Oliver Kopp, Niels Lohmann (Eds.)

Services and their Composition

5th Central European Workshop, ZEUS 2013
Rostock, Germany, 21–22 February 2013
Proceedings

Volume Editors

Oliver Kopp
Universität Stuttgart, Institut für Architektur von Anwendungssystemen
Universitätsstraße 38, 70569 Stuttgart, Deutschland
oliver.kopp@iaas.uni-stuttgart.de

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

Preface

After five years, the Central European Workshop on Services and their Composition (ZEUS) still isn't a classical workshop, where finished scientific results are presented and published. The discussion of ideas being in a first stage remains the focus of ZEUS. We happily perceived that past ZEUS submissions have been extended and got accepted at major conferences. There were also researchers who met on ZEUS and collaboratively published conference papers. We hope that also this year's ZEUS will also foster collaboration of different institutions.

Due to the movement of services into the Cloud, we have broadened the scope of ZEUS to Cloud-enabled applications and RESTful systems. In this year's ZEUS edition, we accepted 10 submissions of which 2 honored the extension of the scope and 8 followed the established scope of ZEUS. We thank all authors for their contributions and look forward to fruitful discussions on the workshop.

Stuttgart/Rostock, February 2013

Oliver Kopp
Niels Lohmann

Organization

Program Committee Co-chairs

Oliver Kopp	Universität Stuttgart, Germany
Niels Lohmann	Universität Rostock, Germany

Program Committee

Rafael Accorsi	University of Freiburg, Germany
Dirk Fahland	Technische Universiteit Eindhoven, The Netherlands
Christian Gierds	Humboldt-Universität zu Berlin, Germany
Thomas Heinze	Friedrich Schiller University of Jena, Germany
Meiko Jensen	Independent Centre for Privacy and Data Protection Schleswig-Holstein, Germany
Agnes Koschmider	Karlsruher Institute of Technology, Germany
Matthias Kunze	Hasso Plattner Institute at the University of Potsdam, Germany
Andreas Lehmann	Universität Rostock, Germany
Philipp Leitner	Vienna University of Technology, Austria
Henrik Leopold	Humboldt-Universität zu Berlin, Germany
Stephan Reiff-Marganiec	University of Leicester, UK
Thomas Ruhroth	TU Dortmund, Germany
Andreas Schönberger	Universtität Bamberg, Germany
Silvia Schreier	FernUniversität in Hagen, Germany
Christian Stahl	Technische Universiteit Eindhoven The Netherlands
Thomas Stocker	University of Freiburg, Germany
Jan Sürmeli	Humboldt-Universitaet zu Berlin, Germany
Ruben Verborgh	Universiteit Gent, Belgium
Matthias Weidlich	Technion – Israel Institute of Technology, Israel
Andreas Wombacher	University of Twente, The Netherlands
Marco Zapletal	Vienna University of Technology, Austria

Steering Committee

Oliver Kopp	Universität Stuttgart, Germany
Niels Lohmann	Universität Rostock, Germany
Karsten Wolf	Universität Stuttgart, Germany

Table of Contents

Control Flow Unfolding of Workflow Graphs Using Predicate Analysis and SMT Solving

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

[1] Friedrich Schiller University of Jena
[t.heinze,wolfram.amme]@uni-jena.de
[2] IBM Research & Development Boeblingen
smoser@de.ibm.com

Abstract. We present an extension of our previously introduced technique for unfolding conditional control flow in extended workflow graphs. This technique allows for a more precise process-to-Petri-net-mapping which is crucial for business process verification. Our new technique derives data flow information about the state space of process data by means of predicate clauses using a novel CSSA-Form-based analysis. The derived information is then exploited in an adjusted unfolding algorithm to resolve conditional control flow utilising the SMT solver YICES.

1 Introduction

Verification of business processes today is typically done using Petri-net-based process models, which allows for a natural modeling and analysis of vital aspects like parallelism and message exchange. The quality of verification thereby strongly depends on the precision of the process-to-Petri-net mapping. In particular, there exists an inherent tradeoff between verification effectivity and precision, as typical properties for business process verification, e.g., soundness or controllability, are in general undecidable for full-specified business processes.

For this reason, more often than not, methods for business process verification omit process data so that the used Petri-net-based process models merely represent the processes' (unconditional) control flow. By this means, the conditional control flow of a process, i.e., its data-dependent branchings and loops, is mapped to nondeterminism which only over-approximates the process's actual behaviour (under the fairness assumption). However, as research has shown lately, such an approach comprises the danger of an erroneous verification, by means of both, false-positive and false-negative verification results [2,7].

In order to tackle this problem, in our previous work [2,3], we have developed a *control flow unfolding technique* which allows for an increase in the precision of the process-to-Petri-net mapping. More specifically, the developed technique transforms certain kinds of conditional control flow into unconditional control flow for a business process, without inferring the process's execution semantics. As a result, when mapping a thus preprocessed process to its Petri net model, there is no need to over-approximate conditional control flow using nondeterminism and therefore no possible source of error for verification. In other words, we have

envisioned a compiler, which takes as input a business process and generates as output a Petri net. However, in contrast to a conventional compiler, its objective is not to result in efficient runtime code but rather to produce a most-precise though still effectively verifiable Petri-net-based process model.

Our previous technique exploited data flow information derived by copy propagation [2], i.e., information about constant values, or value range analysis [3], i.e., value ranges for integers, to unfold a process's conditional control flow. In this work, we will sketch a new *CSSA-Form-based analysis* for gaining a more general representation for the state space of process data in terms of predicate clauses and how the such derived information is then used to integrate a *SMT solver* into our unfolding approach, so that its applicability is further widened.

The remainder of the paper is structured as follows: The following section introduces the example process which is used for illustration throughout the paper. In Section 3, we describe the CSSA-Form-based analysis to derive predicate clauses. The use of the thus derived data flow information in our adjusted control flow unfolding technique is explained in Section 4. Finally, after a brief discussion of related work in Section 5, Section 6 concludes the paper.

2 Running Example: Rock-paper-scissors

For illustration, we will use the example shown in Figure 1. On its upper left-hand side, a (business) process is shown in a textual format. The process models the game *Rock-paper-scissors*, where two partners (A and B) play against each other. The idea is, that each player decides whether to take one of the three items: rock, paper, scissors. If A takes scissors and B paper, A takes paper and B rock, or A takes rock and B scissors, player A wins the game and vice versa. If both players take the same item, the game continues with another round.

In order to implement the game, the three items are encoded in the process by using three integers, i.e., scissors becomes 0, paper becomes 1, rock becomes 2. In consequence, the decision whether player A, who has chosen $\$a$, won over player B, who has chosen $\$b$, can be done based on the expression $(\$a + 1)\ mod\ 3 = \$b$ and vice versa. Therefore, the process contains a loop which tests if A and B chose the same item. If so, the loop continues and another round is played. In the loop, A and B state their choice by sending an integer to the process, which is encoded into either 0, 1, or 2. Afterwards, the winner is determined, if existent, using two conditional branchings and an appropriate message is sent back.

When verifying this process with regard to *controllability* [4], i.e., whether partners A and B exist for which the process will always be able to terminate its execution, the process is mapped to a Petri-net-based process model first. The application of a conventional mapping, e.g., using the pattern-based approach described in [4,5], results in the Petri net shown in Figure 1. Note that the loop and conditional branchings are therein mapped to conflicting transitions modeling nondeterminism. Thus, the Petri net only over-approximates the process's control flow such that verifying the process based on this process model results in an erroneous finding that the process is not controllable, while it rather is.

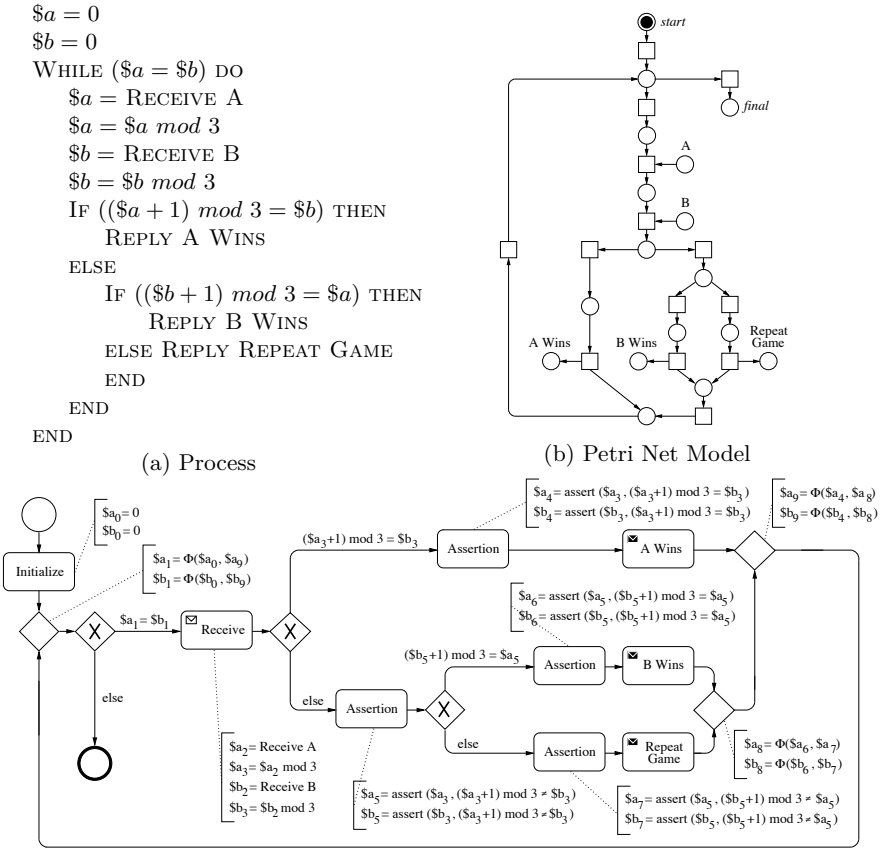

Fig. 1. Running example: Process, Petri net model, and extended workflow graph

3 Predicate Clause Analysis

Using control flow unfolding as described in [2,3], allows for resolving certain kinds of conditional control flow such that nondeterminism can be avoided in a process's Petri net model. To this end, data flow information is derived and used to identify control flow paths where a loop or branching condition is statically evaluable based on the gained information. These control flow paths are then made explicit, i.e., unfolded, and, as a result, the condition can be removed.

In order to derive data flow information about process data, we use *extended workflow graphs* [2]. Workflow graphs provide a well-known format to model the control flow structure of a process. For the representation of process data, workflow graphs are enriched by annotating nodes and edges with instructions and condition expressions in *Concurrent Static Single Assignment (CSSA-) Form* [1], which benefits analysis. The such defined extended workflow graph for

our example process is shown at the bottom of Figure 1. Note that the process's variables are therein (statically) defined only once such that variables become values, which is denoted by subscripts, e.g., $\$a$ becomes $\$a_0, \ldots, \$a_9$. In order to merge conflicting variable definitions into a single value, *Φ-functions* are used, as is done for variable $\$a$'s and $\$b$'s definitions, e.g., $\$a_1 = \Phi(\$a_0, \$a_9)$. Further, several *assertions* have been added exposing the induced constraints for a value referenced in a condition expression, e.g., $\$a_4 = assert(\$a_3, (\$a_3 + 1) \ mod \ 3 = \$b_3)$ guarantees for all dominated uses of value $\$a_3$, which have been renamed to $\$a_4$, that its value satisifies the branching condition $(\$a_3 + 1) \ mod \ 3 = \$b_3$.

For the example process, data flow information resulting from constant propagation or value range analysis does not help control flow unfolding since the information derivable is too limited to allow for evaluating the loop or branching conditions. In contrast, we here employ a novel analysis based on the analysis framework described in [1], which produces a more general notion for the state space of process data in terms of predicates. Thereby, predicates denote instructions or condition expressions as they appear in the process's extended workflow graph. Sets of predicates depict conjunctions of predicates, so-called *predicate clauses*, as they hold on a single control flow path. Sets of predicate clauses are then used, by means of disjunctions, to merge information over multiple paths.

In the following, let *Variables* denote the set of variables and *Predicates* the set of instructions and condition expressions appearing in an extended workflow graph. *Predicates* is augmented with instructions in $\{x = y \mid x, y \in Variables\}$. Function $var(pred)$ returns the set of contained variables for $pred \in Predicates$. Then, for each variable v, we estimate its state space in terms of sets of predicate clauses $inf(v)$ using the following CSSA-based data flow equations:

Incoming Message If variable v is defined by incoming message activity, e.g., RECEIVE, the result is the singleton set $inf(v) = \{\emptyset\}$

Constant Assignment If variable v is defined by constant assignment, i.e., $v = c$, the result is the singleton set $inf(v) = \{\{v = c\}\}$

General Assignment If variable v is defined by expression assignment $v = expr$ with $var(expr) = \{x_1, \ldots, x_n\}$, the result is:

$$inf(v) = \bigcup_{k_1 \in inf(x_1), \ldots, k_n \in inf(x_n)} \{k_1 \setminus kill(v) \cup \ldots \cup k_n \setminus kill(v) \cup \{v = expr\}\}$$

Assertion If variable v is defined by assertion, i.e., $v = assert(x, pred)$ with $var(pred) = \{x_1, \ldots, x_n\}$, the result is:

$$inf(v) = \bigcup_{k_1 \in inf(x_1), \ldots, k_n \in inf(x_n)} \{k_1 \setminus kill(v) \cup \ldots \cup k_n \setminus kill(v) \cup \{pred, v = x\}\}$$

Φ-/π-Function If variable v is defined by Φ-/π-function, i.e., $v = \Phi(x_1, \ldots, x_n)$ or $v = \pi(x_1, \ldots, x_n)$, the result is:

$$inf(v) = \bigcup_{k_i \in inf(x_i)} \{k_i \setminus kill(v) \cup \{v = x_i\}\}$$

where $kill(v) = \{pred \in Predicates \mid v \in var(pred)\}$ for all $v \in Variables$.

