
18. Kallel, S., Charfi, A., Dinkelaker, T., Mezini, M., Jmaiel, M.: Specifying and Monitoring Temporal Properties in Web Services Compositions. *IEEE European Conference on Web Services* (2009)
19. Kazhamiakin, R., Pandya, P., Pistore, M.: Representation, verification, and computation of timed properties in web service compositions. *IEEE International Conference on Web Services* (2006)
20. Levy, Y., Ellis, T.J.: A systems approach to conduct an effective literature review in support of information systems research. *Informing Science Journal* 9 (2006)
21. Mateescu, R., Rampacek, S.: Formal modelling and discrete-time analysis of BPEL web services. *International Journal of Simulation and Process Modelling* (2008)
22. Papazoglou, M.: *Web Services - Principles and Technology*. Prentice Hall (2008)
23. Ponge, J., Benatallah, B., Casati, F., Toumani, F.: Fine-grained compatibility and replaceability analysis of timed web service protocols. In: *International Conference on Conceptual Modeling* (2007)
24. Ponge, J., Benatallah, B., Casati, F., Toumani, F.: Analysis and applications of timed service protocols. *ACM Transactions on Software Engineering and Methodology* (2010)
25. Ramamoorthy, C.V., Prakash, A., Tsai, W.T., Usuda, Y.: *Software Engineering: Problems and Perspectives*. Computer (1985)
26. Song, W., Ma, X., Ye, C., Dou, W., Lü, J.: Timed Modeling and Verification of BPEL Processes Using Time Petri Nets. *International Conference on Quality Software* (2009)
27. Ter Beek, M., Bucchiarone, A., Gnesi, S.: Formal methods for service composition. *Annals of Mathematics, Computing & Teleinformatics* (2007)
28. Tsai, W.T., Lee, Y.h., Cao, Z., Chen, Y., Xiao, B.: RTSOA: Real-Time Service-Oriented Architecture. *IEEE International Symposium on ServiceOriented System Engineering* (2006)
29. Zahoor, E., Perrin, O., Godart, C.: A declarative approach to timed-properties aware Web services composition. *Event* (2010)
30. Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., Sheng, Q.Z.: Quality driven web services composition. *International World Wide Web Conference* (2003)
31. Zhao, X., Cai, C., Yang, H., Qiu, Z.: A QoS View of Web Service Choreography. *IEEE International Conference on e-Business Engineering* (2007)

Weak Conformance of Process Models with respect to Data Objects

Andreas Meyer, Artem Polyvyanyy, and Mathias Weske

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

Abstract. Process models specify behavioral aspects by describing ordering constraints between tasks which must be accomplished to achieve envisioned goals. Tasks usually exchange information by means of data objects, i.e., by writing information to and reading information from data objects. A data object can be characterized by its states and allowed state transitions. In this paper, we propose a notion which checks conformance of a process model with respect to data objects that its tasks access. This new notion can be used to tell whether in every execution of a process model each time a task needs to access a data object in a particular state, it is ensured that the data object is in the expected state or can reach the expected state and, hence, the process model can achieve its goals.

1 Introduction

Process modeling usually comprises two aspects: The control flow perspective and the data flow perspective [1]. Control flow defines possible execution sequences of tasks, whereas data flow provides means for exchanging information between the tasks. Information gets passed between tasks of a process model by writing to and reading from data objects. A data object can be formalized as a set of data states and transitions between the data states, i.e., as a labeled transition system, which is usually referred to as an object life cycle. An object life cycle can be used to identify the current data state of the data object and the set of its reachable data states from the current one [2]. Similarly, the execution semantics of process models is often defined by employing the notion of a process state that defines a set of tasks which can be performed. A process state changes once a task gets accomplished. A process state together with all data states (one for each data object) collectively define a state of a process instance. It is usually accepted that control flow drives execution of process models, i.e., a change in a state of a process instance is triggered by a change of a process state, which in turn may activate changes of data states.

In order to achieve safe execution of a process model, it must be ensured that every time a task attempts to access a data object, the data object is in a certain expected data state or is able to reach the expected data state from the current one, i.e., object life cycles of data objects must conform to the process model; otherwise, the execution of a process model may deadlock, i.e., terminate prior to

reaching the goal state. In this paper, we propose a notion of weak conformance which allows for a precise characterization of the above described intuition, where “weak” reflects the fact that data states are required to be reachable via arbitrary number of data state transitions and not necessarily via a single one.

In a process model which satisfies weak conformance with respect to its data objects, it is assumed that implicit data state transitions get realized by an external entity or by detailed implementations of process model tasks. Relevance for the new notion is based on the need to check for conformance of underspecified process models where, e.g., external events, not captured in the process model, change states of data objects. Events and tasks, being part of but not modeled in the process model, may also change the states of data objects. These modeling artifacts are, for instance, hidden in subprocess structures, so that process models and object life cycles are specified at different levels of detail. Practically, process models still conform to their used data objects if the hidden state changes do not contradict against data object life cycles.

The remainder of the paper proceeds as follows: The next section describes process scenarios – a formalism which integrates control flow and data flow aspects of process modeling. In Section 3, we define the notion of weak conformance of the process model from a process scenario with respect to data objects it operates with. Section 4 is devoted to related works. Finally, Section 5 draws conclusion.

2 Process Scenarios

In this section, we propose *process scenarios* – a formalism for designing concurrent systems which integrates control flow and data flow perspectives. A process scenario consists of two parts: (i) a process model which orchestrates the execution of tasks, and (ii) data objects which describe what information do tasks require to be executed and/or what information do tasks produce. We start the discussion with the definition of the first part – a process model.

Definition 1 (Process model).

A *process model* is a tuple $M = (A, G, D, R, C, F, type, \mathcal{A}, \mu)$, where A is a finite set of *tasks*, G is a finite set of *gateways*, D is a finite set of *data objects*, R is a finite set of *data states*, $C \subseteq (A \cup G) \times (A \cup G)$ is the *control flow* relation, $F \subseteq (A \times (D \times R)) \cup ((D \times R) \times A)$ is the *data flow* relation, $type : G \rightarrow \{xor, and\}$ assigns to each gateway a type, \mathcal{A} is a finite set of *names* such that $\tau \in \mathcal{A}$ (A, G, D, R , and \mathcal{A} are pairwise disjoint), and $\mu : A \rightarrow \mathcal{A}$ assigns to each task a name.

We use subscripts, e.g., A_M, G_M , and μ_M , to denote the relation of the sets and functions to process model M , and omit subscripts where the context is clear. We refer to the set $A \cup G$ as *nodes* of process model M . If $\mu(a) \neq \tau$, $a \in A$, then a is *observable* in M ; otherwise a is *silent* in M . We expect that every process model M fulfills basic structural correctness requirements: (i) every task of M has at most one incoming and at most one outgoing control flow edge, (ii) every gateway has at least three incident control flow edges, (iii) M has exactly one *source* task and at least one *sink* task (the source has exactly one outgoing and no incoming control flow edges, while each sink has exactly one incoming and no

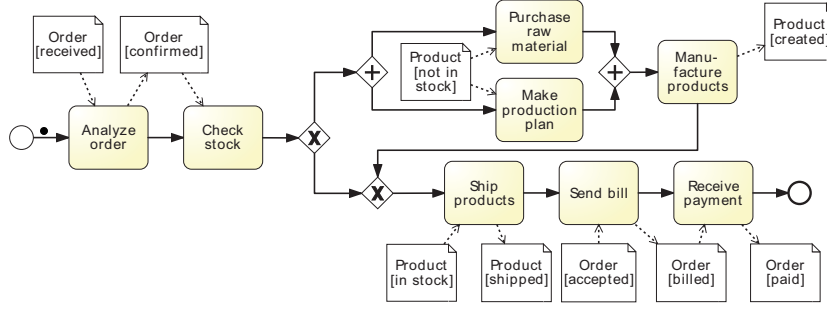


Fig. 1. A simple “order delivery and payment” process model

outgoing control flow edges), (iv) the source and all sinks of M are silent tasks, whereas all other tasks are observable, (v) every node of M is on a path from the source to some sink, and (vi) there exist no data flow edges which involve silent tasks, i.e., $\nexists(a, (d, r)) \in F : \mu(a) = \tau$ and $\nexists((d, r), a) \in F : \mu(a) = \tau$.

We adapt the notation similar to BPMN [3] for visualization of process models. An observable task is drawn as a rectangle that has rounded corners with its name inside. Source and sink tasks are visualized as start and end BPMN events, respectively. Gateways are drawn as diamonds. We call a gateway $g \in G_M$ of M an *xor* (an *and*) gateway, if $type_M(g) = xor$ ($type_M(g) = and$). An *xor* (an *and*) gateway uses a marker which is shaped like “ \times ” (“ $+$ ”) inside the diamond shape. A data object (in a particular data state) is visualized as a BPMN data object. A data object $d \in D_M$ can appear multiple times in the visualization of the process model (also when in a particular data state $r \in R_M$). Control flow and data flow edges are drawn as solid and dashed directed edges, respectively.

The semantics of a process model is defined as a token game. A marking of a process model is represented by tokens on its control flow edges. Given process model M , a *marking* (or a *process state*) of M is a mapping $m : C_M \rightarrow \mathbb{N}_0$ (\mathbb{N}_0 is the set of natural numbers including zero). Fig. 1 shows a process model in its *initial* process state – a process state which puts one token on the only outgoing control flow edge of the source and no tokens elsewhere. Every node of a process model (except silent tasks) can be executed. The execution of an observable task removes one token from its only incoming and adds one token on its only outgoing control flow edge. The execution of an *and* gateway removes one token from each of its incoming control flow edges and then adds one token on each of its outgoing control flow edges. The execution of an *xor* gateway removes one token from one of its incoming control flow edges and afterwards adds one token on one of its outgoing control flow edges; the choice of the incoming edge as well as of the outgoing edge is done nondeterministically. Observe that we abstract from data-based decisions that are usually used to control the semantics of *xor* gateways. Let m and m' be two markings of M . We write $m \xrightarrow{x} m'$ to denote that m changes to m' by executing node x of M . If $\sigma = a_1 a_2 \dots a_n$, $n \in \mathbb{N}_0$, is a sequence of nodes of M , $m \xrightarrow{\sigma} m'$ denotes the fact that there exists a sequence of process states $m_1 m_2 \dots m_{n-1}$ such that $m \xrightarrow{a_1} m_1 \xrightarrow{a_2} \dots m_{n-1} \xrightarrow{a_n} m'$. We call

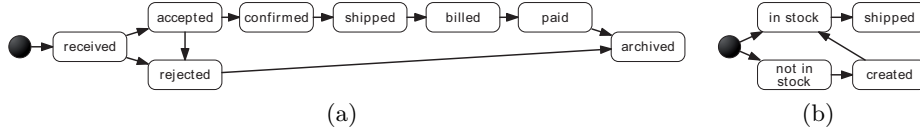


Fig. 2. Object life cycles of (a) “Order” and (b) “Product” data objects

σ an *execution sequence* of M which starts with m . Let a and a' be two nodes of M . With $a \Rightarrow_M a'$ we denote the predicate which evaluates to **true** if $a = a'$ or there exists an execution sequence of M which starts with the initial marking and executes a before a' ; otherwise $a \Rightarrow_M a'$ evaluates to **false**.

Next, we proceed with the definition of an object life cycle.

Definition 2 (Object life cycle).

An *object life cycle* is a tuple $L = (S, \Sigma, \mapsto, i)$, where S is a finite set of *data states*, Σ is a finite set of *actions* (S and Σ are disjoint), $\mapsto \subseteq S \times \Sigma \times S$ is the *data state transition* relation, and $i \in S$ is the *initial* data state.

We use subscripts S_L, Σ_L, \mapsto_L , and i_L , to denote the relation of the elements to the object life cycle L . Note that we omit subscripts where the context is clear. For $s, s' \in S$ and $a \in \Sigma$ we denote by $s \xrightarrow{a} s'$ the fact that $(s, a, s') \in \mapsto$. If $\sigma = a_1 a_2 \dots a_n, n \in \mathbb{N}_0$, is a sequence of actions, $s \xrightarrow{\sigma} s'$ denotes the fact that there exists a sequence of data states $s_1 s_2 \dots s_{n-1}$ such that $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots s_{n-1} \xrightarrow{a_n} s'$. We call σ an *execution sequence* of L which starts with s , and s' is a *reachable data state from data state s via σ* . With $s \Rightarrow_L s'$ we denote the predicate which evaluates to **true** if $s = s'$ or there exists an execution sequence of L which starts with i_L and reaches s before s' ; otherwise $s \Rightarrow_L s'$ evaluates to **false**.

Finally, a process scenario is defined as follows.

Definition 3 (Process scenario).

A *process scenario* is a tuple $H = (M, \mathcal{L}, \omega)$, where M is a process model, \mathcal{L} is a finite set of object life cycles, and $\omega : D_M \rightarrow \mathcal{L}$ assigns to each data object of M an object life cycle.

Note that we assume that for a process scenario $H = (M, \mathcal{L}, \omega)$ it holds that ω is injective and $\bigcup_{L \in \omega(D_M)} S_L \subseteq R_M$. Fig. 1 and Fig. 2 visualize a process scenario. The process model of the scenario is given in Fig. 1. It contains two data objects: “Order” and “Product”. The life cycles of these data objects are shown in Fig. 2(a) and Fig. 2(b), respectively.

3 Weak Conformance

Prior to proceeding with the definition of weak conformance, we define several notions for convenience considerations. Let $f \in F_M$ be a data flow edge of process model M . With f_A, f_D , and f_R we denote the task, data object, and data state component of f , respectively. For instance, if f is equal to $(a, (d, r))$ or to $((d, r), a)$, then (in both cases) $f_A = a, f_D = d$, and $f_R = r$. We call f an *input* data flow edge if $f \in ((D \times R) \times A)$, and an *output* data flow edge if $f \in (A \times (D \times R))$.

Definition 4 (Weak data object conformance).

Given process scenario $H = (M, \mathcal{L}, \omega)$, $M = (A, G, D, R, C, F, type, \mathcal{A}, \mu)$, M satisfies weak conformance with respect to data object $d \in D$ if for all $f, f' \in F$ such that $f_D = d = f'_D$ holds $f_A \Rightarrow_M f'_A$ implies $f_R \Rightarrow_{\omega(d)} f'_R$, and $f_A = f'_A$ implies f is an input edge and f' is an output edge.

Given a process scenario, we say that the process model satisfies weak conformance, if it satisfies weak conformance with respect to each of its data objects. Weak data object conformance is satisfied if for each two succeeding data states of a data object there exists an execution sequence from the first to the second data state in the corresponding object life cycle. Two data states are succeeding in the process model if either (i) they are accessed by the same task with one being part of an input and one being part of an output data flow edge, or (ii) there exists an execution sequence in the process model in which two different tasks access the same data object in two data states.

The process model in Fig. 1 satisfies weak conformance with respect to data object “*Product*” and does not satisfy weak conformance with respect to data object “*Order*”. Indeed, there exists an execution sequence which visits task “*Analyze order*” before task “*Send bill*”, which access data object “*Order*” in data states “*confirmed*” and “*accepted*”, respectively. However, data state “*accepted*” is not reachable in the object life cycle in Fig. 2(a) via data state “*confirmed*”. One can fix this flaw, for instance, by changing the data state of the only input data flow of “*Send bill*” task from “*accepted*” to “*confirmed*”, which also modifies the process model to satisfy weak conformance.

Comparing the proposed notion of weak conformance to the one introduced in [4], the given process model would not satisfy conformance with respect to data object “*Product*”. In [4], the authors rely on process models with fully specified data information. For instance, task “*Ship products*” reads data object “*Product*” in data state “*in stock*”. However, in [4], it is required that there exists a preceding task which writes “*Product*” in that state. As such task does not exist, conformance is not satisfied.

4 Related Work

Process models which follow on the imperative design paradigm have been studied extensively [1]. The increasing interest in the development of process models for execution has shifted the focus from control flow to data flow perspective. The first step in this regard are artifact-centric processes introduced in [5]. Artifact-centric processes connect data objects with the control flow of process models by specifying object life cycles which represent data dependencies and based thereon, the order of task execution. In [6,4], the authors present an approach which connects object life cycles with process models by determining commonalities between both representations and transforming one into the other. In [7], a rule-based approach is described; it allows to connect control flow with data flow and, thus, to automatically create data-driven executable process models. In terms of data-driven execution, case handling [8] plays a major role, as in case

handling data dependencies solely determine the order of task execution. In this paper, we also concentrate on the integrated scenarios which incorporate process models and object life cycles. However, we remove the assumption that all the approaches mentioned above follow, i.e., both representations must completely correspond to each other. Instead, we set object life cycles of data objects as references that describe what can be utilized by process models.

Compliance, or correctness, in process models mostly refers to checks of the process model with respect to a defined rule set containing, for instance, business policies. The field of compliance is well researched [9,10,11,12] and has already been tackled for artifact centric processes, e.g., [13]. A different type of compliance is introduced in [4]. There, compliance between a process model and an object life cycle of one data object used in the process model is defined as the combination of object life cycle conformance (all data state transitions induced in the process model must occur in the object life cycle) and coverage (opposite containment relation). In this paper, we proposed the definition of a similar type of compliance. As we set object life cycles to be the reference, we assume them to be correct and, therefore, we can restrict the compliance check to conformance only. For conformance, instead of working with direct data state transitions we rely on data state reachability.

5 Conclusion

In this paper, we proposed a notion to check for weak conformance between a process model and object life cycles of its utilized data objects. A process model satisfies weak conformance if every time it is allowed to access states of a data object in a specific order (in the process model), these data states can also be reached in an object life cycle of the data object in the very same order.

In future works, we plan to propose an algorithm to perform analysis checks based on the notion of weak conformance introduced in this paper. For process models which do not satisfy weak conformance, one can suggest, whenever applicable, changes to the process model so that the resulting model conforms to its data objects. Process model modifications may also be applicable to already conforming process models in order to simplify their structure while preserving the conformance property. Furthermore, in process scenarios with “large” object life cycles, a conforming process model can determine the relevant aspects so that the object life cycles get tailored towards the specific needs of process scenarios and, in this way, become better understandable.

References

1. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer (2007)
2. Booch, G.: Object-Oriented Analysis and Design with Applications (3rd Edition). Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA (2004)
3. OMG: Business Process Model and Notation (BPMN), Version 2.0 (January 2011) <http://www.omg.org/spec/BPMN/2.0/> accessed March 1, 2012.

4. Küster, J., Ryndina, K., Gall: Generation of Business Process Models for Object Life Cycle Compliance. In: *Business Process Management*, Springer (2007) 165–181
5. Nigam, A., Caswell, N.: Business artifacts: An approach to operational specification. *IBM Systems Journal* **42**(3) (2003) 428–445
6. Ryndina, K., Küster, J., Gall: Consistency of Business Process Models and Object Life Cycles. In: *MoDELS Workshops*, Springer (2006) 80–90
7. Müller, D., Reichert, M., Herbst, J.: A New Paradigm for the Enactment and Dynamic Adaptation of Data-driven Process Structures. In: *Advanced Information Systems Engineering*, Springer (2008) 48–63
8. van der Aalst, W., Weske, M., Grünbauer, D.: Case Handling: A New Paradigm for Business Process Support. *Data and Knowledge Engineering* **53**(2) (2005) 129–162
9. Awad, A.: A Compliance Management Framework for Business Process Models. PhD thesis, Hasso Plattner Institute (2011)
10. Goedertier, S., Vanthienen, J.: Designing Compliant Business Processes with Obligations and Permissions. In: *BPM Workshops*, Springer (2006) 5–14
11. Governatori, G., Milosevic, Z., Sadiq, S.: Compliance checking between business processes and business contracts. In: *EDOC*, IEEE (2006) 221–232
12. Agrawal, R., Johnson, C., Kiernan, J., Leymann, F.: Taming Compliance with Sarbanes-Oxley Internal Controls Using Database Technology. In: *International Conference on Data Engineering*, IEEE (2006) 92–101
13. Lohmann, N.: Compliance by design for artifact-centric business processes. In: *Business Process Management*, Springer (2011) 99–115

Towards Verification of Process Merge Patterns with Allen's Interval Algebra

Sebastian Wagner, Oliver Kopp, and Frank Leymann

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

Abstract Choreographies present how parties collaborate to achieve an agreed business objective. When companies are bought, their processes have to be in-sourced. Thereby, their part in a choreography has to be merged with the part of their acquiring business partner. Merging patterns may be applied to merge reoccurring activity combinations, such as send/receive. It has to be proven that each merge patterns keeps the relations of the original activities of the choreography. As a first step, we show by an example how the relations between activities may be expressed using the Allen calculus. We show for merging a synchronous message exchange, which relations have to be considered for validating an implementation of that merge.

1 Introduction

In today's business scenarios enterprises often have to collaborate to achieve an agreed business objective. This is especially true if sophisticated goods such as planes, cars, engines, etc. have to be developed. The steps that have to be performed by each company are usually defined by the respective business process model or orchestrations. To reach the overall business objective, the collaboration behavior between these different process models can be modeled by a choreography that describes the interaction behavior between the activities of the involved processes usually in form of message exchanges [13]. Choreographies may be modeled using interaction models or interconnection models [3]. In the following, we focus on interconnection models, where the publicly observable behavior of each participant in a choreography is modeled as process and where the communication activities are wired together.

As in-sourcing or back-sourcing becomes more and more common nowadays, the process models of the outsourced partner have to be reintegrated into the choreography. To accomplish that we introduced an approach to consolidate (merge) process models that are part of a choreography [16]: Pairs of sending and receiving activities are transformed to value-assignment activities. In ongoing work, we extend the approach to use *merge patterns* describing merges of structures such as while loops or a one-to-many send. Thereby, we want to show that the patterns keep the control flow dependencies between the activities. In other words, the control-flow dependencies between the activities in the merged choreography have to be the same as the dependencies between the activities in the original choreography. We plan to show that by using the Allen Calculus that is also referred to as interval algebra [1]. This paper presents a first informal mapping of a

subset of BPEL’s constructs to the Allen calculus. For one merge pattern, the properties to be considered are described.