Applying the analysis to the example process then results for variable $\$a_1$:

$$\begin{array}{l}\{\ \{\ \$a_0 = 0\ ,\ \$a_1 = \$a_0\ \},\\ \quad \{\ \$a_3 = \$a_2\ mod\ 3\ ,\ \$b_3 = \$b_2\ mod\ 3\ ,\ (\$a_3 + 1)\ mod\ 3 = \$b_3\ ,\ \$a_4 = \$a_3,\\ \qquad \$a_9 = \$a_4\ ,\ \$a_1 = \$a_9\ \},\\ \quad \{\ \$a_3 = \$a_2\ mod\ 3\ ,\ \$b_3 = \$b_2\ mod\ 3\ ,\ (\$a_3 + 1)\ mod\ 3 \neq \$b_3\ ,\ \$a_5 = \$a_3,\\ \qquad \$b_5 = \$b_3, (\$b_5 + 1)\ mod\ 3 = \$a_5, \$a_6 = \$a_5, \$a_8 = \$a_6, \$a_9 = \$a_8, \$a_1 = \$a_9\ \},\\ \quad \{\ \$a_3 = \$a_2\ mod\ 3\ ,\ \$b_3 = \$b_2\ mod\ 3\ ,\ (\$a_3 + 1)\ mod\ 3 \neq \$b_3\ ,\ \$a_5 = \$a_3,\\ \qquad \$b_5 = \$b_3, (\$b_5 + 1)\ mod\ 3 \neq \$a_5, \$a_7 = \$a_5, \$a_8 = \$a_7, \$a_9 = \$a_8, \$a_1 = \$a_9\ \}\ \}\end{array}$$

and for variable $\$b_1$:

$$\begin{array}{l}\{\ \{\ \$b_0 = 0\ ,\ \$b_1 = \$b_0\ \},\\ \quad \{\ \$a_3 = \$a_2\ mod\ 3\ ,\ \$b_3 = \$b_2\ mod\ 3\ ,\ (\$a_3 + 1)\ mod\ 3 = \$b_3\ ,\ \$b_4 = \$b_3,\\ \qquad \$b_9 = \$b_4\ ,\ \$b_1 = \$b_9\ \},\\ \quad \{\ \$a_3 = \$a_2\ mod\ 3\ ,\ \$b_3 = \$b_2\ mod\ 3\ ,\ (\$a_3 + 1)\ mod\ 3 \neq \$b_3\ ,\ \$a_5 = \$a_3,\\ \qquad \$b_5 = \$b_3, (\$b_5 + 1)\ mod\ 3 = \$a_5, \$b_6 = \$b_5, \$b_8 = \$b_6, \$b_9 = \$b_8, \$b_1 = \$b_9\ \},\\ \quad \{\ \$a_3 = \$a_2\ mod\ 3\ ,\ \$b_3 = \$b_2\ mod\ 3\ ,\ (\$a_3 + 1)\ mod\ 3 \neq \$b_3\ ,\ \$a_5 = \$a_3,\\ \qquad \$b_5 = \$b_3, (\$b_5 + 1)\ mod\ 3 \neq \$a_5, \$b_7 = \$b_5, \$b_8 = \$b_7, \$b_9 = \$b_8, \$b_1 = \$b_9\ \}\ \}\end{array}$$

Note that the Φ-functions and assertions are included by means of simple assignments, each copying the respective operand's value to the function value.

4 Control Flow Unfolding

Having done the analysis, the derived data flow information, i.e., predicate clauses, can be tested for enabling the evaluation of conditional control flow. Thus, given a branching or loop condition, a *SMT solver*, in our case *YICES*[3], is used to check, on the one hand, whether a conjunction of certain predicate clauses for variables referenced in the condition is satisfiable (otherwise it would represent an infeasible path and can be neglected) and, on the other hand, whether the conjunction implies the condition to be either true or false. In the latter, the condition can be statically evaluated for this specific set of predicate clauses and is thus a candidate for control flow unfolding.

In the example process, the loop with condition $\$a_1 = \$b_1$ is such a candidate for unfolding, i.e., the control flow path which is denoted by the predicate clause $\{\$a_3 = \$a_2\ mod\ 3, \$b_3 = \$b_2\ mod\ 3, (\$a_3 + 1)\ mod\ 3 \neq \$b_3, \$a_5 = \$a_3, \$b_5 = \$b_3, (\$b_5 + 1)\ mod\ 3 \neq \$a_5, \$a_7 = \$a_5, \$a_8 = \$a_7, \$a_9 = \$a_8, \$a_1 = \$a_9\}$ for variable $\$a_1$ and the clause $\{\$a_3 = \$a_2\ mod\ 3, \$b_3 = \$b_2\ mod\ 3, (\$a_3 + 1)\ mod\ 3 \neq \$b_3, \$a_5 = \$a_3, \$b_5 = \$b_3, (\$b_5 + 1)\ mod\ 3 \neq \$a_5, \$b_7 = \$b_5, \$b_8 = \$b_7, \$b_9 = \$b_8, \$b_1 = \$b_9\}$ for variable $\$b_1$ is a feasible path since the conjunction of both clauses is satisfiable. Further, the conjunction of the clauses also implies the loop condition $\$a_1 = \$b_1$ always to be satisfied, as can be checked using YICES:

$$\begin{array}{rl}\models & (\$a_3 = \$a_2\ mod\ 3 \wedge \$b_3 = \$b_2\ mod\ 3 \wedge (\$a_3 + 1)\ mod\ 3 \neq \$b_3 \wedge \$a_5 = \$a_3\\ & \quad \wedge \$b_5 = \$b_3 \wedge (\$b_5 + 1)\ mod\ 3 \neq \$a_5 \wedge \$a_7 = \$a_5 \wedge \$b_7 = \$b_5 \wedge \$a_8 = \$a_7\\ & \quad \wedge \$b_8 = \$b_7 \wedge \$a_9 = \$a_8 \wedge \$b_9 = \$b_8 \wedge \$a_1 = \$a_9 \wedge \$b_1 = \$b_9) \rightarrow \$a_1 = \$b_1\end{array}$$

[3] `http://yices.csl.sri.com`

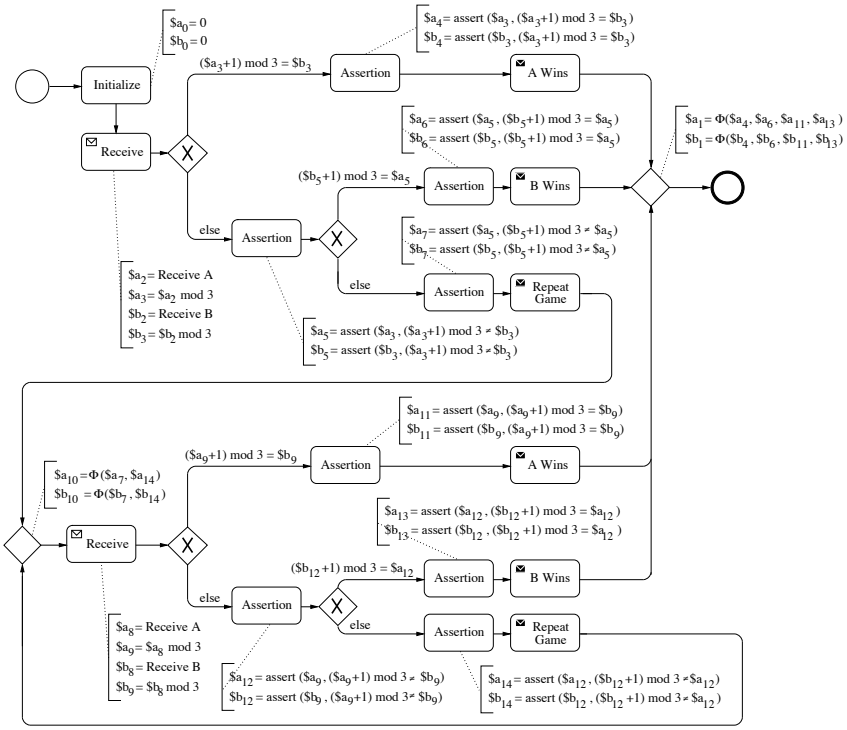

Fig. 2. Extended workflow graph with unfolded loop

Since the derived predicate clauses make it possible to evaluate the loop condition to either true or false for all control flow paths in the example process, the loop can be effectively transformed such that the loop condition is removed. To this end, the control flow paths which are associated to the individual predicate clauses are made explicit by dissolving merge nodes joining these paths through subgraph duplication. In particular, the loop is replaced by copies of it, so-called *loop instances*, based on the derived predicate clauses which therefore act as a kind of invariant for the values of variables $\$a_1$ and $\$b_1$. For instance, predicate clauses $\{\$a_0 = 0, \$a_1 = \$a_0\}$ and $\{\$b_0 = 0, \$b_1 = \$b_0\}$ constitute the invariant $\$a_0 = 0 \wedge \$a_1 = \$a_0 \wedge \$b_0 = 0 \wedge \$b_1 = \$b_0$ which holds for the first loop iteration and allows for evaluating the loop condition to true therein. Thus, the loop condition can be evaluated in each loop instance and afterwards replaced by unconditional control flow to the loop exit or to the same or another instance.

For conducting the above described *control flow unfolding technique*, an adjusted version of the algorithm described in [3] is used, which works with predicate clauses as data flow information and evaluates loop and branching conditions by the help of SMT solver YICES. Applying this algorithm to the loop in the example process results in the extended workflow graph shown in Figure 2. As can be seen, the loop is therein unfolded into two loop instances such that

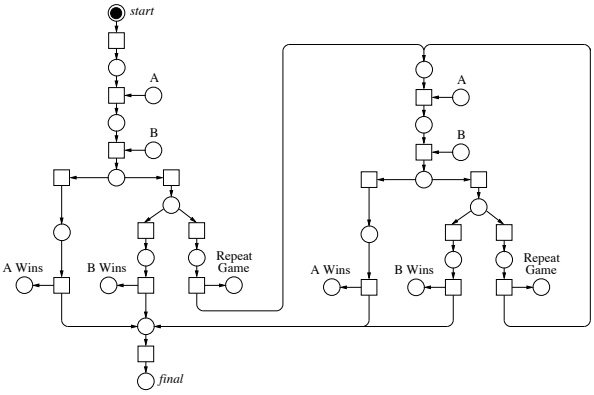

Fig. 3. Refined Petri net model

the loop condition has been eventually resolved. Mapping the thus successfully unfolded process to a Petri-net-based process model yields the Petri net shown in Figure 3. Verifying the process in respect of controllability based on this refined process model then comes to the correct result that the process is controllable.

5 Related Work

The relevance of process data when verifying business processes based on Petri nets is an ongoing research topic. Nevertheless, most approaches to a process-to-Petri-net-mapping either omit data entirely our restrict themselves to data of bounded and limited domain [4,5,7]. Using high-level Petri nets allows for augmenting the process model with (unbounded) data, for which verification methods have been proposed in respect of acyclic processes. However, the application of high-level nets in general leads to undecidability in case of cyclic control flow, and even if data is bounded, state space explosion may hinder a feasible verification. This also applies if high-level nets are unfolded into low-level Petri nets, since an infinite data domain implies an infinite low-level net. In contrast, our unfolding technique always terminates with a finite process model.

A method to integrate SMT solving into Petri-net-based business process verification has already been described in [6]. However, this approach is also restricted to acyclic processes since its termination can else not be guaranteed.

6 Conclusion

In this paper, we presented an extension of our previous technique [3], which allows us to unfold certain kinds of a business process's conditional into unconditional control flow such that a precise mapping of the process to its Petri net model is not impeded by the introduction of nondeterminism. In our previous work, we

have based the control flow unfolding on data flow information derived by copy propagation or value range analysis. However, the information thus derivable can be too limited for resolving certain loop or branching conditions, as is shown by the running example in this paper. In order to enlargen the number of cases our technique is effectively applicable, we now employ a CSSA-based analysis for deriving predicates determining the state space of process data, which are then used in combination with a SMT solver to conduct the unfolding. In this way, we are able to exploit the various background theories available in SMT solvers (supporting real/integer arithmetic, arrays, lists, etc.).

In a current prototype, we have implemented the unfolding technique for a subset of the WS-BPEL language based on value range information. Using the prototype in combination with existing Petri-net-based verification tools, e.g., Fiona [4], then allows for achieving more precise verification results. We plan to integrate the predicate clause analysis and adjusted unfolding algorithm described here into this prototype. Building on that, the thorough evaluation of the control flow unfolding approach remains the main issue for future work.

References

1. Amme, W., Martens, A., Moser, S.: Advanced verification of distributed WS-BPEL business processes incorporating CSSA-based data flow analysis. International Journal of Business Process Integration and Management 4(1), 47–59 (2009)
2. Heinze, T.S., Amme, W., Moser, S.: A Restructuring Method for WS-BPEL Business Processes Based on Extended Workflow Graphs. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) Business Process Management, 7th International Conference, BPM 2009, Ulm, Germany, September 8-10, 2009, Proceedings. pp. 211–228. No. 5701 in Lecture Notes in Computer Science, Springer (2009)
3. Heinze, T.S., Amme, W., Moser, S., Gebhardt, K.: Guided Control Flow Unfolding for Workflow Graphs Using Value Range Information. In: Schönberger, A., Kopp, O., Lohmann, N. (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. pp. 128–135. No. 847 in CEUR Workshop Proceedings, CEUR-WS.org (2012)
4. 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)
5. Lohmann, N., Verbeek, E., Ouyang, C., Stahl, C.: Comparing and evaluating Petri net semantics for BPEL. International Journal of Business Process Integration and Management 4(1), 60–73 (2009)
6. Monakova, G., Kopp, O., Leymann, F.: Improving Control Flow Verification in a Business Process using an Extended Petri Net. In: Kopp, O., Lohmann, N. (eds.) Services und ihre Komposition, Erster zentraleuropäischer Workshop, ZEUS 2009, Stuttgart, 2.-3. März 2009, Proceedings. pp. 95–101. No. 438 in CEUR Workshop Proceedings, CEUR-WS.org (2009)
7. Sidorova, N., Stahl, C., Trčka, N.: Soundness verification for conceptual workflow nets with data: Early detection of errors with the most precision possible. Information Systems 36(7), 1026–1043 (2011)

Consolidation of Interacting BPEL Process Models with Fault Handlers

Sebastian Wagner, Oliver Kopp, and Frank Leymann

Institute of Architecture of Application Systems, University of Stuttgart, Germany
{wagnerse,kopp,leymann}@iaas.uni-stuttgart.de

Abstract The interaction behavior between processes of organizations and their suppliers can be modeled by using choreographies. When an organization decides to gain more control about their suppliers and to minimize transaction costs they may decide to insource these companies. This also requires the integration of the partner processes into the organization. In previous work we proposed an approach to merge (consolidate) interacting BPEL process models of different partners into a single process model by deriving control flow links between the process models from their interaction specification. In this work we are detailing this consolidation approach. Thereby, special attention is turned on extending the consolidation operations in a way that process models with fault handlers can be merged.

1 Introduction

To reduce transaction costs or to gain more control companies often decide to integrate suppliers into their organization (in-sourcing, mergers, and acquisitions). This requires the integration of the processes and the organizational structure of the two companies. In this work we focus on the integration on the process-level. More precisely, we want to merge (or consolidate) complementing process models whose interaction behavior is described by a choreography.

Process modeling languages such as BPEL [10] or BPMN [11] offer different language constructs to raise and handle faults that work similar to *throw-catch* constructs in traditional programming languages such as Java. As fault handling constructs can also cause message exchanges between interacting processes they are also affected by the consolidation. In this paper we want to describe a technique to merge BPEL process models that communicate via fault handlers. Moreover, we describe an extension of the merge operations proposed in [13] and [14].

We use BPEL4Chor [2] to model BPEL choreographies as this language provides a means to define message links between the communication activities of the interacting BPEL processes.

We assume that the reader is familiar with BPEL. Nevertheless, we give a brief overview about BPEL's fault handling concepts in Sect. 2. In Sect. 3 an overview on the merge operations is provided and extensions to them are discussed. Then, Sect. 4 presents a technique to merge processes that communicate via BPEL fault handlers. After discussing related work in Sect. 5, Sect. 6 concludes this paper and provides an outlook on future work.

2 BPEL Fault Handling Basics

BPEL offers three language constructs to repair faulty situations during process execution, namely *fault handlers*, *compensation handlers* and *termination handlers*. If a fault occurs within a `scope` all running activities within this `scope` are terminated and its fault handlers are called. A fault handler is represented by a `catch` or `catchAll` block. Thereby, multiple `catch` blocks can be defined for a `scope`. Each `catch` block catches a particular fault that may be thrown during execution of the `scope` and contains BPEL activities to handle this fault. A `catchAll` block contains logic to catch all other faults that do not match to a particular `catch` block. If no explicit fault handlers are defined for a `scope` it has an implicit default fault handler attached to it. If any kind of fault occurs during the execution of the `scope` the default fault handler triggers compensation handling for its child `scopes` (see below) and finally rethrows the fault to its parent `scope`. If this `scope` does not provide a fault handler for this fault either, it is propagated up to its parent `scope` and so on until the process `scope` is reached. If the process `scope` cannot catch the fault the process fails and is terminated.

Compensation handlers contain activities to undo work that was successfully performed by a `scope` they are attached to, e. g., canceling a flight that was booked by the activities of the `scope`. Hence, they are only executed if their associated `scope` has completed successfully.

To control the termination of a `scope` that is still running a termination handler can be attached to it. Within the termination handler activities can be defined that are performed before the actual termination of the `scope`. If no explicit termination handler was defined for a `scope` its default termination handler compensates its child `scopes`. A more detailed discussion about fault and compensation handling concepts in BPEL was provided by Khalaf et al. [4].

3 Asynchronous and Synchronous Consolidation

We introduced the consolidation operations to merge asynchronous and synchronous communicating process models in [13]. The aim of the consolidation is that the atomic activities of the different participants in the merged process model have the same control flow relations as in the original choreography. The basic idea behind the consolidation algorithm is that the message links imply control flow relations between the activities of the communicating process models. The message link *m* in Fig. 1 implies for instance that the successor *RC*• of the `receive` activity is always performed after the predecessor activity •*S* of the `invoke` activity *S* in process model *A* as *RC*• cannot be performed before *S* completed. However, no statement can be made about the execution sequence

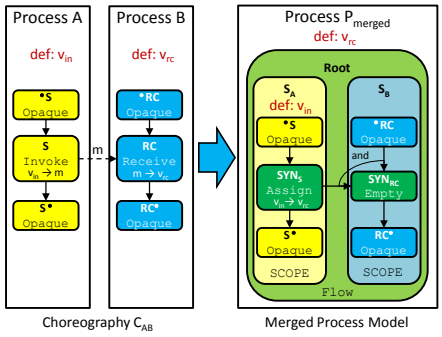

Figure 1. Asynchronous Merging Operation

between $S\bullet$ and $RC\bullet$, e. g., if they have to be performed simultaneously or if $S\bullet$ is performed before $RC\bullet$ and so on. This is different from the synchronous scenario depicted in Fig. 2. There $RC\bullet$ is always performed before $S\bullet$. As activity S does not complete until it received a response message from RP that is performed after $RC\bullet$. Given these implicit control flow dependencies, the sending and receiving activities can act as merge points. Therefore the consolidation operation *materializes* the implicit control flow to explicit control flow relations between the activities.

The consolidation algorithm to merge an arbitrary number of process models is described in the following. As a prerequisite we assume that the choreography is modeled correctly [3] and deadlock free [8]. Moreover, we assume there exists just one instance of each participant per choreography instance, i. e., interaction patterns involving multiple instances of one participant such as *one-to-many send/receive* [1] are not supported yet by the consolidation algorithm. Another restriction we make is that a repeatable constructs such as a BPEL `ForEach` loop do not contain any communication activities that are replaced by control flow links between the process models to be merged. As this would violate the BPEL restriction that repeatable constructs must not be crossed by control flow links [SA00070] [1].

In a first step a new process model P_{merged} is created that contains a `flow` as root activity. For each of the process models P_1 to P_n to be merged a separate `scope` is created in the `flow` activity of P_{merged}. This ensures that the `scope` activities are performed simultaneously. Each scope contains the root activity (along with its child activities) of one of the process models P_1 to P_n. The purpose of the `scope` is to isolate the activities of the process models from each other as they were also isolated in the original choreography. To avoid that an uncaught fault thrown in one scope causes the other scope to crash (as uncaught faults are propagated up to the process scope) a `catchAll` fault handler is added to the scopes as shown in Fig. 2. The `catchAll` contains a `compensate` activity to emulate the default compensation that would have been triggered in an original process without an explicit fault handler. In case an explicit fault handler was defined in a `catchAll` block on the original process scope nothing is changed. If a process scope of a process to be merged has already a a `catch` block defined simply the `catchAll` block is added to this fault handler.

Then the message links are materialized to control flow links. In the asynchronous case the `invoke` activity S is replaced by an `assign` activity SYN_{S} and the `receive` activity RC by an `empty` activity SYN_{RC}. SYN_{S} emulates the message transfer between between the former `invoke` and `receive` activity, i. e., it copies the message from the input variable v_{s} of the `invoke` to the variable v_{rc} of the `receive` activity where the message was copied to before. To perform the assignment the declaration of variable v_{rc} is lifted to the parent `scope` that encloses the two `scopes` that contain the participant activities. Otherwise, SYN_{S} could not access v_{rc}. The `empty` activity SYN_{RC} replaces the former `receive` RC. To avoid name clashes between variables it might be necessary to adapt the variable names accordingly during the consolidation. The incoming and outgoing links of S and RC are mapped to SYN_{S} and SYN_{RC}, respectively. An additional link from SYN_{S} to SYN_{RC} is created. This link ensures that SYN_{RC} is not started before SYN_{S} was executed.

[1] Static Analysis (SA) Fault Codes defined in the BPEL specification [10]

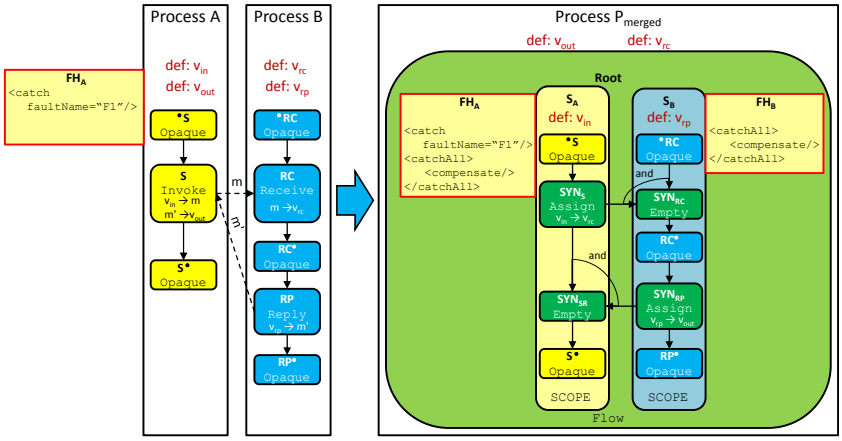

Figure 2. Synchronous Merging Operation

The synchronous merge is sketched in Fig. 2. There additionally the `reply` activity *RP* is replaced by the `reply` activity SYN_{RP} to emulate the transfer of the response message sent via message link m'. SYN_{RP} copies the value of the former `reply` variable v_{rp} to the output variable of the former `invoke` activity v_{out}. The declaration of v_{out} has to be lifted to the parent scope as well to make this variable accessible for SYN_{RP}. The `empty` activity SYN_{SR} is added for the same reason SYN_{RC} was added. The control links of *RC* are mapped to SYN_{RP} and the outbound links of *S* are mapped to SYN_{RC}. Moreover, a new link is created to connect SYN_{S} and SYN_{SR} and another one between SYN_{RP} and SYN_{SR} to ensure that the successors of the former `invoke` activity are not started before.

4 Consolidation in the Context of Fault Handlers

In this section we discuss the challenges that arise when materializing the control flow from message links between communication activities that reside within BPEL fault handlers. Thereby, we focus on the *cross boundary link* constraint imposed by the BPEL specification [SA00071]. This constraint specifies that no control link must point to a target activity within a fault handler from outside the fault handler, i. e., no link must point into a `catch` or `catchAll` block.

In the following we distinguish between three different scenarios (i) fault handlers without communicating activities (ii) fault handlers with only outgoing message links and (iii) fault handlers with at least one incoming message link.

The first scenario is trivial as there is no communication between the fault handlers of the two process models that have to be merged. Consequently, they can be simply merged with the consolidation operations introduced in 3.

The second is scenario is depicted in Fig. 3. In process model *A* the fault handler FH_{A} is attached to the `scope` S_{A}. FH_{A} contains an asynchronous `invoke` activity *A*4 that is related to the corresponding `receive` activity *B*2 in process model *B* via message

link m. Note, that for simplicity reasons the `flow` activity containing S_A and S_B is not explicitly depicted in Fig. 3 and in the following figures.

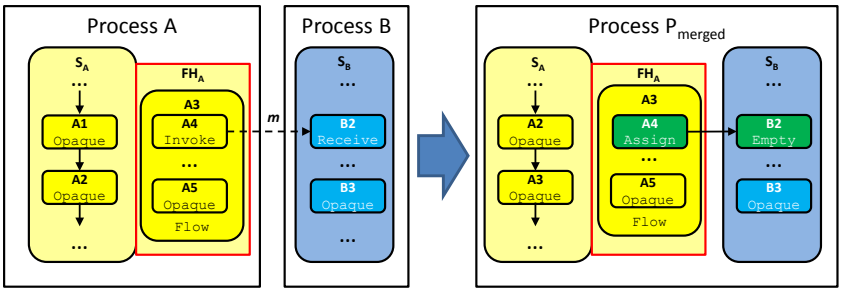

Figure 3. Scenario 2: Message Link pointing from a Fault Handler

To merge the process models in a first step the merge operations introduced in Sect. 3 are applied. This results in process model P_{merged}. As the new control flow links materialized from the message links leave the fault handler boundaries outbound only, the cross boundary link constraint is not violated.

The scenario in Fig. 4 is very similar to the previous one except that `invoke` activity $A4$ is synchronous, hence, a second message link from the `reply` activity points back to $A4$. The synchronous consolidation operation creates from the message link m_2 the control flow link l_2. This link crosses the fault handler boundary of FH_A inbound in order to realize that $A5$ is performed after $B2$ or $B3$ respectively. This violates the cross boundary link constraint.

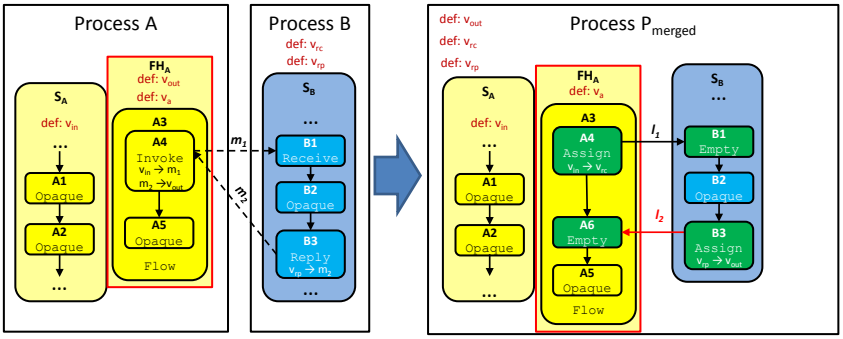

Figure 4. Scenario 3: Message Link pointing into a Fault Handler

To develop a solution to overcome this problem control flow links pointing into a fault handler have to be avoided, hence, the control flow has to be modified accordingly. To resolve the incoming link issue we can either remove the fault handler completely or we move the activities with incoming links out of fault handler. Removing the fault handler completely is not an option as a fault handler can be activated any time during runtime of a scope if a fault is thrown. This behavior cannot be emulated by using other BPEL constructs. Hence, we suggest a solution where the fault handler is kept and the fault handling activities are factored out of it.

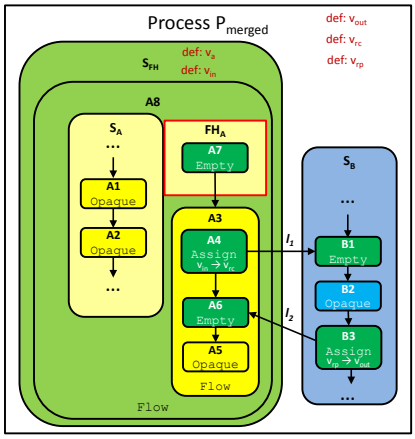

Figure 5. Merged Process Model with outfactored Fault Handler Logic

A scope S is given that has several fault handlers $FH^1{}_S$ to $FH^n{}_S$ represented by `catch` or `catchAll` blocks. After asynchronous and synchronous merges were performed for each fault handler $FH^i{}_S$, it is checked if it contains activities that are target of a link originating from outside of $FH^i{}_S$. If this is the case for at least one fault handler $FH^i{}_S$, a new `scope` S_{FH} is created that contains a `flow` activity. S_{FH} is required to ensure that the activities have still access to the data context of the fault handler they were moved from (see below). The `flow` activity within S_{FH} serves as container for the scope S and its fault handlers. Then for each fault handler $FH^i{}_S$ with an incoming link its root activity $root^i{}_{FH}$ is moved to the scope S_{FH} and replaced by a new `empty` activity $e^i{}_{FH}$. Between $e^i{}_{FH}$ and $root^i{}_{FH}$ a control link is created where $e^i{}_{FH}$ acts as source. This ensures that the fault handling logic contained in $root^i{}_{FH}$ is always performed when $FH^i{}_S$ is activated. All fault handlers that have no incoming control flow link remain unchanged. Figure 5 shows the merged process model P_{merged} where the root activity $A3$ of the fault handler was factored out. The new `empty` activity $A7$ acts as source for the control link pointing to $A3$.

The activities and control flow links that were moved out of the fault handler cannot access the local data context (local variable, partner link declarations etc.) that was either defined in scope S or in the `catch` block anymore as they reside in the parent scope S_{FH}. To make the data context visible again, the defined data have to be moved to the parent scope S_{FH} of the fault handling activities. In Fig. 5 this affects the variables v_a and v_{in}, hence, they are lifted from scope S to S_{FH}.

The solution described above keeps the control flow relations between the (non-communicating) basic activities as defined in the choreography. If S_A in Fig. 5 completes successfully the fault handler FH_A is uninstalled and all links originating from activity $A7$ are marked as dead. Thus, activity $A3$, its child activities and also all activities within S_B are not activated either (dead path elimination). This is the same behavior as modeled in the choreography. In case a fault is thrown and caught by FH_A activity $A7$ is executed and its outgoing links are activated. This causes the former root activity of FH_A $A3$ and its children to be executed. This in turn causes all activities in S_B to be performed. When S_A completes successfully but S_{FH} was not completed yet by the process engine (which

usually happens immediately after S_{A} was completed) and S_{FH} terminated due to an error in its parent `scope` the default termination handler compensates all successfully executed child `scopes` of S_{FH}. In case a fault is thrown during the execution of the fault handler FH_{A} this fault is simply thrown to its parent `scope`. Also this behavior is kept as S_{FH} does not catch any fault, i. e., it simply rethrows the fault to its parent `scope` that used to be the parent scope `scope` of S_{A}. This also happens when S_{A} throws a fault that is not caught by fault handler FH_{A}.

5 Related Work

Compared to many other techniques that merge processes that are semantically equivalent such as different variants of the same process, we aim to merge collaborating processes. Mendling and Simon [9] propose for instance an approach where semantically equivalent events and functions of Event Driven Process Chains [12] are merged. Küster et al. [5] describe how change logs can be employed to merge different process variants that were created from the same original process.