Consequently, the remainder of this paper is structured as follows: Section 2 provides a brief overview about the choreography notation BPEL4Chor and the Allen calculus. Section 3 provides an overview on the merge approach and a rendering of the choreography using the Allen calculus. Section 4 presents the properties to be kept when applying the merge pattern for asynchronous communication. After discussing related work in Sect. 5, Sect. 6 concludes the work and provides an outlook.

2 Preliminaries

The consolidation approach that is described here is designed for BPEL process models [12] that are part of a BPEL4Chor [4] choreography as BPEL is still the de-facto standard for describing and enacting processes. Even if BPEL is not formalized, we use the understanding of one of its inventors to capture the relations between activities formally. If we use a formal meta model, the mapping of BPEL to a meta model still is subjective.

BPEL offers the `invoke` activity to send messages. In its synchronous form, it also waits until a `reply` message is received. In the asynchronous form, it solely sends a message. Messages may be received by `receive` activities. A `reply` to a synchronous call is realized by a `reply` activity. The terms “synchronous” and “asynchronous” do not state anything about the underlying messaging transport used. For instance, if Java Messaging Service [15] is used, the transport is always asynchronous even if the operation invoked at the partner is a request/response operation.

To model a choreography BPEL4Chor provides message links to interconnect the activities of the involved process models. For asynchronous `invoke/receive` communication between two processes BPEL4Chor requires that one message link has to be modeled between the two activities. In a synchronous communication scenario two message links have to be modeled, one from `invoke` to `receive` activity and another one from the `reply` to the `invoke` activity. BPEL offers a rich set of control-flow constructs. It offers block-structured constructs (such as `while` for while loops) and graph-based constructs (such as `flow` with links to model acyclic graphs) [6]. We use the graph-based part using a `flow` activity. We assume that BPEL’s dead path elimination is activated and the default join condition is used. This causes an activity to be executed if at least one of its incoming links is “not dead”.

In this paper we present the idea to use the Allen’s interval algebra to verify the correctness of a merge pattern. Currently, there is no merge pattern for BPEL’s scopes and loops. Therefore, we omit loops, event handling, fault handling, termination handling, and compensation handling in this paper.

To verify merge patterns, the control flow relations between the activities of the BPEL4Chor choreography are captured using Allen’s interval algebra [1]. This algebra defines 13 distinct basic relations that can be defined between two intervals a and b that are depicted in Fig. 1. Using these basic relations, more complex relations between two intervals can be defined, e. g., the relation $a\{<,d\}b$ denotes that A exists either before or during B . The composition operation $R' \otimes R''$ of two intervals R' and R'' is provided to

calculate the transitive relations between the intervals a and c , where $aR'b$ and $bR''c$. To derive the composition of two relations, their basic relations are pairwise composed, i. e., $R' \otimes R'' = \{r' \otimes r'' | r' \in R', r'' \in R''\}$. The result of a composition of the basic relations is defined by the composition table that is provided by Allen [1]. For the intervals $a\{<\}b$ and $b\{<\}c$ the composition operation is $R' \otimes R'' = <$, i. e., a before c .

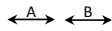
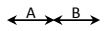
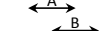
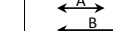


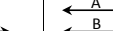
A before B: A<B B after A: B>A 	A meets B: AmB B met-by A: AmiB 	A overlaps B: AoB B overlapped-by A: AoiB 	A starts B: AsB B started-by A: AsiB 	A finishes B: AfB A finished-by B: AfiB 	A during B: AdB B contains AdB 	A equals B: AeB 
--	---	---	--	---	--	--

Figure 1. Allen’s Interval Relations

In the approach described in this work we use the Allen calculus to determine the relations between activities instead of intervals. The advantage of using the Allen calculus is that it is a full algebra providing a set of operations for determining transitive relationships between activities. The graph-based part of BPEL defines predecessor and successor relationships. The block-structure of BPEL defines relations between composite activities (e. g., *while*) and their children. Allen’s calculus is capable to capture both the graph-based and the block-structured part of BPEL. The *during* relation can for instance be used if we want to model the relation between a BPEL loop or a BPEL scope and its child activities. Using an equivalence notion of the linear time/branching time spectrum [5] is no option. It is not possible to express a *during* relationship as the notions treat state machines only. There are no nested states in state machines.

3 Choreography-based Process Consolidation

The approach of choreography-based process consolidation was introduced in [16]. Figure 2 presents an example choreography to illustrate the description using the Allen calculus. Process A sends a message to process B or process C. Process B synchronously calls a process D. A result message is sent from process B or process C to process A. The choreography has been merged into a single business process: All pairs of communication activities have been merged.

Message links in the choreography imply control flow relations between the involved processes. For instance, message link m_1 implies that activities $C1$ and $C2$ are always performed after $A1$ was executed. The consolidation approach replaces these implicit relations by an explicit control flow. Different interaction scenarios between the collaborating processes define different control flow relations between their activities. For instance, an asynchronous send/receive has different implications on the control flow relations between the activities of the involved processes than a synchronous send. Hence, different merge operations have to be applied. To goal is to merge process models into a single process model in a way that the explicit and implicit control flow relations specified by the choreography are kept. Consequently, the relations that exist between all activity pairs of the choreography have to be same in the new process model. Table 1

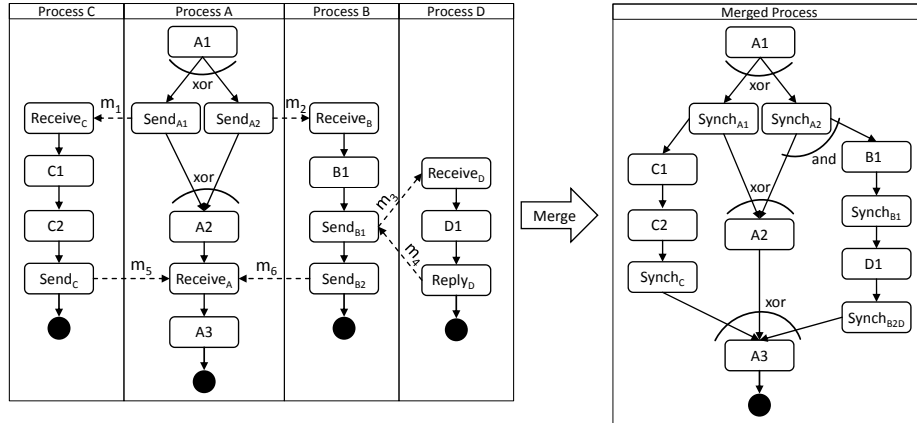


Figure 2. Example Choreography

depicts the pairwise relationships between the activities of the example choreography. The send and receive activities are omitted in the table as they are removed during the consolidation.

	A1	A2	A3	B1	C1	C2	D1
A1	\emptyset	<	<	<	<	<	<
A2	>	\emptyset	<	\mathcal{R}	\mathcal{R}	\mathcal{R}	\mathcal{R}
A3	>	>	\emptyset	>	>	>	>
B1	>	\mathcal{R}	<	\emptyset	\mathcal{R}	\mathcal{R}	<
C1	>	\mathcal{R}	<	\mathcal{R}	\emptyset	<	\mathcal{R}
C2	>	\mathcal{R}	<	\mathcal{R}	>	\emptyset	\mathcal{R}
D1	>	\mathcal{R}	<	>	\mathcal{R}	\mathcal{R}	\emptyset

Legend:

- Rows list the left part of relation relation
- Columns list the right part each relation
- <: set consisting of the single relation “before”
- >: set consisting of the single relation “after”
- \emptyset : no relation
- \mathcal{R} : all relations hold

Table 1. Relations between activities in the example choreography

The actual merge operation consists of several steps that are described informally in following. To simplify the description, we describe the merge of all participant behavior descriptions of a choreography into a single orchestration.

First, a new process model is created with a flow activity as top level element. All participant behavior descriptions of the choreography are copied into this flow activity to reflect the parallel execution of the orchestrations that exist in the choreography. Then, on each single interaction between two or more participant behavior descriptions a merge pattern is applied to replace the message links by control flow links. The type of pattern that is applied depends on the interaction type.

To validate the merge patterns, it has to be shown that the relation set R of the new orchestration models equals the relation set R_C of the original choreography: All atomic activities contained in the new orchestration *and* the original choreography have the

same relations. That means, the relations of all activities not removed or inserted remain the same.

4 Properties of Synchronous Communication

In the following section, we treat the properties a merge pattern for merging synchronous communication has to keep. In the context of BPEL, synchronous communication denotes that the activity sending the request also receives the response message. The synchronous communication pattern is implemented in BPEL4Chor by a synchronous `invoke` activity that is related to a `receive` activity via a message link m . The `receive` is directly or indirectly followed by a single logical `reply` activity that is also connected to the `invoke` via a message link m' . “Single logical `reply` activity” describes that there may be multiple `reply` activities belonging to the `receive` activity, but that there may only be one of them executed after the `invoke` has been executed. In this paper, we assume that there is exactly one `reply` activity given for a `receive` activity. Furthermore, we assume that there is exactly one `invoke` activity for the `receive` activity. BPEL4Chor allows multiple `invoke` activities for one `receive` as long as the `invoke` activities are mutually exclusive.

An example of a synchronous interaction is given in Fig. 2 where $Send_{B1}$ is connected to $Receive_D$ via m_3 and $Reply_D$ is connected to $Send_{B1}$ via m_4 . One important characteristic of synchronous communication is that the sender blocks until it receives the response. Technically spoken, this means that the `invoke` does not complete until it receives a message from the `reply`. This behavior has implications on the control flow relations between the activities that are depicted in Table 2.

For the proof, we require that there are no consecutive interactions between two partners. If there are, we regard that part of the process as a sequence of the first interaction, followed by an empty at each partner, followed by the second interaction. An empty activity does nothing. In short, this rewrite is necessary as we regard the direct predecessors of the communication activities and want to assume that they can happen in any order.

	$\bullet s$	$s \bullet$	$\bullet rc$	$rc \bullet rp$	$rc \bullet \overline{rp}$	$\bullet rp$	$rp \bullet$
$\bullet s$	\emptyset	$<$	\mathcal{R}	$<$	$<$	$<$	$<$
$s \bullet$	$>$	\emptyset	$>$	$>$	\mathcal{R}	$>$	\mathcal{R}
$\bullet rc$	\mathcal{R}	$<$	\emptyset	$<$	$<$	$<$	$<$
$rc \bullet rp$	$>$	$<$	$>$	\emptyset	\mathcal{R}	$<$	$<$
$rc \bullet \overline{rp}$	$>$	\mathcal{R}	$>$	\mathcal{R}	\emptyset	\mathcal{R}	\mathcal{R}
$\bullet rp$	$>$	$<$	$>$	$>$	\mathcal{R}	\emptyset	$<$
$rp \bullet$	$>$	\mathcal{R}	$>$	$>$	\mathcal{R}	$>$	\emptyset

Legend:

- $\bullet a$ – the set of all directly preceding activities of a
- $a \bullet$ – the set of all directly succeeding activities of a
- s – the `invoke` activity
- rc – the `receive` activity
- rp – the `reply` activity
- $rc \bullet rp$ – all direct successors of rc being on a path to rp .
- $rc \bullet \overline{rp}$ – all direct successors of rc not being on a path to rp .

Table 2. Relations in a synchronous scenario

No statement about the relations between the direct predecessor and successor activities of s and rp can be made if just the direct predecessors and successors of s , rc , or rp are considered. However, it is clear that dependencies must exist between $\bullet s$ and $\bullet rc$, because they are transitively connected via prior control links or message links. The predecessor and successor relation within one participant behavior description (e. g., $rc^{\bullet rp} \{<\} s^{\bullet}$) are trivial as they are only defined by the control links.

Concerning the relations of the successor activities of rc two kinds of successor activities have to be distinguished, namely $rc^{\bullet rp}$ and $rc^{\bullet rp}$. The successor activity $rc^{\bullet rp}$ is no direct or indirect predecessor of rp . Hence, it has no explicit relation to the successor of the send activity, i. e., $rc^{\bullet rp} \{\mathcal{R}\} s^{\bullet}$. For the successor activity $rc^{\bullet rp}$ that resides on the path to the reply activity rp exists a direct relation $rc^{\bullet rp} \{<\} s^{\bullet}$. This relation is implicitly defined by the message link. As s^{\bullet} can be only performed after s completed and s can only complete after it got a response from rp which in turn is not completed before $rc^{\bullet rp}$. For instance, in the scenario presented in Fig. 2 $D1$ has to be performed before $Send_{B2}$ and after $Send_{B1}$. This is only the case if we make the assumption that the reply activity rp completes immediately after it sent the response to the send activity s , thus $rp \{<\} s$. In an asynchronous communication, however, the relation $rc^{\bullet rp} \{\mathcal{R}\} s^{\bullet}$ would exist. This is because of the operational semantics of BPEL: s sends a message to rc and completes even if rc or $\bullet rc$ are not activated yet.

Concerning the reply activity rp , we make the assumption that it completes immediately after it sent the response to the send activity s , thus $rp \{<\} s$.

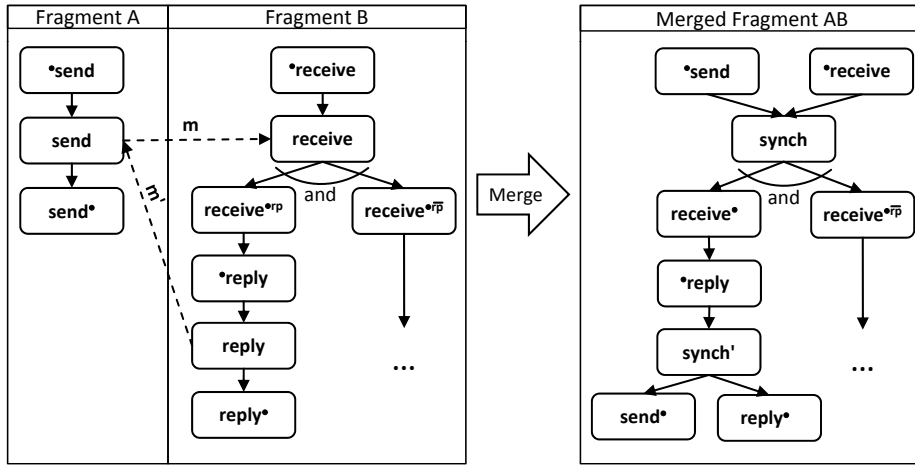


Figure 3. Merge of Synchronous Interactions

Fig. 3 depicts the merge of two process fragments that are communicating synchronously. The new orchestration at the right side of Fig. 3 keeps all control flow relations of the choreography that are sketched in Table 2. The sending activity s and the receiving activity rc are combined to a synchronization activity $synch$. This synchroniza-

tion activity is used to assign the values that were transported by the message m in the choreography. Likewise, $synch'$ is inserted to assign the values that were transported in the message m' from the reply activity rp to s .

5 Related Work

In contrast to techniques that merge processes that are semantically equivalent we aim to merge collaborating processes. An approach for process merging is the work by Mendling and Simon [11] where semantically equivalent events and functions of Event Driven Process Chains [14] are merged. An approach to merge processes that origin from the same process using change logs is described by Küster [7].

Instead of directly generating a BPEL orchestration out of a BPEL4Chor choreography, an intermediate format may be used. There is currently no approach keeping the structure of the generated orchestration close to the structure of the original choreography. For instance, Lohmann and Kleine [9] do not generate BPEL scopes out of Petri nets, even if the formal model of Lohmann [8] generates a Petri net representation of BPEL scopes.

An overview of existing BPEL formalizations and verification approaches is provided by Breugel [2]. There is no verification approach using Allen's calculus. Lohmann et al. [10] showed how BPEL4Chor can be verified using a Petri Net representation. It is not yet shown how that mapping may be used to show equivalence between a choreography and the merged orchestration. In our work, we want to keep the ordering of the internal activities, which is more than behavioral equivalence.

Weidlich et al. [17] use behavioral profiles to capture the relations between activities in process models for compliance checking. In contrast to our approach the work does not consider the relations between activities in choreographies. Moreover, only predecessor and successor relations can be captured there. Hence, it is not possible to capture the relations between parent and child activities (block-structure) which can be accomplished with the Allen calculus.

6 Conclusion and Outlook

This paper presented how relations between activities may be expressed using Allen's calculus. The derivation from choreographies and orchestration has been outlined by using an example. We used the relations to show that a merge of a choreography model into an orchestration model does not change the relations of the non-merged activities.

The capturing of interval relations has been done manually. This procedure will be kept when verifying other merge patterns. This especially includes merging BPEL's scope and loop activities. To verify such patterns, we surely will have to use the *during* relation of Allen's calculus. In our future work we will investigate if we need all relations of Allen's calculus or if the subset consisting of *before*, *after*, and *during* is sufficient.

Acknowledgments This work was partially funded by the BMWi project Migrate! (01ME11055) and the BMWi project CloudCycle (01MD11023).

References

1. Allen, J.F.: Maintaining Knowledge about Temporal Intervals. *Commun. ACM* 26(11), 832–843 (1983)
2. van Breugel, F., Koshkina, M.: Models and Verification of BPEL (2006), <http://www.cse.yorku.ca/~franck/research/drafts/tutorial.pdf>
3. Decker, G., Kopp, O., Barros, A.: An Introduction to Service Choreographies. *Information Technology* 50(2), 122–127 (Feb 2008)
4. Decker, G., Kopp, O., Leymann, F., Weske, M.: Interacting services: From specification to execution. *Data & Knowledge Engineering* 68(10), 946–972 (Apr 2009)
5. van Glabbeek, R.J.: The Linear Time-Branching Time Spectrum (Extended Abstract). In: *CONCUR*. pp. 278–297 (1990)
6. Kopp, O., Martin, D., Wutke, D., Leymann, F.: The Difference Between Graph-Based and Block-Structured Business Process Modelling Languages. *Enterprise Modelling and Information Systems* 4(1), 3–13 (June 2009)
7. Küster, J., Gerth, C., Förster, A., Engels, G.: A Tool for Process Merging in Business-Driven Development. In: *Proceedings of the Forum at the CAiSE (2008)*
8. Lohmann, N.: A Feature-Complete Petri Net Semantics for WS-BPEL 2.0. In: Dumas, M., Heckel, R. (eds.) *WS-FM'07: Web Services and Formal Methods*, 4th International Workshop. *Lecture Notes in Computer Science*, vol. 4937, pp. 77–91. Springer (2007)
9. Lohmann, N., Kleine, J.: Fully-automatic Translation of Open Workflow Net Models into Simple Abstract BPEL Processes. In: *Modellierung*. *Lecture Notes in Informatics*, vol. P-127. Gesellschaft für Informatik e. V. (2008)
10. Lohmann, N., Kopp, O., Leymann, F., Reisig, W.: Analyzing BPEL4Chor: Verification and Participant Synthesis. In: *WS-FM 2007: Forth International Workshop on Web Services and Formal Methods*. Springer-Verlag (2007)
11. Mendling, J., Simon, C.: *Business Process Design by View Integration*. In: *BPM Workshops*. Springer (2006)
12. OASIS: *Web Services Business Process Execution Language Version 2.0 – OASIS Standard (2007)*
13. Peltz, C.: Web Services Orchestration and Choreography. *IEEE Computer* 36(10), 46–52 (2003)
14. Scheer, A.W., Thomas, O., Adam, O.: *Process Aware Information Systems: Bridging People and Software Through Process Technology*, chap. *Process Modeling Using Event-Driven Process Chains*. Wiley-Interscience (2005)
15. Sun microsystems: *JSR-000914 Java™ Message Service (JMS) API (2002)*, version 1.1 April 12
16. Wagner, S., Kopp, O., Leymann, F.: Towards Choreography-based Process Distribution In The Cloud. In: *Proceedings of the 2011 IEEE International Conference on Cloud Computing and Intelligence Systems (2011)*
17. Weidlich, M., Mendling, J., Weske, M.: Efficient Consistency Measurement Based on Behavioral Profiles of Process Models. *IEEE Trans. Software Eng.* 37(3), 410–429 (2011)

Towards Automatic Generation of Process Architectures for Process Collections

Rami-Habib Eid-Sabbagh

Hasso Plattner Institute at the University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam, Germany
rami.eidsabbagh@hpi.uni-potsdam.de

Abstract. With the prevalence of business process management in the private and public sector, large process collections are created and shift into focus. To be able to harvest the underlying information, process collections need to be made easily accessible providing intuitive navigation and search.

Process collections are often structured into folders that are labeled along functional, organizational or goal-based lines. As those structures are tediously created manually, they often offer only a single view on the underlying data. However, users use such process collections with different intentions.

This paper presents a generic approach for automatically creating process architectures from unstructured process collection to offer browsing and user centered navigation structures, as well as reduce time of creation. The approach uses the characteristics of clustering algorithms to group processes and label them accordingly. Improvements for further development in the near future will be investigated and outlined as well.

1 Introduction

In recent years BPM has gained momentum in the private and public sector. As a result, process collections of different size, quality, and purpose have emerged. Especially for large collections, the need for an inherent and intuitive structure and navigation is of importance for the retrieval of process models. The knowledge stored in process collections is often not treasured to the best possible extent. The lack of consistent ordering poses strong challenges for the retrieval of process models. According to Baeza-Yates and Ribeiro-Neto [1] providing browsing capabilities on large repositories allows for efficient retrieval of large data sets, especially when users explore information collections without specific intent in mind.

Offering structured overview of business processes in a process collection is one of the aims of process architectures according to Dijkman et al. [2]. They define a process architecture as the relations between processes within a process collection, as well as the guidelines for organizing them. A process architecture shall ensure consistent and integrated process collections; hence enable navigation and easier information retrieval from the process collection.

In well-organized projects, the process elicitation process follows guidelines for structuring process models according to a process architecture. However, in many cases, process architectures have not been developed before the modeling phase. As a result, many process collections are semi- or unstructured.