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 [7] do not generate BPEL scopes out of Petri nets, even if the formal model of Lohmann [6] generates a Petri net representation of BPEL `scopes`.

6 Conclusion and Outlook

In this work we extended the process consolidation approach presented in [13] and [14]. We have shown, how to isolate the activities of the different partners from each other by using BPEL `scopss` and we also extended the asynchronous and synchronous merge operations to reduce the number of control flow links that may be created during the consolidation operation. The main contribution of this work is a technique to merge process models that interacted via fault handlers before they were merged. To satisfy the constraint that no control links must point into a fault handler we have shown a technique to factor the fault handling activities out of the handler.

In future works we also have to propose a way to merge process models that interact via compensation handlers and event handlers. This is even more challenging as they allow neither inbound nor outbound control flow links. Another issue we have to address is that our current merge operations create process models that violate the *peer-scope-dependency* rule. Basically, this rule states that two `scopes` enclosed within the same parent `scope` must have no cyclic control-flow dependencies, otherwise the compensation order of these `scopes` cannot be determined. However, in practice this rule is not enforced by engines such as the Apache ODE[2] or BPEL-g[3].

[2] `http://ode.apache.org/`
[3] `http://code.google.com/p/bpel-g/`

In this paper, we informally argued that the consolidation approach is correct. A first approach to provide a more formal validation has been presented in [14]. Our ongoing work is to evaluate the Petri net formalizations with respect to formal foundations for our merging approach.

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

References

1. Barros, A., Dumas, M., ter Hofstede, A.: Service Interaction Patterns. In: BPM. Springer (2005)
2. Decker, G., Kopp, O., Leymann, F., Weske, M.: Interacting services: From specification to execution. Data & Knowledge Engineering 68(10), 946–972 (Apr 2009)
3. Decker, G., et al.: Non-desynchronizable Service Choreographies. In: ISCOC 2008
4. Khalaf, R., Roller, D., Leymann, F.: Revisiting the Behavior of Fault and Compensation Handlers in WS-BPEL. In: OTM 2009
5. 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)
6. Lohmann, N.: A Feature-Complete Petri Net Semantics for WS-BPEL 2.0. In: WS-FM'07: Web Services and Formal Methods, 4th International Workshop (2007)
7. Lohmann, N., Kleine, J.: Fully-automatic Translation of Open Workflow Net Models into Simple Abstract BPEL Processes. In: Modellierung. Gesellschaft für Informatik e. V. (2008)
8. Lohmann, N., Kopp, O., Leymann, F., Reisig, W.: Analyzing BPEL4Chor: Verification and Participant Synthesis. In: WS-FM'07: Web Services and Formal Methods, 4th International Workshop (2007)
9. Mendling, J., Simon, C.: Business Process Design by View Integration. In: BPM Workshops. Springer (2006)
10. OASIS: Web Services Business Process Execution Language Version 2.0 – OASIS Standard (2007)
11. Object Management Group (OMG): Business Process Model and Notation (BPMN) Version 2.0 (2011), OMG Document Number: formal/2011-01-03
12. 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)
13. 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)
14. Wagner, S., Kopp, O., Leymann, F.: Towards Verification of Process Merge Patterns with Allen's Interval Algebra. In: ZEUS. CEUR, Bamberg (2012)

Business Process Mining for Collaborative Service-Oriented Systems – "Duality" of Process Representations and the Need for Statistical Treatment

Jörg Becker, Dominic Breuker

Department of Information Systems / ERCIS at the University of Muenster, Germany
Leonardo-Campus 3, 48149 Muenster, Germany
{becker,breuker}@ercis.uni-muenster.de

Abstract. Service-oriented systems are deployed by companies to support business processes, especially in inter-organizational collaborative settings. Process mining provides techniques to visualize and understand the emergent behavior in such systems based on data, as compared to what employees believe it is like. In highly unstructured settings however, these techniques have to deal with incomplete data, which still is a demanding challenge. In order to address it, we propose an alternative "dual" interpretation of mining results and outline the theoretical basis upon which process mining techniques could be extended and modified to deliver such results.

Keywords : Service-Orientation, Business Process Mining, Incompleteness

1 Motivation

With today's pressure to rapidly adapt to highly dynamic business environments, collaborations between different business partners have to be set up fast and in a flexible manner. Ad-hoc formation of virtual teams is an increasingly observable collaboration pattern. Technology can be considered one of the main facilitators of this development [1]. Service-oriented systems are perceived as a particularly suitable means of implementing collaborative services virtual teams rely on, including knowledge and resource sharing as well as communication and interaction [2].

For companies, it is important to be aware of their work practices in order to manage and align their activities. Business processes, codified within models, constitute a popular concept to do this [3]. Creating adequate business process models though is typically a laborious task as comprehensive interviews with process participants have to be conducted in order to identify and model the processes' structure [4]. Process mining techniques constitute a data-driven alternative. They allow analyzing business processes as they actually take place, provided that event data can be obtained from information systems involved in the processes' execution [5]. Consequently, process mining techniques could be applied to mine collaborative service behavior and to create business process models of what is going on in a company or even in virtual teams spanning multiple organizations.

Traditional modeling techniques such as Event-driven Process Chains (EPC) or the Business Process Model and Notation (BPMN) are successful in representing business processes of repetitive and well-standardized nature. Unstructured processes though are seen as hardly amenable to traditional process modeling due to uncertainty regarding their outcome as well as the steps and resources needed to produce it [6]. With process mining techniques being designed for processes of the first kind, the question is to which extent they are applicable to unstructured collaborative processes in service-oriented systems that belong more to the latter.

2 Related Work

Research investigating process mining to the field of services includes technical aspects such as logging in service-oriented architectures [7] but also case studies, e.g., about using process mining in the IBM WebSphere environment [8]. Challenges arising in this context include correlations between related process instances as well as restricted service behavior due to context [9]. Other challenges tackled in the literature include identifying events belonging to the same process instance [10].

However, these works target well-structured processes. With respect to collaborative service-oriented systems, little research has been done. Incompleteness, describing a situation in which a mining algorithm is provided with a dataset not including all necessary information, is a challenge process mining techniques must deal with. It is of particular importance in collaborative, unstructured settings [11]. The obvious reason is that the huge number of possibilities for performing an unstructured process leaves no hope for obtaining an event log enumerating them all. For this and other reasons, some researchers move away from mining holistic process descriptions towards aggregated features (e.g., number of interactions between individuals) [12].

3 Research Outline

The question we want to investigate is if it is possible to adapt process mining techniques in a way such that they can be applied in settings in which event logs are expected to be far from complete. Naturally, we cannot expect to generate an exact description of the underlying real-world process. This raises the question what any process model generated on an incomplete dataset is supposed to represent. A possible answer can be obtained through the following line of reasoning.

The normal way of thinking about processes models is that they specify the allowed behavior of a system. As an example, consider a simple process in which activity A is performed first, and then either activity B or C is performed after that, which both terminates the process. This *positive view* defines the process as a set of possibly observable instances: $\{AB, AC\}$. In such a setting, it makes sense to apply an algorithm to learn this allowed behavior. In a dual way though, one could equivalently think of a *negative view* that defines the process as the set of unobservable instances. In the example above, $\{AA, BA, BB, BC, CA, CB, CC, \dots\}$ would be this set. While this set will be huge for highly structured processes, the opposite might be the case for

unstructured ones. For this reason, algorithms searching for disallowed behavior in unstructured processes might use scarce data more efficiently.

But how can an algorithm infer disallowed behavior if only allowed behavior is observed in event logs? The answer is delivered by statistics. If direct negative evidence is unavailable one can work with indirect negative evidence instead, provided one is willing to assume a suitable statistical model. To illustrate this, consider the example of an unfair coin always showing heads when tossed. This could be inferred directly from the information that observing tails is the impossible event. Alternatively, one could observe that heads occurs surprisingly often, with each additional heads increasing the confidence that tails is a highly unlikely event. Statistical tests provide an established theoretical framework to incorporate such reasoning into process mining algorithms. Investigating how this can be accomplished is our approach to developing process mining techniques applicable to unstructured, collaborative service-oriented systems.

References

1. Lipnack, J., Stamps, J.: Virtual teams: People working across boundaries with technology. John Wiley & Sons, New York (2000)
2. Jerstad, I., Dustdar, S., Thanh, D. V.: A service oriented architecture framework for collaborative services. In: 14th IEEE International Workshops on Enabling Technologies Infrastructure for Collaborative Enterprise (WETICE'05). Linköping, Sweden (2005) 121–125
3. van der Aalst, W.M.P., Hofstede, A.H.M. ter, Weske, M.: Business Process Management: A Survey. Lecture Notes in Computer Science 2678 (2003) 1–12
4. Kettinger, W.J., Teng, J.T.C., Guha, S.: Business Process Change: A Study of Methodologies, Techniques, and Tools. MIS Quarterly 21(1) (1997) 55–80
5. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer, Berlin / Heidelberg (2011)
6. Seidel, S., Müller-Wienbergen, F., Rosemann, M.: Pockets of Creativity in Business Processes. Communications of the Association for Information Systems 27(1) (2010) 415–436
7. Dustdar, S., Gombotz, R.: Discovering web service workflows using web services interaction mining. International Journal of Business Process Integration and Management 1(4) (2006) 256–266
8. van der Aalst, W.M.P., Verbeek, H.M.W.: Process Mining in Web Services: The WebSphere Case. IEEE Data Eng. Bull 31(3) (2008) 45–48
9. van der Aalst, W.M.P..: Service Mining: Using Process Mining to Discover, Check, and Improve Service Behavior. IEEE Transactions on Service Computing (2012)
10. Motahari-Nezhad, H.R., Saint-Paul, R., Casati, F., Benatallah, B.: Event correlation for process discovery from web service interaction logs. The VLDB Journal 20(3) (2011) 417–444
11. van der Aalst, W.M.P.: Exploring the CSCW spectrum using process mining. Advanced Engineering Informatics 21(2) (2007) 191–199
12. Truong, H.L., Dustdar, S.: Online Interaction Analysis Framework for Ad-Hoc Collaborative Processes in SOA-Based Environments. Transactions on Petri Nets and Other Models of Concurrency 2 (2009) 260–277

Improving Process Monitoring and Progress Prediction with Data State Transition Events

Nico Herzberg and Andreas Meyer

Hasso Plattner Institute at the University of Potsdam
{nico.herzberg, andreas.meyer}@hpi.uni-potsdam.de

Abstract. Monitoring business processes during their execution is one important aspect of business process management. Process monitoring requires observed events, which are recorded in information systems, to reason about, amongst others, process progress. Especially in manual executing process environments, the observed events are most likely sparse. Therefore, we introduce an approach increasing the number of observed events by capturing data state transition events, which occur after successfully writing a data object during process execution.

1 Introduction

In the field of business process management, monitoring is used to observe process behavior and probably to react upon events as well as to predict upcoming process steps during process execution. Processes automated by information systems, i.e., process engines, can be monitored very well because the system usually provides logging capabilities and therefore, the progress is easily recognizable. In contrast, in environments with processes to be mainly executed manually, such as in health care, many occurring events are not captured. Thus, event information about such processes is incomplete and exact proposition about progress, e.g., by applying the concept of event monitoring points [1], is not possible. An event monitoring point is correlated to certain events captured by an IT system connected to a specific event data source, e.g., a database or a bar code scanner, and informs when certain states transitions as, for instance, enable, begin, or terminate, occur in a process activity. Probabilistic means, as for instance explained in [4], can provide an indication about process progress but are an approximation only.

Nevertheless, progress recognition of the complete process and an holistic view on the process are goals in manual executing process environments also. Introducing additional documentation and logging activities during process execution can be a solution, however, this is not intended, because it creates additional effort to the process participants, e.g., the doctors and nurses. Therefore, we will introduce an approach utilizing events from data object creation or modification to increase the number of observable events used for process monitoring and prediction of process progress. We call them *data state transition events*. After the write of a data object in a specific data state, we can assume this data object to be existent. Besides arguing about process progress, the registration of

events helps in deciding about future process execution and the chance of proper completion of a process or the reachability of a specific activity. Additionally, the approach can be utilized to identify incorrect behavior similarly to what is done in the field of data conformance, e.g., [3].

2 Approach

The presented approach enables insights into process execution based on information about a data object. Data objects and their life cycles are the basis for this approach. A data object life cycle (OLC) is represented by a Petri net, where a place describes a data state and a transition represents a data state change from the preceding to the succeeding one. An OLC specifies all allowed data state changes of the corresponding data object. As an example, we refer to an *Invoice* with data states *created*, *sent* and *paid*, whereat these states can be reached in that order only, see Fig. 1 – Data Object Life Cycle Level. Based on this OLC, those transitions are selected that can be monitored with events. We refer to an event as a happening in a particular *point in time* at a certain *place* in a certain *context* that is represented in the IT system landscape, see Fig. 1 – IT Level. The observable data state transitions of the OLC have to be linked to the events that provide the information about the triggering of the transitions. This link is provided by a binding, see Fig. 1 – Event Level, of the particular transition of the OLC to an implementation that extracts the necessary information about the events from a data source including the correlation to a specific data object instance during runtime. For the binding, we use the same technique as presented in our earlier work [1]. In our example, we assume that sending the invoice and receiving the payment can be observed by retrieving the relevant data from a database.

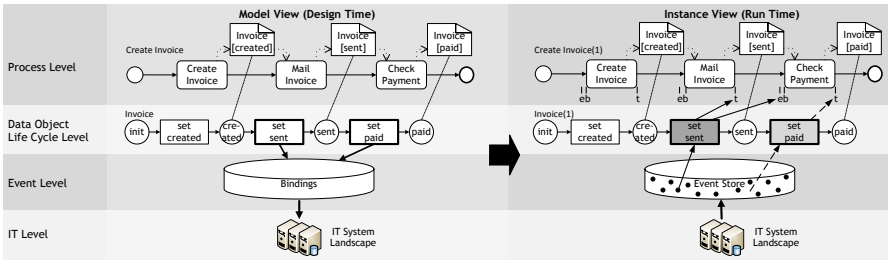

Fig. 1. Scenario describing the approach at design time and run time

During the design of several process models, the data objects can be assigned to particular activities, whereas the activities either read a data object in a certain data state, create a data object in a certain data state, or transfer a data object from one data state to another. We assume that the usage of the data object in a process correlates correctly to the OLC, i.e., the process only uses the specified data states and their transitions as shown in Fig. 1 – Process

Level. Nevertheless, not every possible data object state in the OLC needs to be reflected in the process model.

Based on the assignment of the data objects to the activities, it is possible to provide information about the process execution during runtime. We assume that an activity is enabled if the activity could be executed because of the control flow specification and if the input data object is available in the required data state. Furthermore, we assume that an activity is terminated as soon as the output data object is present in a certain data state. Information about process execution can be provided for activities, which consume or provide data objects in certain data states whose state transition is observable.

Referring to our example, the termination of activity *Create Invoice* and the enablement of *Mail Invoice* cannot be monitored with information about the data object *Invoice* only, whereas the completion of the second activity as well as the enablement of the third activity of the process model can be tracked on IT Level. This is visualized by the edges to the *Bindings* database in Fig. 1 (left part). At design time, the data state transitions to *sent* and *paid* are marked as observable (bold outline). This maps to the aforementioned monitoring capabilities at run time. At run time, the progressing in the process can be recognized by data state transition events occurring in the event store. In Fig. 1 (right part), an event for sending *Invoice(1)* (data object with corresponding instance id) is observed in the event store (solid edge), but there is no event available for the payment although it is expected to happen some time (dashed line). For monitoring process details not observable by the means of data manipulation, the introduced approach can be combined with other approaches, e.g., [1].

For simplicity reasons, we described the approach assuming at most one input data object and at most one output data object per activity. However, the approach is also valid for many input and output data objects.

3 Related Work

In [1], a framework is presented that describes the definition of so-called event monitoring points in business process models based on their activities' states. This work builds the basis of the approach discussed in this paper. As process mining techniques [5], especially conformance checking, are relying on event logs that are complete with respect to the underlying process model, the presented approach could help to enrich existing event logs especially in manual executing process environments and extend them to complete event logs. [3] checks for conformance between process models and data objects at design time while [2] ensures compliance to regulations by restricting the design of artifact-centric, i.e., data-centric, processes. In contrast to these works, our approach targets on run time compliance to data objects.

4 Conclusion

The presented approach uses information about data objects and their data states resp. data state transitions from process models to enable process monitoring.

The observable data about the transition of a data object from one data state to another provides further insights about the process execution. Thus, a more detailed process monitoring can be assured.

References

1. N. Herzberg, M. Kunze, and A. Rogge-Solti. Towards Process Evaluation in Non-automated Process Execution Environments. In *Services and their Composition (ZEUS)*, 2012.
2. N. Lohmann. Compliance by design for artifact-centric business processes. In *Business Process Management*, pages 99–115. Springer, 2011.
3. A. Meyer, A. Polyvyanyy, and M. Weske. Weak Conformance of Process Models with respect to Data Objects. In *Services and their Composition (ZEUS)*, 2012.
4. A. Rogge-Solti and M. Weske. Enabling Probabilistic Process Monitoring in Non-automated Environments. In *BMMDS/EMMSAD*, pages 226–240, 2012.
5. W. M. P. van der Aalst. *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.

Improving Portability of Cloud Service Topology Models Relying on Script-Based Deployment

Johannes Wettinger, Oliver Kopp, and Frank Leymann

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

Abstract Portability is key for services running in the Cloud to prevent vendor lock-in. Today, many Cloud services are portable and can thus be moved from one Cloud provider to another. However, the management of these services is often bound to provider-specific management tooling. Thus, the way of management of a particular Cloud service may completely change when moving it to another Cloud provider. This paper presents concepts to improve the portability of Cloud service topology models that are deployed and managed using scripts. We highlight the challenges of a semi-automatic procedure to generate portable TOSCA-compliant topology model components based on Juju topology model components.

Keywords: portability, service topology, topology model, script-based deployment, Cloud computing

1 Introduction

Reducing the costs of infrastructure and service management is one of the most important aspects of Cloud computing because traditional IT service management is costly. This goal is achieved by automating the whole management of services running in the Cloud. Management of Cloud services is not limited to deploying and decommissioning service instances; it includes several management tasks that need to be performed once a particular service instance has been deployed. As an example, the service instance has to scale up and down depending on the current workload. Today, Cloud providers offer proprietary tooling to automate Cloud service management such as "CloudFormation" and "Auto Scaling" provided by Amazon Web Services[1]. The learning curve is flat because these tools are easy to use. However, when the service is moved to another Cloud provider the management tooling is different. Thus, the service may be managed in a completely different manner. The service itself may be perfectly portable, so it can be moved from one Cloud provider to another. However, this may not be true for the service management. This is why portability is essential for services running in the Cloud, especially when it comes to service management.

[1] Amazon Web Services: http://aws.amazon.com

To achieve management portability, this paper provides two key contributions: (1) an approach to generate standard-compliant topology model components and (2) concepts to improve portability of these generated components.

2 Background

We assume that the structure and management behavior of a Cloud service is specified using a service topology model consisting of several topology model components. As an example, two topology model components may be part of a topology model for deploying and managing a Web application: an "Apache Web server" and a "MySQL database server." A topology model component contains scripts that are typically implemented using a scripting language such as Python or Perl. These scripts realize the management actions that can be performed regarding a service instance of the particular topology model such as deploying and updating its components. We focus on *service deployment* as one of the most important management tasks, based on topology models. Today, there are existing topology model components publicly available that can be used to deploy and manage services in the Cloud. A prominent example is Juju[2]. The community shares more than one hundred topology model components as open source software. Such a component is called a "charm" and can be combined with other "charms" to create a service topology model that can be instantiated and managed in the Cloud. The core of a charm is a set of scripts to enable automated management of a particular service instance. However, these scripts are bound to Ubuntu Linux and thus are not portable. There are standardization efforts going on in the field of model-driven Cloud management that are focusing on management portability: the Topology and Orchestration Specification for Cloud Applications (TOSCA)[3] is an emerging standard supported by a number of prominent companies in the industry such as IBM, SAP, and Hewlett-Packard. TOSCA enables the specification of portable topology models and portable topology model components. However, an ecosystem including an active community sharing topology models and topology model components based on TOSCA is still missing.

3 Generating Standard-Compliant Topology Model Components

One goal of our work is to bring together the standardization efforts of TOSCA enabling management portability with Juju's growing ecosystem and active community. The first step to achieve this goal is outlined in this section: transforming topology model components published by the Juju community to TOSCA-compliant topology model components. Both TOSCA's and Juju's topology models basically specify graphs consisting of nodes and relations between nodes

[2] Juju: `http://juju.ubuntu.com`
[3] TOSCA: `http://www.tosca-open.org`

to define the structure of a Cloud service. In TOSCA, both relations and nodes are explicitly modeled as separate topology model components, whereas Juju specifies nodes as topology model components only. Consequently, two major steps have to be performed: (1) a TOSCA-compliant topology model component has to be generated for each Juju charm; as a result, each node that can be modeled using Juju, can be modeled using TOSCA, too. However, the relations between these nodes cannot be modeled in TOSCA because the corresponding topology model components are missing. (2) Thus, additional TOSCA-compliant topology model components have to be generated for each relation that can be implicitly modeled using Juju.

As an example, for the Juju charms "WordPress application" and "MySQL database server," two corresponding topology model components are generated that can represent nodes in a TOSCA topology model (service topology). For the relation "WordPress application connects to MySQL database server," which can be implicitly modeled in Juju, a separate topology model component is generated that can represent the corresponding relation in a TOSCA topology model.

4 Improving Portability of Generated Topology Model Components

The generated topology model components as described in Section 3 are TOSCA-compliant and thus follow an emerging standard. Service topology models using these components can be deployed and managed using an arbitrary TOSCA engine such as OpenTOSCA[4] or IBM SmartCloud Orchestrator[5]. This is already an improvement of portability because the original topology model components shared by the Juju community can be processed by the Juju engine only. However, the scripts inside the topology model components are still restricting the portability in two ways: (1) the scripts use a set of commands and environment variables that are available on each virtual machine managed by Juju. (2) The scripts are designed to be executed on Ubuntu Linux; as a result, their execution fails on other Linux variants and other platforms.

The first restriction can be compensated by generating wrapper scripts that prepare the execution environment and then call the actual scripts originating in Juju charms. These wrapper scripts receive their input from the TOSCA engine and expose commands and environment variables that are used by the actual scripts. The second restriction is a greater challenge: the generated topology model components have to be refined to further enhance their portability either by improving the existing scripts so they also run on other platforms or by creating and attaching additional scripts to support other platforms. These additional scripts can be created by copying an existing script and semi-automatically adapting it to be executable on another platform. This alternative realizes separation of concerns and thus is the preferred one in contrast to directly modifying an existing script.

[4] OpenTOSCA: `http://www.iaas.uni-stuttgart.de/OpenTOSCA/indexE.php`
[5] IBM SmartCloud Orchestrator: `http://ibm.co/CPandO`

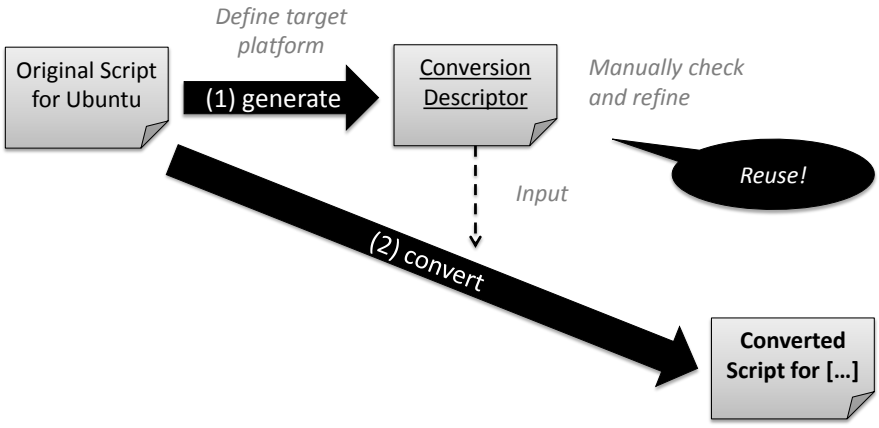

Figure 1. Conversion descriptors as a means of improving the portability of scripts

We are currently designing a semi-automatic, modular, and extensible procedure to convert a script that is implemented for a specific platform to be executed on another one. Figure 1 presents an example to show the basic concept of the procedure: based on the original script that is bound to a specific platform such as Ubuntu Linux and the definition of a particular target platform a conversion descriptor gets generated. It specifies the changes that are required to make the original script executable on a particular target platform such as Red Hat Linux. Then, the generated conversion descriptor gets checked and refined manually because some of the specified changes may be incorrect. Additionally, changes that are actually required may not represented at all. Based on the refined conversion descriptor the actual conversion procedure gets performed to create a new version of the script that is executable on the target platform. The purpose of generating a conversion descriptor is twofold: first, it enables a manual review regarding correctness and completeness. Second, the conversion descriptor can be reused and further refined for future versions of the original script.

The concepts described in Section 3 and Section 4 enable the creation of portable topology models based on TOSCA using the generated and refined topology model components owning a high degree of portability. In future, we focus on reusing these concepts to generate additional topology model components originating in communities of configuration management tools such as Chef[6] or Puppet[7].

Acknowledgments The research leading to these results has partially received funding from the 4CaaSt project part of the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 258862. Further, this work was partially funded by the BMWi project CloudCycle (01MD11023).

[6] Chef: `http://www.opscode.com/chef`
[7] Puppet: `http://www.puppetlabs.com`

Towards Integrating TOSCA and ITIL

Christoph Demont[2], Uwe Breitenbücher[1], Oliver Kopp[1], Frank Leymann[1], and Johannes Wettinger[1]

[1] Institute of Architecture of Application Systems, University of Stuttgart, Germany
{breitenbuecher,kopp,leymann,wettinger}@iaas.uni-stuttgart.de
[2] Informations- und Kommunikationszentrum, University of Stuttgart, Germany
christoph.demont@izus.uni-stuttgart.de

Abstract The integration of low level management functionalities provided by TOSCA and high level processes as defined by ITIL may provide significant improvement opportunities to the application provider as on both levels workflow technology can be employed. In this paper, we present Key Performance Indicator Analysis Plans as first idea how both approaches can be integrated.

1 Introduction and Fundamentals

Topology and Orchestration Specification for Cloud Applications (TOSCA) [3] supports automating Cloud application management by providing a formal method to model the structure of applications as topology and their management functionalities through so called management plans. These plans provide management functionality such as instantiating a service, backup data or scaling out applications. A major issue in this field is the question how to embed these plans in the overall management strategy of the application provider. The correct time to execute a management plan often depends on certain situations which may be expressed as Key Performance Indicator (KPI) [2] values. In this paper we propose an idea showing how to integrate TOSCA with the IT Infrastructure Library (ITIL) [1] to manage the execution of plans in a well-defined but flexible manner by introducing so called KPI Analysis Plans (KPI-APs). Our idea supports using workflow technology on both layers to achieve robust and reliable management. First, we explain TOSCA and ITIL and present our integration idea afterwards before we conclude and give an outlook.

TOSCA is an emerging standard supported by a number of prominent companies in the industry such as IBM, SAP, and Hewlett-Packard. It enables the specification of portable and holistic service models that can be used to automatically instantiate concrete services in the Cloud. A service model basically specifies a service topology (graph) consisting of nodes and relationships between nodes to define the structure of a Cloud service. A node is any kind of component that can be deployed into the Cloud, e.g., a virtual machine or a software component hosted on a virtual machine. First, nodes and relationships need to be specified by defining node types and relationship types. These type definitions include abstract operation definitions and concrete implementation artifacts that are

attached to particular operations. As an example, a "MySQL database" node type may own an "install" and a "configure" operation to set up a MySQL database instance. The operations may be implemented by a set of scripts that are part of the service model as implementation artifacts. Second, the actual service topology is defined by connecting nodes derived from node types using relationships derived from relationship types. An example for a very simple service topology is: "MySQL database [hosted on] Ubuntu Linux [hosted on] virtual machine." The complete service model including all its parts is contained in a Cloud service archive (CSAR) [3]. Complex composite applications can be created by combining several CSARs. The person who is in charge of creating and maintaining the service model may create and add management plans (workflows) to the service model in order to define any kind of management activity. Examples for such an activity are service deployment, database backup, or updating an application component. Typically, these plans are defined using a workflow language such as Business Process Model and Notation (BPMN). Plans interact with operations defined by node types and relationship types to perform actions on a particular service instance.

ITIL is a widely adopted approach for IT Service Management. It provides an accepted practical framework to identify, plan, deliver and support IT services to the business [1]. All necessary processes, the structural organization and the tools to be used are described. One of ITILs objectives is to gain cost benefits for the applying organization. The framework also includes recommendations for the definition of Key Performance Indicators (KPI) which are necessary for measuring performance and condition of IT services. One requirement in cloud computing is the utilization of ITIL for defining processes and to derive measures to quantify economic benefits and impact. For monitoring and controlling IT organization, KPIs are interpreted and measures can be taken to improve process performance continuously. On operative level adequate tasks are derived from improvement measures, which need to be executed at the time defined to finally improve the IT services in a continuous improvement process [4]. approved within a comparison of operating costs that ITIL based application management in cloud computing supports meeting QoS requirements, safes costs and leads to an overall better service quality. Therefore, we utilize ITIL as a management framework to optimize Cloud application management automated by TOSCA.

2 Integration Idea

To control and manage an IT organization close to the optimum with ITIL the organization should operate in a workflow-based process oriented way – in organizational and technical respect. This enables benefitting from the workflow technology's properties such as fault handling, recoverability, and compensation mechanisms on both levels. KPIs support the management in its decisions and are modeled on company level as they are used to monitor and control the enterprise performance. Both, technical and organizational KPIs are derived from defined measures out of the processes and need to be integrated.

KPIs are typically contracted in Service Level Agreements (SLA), are continuously renegotiated in service level management processes, and adapted to changing business requirements. To meet these SLA requirements, the specific and adequate measures in TOSCA need to be defined and recorded and to be submitted to the business processes. TOSCA already works in a process oriented manner. The plans are ideal connecting points to and from ITIL based business processes. To connect TOSCA systematically with ITIL based business processes, the measures coming from TOSCA need to be integrated into KPIs defined in ITIL. This problem is tackled in this paper. Of course, vice versa mechanisms for triggering plans and dynamic plan generation or adaption of existing plans need to be provided in future work.