(Re-) structuring already existing process collections becomes a strenuous manual task, starting with the design of a process architecture. Later, processes need to be classified manually into specific categories. For collections of hundreds or several thousand process models, e.g., SAP Reference Model, Dutch administration, or China CNR Corporation Limited [13], this is not efficient. The manual selection of categories bears possibilities of wrong subjective categorization. Defining crisp and unambiguous rules for classifying process models into categories is rather difficult.

This paper will not focus on defining process architectures in the beginning of business process modeling projects in regard to modeling responsibilities, guidelines, undocumented processes or other issues of process architecture design. It rather presents a generic system design and algorithm architecture along with a concrete example that creates a process architecture based on syntactic similarity of process names. This approach shall provide better navigation through process knowledge, as well as improve efficiency and adequacy over the manual creation of process collection structures. It may even bear the possibility to create process architectures with different focus according to the users' interest.

The paper is structured as follows, Sect. 2 will introduce current research on the structuring of process collections, process architectures and hierarchical clustering, Sect. 4 presents a conceptual system architecture, Sect. 5 sketches an algorithm for creating process architectures, followed by Sect. 6 which will elaborate on future work and improvements of the presented ideas.

2 Related Work

Different approaches to structuring process collections or creating process architectures have been developed and proposed. Most of them are based on manual classification techniques. Weske [17] presents a hierarchy consisting of strategic level, organizational level, operational level and implemented business processes in which business process can be classified according their scope. In contrast to that, Leymann and Roller [7] define a classification of business processes along the dimensions of structure and repetition.

Dijkman et al. [2] present a wide overview of different approaches of designing process architectures. They classify them into five categories; goal-based, action-based, object-based, reference-based and function-based approaches. Each concept shows a different view on a process collection focusing on different aspects of process models.

Scheer et al. [14] design a process architecture consisting of four levels; process engineering, process planning and control, workflow control, and application systems. However, this is rather a classification about the usage of process models in operational activities. Having a different focus, Fettke et al. [4] classify business process reference model approaches according to domain independent and domain dependent characteristics that are functional area and economic activity.

In Smirnov et al. [16] the need for a fast overview of a process's main characteristics is highlighted based on an empirical survey of health insurance workers and validated by BPM consultants. Process architectures have similar aims, e.g. offering information and an overview of the main characteristics of processes in a particular category. Similarly, Melcher and Seese [10] aim to provide more abstract information on process models. They visualize process metrics of process models in a heatmap based on hierarchical clustering methods and a cosine similarity function.

Coming from a different domain but facing similar problems in large multimedia collections Lew et al. [6] emphasize two main necessities, searching for a single item, and browsing and summarizing the information covered by a media collection. Summarizing process information in process architecture categories and providing navigation capabilities are aims of the approach presented in the following sections.

Qiao et al. [11] present a highly effective technique for similarity search of business processes by using clustering algorithms that use structure matching and language modeling. They point out that the clusters found, consist of similar processes as well as provide information about their common characteristics.

Jung et al. [5] propose another technique to find structurally similar process models by adapting a cosine similarity measure to match activity and transition elements of process models. They use an agglomerative clustering algorithm to find similar processes and to create new, or re-engineer business processes in an organization.

Most of the approaches use similarity measure and clustering algorithms to find similar process models or similar process elements focusing on structural [11, 5, 10], semantic aspects within a process model [15], or selected characteristics of process models [10]. However, these approaches only focus on a single process model, or result in displaying only a subset of a process collection.

The reduction of complexity and the visualization of the most important characteristics of process models is the aim of many approaches. Abstraction, as well as providing browsing and summarization of process information can be achieved by creating process architectures. According to Baeza-Yates and Ribeiro-Neto [1] providing browsing capabilities leverages information about the information collection by putting information into context with its environment. So far most approaches for creating process architectures lack automatic support which can be overcome by using hierarchical clustering algorithms for designing process architectures.

3 Hierarchical Clustering

Most of the process architecture styles presented in [2] share hierarchical characteristics to organize their processes whereas they differ according their categorization functions. Hierarchical clustering provides these functionalities and results in a hierarchy according to a selected group of features. In this regard hierarchical clustering seems promising for constructing hierarchical process architectures.

Hierarchical clustering algorithms can be distinguished into agglomerative clustering (bottom-up) and divisive clustering (top-down). Agglomerative clus-

tering algorithms are single-link, average-link, complete-link and centroid-based which differ on their similarity measures. Divisive clustering (top-down) starts with one cluster that iteratively becomes split into smaller clusters. In contrary to agglomerative clustering, divisive clustering is more complex and uses flat clustering algorithms, like k-means clustering in its subroutine [8].

In van Rijsbergen [12] three adequacy requirements for clustering are named, stable clusters for an increasing set of items, tolerability of small errors in the data set, and independence from initial ordering of the items to be clustered. According to van Rijsbergen [12] hierarchical single-link clustering satisfies those requirements. The advantages of this clustering method are efficient search strategies and fast construction of hierarchies in comparison with less efficient search strategies of non-hierarchic structures. Creating hierarchical cluster structures is of high complexity in contrast to K-means clustering and EM-clustering algorithms with low complexity.

Despite that, the use of agglomerative single link clustering (hierarchical clustering) is promising for automatically creating process architectures considering its simplicity and robustness. It fulfills the adequacy requirements making it easily applicable to heterogeneous as well as different process collections. The drawback of exponential computational complexity can be disregarded as creating process architectures is not an everyday task.

4 Conceptual Architecture

Fig. 1 depicts a conceptual system for creating process architectures. It shall provide the flexibility to create different kinds of process architectures by using different similarity measures, cluster algorithms, and different approaches for the labeling of clusters. It consists of four main components, the preprocessor, the clusterer, the label analyzer, and a GUI. The preprocessor takes the process collection as input and formats the attributes, labels, and elements of process models for clustering. The cluster module clusters the preprocessed data according

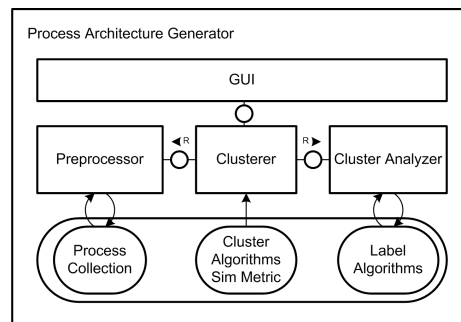


Fig. 1. Process Architecture Generator

to the selected cluster algorithm. This shall allow exploring the ability of relating clustering algorithms to particular user views. The results of the clustering process

are clusters arranged in a hierarchical tree structure. The resulting hierarchy tree diagram is depicted as dendrogram [9], see Fig. 4.

After the clustering process, the label analyzer module analyzes each member of a cluster and chooses a label for the cluster considering common characteristics of process models e.g. metadata, labels, or input and output. Algorithms for selecting adequate labels for clusters using semantic approaches from the field of natural language processing will be part of future work.

5 Sketch of Algorithm

The general approach for creating process architectures from process collections, shown in Fig. 2, consists of three steps; preprocessing, clustering, and labeling of clusters, which will be presented along with a concrete example Fig. 3. The approach takes a model collection as input and creates a process architecture with labeled clusters as output. In the preprocessing step, process models and

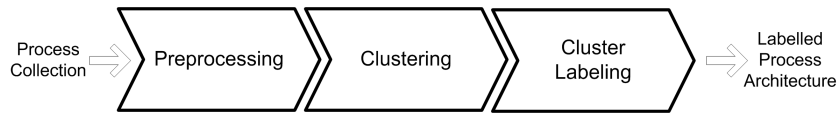


Fig. 2. General algorithm for creating process architectures from process collections

their metadata are cleansed. Metadata values are normalized and missing data is dealt with. String values are converted into numerical values. There are different approaches that can be applied to deal with missing data and string values of metadata. In the example algorithm the preprocessing step consists of extracting

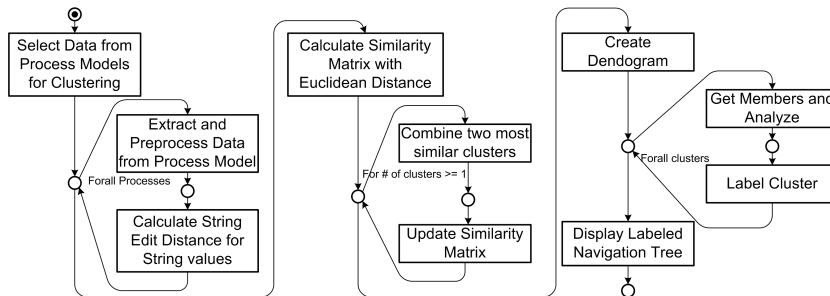


Fig. 3. Example Algorithm for creating a process architecture from a process collection

the process names from the process models and converting the strings into numerical values by calculating the string edit distance. The output of the step is multidimensional vectors representing the process models. In future research this step could be improved by using natural language processing techniques for dealing with semantics of process names, or labels of control flow elements. For example synonyms could be detected and given the same value. Analyzing the

semantic similarity of labels of control flow elements, structural, or behavioral aspects of process models bears many possibilities for exploration. Particular process model elements shall be aligned to styles of process architectures and form the input for generating those automatically; e.g. activities could be used to generate action-based process architectures. The cluster function takes the

<p>Function PreprocessProcessModels; Input <i>ProcessCollection PC, List SelectionOfMetadata;</i> Output <i>ListPreprocessedModels PM ;</i> Function Cluster; Input <i>PreprocessedModels PM, ClusterAlgorithm CA, SimFunction SF ;</i> Output <i>Dendogram D, SetClusters C;</i> Function ClusterLabeling; Input <i>Dendogram D, SetClusters C, AnalysisAlgorithm AA ;</i> Output <i>LabeledDendogram LD, SetOfLabeledClusters LC ;</i> Example: Function PreprocessProcessModels; Input <i>SAPReferenceModel, Processname;</i> Output <i>multVectors</i> Example: Function HierarchicalBottomUpCluster; Input <i>multVectors, HierarchicalSLBottomUp, EuclideanDistance ;</i> Output <i>(see Fig. 4(a)) Dendogram, Clusters</i> Example: Function ClusterLabeling; Input <i>Dendogram, Clusters, SimNamePartsAndCountEvents ;</i> Output <i>(see Fig. 4(b)) LabeledDendogram, labeledClusters</i></p>
--

Function General and Example Function Interface Descriptions

preprocessed process models, e.g., a multidimensional vector as input as well as the clustering algorithm of interest and a similarity function to calculate the similarity between the different processes in the process collection. The output is a dendogram and a set of clusters.

The input of the example algorithm is a list of multi-dimensional vectors representing the process models, the bottom-up single-link clustering algorithm and the Euclidean distance function. A hierarchical clustering algorithm is chosen due to the hierarchical nature of most process architectures and the good characteristics of hierarchical clustering algorithms. The Euclidean distance function is used to calculate the similarity matrix for the process models. In future, this step can be realized with more sophisticated similarity functions. Considering each process model as one cluster in the beginning, the hierarchical cluster algorithm will join the two clusters with the minimal distance between any two items p_i and p_j with p_i in cluster i and p_j in cluster j until only one cluster exists containing all other clusters and the inherent process models. After each iteration, the distance matrix must be updated with the similarity value of the newly created cluster. The clustering process results in a dendogram, a hierarchical structure tree that links all the clusters generated.

In the future also flat clustering algorithm can be used with presented framework. The next step defines the labeling process of the clusters. The general

algorithm takes the dendrogram, the set of clusters and an analysis algorithm as input. Here different strategies that need further elaboration can be applied. E.g. only the input and output labels of each member in the cluster could be extracted and counted.

In the example, the analysis algorithm examines the names of each member of cluster and figures out a word that describes the processes in the cluster. It also counts the number of activities, start events, and end events of each process. The label is put together from a word that is common for each process model in the cluster as well as the range of start events, end events and activities as displayed in Fig. 4(b). In this way context information on the process models in each cluster is provided while browsing. The technique described in the example is rather a simple technique for identifying labels which also leaves room for improvement in the future.

An example output of the clustering algorithm with five process models from the sap reference model collection is depicted in Fig. 4(a) and Fig. 4(b). The

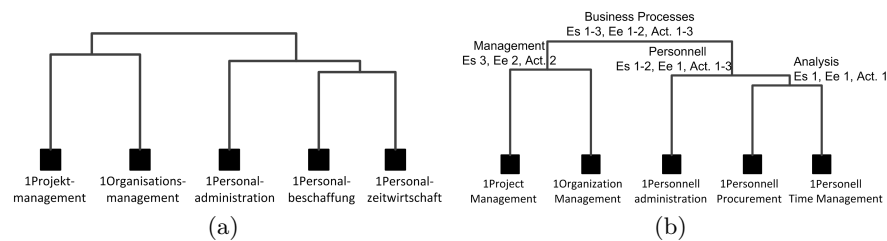


Fig. 4. Example of an unlabeled dendrogram (a) and a labeled dendrogram (b)

implementation of the algorithm has not been fully realized and first results with large process collections can only be presented in the near future.

Empirical surveys can be used to validate both the automatically created process architectures in regard to their improvement of browsing capabilities and the labels chosen for the different clusters. The architectures can be replicated in process collections and tested with users to investigate the quality of their browsing capabilities. In a similar way, the quality of labels representing the process clusters can be assessed by users in an empirical survey. A current research project, the national process library (npl), offers a suitable use case for this purpose [3]. The process architectures generated could be integrated into the npl and used for browsing. This way browsing capabilities of the process architecture could also be compared to the already existing filter mechanism and search engine in the npl.

6 Conclusion and Future Work

This paper presented a conceptual framework for automatically generating process architectures from process collections. A general and exemplary algorithm as well as the input and output of the different steps were presented to depict the process of generating process architectures for process collections. Suggestions for improvements of the quality of clustering and labeling of clusters were mentioned

as future research agenda. Using more sophisticated similarity measures for clustering as well as natural language processing techniques for analyzing semantic information from control flow labels may bear the most potential here. Also, the preprocessing step can be varied in respect to semantics which likely improves results, e.g. only nouns will be extracted. In general the presented approach is very flexible and lays the foundation for future exploration of process collections and the interdependencies of its process models.

References

1. Baeza-Yates, R., Ribeiro-Neto, B.: *Modern Information Retrieval*. Addison Wesley (1999)
2. Dijkman, R.M., Vanderfeesten, I., Reijers, H.A.: *The Road to a Business Process Architecture: An Overview of Approaches and their Use* (2011), http://cms.ieis.tue.nl/Beta/Files/WorkingPapers/wp_350.pdf
3. Eid-Sabbagh, R.H., Kunze, M., Weske, M.: *An Open Process Model Library*. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) *BPM Workshops*. LNCS, vol. 100, pp. 26–38. Springer, Berlin, Heidelberg (2012)
4. Fettke, P., Loos, P.: *Classification of reference models: a methodology and its application*. *Information Systems and e-Business Management* 1(1), 35–53 (Jan 2003)
5. Jung, J.y., Bae, J., Liu, L.: *Hierarchical Business Process Clustering*. 2008 IEEE International Conference on Services Computing 2, 613–616 (2008)
6. Lew, M.S., Sebe, N., Djeraba, C., Jain, R.: *Content-based multimedia information retrieval*. *ACM Transactions on Multimedia Computing, Communications, and Applications* 2(1), 1–19 (Feb 2006)
7. Leymann, F., Roller, D.: *Production Workflow: Concepts and Techniques*. Prentice Hall (2000)
8. Manning, C.D., Raghavan, P., Schütze, H.: *Introduction to Information Retrieval*. Cambridge University Press (2008)
9. Manning, C.D., Schuetze, H.: *Foundations of Statistical Natural Language Processing*. The MIT Press (1999)
10. Melcher, J., Seese, D.: *Visualization and Clustering of Business Process Collections Based on Process Metric Values*. In: 2008 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. pp. 572–575. IEEE (Sep 2008)
11. Qiao, M., Akkiraju, R., Rembert, A.: *Towards efficient business process clustering and retrieval: combining language modeling and structure matching*. In: *Business Process Management*. pp. 199–214. LNCS, Springer (2011)
12. van Rijsbergen, C.J.: *Information Retrieval*. Butterworth (1979)
13. Rosa, L., Arthur, H.M., Efficient, L., Jin, T., Wang, J., La, M.: *Efficient and Accurate Retrieval of Business Process Models through Indexing*. *Computers in Industry* (2010)
14. Scheer, A.W., Nüttgens, M., van der Aalst, W., Desel, J., Oberweis, A.: *BPM*, LNCS, vol. 1806. Springer, Berlin, Heidelberg (Mar 2000)
15. Smirnov, S., Reijers, H., Weske, M.: *A semantic approach for business process model abstraction*. In: *AISE*. pp. 497–511. Springer (2011)
16. Smirnov, S., Reijers, H.A., Nugteren, T., Weske, M.: *Business process model abstraction: theory and practice*. Universitätsverlag Potsdam (2010)
17. Weske, M.: *Business Process Management: Concepts, Languages, Architectures*. Springer; 1 edition, 1 edn. (2007)

Towards Process Evaluation in Non-automated Process Execution Environments[★]

Nico Herzberg, Matthias Kunze, Andreas Rogge-Solti

Hasso Plattner Institute at the University of Potsdam
Prof.-Dr.-Helmert-Strasse 2-3, 14482 Potsdam, Germany
{nico.herzberg, matthias.kunze, andreas.rogge-solti}@hpi.uni-potsdam.de

Abstract. Process models gained more and more significance to carry out an organization's operations. Besides documentation purposes, organizations strive to evaluate their executed processes in terms of performance and conformance. However, this is far from trivial: As most processes are still carried out manually, only few effects can be tracked and are typically not related to process instances. In this paper, we propose an architecture that defines event monitoring points: Elementary state transitions of a process instance that are bound to a configuration to discover events from a process agnostic technical environment. We discuss applications of this architecture, towards monitoring, performance measurement, and execution conformance.

1 Introduction

Central to managing an organization in a process-oriented fashion are process models, as they explicitly capture the operations carried out and are used, among others, for documentation, certification, and enactment. There is a large body of work that addresses automatic orchestration of business processes through process-aware information systems (PAIS), while the majority of processes are still carried out manually.

In the latter case, it is difficult to track the execution of a process for different reasons: (a) Tasks cannot be tracked, if they have no observable side effect in IT systems, e.g., examining a patient on a ward round, and thus would need to be recorded by hand, which is time consuming and prone to errors. (b) In the absence of a PAIS, there is no central system to collect information about process advancements and thus, (c) valuable information about progress is contained in various systems, such as ERP or CRM systems, but cannot easily be related to a process model. As a result, process execution information is scattered among the IT landscape and only few events of a process can be captured at all. Reconstruction of these events requires explicitly defined methods to access systems, combine relevant information, and correlate it with a process instance. Approaches towards monitoring, performance measurement, and conformance verification assume the presence of a complete event log, which is not the case according to the above discussion. Hence, they cannot be properly applied.

[★] This work is supported and funded by the German Federal Ministry of Education and Research (01IS10039B)

In an ongoing research project, PIGE¹—Process Intelligence in Health Care—we encountered above obstacles in the University Hospital of Jena, where the performance of clinical pathways, i.e., disease treatment processes, shall be measured and evaluated against key performance indicators.

In this paper, we present a basic architecture that explicitly addresses these issues, and propose a solution, where so-called event monitoring points are defined at particular points of a process. An event monitoring point is bound to a certain state transition in the process and contains information, how this event can be discovered in a heterogeneous IT landscape. We further discuss, how this can be used to monitor process instances, apply key performance indicators for performance measurement and examine running processes for their conformance to given models.

2 Architecture

As our approach towards process evaluation is tightly coupled with process models and state transitions, we first introduce these concepts and then illustrate how this manifests in a system overview.

2.1 Fundamentals

Our definition of a process model subsumes a connected graph consisting of nodes N and edges F . This covers commonly used process modeling languages, such as BPMN [8] and EPC [4].

Definition 1 (Process Model). *A process model is a tuple $P = (N, F)$, where N is the set of control flow nodes and $F \subseteq N \times N$ is the flow relation that captures ordering constraints of the process execution.*

Note, that other modeling notations, such as value chains, where N represents coarse grained units of work and F represents the execution order of these work units is also covered by this generic definition. In BPMN, N is partitioned into *activities*, *gateways*, and *events*, whereas F represents sequence flow among these nodes, for an example refer to Fig. 1. For each of these node types, we envision a state-based life cycle model, where we reserve the flexibility, to assign a unique life cycle model to any node. Life cycles of process nodes have been exhaustively discussed in literature [11,9]. Thus, we employ a generic life cycle model.

Definition 2 (Life Cycle Model). *A life cycle model $L = (\Sigma, S, T)$ consists of an event alphabet Σ , states S and state transitions T . \mathcal{L} is the universe of life cycle models.*

Let $P = (N, F)$ be a process model. There exists a function $lc : N \rightarrow \mathcal{L}$ that assigns a life cycle model to every node $n \in N$ of P .

State transitions are the most elementary facts that can be leveraged to monitor progress during process enactment. The set of all state transitions of a process model is comprised by $\bigcup_{n \in N} \{(n, t) | t \in T_{lc(n)}\}$, each of which could be potentially captured. However, as

¹ <http://pige-projekt.de>

explained in Section 1, only a subset of those can or shall actually be monitored. We refer to these state transitions of interest as event monitoring points.

Definition 3 (Event Class, Event Monitoring Point). C is a set of event classes indicating the nature of an event. Let $P = (N, F)$ be a process model. An event monitoring point is a tuple $M = (n, t, c)$, where $n \in N$ is a node, $t \in T_{l(n)}$ is a state transition within the life cycle of node n , and $c \in C$ is the event class to be monitored.

Event monitoring points of a process model are selected state transitions, for which information can be retrieved from the process environment and they can be bound to an implementation. Event classes are used to specify the measurement an event monitoring point shall provide, e.g., time, counters, or cost.

Definition 4 (Binding, Implementation). Let M be the set of defined event monitoring points of a process model P . A binding is a function $bind : M \rightarrow \mathcal{I}$ assigning an implementation to an event monitoring point, where \mathcal{I} is the universe of implementations, i.e., rules and methods to capture an event in the process execution environment.