Thus, the integration of TOSCA and ITIL consists of two cyclic steps influencing each other directly: Analysis and Monitoring of Key Performance Indicators and triggering TOSCA management plans based on these results to react to the measured KPIs. The analysis of an application's KPIs is typically a complex challenge. Although there are monitoring frameworks and tools, the integration of different KPI measurements of different application components is difficult and mainly depends on the overall characteristic of an individual application: For some applications a certain combination of KPIs may be appropriate while other applications would lose their key success factors. Thus, there are two challenges: (i) Integrating different KPIs measurements of different application components in a (ii) customizable but well-defined manner. We argue, that a generic approach is not suitable as especially the ability to customize the integration based on individual requirements is of vital importance as stated above. Thus, to tackle this problem, we introduce an analysis method which is tightly coupled to an individual application but benefits from standardized and reusable artifacts: *Key Performance Indicator Analysis Plans (KPI-AP)*. These plans are TOSCA management plans implemented by the application developer of the application itself following well-defined KPI metrics. They are responsible for measuring high level KPIs of the overall application by orchestrating individual low level KPI measurement operations provided by the application's components and provide a standardized way to integrate reusable artifacts and enable portable KPI measurement as they are contained in CSARs. This allows different application providers to embed the plans in their overall ITIL management processes as their functionality and KPI results are well-defined.

3 Conclusion and Outlook

In this paper we presented a first idea showing how to integrate TOSCA and ITIL by using Key Performance Indicator Analysis Plans. These plans provide a portable and self-contained way for integrating KPI measurements into the application providers overall management. In future work, we want to analyze how to embed requesting TOSCA management fully automated into ITIL processes and how the presented KPI-APs may be employed to achieve this.

References

1. APM Group Ltd: Official ITIL Website (2013), http://www.itil-officialsite.com/
2. Gabler Wirtschaftslexikon: Key Performance Indicator (KPI) (2013), http://wirtschaftslexikon.gabler.de/Definition/key-performance-indicator-kpi.html
3. OASIS: Topology and Orchestration Specification for Cloud Applications Version 1.0. Working Draft 14 (November 2012), http://www.tosca-open.org
4. Rajan, S.S.: Cloud Computing Application Management and Lean ITIL. Cloud Computing Journal (October 2011)

All links were last followed on April 2, 2013

Fast soundness verification of workflow graphs

Thomas M. Prinz

Friedrich Schiller University Jena, Germany
Thomas.Prinz@uni-jena.de

Abstract. This paper shows a new approach to check the soundness of workflow graphs. The algorithm is complete and allows a very good localization of the structural conflicts, i.e. local deadlocks and lack of synchronizations. The evaluation shows a linear processing time in the average and an up to quadratic in the worst case.

Keywords: Soundness, Workflow Graphs, Deadlock, Lack of Synchronization, Localization

1 Introduction

Creating a business process is accompanied by control-flow errors which is shown by various studies [1,2]. These errors evoke wrong or unexpected results and restricts the correct simulation and execution of business processes [1].

We deal with business processes as workflow graphs [3,4,5]. Formally, a *workflow graph* is a directed graph $WG = (N, E)$ at which N consists of *activities*, *fork*, *join*, *split*, *merge* nodes and one *start* node as well as one *end* node. (1) An activity has exactly one incoming and exactly one outgoing edge, (2) a fork or split node has exactly one incoming and at least two outgoing edges, (3) a join or merge node has at least two incoming edges and exactly one outgoing edge, and (4) each node $n \in N$ lies on a path from the start to the end node. Figure 1 illustrates an example of a workflow graph. We use the notion $\bullet n$ to describe all predecessor nodes and $n\bullet$ to describe all successor nodes of a node n.

We call a workflow graph *simple* if and only if the predecessors and successors of each fork, join, split, merge, start or end node are activities, e.g., the workflow graph of Figure 1 is simple. Workflow graphs observed by this paper are *simple* and could have cycles. All splits and merges have an XOR semantic and all forks and joins have an AND semantic. The restriction of simple workflow graphs is not a limitation, because a transformation of a common workflow graph to a simple one is possible in $O(E)$ (see appendix). The execution of such a workflow graph starts in an *initial state* that means the outgoing edge of the start node owns one token.

As structural correctness criterion of workflow graphs, the classical notion of soundness [4] is used in this paper. This notion was introduced on workflow nets. [1] has shown that a workflow graph is a so called free-choice workflow net and the soundness of such a graph corresponds with the absence of *local*

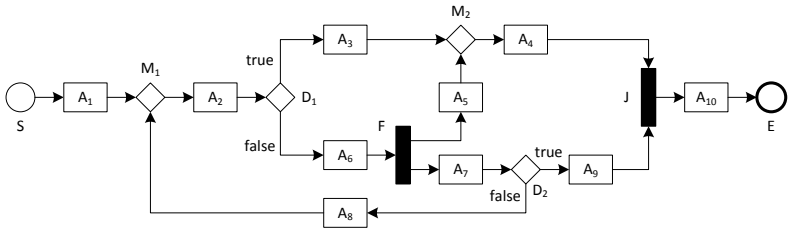

Fig. 1. A workflow graph containing a local deadlock and lack of synchronization

deadlocks and *lack of synchronizations* [3,1]. To introduce both error types, we take a look at the simple example of Figure 1 taken from [1] that includes a lack of synchronization and a local deadlock.

If a token travels the *true* edge leaving the split D_1 in our example, then the token will reach the join J via the upper incoming edge. However, there is no other token that will ever arrive at the lower incoming edge of J. Such a reachable state is called *local deadlock*. Formally, a *local deadlock* is a reachable state s of the workflow graph that has a token on an incoming edge e of a join node such that each reachable state from s also contains a token on e [1].

When a token reaches the *false* edge leaving the split D_1, the token enables the firing of fork F. So one token reaches M_2 and it is possible that the other token will firing the cycle D_2, M_1, D_1, F. The result is a possible state with multiple tokens on the incoming edge of M_2. Such a reachable state is called *lack of synchronization*. Formally, a *lack of synchronization* is a reachable state s containing an edge that has more than one token [1].

The analysis of the absence of lack of synchronizations and local deadlocks is well established by various studies (see [3,4,5,6] for example). However, the processing time of the current approaches is at least cubic or the workflow graphs are critically restricted (acyclic, incomplete or well-formed). The diagnostic information also ranges from no information (fast algorithms) over displaying the failure trace (Petri net based techniques) to detailed information (for restricted workflow graphs). A good overview over such techniques gives the introduction of [6]. Therefore, it exists a divergence of fast or informative algorithms.

The remainder of this paper is structered as follows: Sect. 2 introduces the basic ideas and marginal cases of our new approach. These ideas will be complemented in Sect. 3 following an evaluation of our approach in Sect. 4. Finally, Sect. 5 concludes the paper.

2 Basic idea and marginal cases

Our basic idea is to detect local deadlocks and lack of synchronizations structurally by finding *entry points* which control the firing of a join node. Then we assume a state allowing the firing of such an *entry point*. Further, we find each local

deadlock and lack of synchronization *isolated*. That means, such an error can not occur if a local deadlock or lack of synchronization occurs before. These isolated local deadlocks and lack of synchronizations are called *potential*. The task is to determine entry points to detect *potential LD*s and lack of synchronizations.

Therefore, we introduced so-called *bottle necks*, which are fork or split nodes, e.g., the nodes D_1, F and D_2 of Figure 1. A *bottle* of a bottle neck n_b is a subgraph $G_{n_b} = (N_{n_b}, E_{n_b})$ of the workflow graph such that each path from the start node to a node $n \in N_{n_b}$ contains n_b or a merge node $n_m \in N_{n_b}$. From this it follows that, N_{n_b} contains only a join node n_j if $\bullet n_j \subseteq N_{n_b}$ is valid. If the bottle n_b contains a merge node n_m that has a predecessor node n_p beeing not in N_{n_b}, then no state should be reachable from the initial state which has a token on the incoming edge of n_b and on the edge $(n_p; n_m)$ (see Figure 2). A lack of synchronization is reachable having two tokens on the outgoing edge of n_m.

For example the bottle of D_1 of Figure 1 is the sub graph containing all nodes except S and A_1 and the bottle of F_2 of Figure 3 contains the nodes A_3 and A_4.

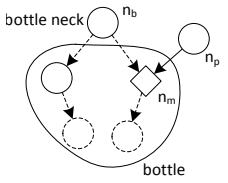

Fig. 2. A bottle that contains a merge node

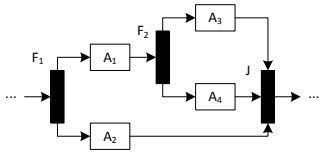

Fig. 3. A sub graph of a workflow graph

After all, an *entry point* of a join node n_j is a bottle neck with a bottle containing n_j, e.g., the entry points of J of Figure 1 are D_1, F and D_2 and an entry point of J of Figure 3 is F_1.

If a join node n_j has no entry point, then there exists no bottle neck which has a bottle containing all predecessors of n_j. That means at least one predecessor of n_j is only reachable from the start node when n_j fires. A local deadlock is reachable.

Lemma 1. *If a join node has no entry point, a potential local deadlock occurs.*

An entry point of a join node is named *immediate* if and only if there exists at least one path from the entry point of the join node containing no other entry point of the join. If we determine all immediate entry points of a join node, then the join node is only reachable over these nodes. So all entry points of J in our example of Figure 1 are immediate, e.g., there is a path (A_5, M_2, A_4) from F to J with no entry point.

Now, we consider an immediate entry point of a join node n_j being a split node n_s. There is a path from n_s to n_j containing no other immediate entry

point of n_j. So if no local deadlock occurs on this path, it is possible that after firing n_s a token reaches exactly one incoming edge of n_j. n_j cannot fire. If we assume that, at this point the entry point or another entry point fires, a token can reach the same incoming edge of n_j over again. So either a local deadlock occurs or a lack of synchronization is possible.

Lemma 2. *If a join node has at least one split node as immediate entry point, a potential local deadlock is reachable.*

We call an entry point of a join node n_j *complete* if and only if no path from the entry point to n_j contains an other entry point of n_j. We assume that, n_j has a non-split immediate entry point n_e which is not complete. We know, there is a path P_1 from n_e to n_j which contains no other immediate entry point. Let us assume that, P_1 contains the edge $e = (x, n_j)$. We know, there is also a path P_2 from n_e to n_j containing another immediate entry point n_i. Because n_i is also an entry point of n_j, there is a path P_3 from n_i to n_j containing also the edge e. If n_e fires and there is no local deadlock on the observed paths, then a token reaches e and the incoming edge of n_i. n_i can also fire and a token reaches also e. A lack of synchronization occurs. This situation is only avoidable if a local deadlock occurs.

Lemma 3. *If a join node has at least one immediate entry point being not complete and not a split node, a potential lack of synchronization is reachable.*

Definition 1 (Well-formed). *The immediate entry points of a join node are called* well-formed *if and only if none of the lemmata 1, 2 and 3 is valid.*

3 Processes

Let us consider all paths from all immediate entry points to exactly one predecessor of a join node n_j (and so to an incoming edge) which contains no entry point of n_j. We merge all these paths to a sub graph $P = (N_P, E_P)$ called *process* of n_j, because (related to a business process) such a *process* can be executed from one entity (like a human or machine). n_j is named *terminator* $T(P)$ of the process, because this node ends the execution of the process. Supplementary, we define a *main process* for generalization. A *main process* is the process containing all nodes of a workflow graph except the start and end node. For example, the processes in figure 1 are $\{A_3, M_2, A_4, A_5\}$ and $\{A_7, D_2, A_9\}$ and the main process is $N \setminus \{S, E\}$. At this point, we need the simpleness of a workflow graph, because every process should have at least one node.

Hence, we observe a join node n_j having only well-formed immediate entry points. If we assume that on the join node n_j with its complete entry points N_{EP} occurs a local deadlock, then we conclude that at least one process P of n_j contains a split node n_s with $n_s\bullet \not\subseteq N_P$. On a local deadlock is at least one token on an incoming edge and at least no token on another. We assume P is the process which contains the predecessor node n_p of n_j and (n_p, n_j) bears no token.

One entry point $n_e \in N_{EP}$ has to been fired. There exists a path from n_e to n_p. So if we assume there is no local deadlock on this path, a token can travel from the outgoing edge of n_e to the outgoing edge of n_p. How could a token leave this path? The only way is a split node which lies on the path and has a successor node with no path from this node to n_p. That means this successor node is not in the process. So our assertion is correct.

Lemma 4. *If a process $P = (N_P, E_P)$ contains a split node n_s with $n_s\bullet \not\subseteq N_P$ a* potential *local deadlock is reachable.*

For example the process $\{A_7, D_2, A_9\}$ contains the split node D_2 which satisfies the conditions to cause a potential local deadlock.

Processes have nice properties on closer inspection. If a process P contains a join node n_j, each process P' which ends in n_j is a sub graph of P called *sub process* (written $P' \subset P$) and P is called *super process* respectively. This results on the definitions of processes and entry points. Furthermore, the process P is called *active* on a node n if there is no sub process of P containing n. We called it *active*, because an entity processes P *actively* regarding to a business process engine, while *passive* processes have to wait.

Each lack of synchronization starts in a fork node which produces two tokens joining the same edge later. So the detection of lack of synchronizations is the detection of processes which are active on the successor nodes twice or more. Because of the definition of an active process, there is no join node on the path to its terminator splitting the process. It has to be a merge node. This results in a lack of synchronization (see Figure 4 and 7). But an active process could be hidden by another active process (see Figure 5 and 6). If a sub process P' of a process P starts in a fork node n_f, we can assume a (maybe infinitesimal) short time between the firing of n_f and the activation of P', in that P is also *active*. We name these hidden active processes simply *hidden active*. Altogether we can express the following lemma.

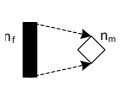

Fig. 4. Case 1

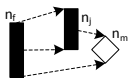

Fig. 5. Case 2

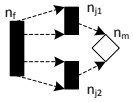

Fig. 6. Case 3

Fig. 7. Case 4

Lemma 5. *Let M be a multi set containing all active and hidden active processes of all successors of a fork node. If M contains a process twice or more, then a potential lack of synchronization is reachable.*

For example the fork node F in our example contains the active main process twice in its successor nodes (once active and once hidden active).

Now, we have determined all potential local deadlocks and lack of synchronizations by handling marginal cases and processes. Thus, it is possible to exactly localize them by the entry points, the terminators, the processes and split and fork nodes. Because of the fact, that a potential local deadlock is not only reachable from the initial state when another potential local deadlock or lack of synchronization occurs, and a potential lack of synchronization is not only reachable when a potential local deadlock occurs, we can determine the soundness of a workflow graph. Altogether we can formalize the following lemma.

Theorem 1. *A workflow graph is* sound *if and only if it contains neither a potential local deadlock nor a potential lack of synchronization.*

The basic algorithms to determine the entry points, immediate entry points and processes are in the appendix. It is reasonable that $|E| \leq 2 * |N|$ is valid for simple workflow graphs. Altogether, the processing time is in worst case quadratic to N.