This definition allows to implement event monitoring points in several ways, e.g., as a database query, as a service request, a calculation method, as a stream processing filter, or reading a log entry. An important aspect of the binding is correlation, i.e., identification of process instances and events that refer to this process instance. As we assume no central process orchestration control, an implementation needs to account for correlation by combining data retrieved from accessed systems.

2.2 System Overview

Our example illustrates a sample business process that describes the admission and examination of a patient, who needs a liver transplant. If the patient is eligible for transplantation, she is enlisted to a European-wide register, Eurotransplant.

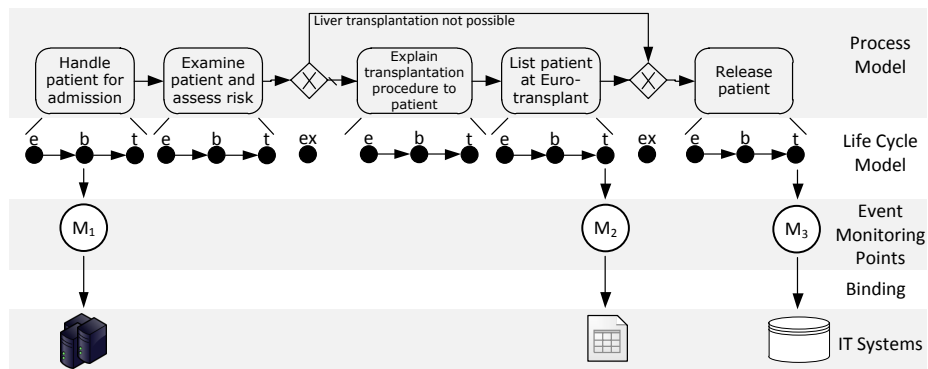


Fig. 1. Infrastructure: Example Process with life cycles (enable, begin, terminate) for activities and (execute) for gateways, selected event monitoring points, and bindings to process environment.

As explained in Section 1, we assume a process model that is the basis for process execution, which takes place in a certain environment. Part of this environment are IT systems that do not necessarily log events of process enactment, but capture the effects of activities nonetheless. These data are produced as a side effect while using IT systems to record information and are very valuable to describe the progress of a process execution. In the example, the information about the patient admission is stored in a hospital information system, the information about the confirmation of Eurotransplant listing is stored in a spread sheet file, and the patient release information is stored in a separate database.

Based on the available information, state-transitions of interest are chosen and defined as event monitoring points with the corresponding event class. In the example in Fig. 1, we keep matters simple and consider only event monitoring points of class *time*. Nevertheless, it is possible to have more than one event monitoring point defined for a certain state-transition, when it is necessary to monitor multiple event classes. In the example, we identified three event monitoring points:

- M_1 is the state-transition *begin* of activity *Handle patient for admission*
- M_2 is the state transition *terminate* of activity *List patient at Eurotransplant*
- M_3 is the state-transition *terminate* of activity *Release patient*

The other activities do not have observable side effects in IT systems, and hence cannot be tracked.

Once the event monitoring points are defined, they can be bound to an implementation. In the example shown in Fig. 1 the implementation of

- M_1 is the call of a web service provided by the hospital information system,
- M_2 is the content of a certain cell in a spread sheet file, and
- M_3 is an SQL query to retrieve data from a relational database.

In the given context, correlation of events to process instances is possible through a unique treatment case id that is stored consistently across all IT systems and can be matched to a patient.

This implementation will be used during the enactment of any instance of the process to access the IT systems and discover the occurrence of events defined in event monitoring points. Once events are extracted and correlated with process instances, they can be used to visualize the data in a monitoring user interface or to set the stage for measurements and KPIs. The causal dependencies between event monitoring points are defined by the process model and the life cycle model of its nodes. This can be leveraged for conformance checking and notification, if a deviation from the process model is detected.

During runtime, the monitoring system based on the discussed architecture is querying the process execution data, resp. event data, online from the IT systems. The method of querying the data is encapsulated in the implementations that are bound to the event monitoring points. There is no notification from the IT systems shown in Fig. 1. Thus, the architecture is following a pull approach, not a push mechanism. Pulling is necessary because in this real-world scenario the IT landscape is not enabled for pushing messages/events most times to interested listeners. In addition the running systems should not be extended by introducing a process monitoring system. Nevertheless, if the IT landscape supports pushing events to the monitoring platform, that would be the

preferable way to reduce bandwidth usage in the network. This implementation detail is not affected by the proposed architecture.

While the proposed architecture and its application are rather generic, we resorted to a simple process consisting of activities and alternative (XOR) gateways, each having brief life cycle models, in the example in Fig. 1. In practice, the architecture could be easily extended to data objects, control and data flow relations, for instance, if a transition from one activity to another would explicitly be manifested in a state change of a central data artifact. Also, every single node could have its own unique life cycle model.

2.3 Case Study

In the hospital setting that we encounter in our research project PIGE, we face the issue that the records about events during a treatment are distributed over several IT systems. Depending on the available systems, some of the steps during treatment are logged in a spread sheet file, some in a SAP Healthcare system and others in separate databases. The University Hospital of Jena elicited detailed process models for the clinical pathways of the liver transplant surgery and the colorectal carcinoma, along with milestones that resemble event monitoring points. The goal of the project is to provide process intelligence and enrich the process models with runtime information about the treatment cases. Process intelligence is enabled by answering analytical questions such as:

- How long does it take from the initial contact with the patient until evaluation for the liver transplant is finished?
- How many emergency patients were treated?
- Which treatment methods were applied?

First, we wanted to enable *monitoring* [5] of a process. One major application of monitoring is the definition of target performance values, which are *key performance indicators* (KPIs) if they are relevant for success, in the model. This allows the detection of deviations from planned time and cost limits in a process. The monitoring system can raise alerts and reminders to inform the responsible process owners and the process participants about delays or exceeding costs. In addition, there is a huge gain of *transparency*, as it becomes visible at which stage a current process instance is in a process model. Note that, while process models can be quite detailed, in the given setting only few event monitoring points can be defined. Thus, there exist unmonitored blank spots in the process model, where KPIs cannot be attached to. Second, the *prediction* of time and cost of an instance becomes much more accurate, when real-time execution information of a process is available and bound to a model. It can be used for improving efficiency by planning resources more accurately. Third, *conformance checking* helps to detect deviations, e.g., missing necessary steps, or the absence of recording them in specified IT systems. Reacting to deviations is very important, as reminders for drug administration and other treatment steps are beneficial to increase quality of care.

In the PIGE research project, we want to assess existing process evaluation methods and tailor them for this specific setting, where execution information for only few activities in the process model is available.

3 Related Work

One problem that has to be addressed when different event sources for a process exist is correlation of events to one process instance. Motahari-Nezhad et al. [7] provide algorithms to determine correlation sets on different attributes of events for distributed environments. They use methods of atomic, conjunctive, and disjunctive correlation conditions and heuristics to find correlating groups. The aspects of correlation are also relevant for this paper, while the focus is on how to map correlated information from different sources to a process model in a flexible architecture.

Process mining [10] is a discipline that can be used on top of correlated information merged in an *event log* to extract all kinds of process information, e.g., process models generated from real-life event data, execution times and conformance checks to existing models. The main difference to the architecture presented in this paper is that we utilize a top-down approach of connecting (detailed) process models to process information, while process mining is a bottom-up approach based on logs.

Closely related to process monitoring is the topic of process performance measurement, or business process intelligence, that addresses “managing process execution quality by providing several features, such as analysis, prediction, monitoring, control, and optimization” [3]. There is a considerable body of work that addresses means to capture and store process execution data and offer it for evaluation purposes [3,6,1]. Del-Rfo-Ortega et al. [2] present a comprehensive ontology to define process performance indicators that measure execution time, occurrence, and costs of processes. However, the majority of such approaches rely on a complete log, i.e., a protocol of every state transition of a process instance. In contrast, the architecture we presented lays the ground work for these approaches in the absence of a complete log.

4 Discussion and Future Work

This paper shows a general and flexible architecture to monitoring and performance evaluation for non-automated process execution environments. In practice, manually executed processes are common, because automation is not always profitable or feasible depending on the process and domain. However, some process information is often available in IT systems and can be exploited for process monitoring. With the described architecture it is possible to define event monitoring points in a process model and bind them to respective implementations. During runtime, these implementations are used to pull events from the IT systems updating the process model for monitoring purposes.

This setting raises additional questions for future work that includes dealing with observed deviations from the modeled process, e.g., missing events, duplicate events, or violation of ordering constraints imposed by the process model.

A complex model of many activities that only has few event monitoring points may not be well suited to display the state of a process instance, as it lacks information for most of the activities. A better representation would merge fragments of a process model, based on the available monitoring data. The result would be an abstraction, where a node represents a precisely measurable entity of work. On the other hand, it may be useful to abstract the process into a very coarse grained representation, e.g., to communicate it to

external stakeholders, which in turn requires aggregating and adjusting event monitoring points.

In a separate stream of work, we aim at implementation strategies for this approach. Currently, the binding of an event monitoring point is laborious work on the edge between business requirements, e.g., KPI definitions, and technical capabilities. Therefore, we envision decoupling the work of process experts and implementers by means of a service-based bindings. An additional layer decouples the IT systems from event monitoring points and can provide a flexible event distribution model, e.g. publish-subscribe, or caching. These services can be used by the by process modelers to configure event monitoring points. An adequate language for event monitoring point configuration is required to ease the currently laborious implementation process.

References

1. B. Azvine, Z. Cui, DD Nauck, and B. Majeed. Real time business intelligence for the adaptive enterprise. In *IEEE-CEC, 2006.*, pages 29–29. IEEE, 2006.
2. A. Del-Río-Ortega, M. Resinas, and A. Ruiz-Cortés. Defining process performance indicators: an ontological approach. *OTM 2010*, pages 555–572, 2010.
3. D. Grigori, F. Casati, M. Castellanos, U. Dayal, M. Sayal, and M.C. Shan. Business process intelligence. *Computers in Industry*, 53(3):321–343, April 2004.
4. G. Keller, M. Nüttgens, and A.W. Scheer. Semantische Prozessmodellierung auf der Grundlage "ereignisgesteuerter Prozessketten (epk)". *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, 89, 1992.
5. D.W. McCoy. Business activity monitoring: Calm before the storm. *Gartner Research, ID: LE-15-9727*, 2002.
6. F. Melchert, R. Winter, M. Klesse, and N.C.Jr. Romano. Aligning process automation and business intelligence to support corporate performance management. In *AMCIS'2004*, pages 4053–4063, New York, 2004. Association for Information Systems.
7. H.R. Motahari-Nezhad, R. Saint-Paul, F. Casati, and B. Benatallah. Event correlation for process discovery from web service interaction logs. *VLDB Journal*, 20(3):417–444, 2011.
8. OMG. Business Process Model and Notation (BPMN) 2.0 Specification, January 2011. <http://www.omg.org/spec/BPMN/2.0/PDF>.
9. N. Russell, W.M.P. van der Aalst, A.H.M. ter Hofstede, and D. Edmond. Workflow resource patterns: Identification, representation and tool support. In *Advanced Information Systems Engineering*, pages 216–232. Springer, 2005.
10. W.M.P. Van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer-Verlag New York Inc, 2011.
11. M. Weske. *Business process management: concepts, languages, architectures*. Springer-Verlag New York Inc, 2007.

Towards a Human Task Management Reference Model

Daniel Schulte

FernUniversität in Hagen, 58084 Hagen, Germany,
Daniel.Schulte@FernUni-Hagen.de

Abstract. Business process engines and workflow engines (but also web applications and emails) provide information about human tasks to people. Although many of these systems support some kind of human task management, no extensive analysis of involved components has been undertaken.

This paper discusses some of these systems exemplarily and defines a first human task reference model to stimulate debates on ways how to manage human tasks crossing system and organization boundaries.

1 Introduction

A workflow is the “computerised facilitation or automation of a business process” [8] and may contain automated and manual activities, also referred to as *human tasks*. As business processes are often considered as “enacted by a single organization” [19], the business process instances “can be controlled by a business process management system as a centralized software component” [19]. Interorganizational processes are realized as process choreographies where several process instances interact with each other via message exchanges.

The management of human tasks comprises, among others, the assignment of tasks to potential workers and personal task management. Some of these management facilities are integrated into Workflow Management Systems (WfMS) and Business Process Management Systems (BPMS) [18, 19].

Today, many process automations with different characteristics are offered, e. g., control flow-driven BPMSs and data-driven Scientific WfMSs, WS-* and REST oriented solutions, processes deployed locally or in the cloud, and engines supporting unstructured, knowledge-intensive business processes [15]. At the same time, people are working in parallel in different virtual, cross-company and interdepartmental teams [12]. Hence, different process automations may be used in parallel within departments and organizations and humans may be working with different process automations in different virtual teams. Hereby, they will be confronted with different human task management solutions, too.

To support humans better —esp. those who are engaged in virtual teams— a platform- and process-independent personal task management is required. This personal task management system needs to collect all tasks of a human spread over different process engines (and other “task-aware” applications) and provide task

management facilities. A step towards this vision is a human task management reference model that allows us to

- identify all affected components and their relations to each other,
- understand possible invocation patterns between the components, and to
- determine data that needs to be potentially exchanged.

These insights will support the independent development of components for human task management and provide the potential to use process engines optimized for their application scenarios in parallel. It may foster distributed orchestrations and reuse of process engines as organizational affiliations of humans do not restrict process execution. Since not only WfMS and BPMS are aware of tasks but (web) applications like Teambox [16], too, the discussion will take a broader look at human task management.

Section 2 introduces basic terms. Section 3 discusses some systems that contain and manage human tasks and shows their diversity, Sec. 4 introduces a first reference model for human task management. Section 5 discusses this reference model and Sec. 6 concludes this paper.

2 Human Tasks and their Management

Van der Aalst and van Hee define a task as “a logical unit of work” and differentiate between manual, automatic and semi-automatic tasks [1]. In the area of human task management, we look at manual tasks that are “entirely performed by one or more people, without any application” [1] and semi-automatic tasks that involve persons and applications.

The management of human tasks considers questions like:

- How can the execution of human tasks be supported?
- How can human tasks be assigned to (potential) workers?
- How can workers be informed about their tasks?
- How can workers manage their tasks, e. g., keep track of their tasks, schedule them or delegate them?

Thus, the management of human tasks comprises, among others, the assignment of tasks to potential workers, called staff resolution [18], claiming of tasks by potential workers, which may remove the item of other potential workers’ worklists [18], and also personal task management with “reminding [...] of current tasks, tracking task status, and maintaining relevant information” [21].

3 Systems touching Human Task Management

Many different systems cover (at least some aspects of) the management of human tasks. Due to lack of space we will only introduce selected systems and discuss their approach to human task management briefly.

The *workflow reference model* [8] focuses on workflows and identifies interfaces to enable interoperable workflow products. It discusses, inter alia, workitem handling that allows users to fetch and filter their workitems “irrespective of the nature of actual product implementation” [8].

The central component of the reference model in [8] (see Fig. 1) is the workflow enactment service with its workflow engines that provide the execution environment for workflow instances. This component offers interfaces

- for process definition tools to exchange process definitions that can be analyzed and modeled with external tools,
- for workflow client applications to access worklists and workitems but also to instantiate and control processes,
- for invoked applications that can be used by the workflow for automated executions of tasks,
- for other workflow enactment services to invoke activities and sub-processes or to transfer data, and
- for administration & monitoring tools to manage users and roles, among others.

The workflow enactment service and the workflow client applications provide some human task management facilities jointly: Human tasks are controlled by the workflow engines within the workflow enactment services incl. some information about them. Workflow client applications can access workitems using the ‘Interface 2’ of the workflow enactment service and can mark them as completed or change their states. They can also instantiate and control workflows (and consequently initiate human task). Hence, workflow client applications allow users to fetch and work on tasks as task workers as well as to initiate them.

The business process community has developed WS-BPEL as an executable language for business processes. For human tasks, the complementary specifications *BPEL4People* [3] and *WS-HumanTask* [2] were added.

The WS-HumanTask specification defines an XML-based description of human tasks assuring portability as the task can be deployed in different environments. A lifecycle specification for tasks and a programming interface assure interoperability. The programming interface, for example, can be used by task list clients to display information about tasks to users. Requesting applications can use a callable WSDL interface to initiate human tasks and —with deeper integration— use WS-HT protocol messages to influence the lifecycle of tasks.

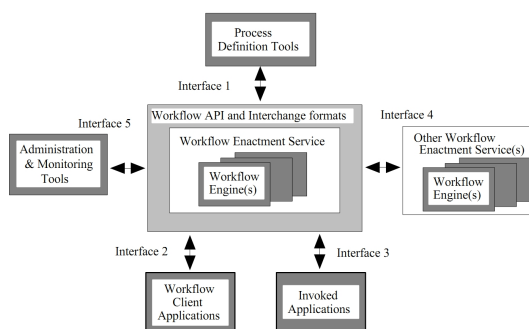


Fig. 1. Workflow Reference Model [8]

The BPEL4People specification is based on the WS-HumanTask specification and adds people activities to BPEL processes to use human tasks as activity implementations. Human tasks can be defined as part of the BPEL processes and thus executed by BPEL engines that implement BPEL4People. Alternatively, the processes can invoke human tasks from other environments using web services protocols.

The assignment of people to human tasks can be defined by logical people groups, literals or via expressions. The staff resolution is done by the task infrastructure which manages information about the tasks.

Teambox is one of many different collaboration tools provided as web applications that offers online project management facilities including task management [16]. Teambox allows the organization of tasks and artifacts in projects as well as the invitation of other users to these projects for collaboration. Tasks can be defined manually, added to projects, and provide a simple lifecycle that is also managed manually. Tasks can be assigned to people and commented, and files can be attached to tasks. All task management facilities are contained within the application but information about tasks can also be sent to users by email.

Many other web applications rely on *email* as a tool for notification about human tasks. Individuals and groups of individuals may use tools like EasyChair [6] or ConfTool [5] for conference management, Google Docs [7] for word processing and Moodle [11] in lifelong learning scenarios. Especially small teams collaborating over limited periods of time may benefit from these applications (regardless of whether provided by individual team members or by third parties).

Often, processes are firmly implemented in these web applications and contain human tasks. The humans concerned are informed by email, and email applications are regularly used for the management of these tasks [20, 21] incl. management facilities like the delegation of tasks to other humans by forwarding emails. In addition, emails allow the direct information and assignment of tasks to users.

Although these applications and systems have very different characteristics, there are some common but also individual components and facilities regarding human task management. Section 4 introduces a human task management reference model to create a shared understanding of important aspects.

4 Human Task Management Reference Model

A first version of a conceptual reference model for human task management is depicted in Fig. 2. In the following we explain the model's core elements and their relationships.

The central component of a human task management solution is a *personal task manager* that allows a human to overview all his current tasks, track their states, and maintain relevant information [21] (also called worklist or task list; it is provided by all systems analyzed in Sec. 3 except email; in the case of email, inboxes are used for it regularly [21]). Analogously, groups of people can use a *shared task manager* that

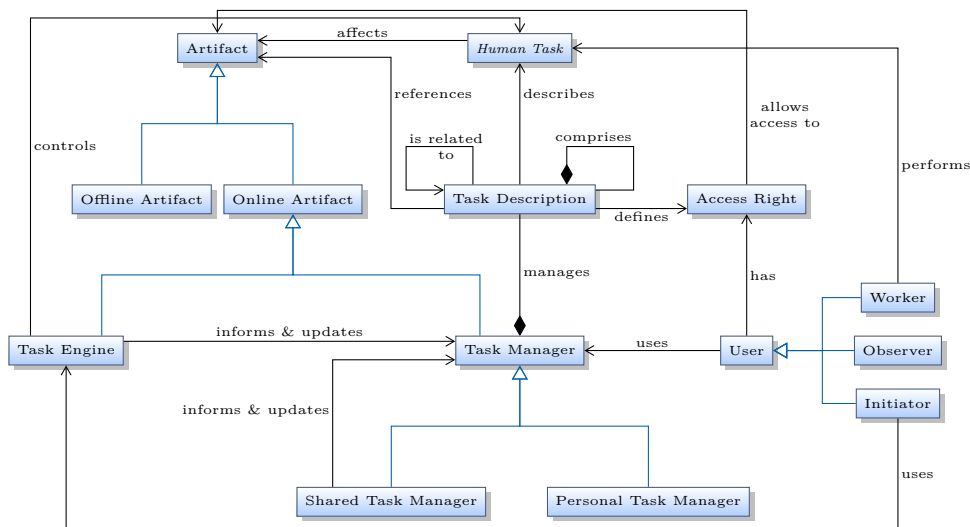


Fig. 2. Human Task Management Reference Model

- can provide direct access to personal task management facilities (the workflow reference model discusses this option briefly), but
- can also act as a simple distributor that passes on information about tasks to other *task managers*, i. e. personal or shared task manager (Teambox may be used this way if users receive their tasks by email; mailing lists may also serve this purpose), and
- can provide staff resolution facilities and passes on information about tasks to *task managers* of selected individuals (as done by the task infrastructure).

The execution of a task is controlled by a *task engine* that defines in which steps a task is executed (the workflow engine and the task infrastructure provide such functions). The task engine provides information about tasks as well as updates of this information to task managers (the workflow engine and the task infrastructure allow clients to fetch this information). The definition as well as the goal and scope of a human task are part of the task engines realization (defined by the deployed process or task, for example).

Task managers manage the information received from task engines and other task managers as *task descriptions*, which describe *human tasks* and contain information about them such as name, status, description, priority, expiration, and progress. They can comprise subtasks and formulate relations to other tasks, e. g., predecessor or successor relations. If real world artifacts —*offline artifacts* such as a certain punching machine or a certain car— or *online artifacts* —e. g., an online document or a web application— are affected by a task —e. g., needed to perform it—, they could be referenced. For the latter, hyperlinks may be appropriate.

Access Rights to artifacts are held by users, for instance, in form of a front door key for a machine hall or credentials for a web application, but may also be contained in task descriptions, e. g., as credentials, code of a combination lock, or a description where to find the keys (the described systems focus mainly on displaying tasks but the responsible clients are not described in detail).

Users — especially as (potential) *worker* or *observer* of a task— use task managers (usually personal task managers) to overview and manage their tasks. *Initiators* of tasks use task engines to initiate and, if necessary, to influence and manipulate task instances.

5 Discussion

The reference model provides terms for components and relations between them for the human task management area to allow the discussion and comparison of different solutions. The relation between task engines and task managers is, for example, implemented very differently: The workflow reference model discussed in Sec. 3 defines a *pull* model, where workflow enactment services (in the role of task engines) define an interface to retrieve and manipulate work items, whereas the email based solutions use a *push* model, where task descriptions are sent to the email applications (in the role of task managers). The *informs and updates* concept between task engines and task managers should therefore not be understood as a directed information flow but as a logical relationship. It shall promote the discussion of advantages and disadvantages of the different implementations dependent of different use cases.

Because of the different perspectives of the WfMSs and BPMSs on the one side and the human task management reference model on the other side, they have only few elements in common. Most process automation functionalities (incl. portable specifications) are subsumed in the task engine component of the human task management reference model whereas many human task management aspects are not explicitly identified in the other models and systems.

The reference model focuses on the management of human tasks. The analysis of tasks in real world processes and the design of tasks for humans are not covered. These aspects are, among other things, discussed by industrial and organizational psychology that examines the task design to improve work conditions towards health and personality-enhancing working conditions [17] and by the user interface design that uses task models to understand and develop user interfaces for interactive systems [10].

The human task reference model is inspired by the systems discussed in Sec. 3. But many other systems provide support for human task management, e. g., Outlook, Bugzilla [4] and Remember the Milk [13], which focus on specialized application areas (software development or manual managed to-do lists). Additional concepts have also been developed to improve the current state of human task management, especially for the human task management based on emails [9, 20, 21]. Therefore, further systems need to be analyzed to refine the model and get empirical evidence that the model is complete and consistent. Additionally, the

analysis of the discussed systems has to be deepened to work out their similarities and differences.

The findings of these analyses will be used to support the development of a web-scale human task management [14], which applies the insights to real world cases.

6 Conclusion

Different systems contain human task management facilities. They consist of very different components and support human task management in various ways. To develop a common understanding of human task management and stimulate debates on ways to manage human tasks crossing system and organization boundaries we introduced a first human task reference model.

Therefore, the proposed reference model pursues three targets: (1) foster the discussion of human task management, (2) provide a framework to analyze and compare existing human task management solutions and approaches, and (3) support the development of distributed and decentralized human task management solutions independent of concrete process automation systems and web applications (it shall allow humans — especially those involved in multiple projects — to overview and manage their tasks efficiently although the information about outstanding tasks may be distributed over different systems).

To improve the reference model and our understanding of human task management, additional solutions like simple to-do list tools, PIMs incl. Outlook, and CSCW workspaces will be discussed and used to refine the model in future work. In addition to components, interaction patterns as well as interfaces need to be analyzed. The usage of different process automation systems and task-aware web applications in parallel and the choice of humans to work on human tasks beyond company boundaries may be a long-term goal.

References

1. van der Aalst, W.M.P., van Hee, K.M.: *Workflow Management: Models, Methods, and Systems*. MIT Press, Cambridge, MA (2002)
2. Agrawal, A. et al: *Web Services Human Task (WS-HumanTask)*, Version 1.0. Technical Report, Active Endpoints Inc. et al (2007)
3. Agrawal, A. et al: *WS-BPEL Extension for People (BPEL4People)*, Version 1.0. Technical Report, Active Endpoints Inc. et al (2007)
4. Bugzilla, <http://www.bugzilla.org/>
5. ConfTool, <http://www.conftool.net>
6. EasyChair, <http://www.easychair.org>
7. Google Docs, <https://docs.google.com/>
8. Hollingsworth, D.: *The Workflow Reference Model*. Technical report, Workflow Management Coalition (1995)
9. Li, W., Zhong, N., Yao, Y., Liu, J.: *An Operable Email Based Intelligent Personal Assistant*. *World Wide Web* 12(2), 125–147 (2009)

10. Limbourg, Q., Pribeanu, C., Vanderdonckt, J.: Towards Uniformed Task Models in a Model-Based Approach. In: Johnson, C. (eds.) DSV-IS 2001. LNCS, vol. 2220, pp. 164–182. Springer, Berlin (2001)
11. Moodle, <http://moodle.org>
12. Powell, A., Piccoli, G., Ives, B.: Virtual teams: a review of current literature and directions for future research. *ACM SIGMIS Database*, 35, 6–36 (2004)
13. Remember the milk, <http://www.rememberthemilk.com/>
14. Schulte, D.: Web-scale Human Task Management. ECSCA 2011. LNCS, vol. 6903, pp. 190–193. Springer, Berlin (2011)
15. Stoitsev, T., Scheidl, S., Spahn, M.: A Framework for Light-Weight Composition and Management of Ad-Hoc Business Processes. In: Winckler, M., Johnson, H., Palanque, P.A. (eds.) TAMODIA 2007. LNCS, vol. 4849, pp. 213–226. Springer, Berlin (2007)
16. Teambox : Collaboration Software - Online project management tool for teams, <http://teambox.com/>
17. Ulich, Eberhard: *Arbeitspsychologie* (German). 7th ed. Schäffer-Poeschel, Stuttgart (2011)
18. Unger, T., Roller, D.: Applying Processes for User-Driven Refinement of People Activities. 14th IEEE International Enterprise Distributed Object Computing Conference Workshops, pp. 9–14. IEEE Computer Society, Los Alamitos (2010)
19. Weske, M.: *Business Process Management: Concepts, Languages, Architectures*. Springer, Berlin (2007)
20. Whittaker, S., Sidner, C.: Email Overload: Exploring Personal Information Management of Email. Tauber, M.J. (eds.): SIGCHI Conference on Human Factors in Computing Systems (CHI 1996), pp. 276–283. ACM Press, New York (1996)
21. Whittaker, S., Bellotti, V., Gwizdka, J.: Email in personal information management. *Communications of the ACM* 49, 68–73 (2006)

Contextsensitive Online Adaption of Workflows

Johannes Kretzschmar, Clemens Beckstein

`johannes.kretzschmar@uni-jena.de`

`clemens.beckstein@uni-jena.de`

Friedrich Schiller University, Jena, Germany

Abstract. A dynamic and complex process environment forces the automated execution and monitoring of processes to face a lot of hard problems. This paper shows how the execution of agile workflows can be assisted by AI planning techniques. In contrast to previous approaches, this method allows for an online adaption of simple process models to changes in the process environment. It can handle a manifold of unexpected events and guarantees the soundness and completeness of the adapted workflow.

1 Introduction

Business processes today are commonly automated by formalizing and instantiating them to workflows. Established comprehensive industry standards support modeling, processing and monitoring of these workflow instances by a workflow management system [8]. The inclusion of more dynamic and complex processes and process environments, demands an extended support for agile workflows [17]. Agility allows the adaption of workflows to face a dynamic environment. The classical distinction between modeling and execution of workflows is progressively given up in order to adjust to changes and new demands at runtime. The adjusting procedure should be fully automated, efficient, correct and able to process expected as well as unexpected events. Workflow elements that are not affected by events should be executable during the adaption and not prolong the execution time of the underlying business process. In order to guarantee the correctness of the adapted workflows a comprehensive formal semantic annotation of the process and its environment is needed. A more expressive model of business processes on the one hand and a formal process semantic on the other one suggest to combine AI planning techniques and workflow modeling for a solution of the adaption problem.

Here, we will present and discuss an approach to workflow adaption via plan repair. First, we will formally define how processes based on a simple plan model can fail, i.e. the corresponding failure model. Next, we will give a sketch and an example for adapting a plan by fragmentation and partial plan repair. Having related our work to previous approaches we will then conclude with a short summary and an outlook on future work.

2 Planning and Failing

For our approach, we assume a very basic workflow model. It consists of semantically annotated real world operations and allows parallel and sequential processing of (sub)workflows. This simple model can easily be transformed to a partial order plan and vice versa.

A partial order plan in set theoretically representation is a tuple $\pi = (A_\pi, \prec)$ consisting of a set A_π of partially ordered actions [14]. The corresponding planning domain $\Sigma = (S, A, \gamma)$ is defined by a set of states $S \subseteq 2^L$ over a finite set L of propositional symbols (so called fluents), a set of actions $A \supseteq A_\pi$ and a state-transition function γ , which defines the effect of an applied action. An action $a = (\text{precond}(a), \text{effect}^+(a), \text{effect}^-(a)) \in A$ is applicable in a state $s \in S$, if $\text{precond}(a) \subseteq s$. The application of a in s results in a state $s' = \gamma(a, s) = (s \cup \text{effect}^+(a)) \setminus \text{effect}^-(a)$. In the following $\Gamma^*((A, \prec), s_0)$ denotes the set of states, which result from processing all linearizations of a plan (A, \prec) in s_0 . A linearization of (A, \prec) is a sequential order of all actions $a \in A$ that is compatible with \prec . A partial order plan π is a solution for a planning problem $\mathcal{P} = (\Sigma, s_0, g)$ with a set of goal fluents g , if for all $s_{\text{end}} \in \Gamma^*((A, \prec), s_0) : g \subseteq s_{\text{end}}$.

Although there is the assumption of a static world in classical planning, we have to handle the dynamics of real life scenarios as they are typical for workflow applications. Plan execution therefore has to deal with a wide range of events, which were not considered at the time of planning. These can affect all aspects of a plan and the planning domain: states, sets of actions and goals. In the following we assume that the actions of a partial order plan π are processed one after the other. As a consequence, the execution state $s_{\pi'}$ that results from the linear plan fragment $\pi' \subseteq \pi$ of already executed actions is uniquely defined. An unexpected event can influence this state by implicitly adding or deleting fluents. Therefore we can define an (unexpected) external event as a pseudo action $\epsilon = (\emptyset, \text{effect}^+(\epsilon), \text{effect}^-(\epsilon))$ which transforms $s_{\pi'}$ to the new state $s' = \gamma(\epsilon, s_{\pi'})$. Internal unexpected events (planned actions that failed) can be formally treated in the same way. Process dynamics can also unexpectedly influence the set of actions available for planning thus changing Σ to the new domain $\Sigma' = (S, A', \gamma')$. Our approach also allows for (unexpectedly) changing the goals from g to g' during plan execution. We assume, that an event or goal change always occur in between the processing of two actions and that the plan domain is fully observable. In real world scenarios unexpected events of the mentioned types can happen in combination.

Once an unexpected event e_u is recognized in state s' after execution of the plan fragment π' its impact on the overall plan π has to be assessed. If the remaining plan $\pi_R = \pi \setminus \pi'$ is not a solution for the planning problem $\mathcal{P}' = (\Sigma', s', g')$ the plan π *failed*. One way to fix the plan would be to find a solution for \mathcal{P}' that is to fix π by re-planning. The fix we propose is to *repair* π by isolating and partially replanning only that part of π_R which is affected by e_u . A plan fragment is called *affected* by e_u , if it contains failed actions. An action $a \in \pi_R$ is failed, if $a \notin A'$ or if there is a linearization of π_R , where a is not applicable.

3 Fragmentation and Replanning

In order to affect the plan or workflow execution as little as possible we identify minimal parts of π_R that fail and try to find alternatives for them.

For this purpose we first choose an *inner fragment* $\pi_E = (A_E, <)$ of π_r by selecting a connected plan fragment, which contains all failed actions $A_{\text{fail}} \subseteq A_E$. A plan fragment π_E is called *connected*, if for all $a, b \in A_E$ and $c \in A_\pi$, we have that $c \in A_E$ if $a < c < b$. A_E may also be empty if π_R only fails due to an unexpected goal change. The *outer fragment* π_F is the plan containing all actions, which are unordered to every action in π_E and not contained in π_E . So π_F is connected as well. Finally we define the *remaining fragment* π_P as the plan fragment containing all actions $a \in A_R$ which are neither in the inner nor the outer fragment. By definition A_E, A_F and A_P are disjoint, but contain all actions of π_R in union. Therefore we call (π_E, π_F, π_P) a *fragmentation* of π_R (see fig. 1).

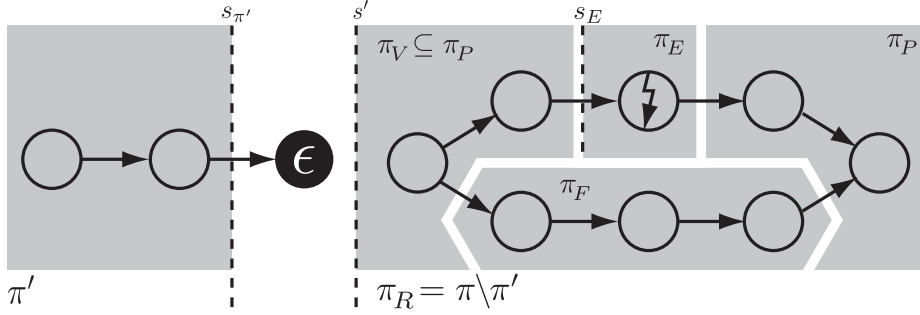


Fig. 1. a valid fragmentation of the remaining plan π_R

The plans π_E and π_F in a fragmentation (π_E, π_F, π_P) are independent and π_E contains all failed actions of π . The plan repair mechanism we propose searches for an alternative π'_E for π_E which is similar to the original, can be executed independently of π_F and together with π_F and π_P is a solution of \mathcal{P}' . To ensure the independent (unordered) execution of π_F in relation to π'_E , we introduce the so called context of π_F . The *context* \mathcal{C}_F of π_F is a pair $(\mathcal{H}_F, \mathcal{S}_F)$ of two fluent sets: the *soft criteria* \mathcal{S}_F and the *hard criteria* \mathcal{H}_F . \mathcal{S}_F contains all fluents which are used in preconditions or added, but not deleted by actions in π_F .

$$\mathcal{S}_F := \left\{ m : \bigvee_{a \in A_F} \bigwedge_{b \in A_F} (m \in \text{effect}^+(a) \cup \text{precond}(a)) \wedge \neg (m \in \text{effect}^-(b)) \right\}$$

The fluents in \mathcal{S}_F can also be used, added but not deleted in π'_E without resulting in plan flaws. On the other hand, fluents are not allowed to be added or used if they are deleted in π_F . These fluents make up $\mathcal{H}_F := \{m : \bigvee_{a \in A_F} m \in \text{effects}^-(a)\}$.

With the help of the context, it is possible to formulate an extended planning problem $\mathcal{P}_E = (\Sigma', s_E, g_E, \mathcal{C}_F)$. Compared to a classical planning problem like \mathcal{P} an extended planning problem also takes into account the dynamic environment described by the context \mathcal{C}_F . As described in section 2, a failure may occur after the state where the unexpected event took place that triggered the plan repair. This happens for example, if a state change event affects the applicability of prospective actions in the plan. We therefore identify a fragment π_V of π_P that contains all actions $a \in A_P$ with $a \prec a_E$ for all $a_E \in A_E$. The plan fragment π_V is used to produce the starting state s_E for the extended planning problem \mathcal{P}_E from the state s' which contains those fluents that for sure hold once π_V is completely executed: $s_E := \bigcap \{s_i : s_i \in I^*(\pi_V, s')\}$. Thus our method is able to repair parts of the plan which are far ahead of the current execution state without expanding the inner fragment. Finally, the goal g_E is constructed from open preconditions in π_P and the goal set g' . To simplify this procedure, the goal is represented by an implicit goal action $a_{goal} = (g', \emptyset, \emptyset)$ which is inserted into π_P after all other actions $a \in A_P$. The goal g_E for \mathcal{P}_E is then chosen to:

$$\left\{ m \in L : \bigvee_{a \in A_P} m \in \text{precond}(a) \text{ and } \bigwedge_{b \in A_P \cup A_F} (b \prec_{\pi} a \rightarrow m \notin \text{effects}^+(b)) \right\}.$$

In order to handle extended plan problems the planning procedure uses a generalized definition of applicability which uses the context $\mathcal{C}_F = (\mathcal{H}_F, \mathcal{S}_F)$ to avoid flaws between the computed plan solution and π_F . An action a is applicable in a state s wrt. a context \mathcal{C}_F , if

$$\text{precond}(a) \subseteq s \text{ and } \bigwedge_{m \in \text{precond}(a)} m \notin \mathcal{H}_F \text{ and } \bigwedge_{n \in \text{effects}^-(a)} n \notin \mathcal{S}_F.$$

The parameters for the extended planning problem and the resulting applicability of actions are all determined by the selection of the inner plan fragment of π_E : the smaller π_E , the bigger \mathcal{C}_F and the more constrained is the planning process. For a good choice of π_E we propose to first attempt the repair process with the smallest π_E possible followed by iterative attempts with bigger fragments until $\pi_E = \pi_R$ which amount to classical replanning. It can be shown, that the solution π'_E of \mathcal{P}_E can replace the defective π_E without producing flaws wrt. π_F . By processing π'_E all open preconditions of π_P and all (maybe changed) goal fluents will be satisfied: the repaired plan π'_R is a sound and correct solution for the plan problem \mathcal{P}' that summarizes the failure of the original plan.

4 A Real World Usecase

The generic repair method, introduced in this paper, was developed in the context of the MOPS project¹. The target of this joint research project is adaptive

¹ MOPS is funded by the European Union (European Regional Development Fund) and the Federal State of Thuringia of Germany.

planning and secure execution of mobile processes for human agents in dynamic scenarios [1]. Because the processes in those scenarios typically are long living, a comprehensive event and failure model as well as support for process adaption during runtime is needed. The workflow technology of MOPS is based on BPEL descriptions. In order to bridge the expressive gap between the BPEL model and the plan model, the plan activities are encapsulated as semantically annotated services. As a consequence, workflows can contain complex control flow structures and detailed local event handling mechanisms, and still can be planned and manipulated automatically on an abstract level.

The application of MOPS is based on a generic scenario of service staff completing missions (on site repair jobs, delivery tasks, facility management). In order to exemplify our repair method, let us assume an extremely reduced scenario with 3 service technicians and 3 missions that can be represented by the following simple set-theoretic planning domain with the fluent set $L = \{a1, \dots, a3, done1, \dots, done3, 1doing1, \dots, 3doing3\}$. For every technician, there is a fluent aX , which means “technician X is available”. Likewise there is a fluent $doneX$ for every job, which means “job X is successfully accomplished.” For every job and every technician there is a fluent $XdoingY$, which means “technician X is assigned to job Y ”. The planning domain further contains two types of actions. The first one actually assigns a job to a technician, if the technician is available. $assign1to2 = (\{a1\}, \{1doing2\}, \{a1\})$, e.g., allocates job2 to technician1. The second type of action initializes the execution of a job by its assigned technician. $1do2 = (\{1doing2\}, \{done2, a1\}, \{1doing2\})$, e.g., means “technician 1 is doing job 2”. The solution of the planning problem $\mathcal{P} = (\Sigma, s_0, \{done1, done2, done3\})$ with $s_0 = \{a1, a2, a3\}$ then results in a plan like π as shown in fig. 2.

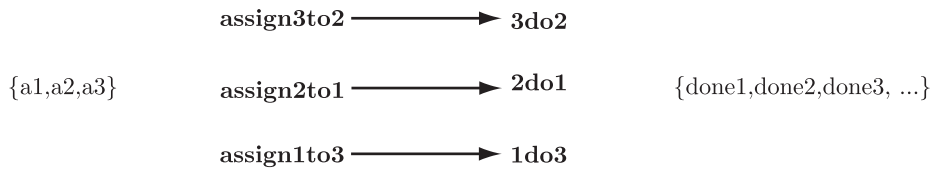


Fig. 2. a sound and correct solution π for \mathcal{P}

Let us now assume that the current execution state $s_\pi = s_0$ and that in this state a technician reports to be sick resulting in the new actual state $s' = \gamma((\emptyset, \emptyset, \{a3\}), s_\pi) = \{a1, a2\}$. This event leads to a failure of the remaining plan π' , because action $assign3to2$ is no longer applicable. Because this action is the only action affected by the failure we can reasonably set $A_E = \{assign3to2\}$, which implies $A_F = \{assign1to3, 1do3, assign2to1, 2do1\}$ and $A_P = \{3do2\}$. This fragmentation leads to a context \mathcal{C}_F with $\mathcal{H}_F = \{a1, a2, 1doing3, 2doing1\}$ and $\mathcal{S}_F = \{done3, done1\}$ and the extended planning problem $\mathcal{P}_E = (\Sigma, \emptyset, g_E, \mathcal{C}_F)$ with the open precondition of $3do2$ as goal $g_E = \{3doing2\}$ It is impossible to find a solution for this problem because job

2 can not be assigned to any other technician due to $\{a1, a2\} \subset \mathcal{H}_F$ (an assignment action deletes the availability fluent for its agent and therefore is not applicable as long as the fluent is contained in the context). If we now widen the inner fragment to $A_E = \{assign3to2, 3do2\}$ (hence $A_P = \emptyset$) then the corresponding new extended planning problem $\mathcal{P}_E = (\Sigma, \emptyset, \{done2\}, \mathcal{C}_F)$ still has no solution because A_F and \mathcal{C}_F remained the same. A further widening of the inner fragment to $A_E = \{assign3to2, 3do2, 2do1\}$ ($A_P = A_V = \{assign2to1\}$) diminishes the outer fragment to $A_F = \{assign1to3, 1do3\}$ and the context to $\mathcal{S}_F = \{done3\}$ and $\mathcal{H}_F = \{a1, 1doing3\}$. The resulting extended planning problem $\mathcal{P}_E = (\Sigma, \{2doing1\}, \{done1, done2\}, \mathcal{C}_F)$ is now solvable with the plan alternative π'_R as shown in fig. 3



Fig. 3. the adapted plan π'_R with alternative π'_E and corresponding fragmentation

This little example already shows that the choice of the inner fragment A_E significantly influences the context and therefore the degrees of freedom for the replanning problem. For this choice we propose to use the control-flow structure, respectively the partial plan order: in order to maximize the part of the workflow that can be executed concurrently to the adaption process for the failed part, the outer fragment A_F and A_V of the remaining fragment should stay as big as possible.