4 Evaluation

We have implemented the algorithm in *Java* and stopped the soundness verification when the algorithm found the first potential error. It followed the approach of [1]. The benchmark was taken from `http://www.service-technology.org/soundness`, is splitted in 5 libraries (A, B1, B2, B3 and C) and was also used by [1]. The input was a *PNML file* which was parsed and transformed from a Petri net to a workflow graph. So we could compare it with other tools like LoLA (`http://www.informatik.uni-rostock.de/tpp/lola/`).

Our runtime environment was a system with a 64 bit Intel® Core™2 CPU E6300 processor and 2 GB main memory. The system rans a Linux 3.1.0-1.2-desktop x86_64 kernel and an Oracle OpenJDK JRE 1.6.0_22. Because of the hot spot compiler of the JRE, we created a startup as long as the optimizing system needed to optimize the most hot methods. We ran each of the 5 libraries 10 times, removed the two best and worst results and calculated the average run time. Furthermore, the number of nodes, edges and explored nodes were determined. We chose LoLA for comparison and ran it on the same machine like our algorithm and subtracted the read and build time of the petri net. Altogether, we determined only the validation time and no transformations. Table 1 shows the results.

Compared to the results of LoLA we determined the same sound and unsound processes. So it accompanies with the correctness of our approach.

The results showed, the number of edges grows linear with the number of nodes and that the number of split, fork and join nodes is less ($< 20\%$). It accompanies with the nearly linear number of max. approx. 11-times inspections of nodes. So the runtime results showed that our approach is *nearly linear* in the worst case. This is also represented by the analysis times of the libraries. They were approx. 150-times faster as the times of LoLA in average. Although,

Library:	A	B1	B2	B3	C
Processes/Sound	282/152	288/107	363/161	421/207	32/15
Avg./Max. $\|N\|$	84/292	82/402	83/437	93/501	136/567
Avg./Max. $\|E\| \setminus \|N\|$	1.3/1.9	1.3/1.9	1.3/1.9	1.2/1.7	1.1/1.3
Avg./Max. $\|vis.N\| \setminus \|N\|$	3.6/7.1	3.4/7.9	3.5/10.8	3.4/7.7	2.4/7.1
Analysis time [ms]	16.4	15.4	20.7	28.4	1.7
Analysis time LoLA [ms]	2373.0	2395.9	3126.1	3651.3	303.8
Per process avg./max. [ms]	0.06/0.28	0.06/0.36	0.06/0.47	0.07/0.69	0.06/0.31
Per process LoLA avg. [ms]	8.5	8.4	8.7	8.7	9.5

Table 1. Results of the benchmark evaluation

the comparison was difficult regarding the different implementation technologies. Altogether, a single process took always less than one millisecond to be validated.

To verify these results we added our complete algorithm to the *Activiti BPMN 2.0 designer* (`http://activiti.org`). When drawing a workflow graph-like business process, the plugin verifies the process in-time and visualizes all errors. It underscores the efficiancy of our algorithm and the very good error localisation.

5 Conclusion and Outlook

In this paper we presented and evaluated a new approach to detect structural conflicts in workflow graphs, i.e., the soundness. The approach and the resulting algorithm found all structural conflicts and were able to localize them. We showed a soundness verification is possible during the drawing of a business process. This is possible, because the algorithm has an up to quadratic worst case processing time.

The main issues for future work are to present a complete proof of our approach, solving the soundness for OR-splits and OR-merges, to use the introduced processes to verify the operability of workflow graphs and to transform data-extended workflow graphs into a CSSA-based form.

Acknowledgments

This paper was written in the context of the SimProgno research project (support code: 01IS10042B) funded by the German Federal Ministry of Education and Research. Special thanks to Wolfram Amme for discussing the ideas of this paper.

References

1. 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) (May 2011) 448–466
2. Mendling, J.: Empirical studies in process model verification. In Jensen, K., Aalst, W.M., eds.: Transactions on Petri Nets and Other Models of Concurrency II, Berlin, Heidelberg, Springer-Verlag (2009) 208–224

3. Sadiq, W., Orlowska, M.E.: Analyzing process models using graph reduction techniques. Inf. Syst. **25**(2) (April 2000) 117–134
4. Aalst, W.M.P.v.d., Hirnschall, A., Verbeek, H.M.W.E.: An alternative way to analyze workflow graphs. In: Proceedings of the 14th International Conference on Advanced Information Systems Engineering. CAiSE '02, London, UK, UK, Springer-Verlag (2002) 535–552
5. Eshuis, R., Kumar, A.: An integer programming based approach for verification and diagnosis of workflows. Data Knowl. Eng. **69**(8) (August 2010) 816–835
6. Vanhatalo, J., Völzer, H., Leymann, F.: Faster and more focused control-flow analysis for business process models through sese decomposition. In: Proceedings of the 5th international conference on Service-Oriented Computing. ICSOC '07, Berlin, Heidelberg, Springer-Verlag (2007) 43–55

A Appendix

Algorithm 1 Transforming a workflow graph into a simple one with processing time: $O(E)$.

Input: A workflow graph $WG = (N, E)$
Output: A simple workflow graph $WG' = (N', E')$

for all $e = (n_s, n_t) \in E$ **do**
  $N' \leftarrow N' \cup \{n_s, n_e\}$
  if n_s and n_t are not activities **then**
    $N' \leftarrow N' \cup \{n_a\}, n_a$ is a new activity
    $E' \leftarrow E' \cup \{(n_s, n_a), (n_a, n_t)\}$
  else
    $E' \leftarrow E' \cup \{e\}$

Algorithm 2 Determining a bottle for one bottle neck with processing time: $O(N + E)$.

Input: A fork or split node n of a workflow graph $WG = (N, E)$
Output: A set N_B of nodes of the bottle of n

for all $n_s \in n\bullet$ **do**
  determineBottle(n_s)
function DETERMINEBOTTLE(n_c)
  if $n_c \notin N_b$ **then**
    if n_c is a join node **then**
      if $\bullet n_c \subseteq N_B$ **then**
        $N_B \leftarrow N_B \cup \{n_c\}$ and mark n as entry point of n_c
        for all $n_s \in n_c\bullet$ **do**
          determineBottle(n_s, N_B)
    else
      $N_B \leftarrow N_B \cup \{n_c\}$
      for all $n_s \in n_c\bullet$ **do**
        determineBottle(n_s, N_B)

Algorithm 3 Determining a subset of immediate entry points and one process of a join node respectively with processing time: $O(N + E)$.

Input: Predecessor node n_p of a join node n_j and the entry points N_e of n_j
Output: Process $P = (N_P, E_P)$ and a subset of immediate entry points N_i of n_j

```
1: determineProcess(n_p)
2: function DETERMINEPROCESS(n_c)
3:     if n_c ∉ N_P then
4:         if n_c ∈ N_e then
5:             N_i ← N_i ∪ {n_c}
6:         else
7:             N_P ← N_P ∪ {n_c}
8:             for all n_p ∈ •n_c do
9:                 determineProcess(n_p)
```

Detecting Interoperability and Correctness Issues in BPMN 2.0 Process Models

Matthias Geiger and Guido Wirtz

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

Abstract. Although BPMN 2.0 is an international standard widely used in practice, interoperability of process models is still an issue. Even between tools and engines claiming to be BPMN compliant the model exchange is often complicated or impossible as the tools produce incorrect model representations or do not support the standardized BPMN serialization format. In this position paper we present reasons for interoperability issues and show why defining a set of constraints derived from the standard is crucial to fix an important subset of those issues. We are currently developing a tool which can check this set of rules automatically.

Keywords: BPMN 2.0, Interoperability, Correctness, XML Serialization, Standard compliance

1 Motivation

Since its official release in January 2011 the Business Process Model and Notation (BPMN) [8] is used more and more in academia and by practitioners alike. The variety of BPMN models spreads from simple workflow descriptions consisting of only a few sequential tasks for illustration purposes to complex models including data modeling and calls to existing software systems in order to be executed on BPMN compliant process engines. For the former process models, which are often drawn by hand or using tools like Microsoft PowerPoint, interoperability and correctness are not of major interest. But correct and interoperable models are essential when a process definition should be deployed on a BPMN engine like Activiti[1] or when models are to be exchanged between different tools. Model exchange and refinement is often performed in interdisciplinary teams in which usage of the same used modeling tools cannot be assumed. Therefore the need for a standardized BPMN serialization format to enable model interchange which also ensures "correct" process models is widely accepted[2].

In fact, BPMN version 2.0 [8] introduces such a standardized serialization format based on a XML Schema Definition (XSD). Unfortunately, this serialization format is not used or correctly/fully implemented by most tool vendors and therefore real interoperability is still far from given.

[1] http://www.activiti.org

[2] see for example the "BPMN-I" initiative of Bruce Silver (http://www.brsilver.com/2011/04/05/a-profile-for-bpmn-interoperability/)

2 Reasons for Missing Interoperability

The reasons for incorrect models and therefore interoperability issues are manifold but can be divided into two main groups:

- **Vendor Policy:** The usage of a proprietary serialization format, missing import and (especially) export functionality is often intentionally used by vendors. Either the ability to switch to competitive products is limited (vendor lock-in) or a simpler format is used to comply to the internal meta model. This is especially the case when a tool is not initially designed for BPMN models but an existing process and workflow modeling tool is enhanced to deal also with BPMN models.
  For instance, the BizAgi Process Modeler[3] is not able to handle BPMN models saved in the format proposed by BPMN [8]. The Signavio Process Editor[4] supports importing and exporting BPMN compliant XML-files but heavily uses extension elements which hinders the usage of Signavio models in other tools.
- **Implementation Problems:** Although vendors might be willing to provide interoperable model definitions, actual interoperability is not in place, because the constraints raised by the standard are overlooked, misinterpreted or faultily implemented so that incorrect BPMN models might be exported. Since the official release of BPMN 2.0, various ambiguities, inconsistencies and faults have been revealed[5] which further inhibit the successful usage of BPMN [1].

3 Creating a Standard-based Rule Set for XML Serialization and Checking the Extracted Rules

As vendor policies may not be affected easily, we focus on giving support for implementation issues. In order to observe all rules it is needless to say that an overview of all constraints stated in [8] is essential. Unfortunately, BPMN falls short in providing such an overview.

Sources for constraints in the standard document are the running text, tables, class diagrams and XSD excerpts. The extraction of rules from the latter sources is rather straightforward (e.g., mandatory attributes, cardinalities, value constraints). In contrast to this, constraints in the running text are harder to identify and frequently some interpretation of the text is needed. We worked through the standard document and in a first iteration we derived more than 300 rules.

Moreover, in about fifty cases inconsistencies between the text and/or the class diagrams and the XSD have been revealed. Frequently some attributes of elements are defined as mandatory in the text and the class diagrams, but the

[3] `http://www.bizagi.com/modeler/`
[4] `http://www.signavio.com`
[5] e.g., see the issue tracking list `http://www.omg.org/issues/bpmn2-rtf.open.html`

schema definition marks the same attribute as optional. To give an example: Even the definition for the BPMN root element `definitions` is affected by this problem: Table 8.1 in [8, p. 53] states that the attribute `name` is mandatory. In contrast to that, this attribute is completely missing in the XSD excerpt in Table 8.3 [8, p. 54] and is defined as optional in the normative XSD schema[6].

In order to improve and check the completeness and correct interpretation of our rule set, we are currently cross-checking it with other less extensive collections of BPMN constraints[7] and with the consistence checks integrated in various tools[8].

Based on a consolidated list of rules for BPMN models, it is possible to check whether a serialized process model is consistent to this rule set (and therefore to the standard itself).

Parts of the extracted rules can already be checked by performing a XML schema validation. An example for such rules are value limitations, as for the attribute `gatewayDirection` of a BPMN *Gateway*. Here only the values 'Unspecified', 'Converging', 'Diverging' or 'Mixed' are allowed. This restriction is realized as a XSD *SimpleType* restricting *Strings* (see [8], p.91).

However the overwhelming majority of constraints cannot be covered by schema validation. Specific examples can also be found in the context of BPMN *Gateways*: Gateways have incoming *SequenceFlows* which are realized as `incoming` sub elements that refer to a *SequenceFlow* definition using a `xs:QName` reference. With XML schema validation, a reference to an arbitrary or even non-existent BPMN element would be regarded as correct. Moreover, depending on the value of the attribute `gatewayDirection` the number of incoming and outgoing sequence flows has to be limited in a different manner (see [8], p.91) which is also not checkable by schema validation.

To tackle these issues, we are currently developing a tool to check all extracted rules which are not covered by schema validation yet.

4 Related Work

Academic research mainly concentrates on semantic validation, verification and correctness checks for BPMN models [3, 5, 9], assuming that the BPMN models used already comply to the standard. In contrast to this, we concentrate on issues regarding the serialized form of process models. Hence, in this paper the term correctness refers to compliance to the constraints postulated by the standard [8] and not to semantic correctness.

Closer related to our work is [2] which proposes a meta model and a serialization format for the prior BPMN Version 1.1 [7] including some checks regarding reference existence and leveraging XPath for more sophisticated validations. Due to the major revisions in BPMN 2.0, most parts of this approach are outdated

[6] see `http://www.omg.org/spec/BPMN/20100501/BPMN20.xsd`

[7] e.g., see Bruce Silver: *BPMN Method & Style, 2nd edition*, 2011, p. 135-139

[8] e.g., the itp-commerce modeling tool checks a series of constraints (see: `http://help.itp-commerce.com/index.php?id=81&L=0`)

by now, as BPMN 2.0 provides a standardized XML interchange format. Nevertheless, the proposed usage of XML Technologies like schema validation which is now used in the current version of the standard is able to check some basic rules as stated above.

A good example for the importance and practical benefits of a list of relevant constraints for a process language standard is the Web Services Business Process Execution Language (BPEL) [6]. BPEL provides a list of 95 static analysis rules which cannot be checked using a simple XML schema validation, but should be checked by BPEL engines during the deployment process. Using such a list tailored to BPMN, it is much easier to generate test cases and verify standard conformance such as presented for BPEL in the tool betsy [4].

5 Conclusion and Outlook

The main contribution of our work will be an extensive set of constraints stated in the standard and a tool checking these constraints. These contributions provide a basis for the successful practical usage of BPMN. The rule set and the tool may be used by modeling tool vendors and engine developers to check if their software generates, respectively is able to import and deploy standard compliant documents. It might be used during import stages in order to reject non-compliant models or to benchmark different tools/engines regarding their standard compliance.

As a side effect of our work we are able to report several issues to the BPMN 2.1 Revision Task Force to improve the upcoming version of BPMN.

References

1. E. Börger. Approaches to modeling business processes: a critical analysis of BPMN, workflow patterns and YAWL. *Software & Systems Modeling*, 11:305–318, 2012.
2. M. Chinosi and A. Trombetta. Modeling and Validating BPMN Diagrams. In B. Hofreiter and H. Werthner, editors, *CEC*, pages 353–360. IEEE Computer Society, 2009.
3. R. M. Dijkman, M. Dumas, and C. Ouyang. Semantics and analysis of business process models in BPMN. *Information and Software Technology*, 50(12):1281 – 1294, 2008.
4. S. Harrer, J. Lenhard, and G. Wirtz. BPEL Conformance in Open Source Engines. In *Proceedings of the 5th IEEE International Conference on Service-Oriented Computing and Applications (SOCA'12), Taipei, Taiwan*. IEEE, December 17-19 2012.
5. N. Lohmann, E. Verbeek, and R. M. Dijkman. Petri net transformations for business processes - a survey. *T. Petri Nets and Other Models of Concurrency*, 2:46–63, 2009.
6. OASIS. *Web Services Business Process Execution Language*, April 2007. v2.0.
7. OMG. *Business Process Modeling Notation, v1.1*, January 2008.
8. OMG. *Business Process Model and Notation (BPMN) Version 2.0*, January 2011.
9. P. Wong and J. Gibbons. A Process Semantics for BPMN. In *Formal Methods and Software Engineering*, LNCS, pages 355–374. Springer Berlin Heidelberg, 2008.