5 Related Work

A lot of workflow related approaches for adaption while runtime are based on the reusability of pre-modeled workflow fragments. AdaptFlow [6], which is based on ADEPT_{flex} [16], uses explicit event-condition-action (ECA) rules to change a workflow. A similar, and widely-established approach, based on case-based reasoning (CBR) [12], is implemented in CBRflow [18], Phala [10] and CAKE II [13]. In case of a change request, a pre-modeled workflow fragment is chosen from a case repository and integrated into the workflow. The retrieval is based on a similarity measure of the structural, procedural and declarative knowledge. The CBR related methods differ primarily in the type and complexity of the underlying knowledge model. There are also approaches using AI planning techniques like the traditional case-based planner CHEF [7], which can be applied to simple workflow models. CODAW [11], e.g., is using a case repository implemented as a hierarchical task network. Other workflow specific approaches like

BPEL'n'Aspects [9] or StPowla [5] use policy descriptions for implementing adaptive process logic. These descriptions are similar to the pre-modeled knowledge of CBR cases and likewise triggered by ECA rules. In contrast to our approach, all these approaches lack of a specified failure model and a goal oriented adaption. The identification of events and the adaption is restricted to pre-modeled use cases which do not cover unexpected events. Further, it is impossible to recognize and handle any flaws which may occur by the adaption.

Besides case based approaches there are two strategies in AI planning for performing a plan adaption: re-planning from scratch or plan repair. Nebel and Koehler [15] show, that the local adaption and reuse of plans suffers from the inherent structural worst case complexity of planning: failed plans should be re-planned from scratch because in the worst case, the whole plan has to be re-structured anyway. First, this consideration ignores the necessity of concurrent plan execution and plan adaption as it is typical for, e.g., business processes. Further, one of the few general domain independent approaches for plan repair, as discussed by Arangu et. al. in [2] and used in the planner MIPS-XXL [3], performs great using local adaption techniques: an iterative two-step procedure identifies the defective part of the plan and tries to find an alternative by gradually expanding the planning problem, especially the set of planning operators. But in contrast to our approach the scope of plan repair is fixed.

Usually there is no unique solution for a planning problem, that's why alternatives have to be considered. Fox et. al, e.g., discuss in [4] a measure for plan-similarity like those used in CBR: a heuristic assures that the alternative plan is as close as possible to the original failed plan. This is justified by the assumption that the original plan was modeled by or with the help of domain experts. Especially when using plans as workflow models, this measure also permits maintaining commitments between process partners. Our context description of process (fragments) could supply useful informations for a heuristic or similarity measure in first principle planners as well as CBR or CBP approaches.

6 Conclusion and Future Work

We have introduced a domain independent method for adapting partial order plans. In contrast to present approaches, this method follows a formally defined complex failure model that captures the relevant characteristics of the underlying process model. The approach is able to adapt processes at runtime, i.e. concurrently to process execution, and is guaranteed to produce sound and correct solutions.

At the moment there still is a big gap between the expressive power of the plan representation we used and that of state of the art workflow models but we are working hard to generalize our method to more complex representations, which are able to represent additional aspects of today's workflow languages — among them complex control structures as well as explicitly coded data flow, organizational and security aspects.

References

1. MOPS — project description (2011), <http://mops.uni-jena.de/us/Homepage-page-.html>, [Online; accessed 08-February-2012]
2. Arangu, M., Garrido, A., Onaindia, E.: A general technique for plan repair. In: Tools with Artificial Intelligence, 2008. ICTAI '08. 20th IEEE International Conference on. vol. 1, pp. 515–518 (nov 2008)
3. Edelkamp, S., Jabbar, S., Nazih, M.: Large-scale optimal pddl3 planning with mips-xxl. In: 5th International Planning Competition Booklet (IPC-2006) (2006)
4. Fox, M., Gerevini, A., Long, D., Serina, I.: Plan stability: Replanning versus plan repair. In: In Proc. ICAPS. pp. 212–221. AAAI Press (2006)
5. Gorton, S., Montangero, C., Reiff-Marganiec, S., Semini, L.: Service-oriented computing - icsoc 2007 workshops. chap. StPowla: SOA, Policies and Workflows, pp. 351–362. Springer-Verlag, Berlin, Heidelberg (2009)
6. Greiner, U., Ramsch, J., Heller, B., Löffler, M., Müller, R., Rahm, E.: Adaptive guideline-based treatment workflows with adaptflow. In: Proceedings of the Symposium on Computerized Guidelines and Protocols (CGP 2004), Computer-based Support for Clinical Guidelines and Protocols. pp. 113–117. IOS Press (2004)
7. Hammond, K.: Case-based planning: A framework for planning from experience. *Cognitive Science* 14, 385–443 (1990)
8. Jablonski, S. (ed.): Workflow-Management. dpunkt-Verl., Heidelberg (1997)
9. Karastoyanova, D., Leymann, F.: BPEL'n'Aspects: Adapting Service Orchestration Logic. In: Proceedings of 7th International Conference on Web Services ICWS 2009. pp. 222–229. IEEE Computer Society (2009)
10. Leake, D., Kendall-Morwick, J.: Towards case-based support for e-science workflow generation by mining provenance. In: Althoff, K.D., Bergmann, R., Minor, M., Hanft, A. (eds.) *Advances in Case-Based Reasoning, Lecture Notes in Computer Science*, vol. 5239, pp. 269–283. Springer Berlin / Heidelberg (2008)
11. Madhusudan, T., Zhao, J.L., Marshall, B.: A case-based reasoning framework for workflow model management. *Data Knowl. Eng.* 50, 87–115 (July 2004)
12. Minor, M., Bergmann, R., Görg, S., Walter, K.: Towards case-based adaptation of workflows. In: Montani, S., Bichindaritz, I. (eds.) *Case-Based Reasoning. Research and Development, 18th International Conference on Case-Based Reasoning, ICCBR 2010, Alessandria, Italy, July 19-22, 2010. Proceedings*. pp. 421–435. LNAI 6176, Springer (2010)
13. Minor, M., Schmalen, D., Kempin, S.: Demonstration of the agile workflow management system cake ii based on long-term office workflows. In: *BPM (Demos)'09* (2009)
14. Nau, D., Ghallab, M., Traverso, P.: *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2004)
15. Nebel, B., Koehler, J.: Plan reuse versus plan generation: A theoretical and empirical analysis. *Artificial Intelligence* 76, 427–454 (1995)
16. Reichert, M., Dadam, P.: Adept flex - supporting dynamic changes of workflows without loosing control. *Journal of Intelligent Information Systems* 10, 93–129 (1998)
17. Weber, B., Wild, W.: Towards the agile management of business processes. In: Althoff, K.D., Dengel, A., Bergmann, R., Nick, M., Roth-Berghofer, T. (eds.) *Professional Knowledge Management, Lecture Notes in Computer Science*, vol. 3782, pp. 409–419. Springer Berlin / Heidelberg (2005)
18. Weber, B., Wild, W., Breu, R.: Cbrflow: Enabling adaptive workflow management through conversational case-based reasoning. In: *ECCBR*. pp. 434–448 (2004)

Six Strategies for Building High Performance SOA Applications

Uwe Breitenbücher, Oliver Kopp, Frank Leymann,
Michael Reiter, Dieter Roller, and Tobias Unger

University of Stuttgart, Institute of Architecture of Application Systems (IAAS)
{uwe.breitenbuecher, firstname.lastname}@iaas.uni-stuttgart.de

Abstract. The service-oriented architecture (SOA) concepts such as loose coupling may have negative impact on the overall execution performance of a single request. There are ways to facilitate high performance applications which benefit from this kind of architectural style compensating the caused overhead significantly. This paper gives an overview on six high level strategies to improve the performance of SOAs with a central service bus and presents how to apply them to build high performance service-oriented applications without corrupting the SOA paradigm and concepts on the technical level.

Keywords: Service-oriented architecture, High Performance, Strategies

1 Introduction

The key concepts of service-oriented architectures (SOAs) such as loose coupling, interoperability, or abstraction may have negative impact on the overall performance of applications. The reasons are additional costs for time-consuming operations such as message format transformations, dynamic service discovery, etc. [10]. In this paper we present six different improvement strategies which may increase the performance and show how to apply them to build high performance SOA applications. As the presented strategies are applied on a higher level than the operations causing the overhead, the strategies compensate these time-consumptions and additionally increase the overall performance.

In this paper we use two metrics for assessing the performance: Throughput and response time. Throughput denotes the maximum number of requests a SOA application can process in a certain period. Response time measures the time an application needs to respond to a request [14].

The remainder of this paper is structured as follows: Section 2 discusses related work. Section 3 presents six strategies to improve the performance and how to apply them in an abstract service-oriented architecture with a central service bus. Finally, Section 4 concludes and provides an outlook on future work.

2 Related Work

This paper is a first attempt to show how a set of high level strategies can be applied to improve the performance of a SOA application without corrupting the underlying SOA concepts. Other work in the area of SOA performance improvements are focusing on the technical level. One example for technical improvements are performance best practices considering optimization strategies focusing on message processing, message structure, and message design of XML based protocols [11]. These optimizations are different from the presented strategies in this paper in the level of abstraction: The six presented strategies in this paper are applied on a high abstract level while the best practices propose optimizations for concrete technologies. FastSOA [12] is an architecture and software coding practice which considers optimization through native XML environments, a mid-tier service cache, and the use of native XML persistence. It combines the cache strategy presented in this paper with best practices by Endrei et al. [11], but lacks applying the other high level strategies in order to gain a higher overall performance.

3 High Performance Strategies

In this section we present six strategies which enable high performance service-oriented applications without corrupting the underlying SOA concepts [4]. We do not claim that these strategies are complete: They are inspired by the experiences in our research projects – mainly SimTech¹ – and have to be seen as a first attempt to design high performance SOA applications which has to be extended.

We present Parallel Processing, Caching, Dynamic Service Discovery, Dynamic Service Migration, Multiple Service Instantiation, and Multiple Service Invocation. Each strategy targets performance issues on an architectural level.

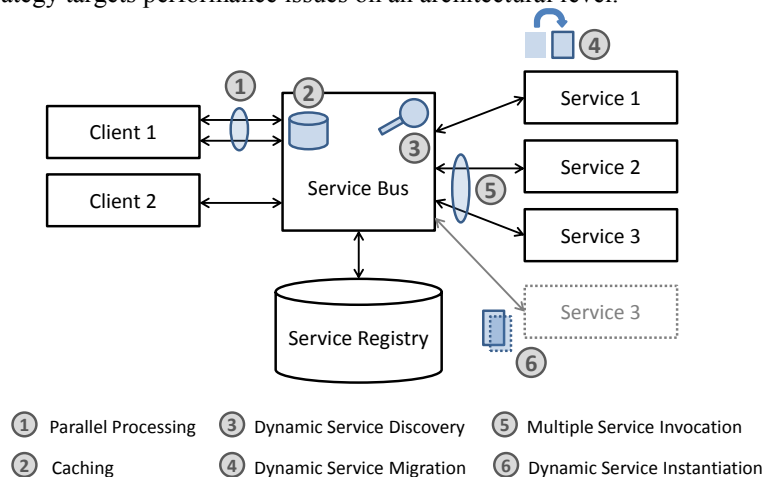


Fig. 1. Six high performance strategies applied to a SOA with central service bus (based on [3])

¹ <http://simtech.uni-stuttgart.de>

The strategies focus on SOAs having a service bus as central component (see Fig. 1): A *service* is an application processing request messages and may returning response messages. A *client* is any application that sends request messages which have to be processed by services to a central component called service bus (a client can be a service, too). The *service bus* is a middleware component providing an integration platform to connect clients with services [6], [7]. It uses a *service registry* which stores all available services combined with a description of their functionality to look up appropriate services [7]. All messages sent by a client are routed through the service bus, which looks up an appropriate endpoint and sends the message to the selected service. After the service finishes the processing, response messages may be routed back to the requesting client.

The following subsections describe the six strategies. Each strategy has a *goal* describing the strategy's impact on the performance in one sentence. The *description* explains in more detail how to apply the strategy and why the performance is improved. The *assumptions* paragraph describes preconditions which have to be met to apply the strategy successfully. *Benefits* of applying the strategies are summarized as well as *downsides and problems* in a separate paragraph.

3.1 Parallel Processing

Goal. The goal of this strategy is increasing the internal throughput of requests in the application to improve the overall application performance.

Description. An application implemented as SOA consists of different services orchestrated together to provide new functionality: The application receives a request, invokes several services and thereby delegates tasks to them needed to provide the overall application functionality. This concept is called "Programming in the large" [9]. If the invocations are independent from each other they can be done in parallel at the same time which increases throughput and therefore the overall processing performance. The distributed computing paradigm of service-oriented architectures enables this feature. The strategy has to be implemented in clients (Fig. 1, Point 1).

Assumptions. It is assumed that each service is hosted on its own physical environment and therefore isolated from each other regarding performance. Thus, multiple concurrent service executions do not influence the performance of each other.

Benefits. The application of this strategy does not need a special performance optimized service bus to achieve high throughput.

Downsides and Problems. The client has to be able to send multiple requests at the same time to the service bus and has to wait for multiple responses which may arrive in various orders. This needs special programming effort as this kind of requesting has to be done asynchronously. There are technologies enabling this kind of service orchestration. One example is BPEL [8]. Another difficulty is identifying which requests can be done in parallel and which requests have to be processed sequentially. For applying this strategy to existing applications, the application flow may have to be changed which can lead to modifications of the overall application architecture.

Application Example. Examples for applying this strategy are all scenarios where requests can be processed independently from each other. For instance, in simulations there are often multiple matrix equations which may be solved at the same time. These equations are independent from each other as they are self-contained in a way that no external information is needed for solving.

3.2 Caching

Goal. The goal of this strategy is avoiding multiple processing of identical requests to speed up the application's performance.

Description. One opportunity to improve the performance of an application's request processing is to avoid the actual request processing at all by exploiting caching. The service bus is the central component which is responsible for any primary service request message consumption: Clients send request messages to the bus which routes the messages to selected services and the responses back to the corresponding requestors [3]. For certain requests the responses are always the same, e.g. a matrix equation solving service returns always the same solution for the same requested equation. These requests can be cached by the service bus to decrease the response time [1]. The strategy has to be implemented in the service bus (Fig. 1, Point 2).

Assumptions. The requests have to be comparable in a way that identical requests can be recognized.

Benefits. The application of this strategy is transparent for the requesting client. Thus, this strategy can be applied without the need for modifying already existing components (of course they have to send all requests to the service bus). If a request is served by the cache the whole service system is discharged.

Downsides and Problems. The identification of cacheable request-response pairs is difficult and causes overhead at the design time of the application. A request which cannot be served by the cache causes additional overhead for cache lookup and management tasks and even decreases the performance for processing this request.

Application Example. In the scientific domain, experiments are executed many times with only little modified input values and therefore internal simulation data within the simulation is often identical [2]. Thus, requests depending on this internal simulation data are also identical and can be cached for further experiment executions.

3.3 Dynamic Service Discovery

Goal. The goal of this strategy is to choose the fastest service for a certain request at runtime to decrease the response time.

Description. One can distinguish between two different binding techniques: Static binding and dynamic binding [10]. The first one enables the client to explicitly define

which service should be used while the latter one sends the request to the service bus which discovers a service matching the functional requirements of the request and then sends the request to this selected service [3]. This service discovery can be enriched by taking non-functional requirements expressing capabilities into account, too [6]: If there are functionally equivalent services, non-functional capabilities of the service are analyzed to select the service guaranteeing the fastest response time. This enables optimized load balancing, too. The Dynamic Service migration strategy (see Section 3.4) may be applied to optimize the services before comparing them. The strategy has to be implemented in the service bus to enrich the service discovery (Fig. 1, Point 3).

Assumptions. To select the fastest service, all available suitable services have to be comparable in their performance for processing a certain request. This performance values have to be predictable automatically (either by the service bus or by the respective services).

Benefits. The application of this strategy is transparent for the requesting client. Thus, this strategy can be applied without the need for modifying already existing legacy components (of course they have to send all requests to a service bus).

Downsides and Problems. This strategy only improves the performance if the overhead caused by the discovery is below the time saved by the faster service. Otherwise dynamic service discovery even slows down the performance.

Application Example. An example is a simulation orchestrating services for complex calculations whose response time depends on a specified requested quality of the output data. Some algorithms offer only low quality of data but guarantee a fast calculation. Other algorithms are designed to achieve high quality of data but are more time-consuming. Depending on the required quality of data (non-functional requirement) the fastest service can be chosen.

3.4 Dynamic Service Migration

Goal. The goal of this strategy is to achieve the fastest response time for processing a request regarding the environment and location conditions a service is hosted on.

Description. There are services whose response time to process a request depends on the power of the environment they are hosted on. The Dynamic Service Migration strategy moves services from less powerful machines to more powerful ones to scale up [13, 15]. Other scenarios increasing the performance are the migration of one service collocated to another service to cut down network costs or the migration of other services hosted on the same environment to other environments to free resources. This component-based migration is possible because of the loose coupling concept of SOA. The strategy may be implemented in the service bus which triggers and manages the migration (Fig. 1, Point 4).

Assumptions. To apply this strategy, the migration of services has to be feasible.

Benefits. The application of this strategy is transparent for the requesting client. Thus, this strategy can be applied without the need for modifying already existing components. Recall that we assume that the services send all requests to a service bus.

Downsides and Problems. The migration of a running service from one machine to another machine is complex and needs management operations which have to be implemented. Especially handling of local data is difficult, because even if a service can be migrated to another machine, a huge amount of data which also has to be transferred can lead to problems: the time savings achieved by the more powerful environment may be too small and the overall processing time (including the time needed for migration) for a single request even increases. To avoid this, the design of the services and the overall architecture of the application have to be aware that this strategy may be applied. This causes additional overhead at the development time and is generally difficult. To find out whether a migration of a service on runtime leads to a faster response time to process a certain request is difficult and depends on many factors. The component managing this migration has to calculate predictions in which scenarios and constellations a migration makes sense.

Application Example. One example taken from our experiences with bone remodeling simulation workflows is the processing of big data sets. The simulations typically process a huge amount of data by sequentially invoking services and passing the data from one service to another service. Because the services are used by multiple simulations, it is not possible to host all services and store all needed data on a single environment. Thus, the migration of services co-located to the data to be processed on runtime may improve the performance in terms of response time because network costs are cut down.

3.5 Multiple Service Invocation

Goal. The goal of this strategy is to choose the fastest service for a certain request at runtime to decrease the response time.

Description. The selection of the fastest service can be difficult, especially if there are completely different ways to process a single request. There are situations where the Dynamic Service Discovery strategy cannot be applied to discover the fastest service because the required values to compare the different services are not calculable. SOA offers a solution to achieve maximum request processing performance by sending the request to all available appropriate services concurrently and taking the response returned by the first responding service. This decreases the response time to an ideal value as the fastest available service is implicitly chosen. To make this work, the different services have to be isolated in a way that they do not affect each other's response time. The strategy has to be implemented in the service bus (Fig. 1, Point 5).

Assumptions. It is assumed that the multiple service invocations have no negative impact on the performance of other request processing services.

Benefits. The application of this strategy is transparent for the requesting client. Thus, this strategy can be applied without the need for modifying already existing components. Recall that we assume that the services send all requests to a service bus.

Downsides and Problems. The concurrent invocation of multiple functional identical services basically produces unnecessary workload for the whole service-oriented environment. To avoid negative impact on other services in terms of performance cloud technology may be used for the provisioning of new services discharging the system (i.e., applying the Multiple Service Instantiation strategy, see Section 3.6).

Application Example. One example from the mathematics domain is solving a matrix equation using numerical or algebraic techniques. For a numerical algorithm starting with random values trying to converge towards the solution by executing multiple iterations, the number of steps and thus the time needed to calculate the solution is not predictable. Thus, for certain equations, data sets and algorithms it is impossible to determine the fastest solving algorithm in advance.

3.6 Multiple Service Instantiation

Goal. The goal of this strategy is to increase the performance by invoking only services having free capacity.

Description. The performance of the overall system decreases if services cannot process a large number of requests any more. If there is no possibility to balance the outstanding workload differently, this strategy solves the problem by instantiating more services functionally equivalent to the overloaded ones [15]. The new instantiated services can be invoked in parallel and hence scale out. The distribution of the workload discharges overloaded services and thus increases the throughput. The strategy may be implemented in the service bus (Fig. 1, Point 6).

Assumptions. The current workload and utilization of a service has to be visible to the service bus to enable the selection of appropriate services and there have to be free resources for hosting the new instantiated services isolated in a way that the concurrent executions do not decrease the performance of each other.

Benefits. The application of this strategy is transparent for the requesting client. Thus, this strategy can be applied without the need for modifying already existing components. Recall that we assume that the services send all requests to a service bus.

Downsides and Problems. The application of this strategy only improves the performance if the additional needed time for instantiation is below the time which is saved by invoking the cloned service. Especially if local data also has to be cloned and transferred to another hosting environment, this leads to additional time-consumptions caused by network costs. A solution to solve this problem of data migration is using stateless services. Applying this strategy requires additional resources in terms of hardware or virtualized systems. Thus, it has to be ensured that this has no negative impact on the performance of other services (e.g. through cloud technology).

Application Example. In service-oriented environments where multiple SOA applications run at the same time certain services may be overloaded because their offered functionality is so general that it is used by many of these applications. In our simulation experiments matrix solving services are frequently overloaded, for example.

4 Conclusion and Outlook

We presented six high level strategies to increase the overall performance of service-oriented applications and showed how to apply them to build high performance SOA applications. As five of the six strategies can be implemented in a performance-driven service bus we plan to implement this bus and integrate our existing migration prototypes [5]. This bus enables performance optimization which is transparent to the orchestrating component and provides a basis for evaluation scenarios for showing that the applied strategies also increase the overall performance in practice.

References

1. Rao, F. Y. et al.: Message Oriented Middleware Cache Pattern – a Pattern in a SOA Environment. In: Fourth "Killer Examples" for Design Patterns and Objects First Workshop (2005)
2. Sonntag, M., Karastoyanova, D.: Next Generation Interactive Scientific Experimenting Based On The Workflow Technology. In: Proceedings of the 21st IASTED International Conference on Modelling and Simulation (2010)
3. Keen, M. et al.: Patterns: Implementing an SOA Using an Enterprise Service Bus. IBM Redbooks (2004)
4. Erl, T.: Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services. Prentice Hall PTR (2004)
5. Binz, T. et al.: CMotion: A Framework for Migration of Applications into and between Clouds. In: Proceedings of SOCA (2011)
6. Leymann, F.: The (Service) Bus: Services Penetrate Everyday Life. In: ICSOC (2005)
7. Chappell, D.A.: Enterprise service bus. Theory in Practice. O'Reilly Media (2004)
8. Weerawarana, S. et al.: Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More. Prentice Hall PTR (2005)
9. DeRemer, F. and Kron, H.: Programming-in-the-Large Versus Programming-in-the-Small. Software Engineering, IEEE Transactions on, SE-2, 80-86 (1976)
10. Papazoglou, M.: Web Services: Principles and Technology. Pearson Prentice Hall (2008)
11. Endrei, M. et al.: Patterns: Service-Oriented Architecture and Web Services. IBM Redbooks (2004)
12. Cohen, F.: FastSOA. Morgan Kaufmann (2007)
13. Hao, W. et al.: Dynamic Service and Data Migration in the Clouds. In: Computer Software and Applications Conference (2009)
14. Weikum, G. et al.: Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann (2002)
15. Lee, J. Y. et al.: Software Approaches to Assuring High Scalability in Cloud Computing. In: IEEE 7th International Conference on e-Business Engineering (ICEBE) (2010)

Guided Control Flow Unfolding for Workflow Graphs Using Value Range Information

Thomas S. Heinze¹, Wolfram Amme¹, Simon Moser², Kai Gebhardt¹

¹ Friedrich Schiller University of Jena
{T.Heinze,Wolfram.Amme,Kai.Gebhardt}@uni-jena.de

² IBM Software Laboratory Böblingen
smoser@de.ibm.com

Abstract. In our previous work, we have introduced a technique to unfold the control flow in workflow graphs based upon static information about constant data values. Using this technique allowed us to safely transform certain kinds of conditional into unconditional control flow, and thus to support a usually data-unaware verification of business processes by more accurate process models. In this paper, a generalisation of this technique is discussed which can be employed in combination with arbitrary information about data values. This way, we show how statically derived value range information is beneficial for unfolding and therefore eliminating conditional control flow in a wider range of cases.

1 Introduction and Motivation

Verification of business processes today is often done using a Petri-net-based process model in which data aspects are being neglected. Prominent examples are the verification of soundness [8], and the verification of its counterpart controllability [6] in case of distributed business processes. The advantage of these data-unaware approaches lies in the feasible and often efficient analysis that is possible when process data is not considered. However, while such a verification is supposed to provide correct results in most cases, in certain circumstances, false-positive as well as false-negative verification results may occur [4, 10].

Given that properties like soundness or controllability relate to control flow, this kind of wrong verification results is mainly due to an imprecise modelling of business processes' control flow. Reasoned by the omittance of data aspects within the Petri-net-based process model, conditional control flow is therein over-approximated by nondeterminism, resulting in an abstraction too coarse. In [4], we advocated the use of static analysis and a process restructuring technique to safely transform certain types of a process's conditional control flow into unconditional control flow, before translating the process into its Petri net model. Consequently, over-approximating the such resolved conditional control flow can be avoided, which yields a more precise process model and thus verification.

The restructuring technique presented in [4] is based on the observation, that a branching or loop condition can be statically evaluated if therein referenced variables are assigned constant values only. Since the values of variables, and therefore the value of the condition, correlate with the control flow path taken

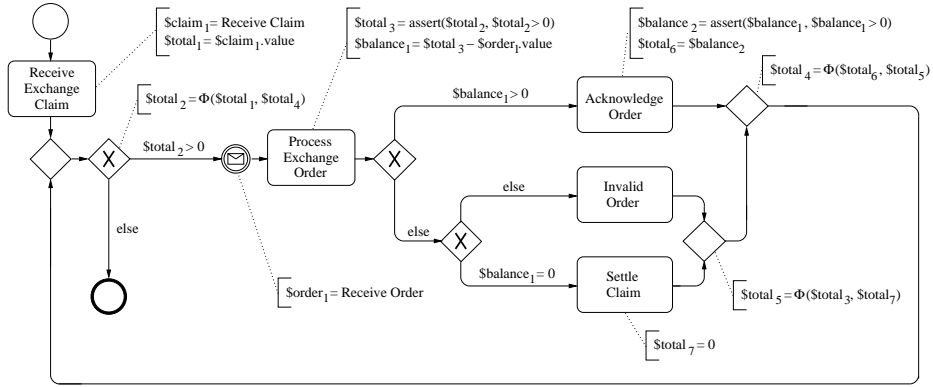


Fig. 1. Running example: Extended workflow graph

at process runtime, separating and duplicating the control flow for each combination of assigned values then allows for the evaluation and substitution of the condition with unconditional control flow in each duplicate. Since the restructuring technique in its current form is thus restricted to analysis information about constant values, only conditions with variables defined over constants, or single messages, can be resolved. In this paper, we discuss a generalisation of the technique to relax this limitation. For that purpose, the generalised technique is enabled to be used in combination with an arbitrary static analysis which yields an abstraction for the values of process data. In particular, we will show how a value range analysis helps in resolving branching or loop conditions in cases condition variables are not necessarily restricted to constant values.

In principle, our restructuring technique can be seen as an unfolding of a process’s control flow. However, existing unfolding techniques restructure the entire process with all variables, though, it is only necessary to unfold those parts and variables related to a branching or loop condition. Further, unfolding at the value level is infeasible in case of infinite data domain such that an abstraction for variables’ values is required. Our restructuring approach overcomes these issues by guiding the control flow unfolding based on static analysis information.

The remainder of the paper is structured as follows: The next section introduces the process representation format and analysis used to derive value range information. In Section 3, we describe our generalised technique for guided control flow unfolding and its use in combination with value range information. Related work is discussed in Section 4. Finally, Section 5 concludes the paper.

2 Workflow Graphs and Value Range Analysis

In order to allow for the static derivation of information essential to our guided control flow unfolding approach, a process representation format is required which is capable of representing both, control as well as data aspects of a business process. We therefore use an extension of workflow graphs [4, 8]. Workflow

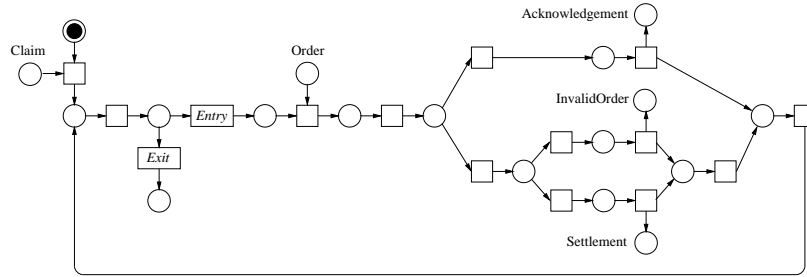


Fig. 2. Petri-net-based process model (Open workflow net [6])

graphs support a simple and flexible modelling of a process's control structure. However, since workflow graphs only map control flow, we augment them with a notation of process data. Thus, nodes and edges of a workflow graph are annotated with data manipulation statements in Concurrent Static Single Assignment (CSSA-)Form [5], which yields an easy to analyse model for a business process's control and data flow. Since we here refer to the verification of fully-specified, i.e., executable, business processes, processes can be translated into their process representation through extended workflow graphs in an automated fashion [2].

In Figure 1, an example process is shown in its representation as extended workflow graph (whose visualisation here closely follows the Business Process Model and Notation). The depicted process models the action of item exchange. A customer therein first specifies the item for exchange by sending message **Claim**. Afterwards, the customer is allowed to order exchange items via message **Order**, where each is acknowledged by message **Acknowledgement**, as long as the total value of orders does not exceed the value of the item for exchange. Otherwise, the last order is rejected, indicated via message **InvalidOrder**. In case the total value of ordered items eventually equals the value of the item for exchange, the claim is settled and message **Settlement** is sent to the customer.

For its realisation, the process is based on a loop whose execution is conditioned. In the Petri-net-based process model, as shown in Figure 2, the loop is mapped to the nondeterministic choice of transitions *Entry* and *Exit*, such that the loop condition is not precisely represented. In consequence, verifying the process using this Petri net model yields erroneous results for properties like controllability, e.g., the process is verified to be non-controllable although it is.

In contrast, the extended workflow graph explicitly models the loop condition: Loop execution is controlled by an integer variable $\$total_2$, representing the difference of the value of the item for exchange and the total value of all already ordered exchange items. Accordingly, if the value of the variable is greater than zero, the loop is executed, and otherwise exited. Therefore, the value of $\$total_2$ is initially set to the value of the item for exchange (message part $\$claim_1.value$), and afterwards updated for each loop iteration with the difference of its current value and the value of an accepted exchange item (message part $\$order_1.value$). As can be seen, all variables are statically only defined

Variable	Derived information	Variable	Derived information
<code>\$claim₁</code>	<i>undefined</i>	<code>\$order₁</code>	<i>undefined</i>
<code>\$claim₁.value</code>	$(0, +\infty]$	<code>\$order₁.value</code>	$(0, +\infty]$
<code>\$balance₁</code>	$[-\infty, +\infty]$	<code>\$balance₂</code>	$(0, +\infty]$
<code>\$total₁</code>	$(0, +\infty]$	<code>\$total₂</code>	$[0, +\infty]$
<code>\$total₃</code>	$(0, +\infty]$	<code>\$total₄</code>	$[0, +\infty]$
<code>\$total₅</code>	$[0, +\infty]$	<code>\$total₆</code>	$(0, +\infty]$
<code>\$total₇</code>	$[0, 0]$		

Table 1. Derived value range information

once, as is indicated by the variables' subscripts. This is the main characteristic of CSSA-Form and vitally supports analysis since variables then coincide with their definition statements. Although, special handling is required if multiple variable definitions have to be joined at a single node of the extended workflow graph. In these cases, statements with so-called Φ -functions are used to merge the confluent definitions into a single value, as is done for the definitions of variables `$total1` and `$total4` by statement `$total2 = Φ ($total1, $total4)`.

To further improve analysis, the implications of branching and loop conditions are annotated in terms of assertion statements. For instance, since the loop in the example process is only executed if the value of variable `$total2` exceeds zero, we know that the variable must be a positive integer within the body of the loop. Therefore, uses of `$total2` in the loop body are substituted with a reference to a new variable which is defined by `$total3 = assert($total2, $total2 > 0)`, indicating that `$total3` equals `$total2` and has a value greater than zero.

Based on the representation of business processes through extended workflow graphs, various static analysis become available. In particular, the use of CSSA-Form facilitates the transfer of analysis techniques from the area of compiler optimisation, like, e.g., constant propagation, global value numbering [5], or value range analysis. Especially the latter analysis provides an abstraction for the values of process data which benefits a guided control flow unfolding.

Value range analysis is a textbook data flow analysis technique which can be used to derive an interval for each integer or floating-point variable and point of a program, such that the variable is guaranteed to take a value in the interval at the given program point. In [2], we have implemented such an analysis for extended workflow graphs and show how it can be used to derive value range information for processes of a small subset of the WS-BPEL language.

An application of the analysis to the example process of Figure 1 yields the value range information shown in Table 1. Note that, therein, each variable is assigned a single interval which is valid at each point of the process, due to the static single definition of variables in CSSA-Form. Further, the analysis defined in [2] is able to exploit data type definitions in business processes for deriving more precise value range information. In case of the example process, the derived intervals for `$claim1.value` and `$order1.value` therefore only comprise positive integers since the corresponding message types are set to `xsd:positiveInteger`.

```

// let  $eWFG$  be an extended workflow graph and let  $loop$  denote a loop therein
 $inf = analyse(eWFG)$ ; //  $inf: Variables \rightarrow AnalysisInformation$ 
normalise  $loop$  in  $eWFG$  and derive instance pattern as is explained in [4];
while ( $\exists guard \in eWFG$  such that  $guard$  is an instance guard) do
   $assertion = \emptyset$ ; //  $assertion: Variables \rightarrow AnalysisInformation$ 
  let  $values$  be the assignment of condition variables valid at  $guard$ ;
  foreach single assignment ( $variable_{condition} \leftarrow variable$ ) in  $values$  do
     $assertion = assertion \cup \{variable_{condition} \mapsto inf(variable)\}$ ;
  end for;
  if ( $evaluate(guard, assertion) == true$ ) then
    let  $instance$  be the loop instance for  $assertion$ ;
    if ( $instance \not\subseteq eWFG$ ) then  $eWFG = eWFG \cup instance$ ;
    end if;
    replace  $guard$  with a control flow edge to  $instance$ ;
  else replace  $guard$  with a control flow edge to the exit node of  $loop$ ;
  end if;
end while;

```

Fig. 3. Generalised algorithm for guided control flow unfolding

3 Guided Control Flow Unfolding

We now describe how the derived analysis information is used to guide control flow unfolding in such a way that conditional control flow can be effectively resolved. In our previous work [4], we only considered branching or loop conditions whose condition variables could be analysed to be only defined by constant values. As a result, it was always possible to resolve conditions using our technique since knowing the constant value for each condition variable allows for inferring the value of the condition. This may not hold true if arbitrary analysis information is used instead. For instance, it is not possible to infer the value of condition $x > 10$ if, e.g., an interval $(0, +\infty]$ has been derived for x . However, using a conservative criteria we are able to check beforehand whether the derived analysis information is sufficient to completely resolve a branching or loop condition.

In principle, unfolding a loop or branching so that conditional control flow is transformed into unconditional control flow is done using two steps. First, the loop or branching is converted into a normal form [4], which is characterised by the separation of all static paths of the control flow, ending in the respective loop header or branching node, that convey distinct values for condition variables. To this end, nodes with Φ -functions merging alternative definitions of condition variables are resolved by duplicating these nodes and their successors for each definition. Thus, after normalisation, definitions of condition variables only converge at the loop header or branching node. Second, in case of a conditional branching, splitting the branching node for each of its predecessors allows for evaluating the branching condition based on derived analysis information. According to the result, the branching is replaced with unconditional control flow leading either to the then- or to the else-part of the branching. For loops, a further step is needed to also separate the remaining paths of the control flow

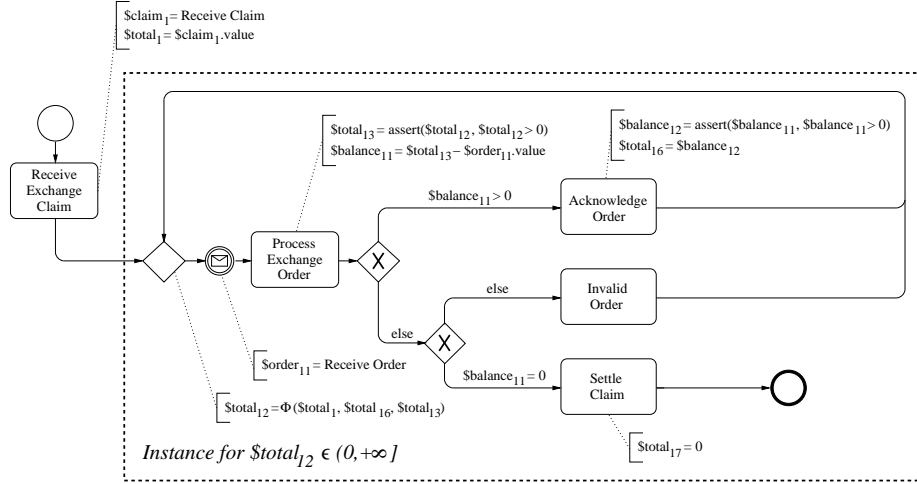


Fig. 4. Unfolded workflow graph

which define distinct values for condition variables, in particular, the dynamic paths coalesced in the loop header node. For that purpose, a loop is divided into duplicates of its loop body, i.e., loop instances, where the execution of each instance is guarded by a copy of the loop condition, i.e., an instance guard. An instance thereby represents a loop iteration for a certain assertion for the values of condition variables. In our previous work [4], assertions constrained condition variables to constant values or messages. However, for a generalised unfolding, we now basically allow arbitrary assertions about variables' values. Eventually, in an iterative procedure, all instance guards are evaluated based on the derived analysis information and, like is done in case of a branching, replaced with unconditional control flow leading either to an instance or to the loop exit node.

In Figure 3, a consolidated view of unfolding a loop is given in terms of an algorithm. Therein, having derived static analysis information for a process's variables using function *analyse*, the loop is first converted into its normal form as explained in [4]. Afterwards, instance guards are iteratively processed by creating an assertion *assertion* for the values of condition variables valid at an instance guard *guard* based on the derived analysis information *inf*. This assertion is then used to evaluate the guard with function *evaluate* and to replace it with unconditional control flow according to the result of the evaluation.

Note that, in the algorithm, the use of analysis information is parameterised using functions *analyse* and *evaluate*. Thus, it is possible to instantiate this general algorithm for exploiting information provided by an arbitrary static analysis by merely declaring implementations for functions *analyse* and *evaluate*, with respect to the analysis. In case of our running example, function *analyse* denotes the value range analysis as described in [2]. Function *evaluate* realises the evaluation of condition expressions based on information derived by function *analyse* and can be implemented using the semantic transfer functions listed in [2].

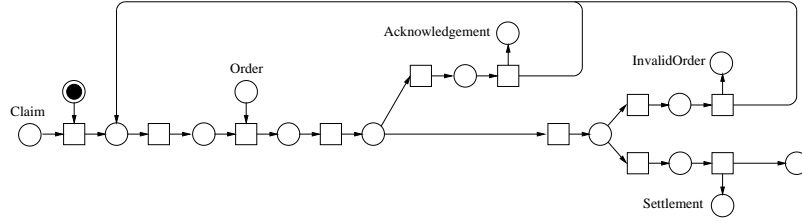


Fig. 5. Petri-net-based process model after unfolding

An application of the algorithm to the loop contained in the example process of Figure 1 allows us to exploit the value range information given in Table 1 for unfolding the loop such that the loop condition is finally resolved. The thus unfolded workflow graph is shown in Figure 4. As can be seen, it was only necessary to create a single instance for assertion $\$total_{12} \in (0, +\infty]$ while unfolding, where variable $\$total_2$ has therein been renamed to $\$total_{12}$.

In Figure 5, the unfolded workflow graph is mapped to a Petri net based on the Petri net semantics for workflow graphs stated in [8]. In so doing, process data, i.e., data annotations in the extended workflow graph, can be discarded without losing precision in representing the loop's control flow since the loop condition is now properly modelled by unconditional control flow. Verifying the thus refined Petri net model with respect to controllability gives then the correct verification result for the example process, i.e., the process is controllable.

In general, our technique provides in this way a heuristics for improving verification in that the more conditional control flow is resolved, the more precise is the Petri-net-based process model and therefore the verification. Furthermore, if termination can be shown, e.g., using termination analysis [10], the verification result is guaranteed to be safe with respect to properties like controllability.

4 Related Work

Improving business process verification based on Petri nets by means of incorporating data aspects is an ongoing research topic. In [10], a termination analysis is introduced for WS-BPEL processes to help in justifying the fairness assumption. The use of high-level Petri nets is, e.g., advocated in [9] for detecting deadlocks in acyclic processes. However, the application of high-level nets in case of cyclic control flow is basically hindered by undecidability, if the domain of process data is unrestricted. This holds also true if high-level nets are unfolded into low-level Petri nets, since infinite domains then yield infinite models. In contrast, our guided unfolding approach is always guaranteed to result in finite models.

Control flow restructuring is also utilised by other program analysis and optimisation methods for improving analysis results [7]. Although, none targets the elimination of conditional control flow or value range analysis in particular. A similar static analysis for the derivation of value ranges to the one used here, which is also applied to WS-BPEL processes, has already been described in [3].

5 Conclusion and Future Work

In this paper, we presented a generalised version of our process restructuring technique [4], which allows us to unfold conditional into unconditional control flow. Compared to our previous work, the generalised technique is enabled to exploit information derived by arbitrary static analysis, as is exemplified for value range analysis, and can therefore be effectively applied in a wider range of cases. Using the technique then helps in compiling more precise process models for data-unaware Petri-net-based approaches to business process verification.

Main issue of future work will be the thorough evaluation of our process restructuring technique. Therefore, we want to implement the technique for structured business processes of the WS-BPEL language. As the value range analysis has already been realised, we currently focus on the implementation of the restructuring algorithm itself. Building on that, we further plan to employ other static analysis, i.e., symbolic methods [1], to our guided unfolding approach.

References

1. Fahringer, T., Scholz, B.: *Advanced Symbolic Analysis for Compilers: New Techniques and Algorithms for Symbolic Program Analysis and Optimization*. No. 2628 in LNCS, Springer (2003)
2. Gebhardt, K.: *Entwurf und Implementierung einer Wertebereichsanalyse für WS-BPEL-Prozesse auf Grundlage erweiterter Workflow-Graphen*. Diplomarbeit, Friedrich Schiller University of Jena (2011)
3. Görlach, K.: *Ein Verfahren zur abstrakten Interpretation von XPath-Ausdrücken in WS-BPEL-Prozessen*. Diplomarbeit, Humboldt University of Berlin (2008)
4. Heinze, T.S., Amme, W., Moser, S.: *Process Restructuring in the Presence of Message-Dependent Variables*. In: *Service-Oriented Computing - ICSOC 2010 International Workshops, PAASC, WESOA, SEE, and SOC-LOG, San Francisco, CA, USA, December 7-10, 2010, Revised Selected Papers*. pp. 121–132. No. 6568 in LNCS, Springer (2011)
5. Lee, J., Padua, D.A., Midkiff, S.P.: *Basic Compiler Algorithms for Parallel Programs*. ACM SIGPLAN Notices 34(8), 1–12 (1999)
6. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: *Analyzing interacting WS-BPEL processes using flexible model generation*. *Data & Knowledge Engineering* 64(1), 38–54 (2008)
7. Steffen, B.: *Property-Oriented Expansion*. In: *Static Analysis, Third International Symposium, SAS'96, Aachen, Germany, September 24-26, 1996, Proceedings*. pp. 22–41. No. 1145 in LNCS, Springer (1996)
8. van der Aalst, W.M.P., Hirsenschall, A., Verbeek, H.M.W.: *An Alternative Way to Analyze Workflow Graphs*. In: *Advanced Information Systems Engineering, 14th International Conference, CAiSE 2002, Toronto, Canada, May 27-31, 2002, Proceedings*. pp. 535–552. No. 2348 in LNCS, Springer (2002)
9. Wagner, C.: *Partner Synthesis for Data-Dependent Services*. In: *Services and their Composition, 4th Central-European Workshop on Services and their Composition, ZEUS 2012, Bamberg, February 23-24 2012, On-Site Proceedings*. pp. 17–24 (2012)
10. Weißbach, M., Zimmermann, W.: *Termination analysis of business process workflows*. In: *Proceedings of the 5th International Workshop on Enhanced Web Service Technologies, WEWST 2010, Ayia Napa, Cyprus, December 1, 2010*. pp. 18–25. ACM Press (2010)

Model support for confidential service-oriented business processes

Andreas Lehmann and Niels Lohmann

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

Abstract. A core motivation of service-oriented execution of business processes is the opportunity to reduce costs by outsourcing certain tasks to third-party service providers. For legal or economic reasons, it might be undesirable that delicate information (e. g., customer data, trade secrets, or financial details) “leak” to the involved third parties. The absence of such leaks — called *noninterference* — can be checked automatically. To this end, a model is required in which each task is assessed as either confidential or public. A drawback of this method is that (1) this distinction has to be made for each task prior to the verification and that (2) an unsuccessful check requires a new confidentiality assessment followed by another verification step.

This paper introduces a full-automatic technique to complete partial confidentiality assessments while guaranteeing noninterference. The proposed technique can be integrated into the design phase of a service-oriented business process and help the modeler choose which tasks can be safely outsourced.

1 Introduction

Service-oriented computing aims at reducing complexity and costs by replacing large monolithic systems by interacting components, called *services*. Such services are offered by service providers and can be flexibly reused in service compositions. As a result, business owners can focus on their core business and outsource other tasks to (possibly cheaper) third-party service providers according to their needs. This trend has led to paradigms such as software as a service, infrastructure as a service, or platform as a service.

The service-oriented execution of a business process adds new challenges, as a business process is usually a very sensitive asset of each company. Though the interplay with third parties can be regulated by contracts, a business owner should never entirely trust other agents. Consequently, only uncritical tasks may be outsourced. To ensure *noninterference* (i. e., the absence of information leaks) in a service-oriented business process, three steps need to be taken: First, the modeler needs to assess each task whether it is confidential or public. This assessment may be straightforward given the nature of the tasks (e. g., processing financial data), but can also be arbitrary for noncritical tasks. Second, the assessment needs to be checked for information leaks. In the context of this paper, we speak of an information leak if a third party can derive confidential runtime information of the business process (e. g., the outcome of choices). Recently [1], we investigated

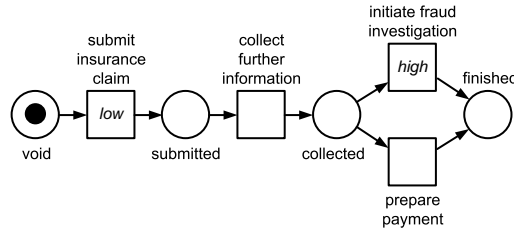


Fig. 1. Petri net model of the insurance business process

noninterference in terms of Petri net models and showed that modern model checking techniques [7] allow to check noninterference of industrial models in fractions of seconds. Finally, the public tasks of the business process can be delegated to third-party service providers, whereas the confidential tasks remain in the responsibility of the business process owner. Apparently, an information leak can be avoided by assessing more tasks as confidential and hence by reducing the number of outsourced tasks. This would, however, contradict the idea of service-orientation.

Contribution. The contribution of the paper is twofold. Instead of requiring a complete confidentiality assessment, we first present an approach that *completes a partial assessment while guaranteeing noninterference*. As a second contribution, we provide a *characterization of all valid assessments*. This enables the modeler to interactively assess tasks by automatically removing any invalid choices. Furthermore, a characterization of all possible assessments can be seen as a first step toward finding a cost-optimal assessment assuming given costs for each transition that cannot be outsourced.

Organization. The rest of this paper is organized as follows. The next section introduces the fundamental concepts of noninterference and a running example we shall use throughout the paper. Section 3 presents our completion approach and a compact representation of all noninterfering assessments. We further discuss several optimizations to avoid combinatorial explosion. In Sect. 4, we provide first experimental results using 559 industrial business process models. Section 5 concludes the paper and sketches a research agenda of future extensions.

2 Background

We consider the Petri net representation of business process models as a basis for the analysis. For this, mappings from common modeling languages, such as WS-BPEL, BPMN, and EPC, exist [6]. To express the confidentiality requirements, we separate the tasks — modeled by Petri net transitions — into two logical security domains: *high* for confidential and *low* for public.

The Petri net in Fig. 1 models a service-oriented insurance claim business process. After submitting the claim, further information is collected and decided whether to initiate a fraud investigation or to prepare the resulting payment before the process finishes. In this example the submitting task is public, because claims can be submitted via a Web site or a call center. The tasks can be

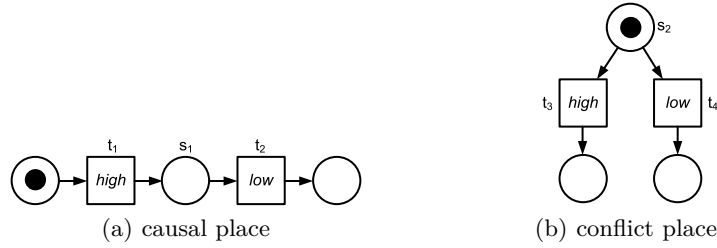


Fig. 2. Patterns for potential causal and conflict places s

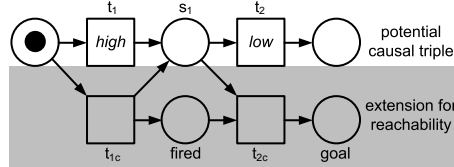


Fig. 3. Pattern for the reachability problem for the causal pattern from Fig. 2(a)

outsourced and the respective transition is labeled *low*. The submission process may contain no confidential data but must only be used to establish the first contact between the insured and his insurance. The task that initiates a fraud investigation is, however, confidential yielding a *high* labeling.

An undesired leak happens whenever information meant to remain in the high domain *leaks* to the low domain. The analysis of noninterference for such Petri net models is carried out with *positive place-based noninterference* (PBNI+) [4]. PBNI+ is an approach to encode and reason about *structural noninterference* (and hence information flow control) in Petri nets. The idea is that some specific places in the net encode different noninterference properties which are leaks from the high to the low domain. In our example “collected” could be such a place, because the following decision depends on it. So in case “collect” is a *high* labeled transition, the transitions “initiate” and “prepare” should also be labeled *high*. In demonstrating the absence of such places in the net, one proves noninterferences.

Figure 2 depicts the two types of possible interference places, the causal case (a) and the conflict case (b). In the causal case, the *low* labeled transition t_2 can only fire after the *high* labeled transition t_1 has fired, so the fact that t_1 (and its corresponding confidential task) has fired is leaked. In the conflict case the two transitions t_3 and t_4 are mutually exclusive, which means that from firing of the *low* labeled transition t_4 one may deduce that the *high* labeled transition t_3 has not fired. Both cases can be expressed as a triple (s, h, l) of a place s , a *high* labeled transition h , and a *low* labeled transition l . In our running example “collected” is both a causal and a conflict place and the triples are (“collected”, “collect”, “prepare”) and (“collected”, “initiate”, “prepare”).

A labeled Petri net is secure in terms of PBNI+ if it contains no such places. Although it appears like a structural property, the behavior of the net needs to be considered to decide PBNI+, because there must be a behavioral dependency

between the creation or consumption of the token on the place s by the involved transition h and l . This dependency can only be checked by taking the behavior of the net (i.e., its state space) into account. Based on our previous work [1], these checks can be expressed as independent reachability problems instead of an examination of the whole state space. Therefore, all checks can be done locally for each specific triple (s, h, l) . Figure 3 depicts the pattern for the causal case (cf. 2(a)) in which the place “goal” is interesting according to reachability. The interested reader is referred to [1].

3 Completion of partial confidentiality assessment

The PBNI+ check has several drawbacks: First, it requires a complete confidentiality assessment; that is, each transition has to be labeled with either *high* or *low*. This means that the modeler needs to make a manual decision for each transition whether the modeled task is confidential or public. Such choices can be very arbitrary, yet still affect overall noninterference. That said, if an information leak was detected, the assessment has to be manually corrected and re-checked.

To this end, we propose to provide a characterization of all valid confidential assessments given a partial (or even empty) confidentiality assessment. Whereas previous work [1] showed that a noninterference check is quite fast, a naive enumeration of all possible assessments has two major downsides:

1. Assuming t transitions in the net, 2^t assessments need to be considered. Even with an average checking time of 30 milliseconds the exponential blowup makes this enumeration not applicable to industrial models with hundreds of transitions.
2. Even if we can determine the valid assessments, an explicit representation is infeasible due to the same exponential blowup. However, only a complete list of all valid assessments gives the modeler maximal freedom to come up with an optimal outsourcing plan.

The rest of this section presents reduction ideas how to tackle each mentioned problem.

3.1 Reducing the number of checks

Considering all possible assessments, one would end up with checking 2^t assignments, if a net has t transitions. For each assignment more than one check (triples in terms of the reachability problem) may be necessary. Therefore it is necessary to reduce the number of checks considerably. In our running example with 4 transitions we already start with $2^4 = 16$ possible assignments. In Tab. 1 all possible 16 assignments are listed.

Based on our observation, all checks are independent from each other, so they can be executed independently [1]. In fact, this does not reduce their number, but all potential critical assignments follow from the structure of the Petri net, because for PBNI+ only potential causal and conflict places are relevant. This means, that only specific parts (the triples) of the net are interesting, which are in $\mathcal{O}(p \cdot t \cdot (t - 1))$ if a net has t transitions and p places. Consider a potential conflict place s with two transition t_1 and t_2 in its postset. Without any assignment on t_1 and t_2 there are two possible triples (s, t_1, t_2) and (s, t_2, t_1) . In the first triple t_1 is

Table 1. All assignments and their necessary checks of our running example.

Assignments				Triples					Checks
submit	collect	initiate	prepare	1	2	3	4	5	
low	low	low	low						0
low	low	low	high		×				1
low	low	high	low	×					1
low	low	high	high						0
low	high	low	low			×	×		2
low	high	low	high		×	×			2
low	high	high	low	×			×		2
low	high	high	high						0
high	low	low	low					×	1
high	low	low	high		×			×	2
high	low	high	low	×				×	2
high	low	high	high					×	1
high	high	low	low			×	×		2
high	high	low	high		×	×			2
high	high	high	low	×			×		2
high	high	high	high						0
Sum				4	4	4	4	4	20

labeled *high* and t_2 is labeled *low* ($[t_1 \mapsto \text{high}, t_2 \mapsto \text{low}]$) and in the second triple it is the other way around. Both other combinations ($[t_1 \mapsto \text{low}, t_2 \mapsto \text{low}]$ and $[t_1 \mapsto \text{high}, t_2 \mapsto \text{high}]$) are not interesting according to PBNI+. Each of these two possible triples will occur in 2^{t-2} of all possible assignments, because of fixing the assignment of the two transitions. In our running example one can identify 5 of these triples:

1. (“collected”, “initiate”, “prepare”): potential conflict place “collected”,
2. (“collected”, “prepare”, “initiate”): potential conflict place “collected”,
3. (“collected”, “collect”, “initiate”): potential causal place “collected”,
4. (“collected”, “collect”, “prepare”): potential causal place “collected”, and
5. (“submitted”, “submit”, “collect”): potential causal place “submitted”.

Table 1 lists all these triples (same enumeration) for all possible assignments. For instance, in line 2, where just “prepare” is assigned *high*, only the second triple needs to be checked, resulting in a single check for this assignments.

Combining these two observations it is not necessary to check all 2^t assignments (by performing $\mathcal{O}(2^t \cdot (p \cdot t \cdot (t - 1)))$ checks), but it is enough to check only the potential critical triples which are in $\mathcal{O}(p \cdot t \cdot (t - 1))$, because they are common through the net structure. Back to our running example: Each of these 5 triples occur $2^{4-2} = 4$ times over all 16 assignments yielding to the sum of 20 checks. However, it is not necessary to perform all 20 checks (× in Tab. 1), but it is sufficient to check each possible triple (columns in Tab. 1) once.

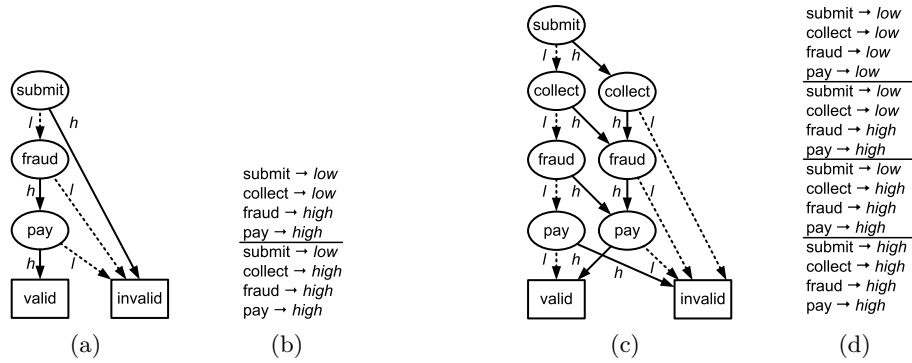


Fig. 4. BDD representation (a) and all valid assignments (b) of running example of Fig. 1. Without initial constraints, more assignments are possible (c, d).

In case the modeler has already assigned some confidentiality, the set of potential critical triples decrease and further triples can be ruled out. In fact our running example has two preassigned tasks (“submit” \mapsto *low* and “initiate” \mapsto *high*), so two triples (columns 1 and 4) are left to decide for all 16 assignments whether they are noninterfering.

To summarize, the main idea is to identify structural causal and conflict triples (columns in terms of the table) once for the net which has polynomial complexity in the net size. Afterwards perform these polynomial many checks (locally and independently) also once and represent all valid assignments in a compact way, which is the content of the following subsection.

3.2 Compact characterization of valid assessments

To fight the exponential blowup of the number of the valid assignments, we employ a *symbolic* representation, namely *binary decision diagrams* (BDDs) [2]. BDDs are successfully used in verification [3] as they can represent sets of bit vectors very compactly.

Figure 4(a) depicts an example of a BDD that represents all valid confidentiality assessments of the running example. The oval nodes are labeled with transition names and represent decisions whether to assess the transition as *high* (continuous outgoing arrow) or *low* (dashed outgoing arrow). After a sequence of decisions, either the node “valid” or “invalid” is reached which describes the status of the resulting assessment. Note that Fig. 4(a) does not mention the “collect” transition: This means that either label is valid for this transition, resulting in 2 valid assessments (cf. Fig. 4(b)). We can further derive that the pay task must be confidential in any case. In case no initial assessment is given (i. e., no transition is initially labeled high or low), the resulting BDD (cf. Fig. 4(c)) characterizes 2 additional valid assessments: setting all transitions to *high* or all transitions to *low* (cf. Fig. 4(d)).

The construction of the BDDs from the noninterference verification results use standard BDD operations for which efficient algorithms exist. In particular,

Algorithm 1 Overall algorithm**Require:** Petri net N

```

1: BDD  $\leftarrow$  true
2: for all relevant potential causal/conflict triples  $(s, h, l)$  do
3:   create net  $N_{(s,h,l)}$  and perform reachability check
4:   if place “goal” can be marked (i.e.,  $s$  is an active causal/conflict place) then
5:     BDD  $\leftarrow$  BDD  $\wedge \neg(h \wedge \neg l)$ 
6:   end if
7: end for
8: return BDD

```

the addition of further constraints (e. g., further assessments of the modeler) can be realized at modeling time and be used to guide the confidentiality assessment.

Algorithm 1 describes how a complete characterization of all valid assessments can be calculated. We begin with a BDD that assigns true (viz. “high”) to all transitions. Then, we check for each potential causal and conflict triple (s, h, l) whether it is an actual violation of noninterference using the reachability check sketched in Fig. 2. In case a violation is found, the respective (partial) assignment is excluded by adding the constraint $\neg(h \wedge \neg l)$ to the BDD. This excludes assignments $[h \mapsto \text{high}, l \mapsto \text{low}]$.

4 Experimental results

The evaluation uses a library of 559 industrial business processes from different business branches, including financial services, ERP, supply-chain, and online sales [5]. They contain no semantic information with respect to the security domains; that is, they are not labeled for security analysis. To this end, this is a good start for our approach, because we can characterize all possible confidential assessments. Table 2 summarizes their experimental results.

As summarized in Tab. 2 we only need to perform 282 checks for the biggest process (no assignments) in contrast to more than 2^{100} checks, which takes 3 seconds on a desktop computer. For this process, the respective BDD has 1,054 nodes.

Table 2. Experimental results of the 559 industrial business processes.

	minimum	average	maximum
transitions (exponent of problem size)	1	20	100
causal triples (cf. Fig. 2(a))	3	34	242
conflict triples (cf. Fig. 2(b))	0	4	90
possible assignments (main factor for checks)	2	1.048.576	$> 10^{30}$
sum of triples (necessary checks)	3	38	282

5 Conclusion

Summary. Confidentiality is important in service-oriented business processes, because business processes are sensitive asset of each company. To express such confidentiality requirements one can use PBNI+, which can be verified on the fly for business processes. So the next step after the verification of a complete assessed business process is to support the modeler in 2 ways: firstly by automatically complete a partial assessed business process and, secondly, by providing a complete characterization of all valid assessments. As shown in this paper, first numbers on runtime are very promising.

Lessons learnt. It is possible to derive all 2^t assessments with only polynomial many checks. The independence shown earlier is essential for this reduction. A polynomial number of checks is feasible for industrial business processes. In order to represent all 2^t assessments, necessary to provide a complete support for all assessments, existing model checking techniques (BDD) are used which proved their scalability in industrial settings.

Future work. Future work aims at two directions: Firstly, provide some interactive design support where only possible choices are offered and obvious ones are set automatically. One way could be an integration into an existing business process modeling tool with a graphical user interface. Second, enhance the approach with costs aspects. Based on the complete representation of all valid assessments one could reason about the costs for each assessment.

Acknowledgement. This work was partially funded by the DFG (German research foundation) in the project WS4Dsec in the priority program Reliably Secure Software Systems (SPP 1496).

References

1. Accorsi, R., Lehmann, A.: Automated and fast information flow analysis for business process models (2012), unpublished manuscript available at <http://www.informatik.uni-rostock.de/~al357/reader.pdf>.
2. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Computers* C-35(8), 677–691 (1986)
3. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond. *Inf. Comput.* 98(2), 142–170 (1992)
4. Busi, N., Gorrieri, R.: Structural non-interference in elementary and trace nets. *Mathematical Structures in Computer Science* 19(6), 1065–1090 (2009)
5. Fahland, D., Favre, C., Koehler, J., Lohmann, N., Völzer, H., Wolf, K.: Analysis on demand: Instantaneous soundness checking of industrial business process models. *Data Knowl. Eng.* 70(5), 448–466 (2011)
6. Lohmann, N., Verbeek, H., Dijkman, R.M.: Petri net transformations for business processes – a survey. *LNCS ToPNoC II(5460)*, 46–63 (2009)
7. Wolf, K.: Generating Petri net state spaces. In: ICATPN 2007. pp. 29–42. LNCS 4546, Springer (2007)

Index of Authors

Amme, Wolfram, 127

Beckstein, Clemens, 111
Breitenbücher, Uwe, 119

Calvanese, Diego, 41
Cortes-Cornax, Mario, 49

Doliwa, Dariusz, 25
Dupuy-Chessa, Sophie, 49
Duske, Kristian, 65

Eid-Sabbagh, Rami-Habib, 88

Gebhardt, Kai, 127
Gierds, Christian, 1
Glesner, Sabine, 33

Heinze, Thomas, 127
Herzberg, Nico, 96
Horzelski, Wojciech, 25

Jarocki, Mariusz, 25

Kopp, Oliver, 80, 119
Kretzschmar, Johannes, 111
Kunze, Matthias, 96

Lehmann, Andreas, 134
Lenhard, Jörg, 57

Leymann, Frank, 80, 119
Lohmann, Niels, 134

Müller, Richard, 65
Meyer, Andreas, 73
Moser, Simon, 127

Niewiadomski, Artur, 25

Penczek, Wojciech, 25
Polrola, Agata, 25
Polyvyanyy, Artem, 73

Reiter, Michael, 119
Rieu, Dominique, 49
Rogge-Solti, Andreas, 96
Roller, Dieter, 119

Sürmeli, Jan, 9
Santoso, Ario, 41
Schulte, Daniel, 103
Skaruz, Jaroslaw, 25
Stöhr, Daniel, 33

Unger, Tobias, 119

Wagner, Christoph, 17
Wagner, Sebastian, 80
Weske, Mathias, 73
Wirtz, Guido, 57