A new approach for WS-Policy Intersection using Partial Ordered Sets

Abeer Elsafie, Christian Mainka, and Jörg Schwenk

Horst Görtz Institute for IT-Security, Ruhr-University Bochum, Germany
{abeer.elsafie, christian.mainka, joerg.schwenk}@rub.de

Abstract. WS-Policy is a framework that can be used to describe assertions for web services message exchange. In the context of Service Oriented Architectures and Clouds, where web services are belonging to, machine-to-machine communication is one of its core ideas. When those machines try to apply WS-Policy, mainly two events can occur: First, the machine-exchanged policies have common assertions – there is an intersection. Second, there is no direct intersection and the participants must reach an agreement by minimal adjustments to the policies. This paper introduces a new approach for reaching intersection by computing adjustments to the policies using partial ordering.

Keywords: WS-Policy Intersection, Partial Ordered Sets, Hasse Diagram

1 Introduction

In the field of web services, requirements and capabilities can be described using XML according to the WS-Policy specification [1]. The policies can be applied to the web services message exchange, which is commonly machine-to-machine communication with multiple participants, for assuring security goals. This leads to the need for WS-Policy intersection, a technique used when two or more web services want to communicate and fulfill each others policy. Currently, this approach can only handle the case that intersection within the participating policies exists [2]. Otherwise it fails and the further communication cannot be achieved.

Hence, our motivation is to find a way to make intersection possible even in the case that there is no direct intersection by adjusting one or both party's policy, e.g. by adding some policy aspects. This is achieved by a multi-layer approach: First, every WS-Policy, which can be seen as a set of Boolean terms, is converted into its *disjunctive normal form* ($\mathcal{DNF}$), so that policies are easy to compare and finding matching terms is simple. In the case that there is a match, the decision for the participants is obviously done. If there is no direct intersection, this paper introduces a model for an arbitrary number of parties, that computes these adjustments using partial order sets to enforce policy intersection for all participants.

2 Foundations

2.1 WS-Policy and Policy Intersection

WS-Policy is a framework for describing policies using XML [3,1]. In the context of web services, it is commonly used to specify which parts of a message should be signed or encrypted using WS-SecurityPolicy [4]. The structure of a WS-Policy can be seen as a Boolean term, but written in XML. It consists of an enveloping `<Policy/>` element which can contain arbitrary *AND* (element: `<All/>`) and *XOR* (element: `<ExactlyOne/>`) expressions. For each term, there exists a *disjunktive normal form* ($\mathcal{DNF}$). It is an XOR-junction of propositions derived from the compact form using boolean algebra [5]. Consider the following example, which does not use any XML for simplicity:

$$\mathcal{A}_1 \wedge (\mathcal{A}_2 \oplus \mathcal{A}_3) \overset{\mathcal{DNF}}{=} \underbrace{\mathcal{A}_1 \wedge \mathcal{A}_2}_{\text{Alternative 1}} \oplus \underbrace{\mathcal{A}_1 \wedge \mathcal{A}_3}_{\text{Alternative 1}}$$

From the $\mathcal{DNF}$, one can easily see the *policy alternative*: They are a bundle of assertions which must be fulfilled.

The *WS-Policy Intersection* process identifies compatible policy alternatives included in all parties policies or returns nothing if there are no matches [6]. Two alternatives are compatible, if the sets of included assertions are identical.

2.2 Ordered Sets and Hasse Diagrams

A *partially ordered set* (poset) is a mathematic tool generalizing the concept of arranging and ordering elements. In a poset, there exists a relation between pairs of elements, e.g. the "$\leq$"-relation, so that the elements can be compared. When this relation exists for each possible pair, then the poset is called a chain (or total ordered set). In addition a poset in which no two distinct elements are comparable is called *antichain*.

A *Lattice* is an ordered set where every pair of elements has a *least upper bound* (LUB) and a *greatest lower bound* (GLB). In our approach we assume that the posets are all Lattices.

A *Hasse or Lattice diagram* is a visualization of the finite poset in the form of a drawing, in which nodes are elements of the poset and arrows between related nodes represent the order relation between these elements [7,8]. In the next section we introduce an example providing a detailed overview of the usage of Hasse diagram.

3 WS-Policy Intersection Model

The evaluation of WS-Policy Intersection consists of two main layers as shown in Figure 1:

The *preparation* layer is responsible for converting each policy into its corresponding $\mathcal{DNF}$. This is achieved either manually or using an software-tool [9] and

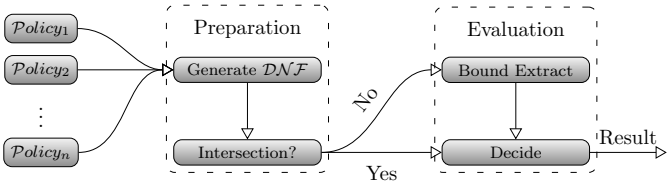

Fig. 1. Evaluating WS-Policy Intersection Model.

is outside the scope of this research. Afterwards, the policy intersection examination unit compares the $\mathcal{DNF}$ policies and forwards the results to the *evaluation* layer. If there is intersection, which means compatible alternatives exist, they are directly forwarded to the *decision making* unit, which chooses the strongest alternative. In the case of no intersection, the *bound extraction* unit takes part. It first identifies all ordered sets, which can be chains like $AES_{128} < AES_{256}$ or anti-chains which cannot be compared, e.g. $Sign_{\mathcal{H}\text{eader}}$ and $Sign_{\mathcal{B}\text{ody}}$. Afterwards, all sets are combined to one Hasse diagram as shown in Figure 2.

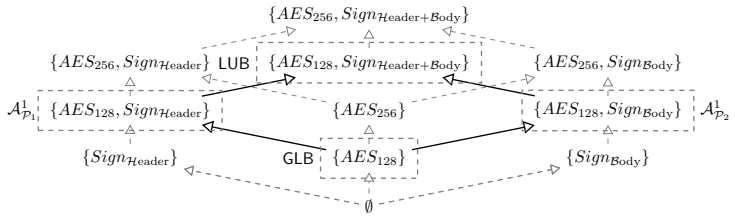

Fig. 2. $\mathcal{S}$igned $\mathcal{P}$art and cryptographic suite combined into one Hasse diagram.

Consider the two policies $\mathcal{P}_1$ and $\mathcal{P}_2$, having the alternatives $\mathcal{A}^1_{\mathcal{P}_1}$ and $\mathcal{A}^1_{\mathcal{P}_2}$ as shown. Obviously, they are not compatible. Using the Hasse diagram, the *least upper bound* (LUB) and the *greatest lower bound* (GLB) can be easily extracted. In general if we consider that the posets used are all lattices, where each two elements have a LUB/GLB, then we can easily use the meet and join for finding these bounds [8]. Finally the bounds are forwarded to the *decision* unit, which has to decide if the GLB or either the LUB should be used.

Note that building the $\mathcal{DNF}$ can drastically increase the size of each policy and thus, building the Hasse diagram might lead to a very large model. Nevertheless, the authors believe to the best of their knowledge that this approach will hold for real examples. We stress that a real implementation and evaluation is needed to prove this.

4 Related Work

Researchers in [10] investigated a mechanism for calculating compatibility of alternatives. An approach for comparing policies and checking compatibility

between alternatives in terms of its assertions to reach intersection is shown in [11] and [12]. Policy reconciliation algorithm, a technique to reach policy agreement between two party communication, is introduced in [13]. Another research using a web ontology language (OWL-DL) is based on the idea that policy assertions and alternatives are mapped in to program classes using OWL to measure compatibility [6]. Our research focuses on how to examine intersection and find solution for policy agreement by means of partial ordering.

5 Conclusions and Future Work

This paper presents a model for WS-Policy Intersection using Partial ordered sets. It is the first solution which is able to (1) handle more than two parties and (2) makes proposals for the case that the policies are not directly compatible.

For future work we plan to investigate a real protocol for multi-party negotiation which is applied to the needs and capabilities of a Web services. Additionally, we will add an implementation to show the practical usability.

References

1. W3C Recommendation, "Web Service Policy 1.5 - Framework," http://www.w3.org/TR/ws-policy/, Sep. 2007.
2. ——, "Web Service Policy Intersection," http://www.w3.org/TR/ws-policy/, Sep. 2007.
3. ——, "Web Service Policy 1.5 - Primer," http://www.w3.org/TR/ws-policy-primer/, Nov. 2007.
4. OASIS Standard, "Web Service Security Policy," http://docs.oasis-open.org/ws-sx/ws-securitypolicy/, Feb. 2009.
5. J. Eldon Whitesitt, *Boolean algebra and its applications*. Courier Dover, 1995.
6. V. Kolovski, B. Parsia, Y. Katz, and J. Hendler, "Representing Web Service Policies in OWL-DL," in *In International Semantic Web Conference (ISWC)*. Springer, Nov. 2005, pp. 461 – 475.
7. W. Strunk, Jr. and E. B. White, *Order Relation*, 3rd ed. Macmillan, 1979.
8. M.-C. van Leunen, *Partial order*. Knopf, 1979.
9. T. A. S. F. Group, "The Apache Software Foundation," https://ws.apache.org/neethi/, Jul. 2012.
10. T. Lavarack and M. Coetzee, "Considering Web Services Security Policy Compatibility," in *The 9th Annual Information Security for South Africa Conference, (ISSA 2010)*. IEEE Press, Aug. 2010, pp. 1 – 8.
11. B. Hollunder, "Domain-Specific Processing of Policies or: WS-Policy Intersection Revisited," in *IEEE 7th International Conference on Web Service (ICWS2009)*. IEEE Press, Jul. 2009, pp. 246 – 253.
12. S. Hudert, T. Eymann, H. Ludwig, and G. Wirtz, "A Negotiation Protocol Description Language for Automated Service Level Agreement Negotiations ," in *Commerce and Enterprise Computing, 2009. CEC '09.* . IEEE Press, Aug 2009, pp. 162 – 169.
13. A. P. P McDaniel, "Methods and limitations of Security Policy Reconciliation," in *2002 IEEE Symposium on Security and Privacy*. IEEE Press, May 2002, pp. 73 – 87.

Challenges in Supporting a Goal-Oriented Enterprise Architecture Analysis

Evellin C. S. Cardoso
Business Process Technology (BPT) Chair
Hasso Plattner Institute, University of Potsdam,
Prof. Dr. Helmert-Str. 2-3, D-14482 Potsdam, Germany
evellin.cardoso@hpi.uni-potsdam.de

Abstract. Enterprise Modelling is a discipline which tries to capture and reason about the distinct dimensions (e.g. structure, strategies and processes) involved in organizations by means of visual models. In this work, we are interested in using Enterprise Architectures to gain an understanding of the enterprise to promote a *goal-oriented enterprise analysis*. This paper describes the current state of art in literature of enterprise architecture and correlated areas and outline research questions that represent the open challenges that must be faced to promote this goal. In particular, the description of literature and the research questions are made in terms of the *languages* that model the enterprise architecture as well as the *techniques* that support architectural analysis.

Keywords: enterprise architecture, enterprise analysis, goal-oriented enterprise analysis, goal-orientation

1 Introduction

Mainly aiming at staying in business or seeking for higher profits, organizations today need support for fostering innovation and boosting production. To achieve both goals, it is crucial that they develop a deep understanding regarding their different dimensions, such as *structure*, *strategies* and *processes*. Such understanding can emerge through the discipline of *Enterprise Architecture* (EA) [1] which tries to capture and reason about the distinct dimensions or viewpoints [1] of the enterprise by means of *visual models*.

Among these viewpoints, the domain of "motivation" has been recognized as an important element of enterprise architectures [2]. Goal modeling allows architects to systematically express the choices behind multiple alternatives and explore new possible configurations for an organizational setting. This is essential for business improvement once changes in a company's strategy and business goals have significant consequences within all domains of the enterprise.

Since changes in all organizational domains must be synchronized with the goal domain, in this work, we are interested in gaining an understanding of how these changes occur in the enterprise by promoting a *goal-oriented enterprise analysis*. The objective of this paper is to describe the current state of art in literature of EA and correlated areas in order to address this research problem. Furthermore, we outline the open challenges to promote this research goal by proposing research questions.

We have noticed during our literature review that such effort in enterprise analysis must initially capture the enterprise architecture in the format of models and subsequently, apply architectural techniques in these models. This observation led us to describe the literature concerning these two aspects, and for this reason: (i) we consider the *languages* to model the EA models and (ii) we also address the *methodologies* and/or *techniques* in EA and related fields that support enterprise model analysis.

The remainder of this paper is structured as follows: Section 2 describes the current state of art regarding the *languages* for modelling the EA (section 2.1) and *techniques* for architectural analysis (section 2.2). Section 3 concludes the paper with an outline of research questions that represent the open challenges that must be faced to promote our research goal.

2 Current State-of-Art in Enterprise Architecture and Related Fields

2.1 Languages for Enterprise Modelling

Architecture at the level of an entire organization is denominated as *Enterprise* A*rchitecture* (EA) and can be defined as "a coherent whole of principles, methods and models that are used in the design and realization of an enterprise's organizational structure, business processes, information systems, and infrastructure" [1]. The first step to use some architectural approach is the documentation of enterprise descriptions through the use of modeling languages.

To cope with the complexity of enterprise architectures, however, the models produced in these modeling languages should capture only the adequate architectural concepts [1]. The right set of concepts that is captured within one model depends on the purpose for which this model is created [1]. In our work, we create our models with a specific concern in mind, that is, we intend to propose a *model-driven technique for goal-oriented enterprise analysis*. With this intent in mind, we set up some requirements that guide our survey among several approaches in literature. We can enumerate these requirements as follows:

1. **Requirement 1 (RQ1).** Since we intend to propose a *model-driven* approach, the proposals must include modeling languages to model the enterprise architecture;
2. **Requirement 2 (RQ2).** Our approach is also *goal-driven*, what makes the inclusion of goal-related concepts an important parameter in our analysis;
3. **Requirement 3 (RQ3).** We have the purpose of providing an *enterprise-wide analysis*, leading us to focus on how the goal domain is integrated with the other viewpoints of the enterprise architecture.

Starting our considerations, the concept of goal is widely used in a number of areas such as Requirements Engineering (RE) [3] [4], Enterprise Modeling [1] [5] and

Business Process Management (BPM) [6] [7] [8] [9]. In particular, we have surveyed only those approaches that provide *modeling languages* (RQ1) that explicitly capture *goal-related concepts* (RQ2). Furthermore, in each approach, we have focused on how these efforts propose to align goals with the other elements of the approaches such as roles, business processes, and so forth (RQ3).

Conclusion of literature review. In order to address RQ2, a careful examination of several areas that we may include RE, EA and BPM revealed that the predominant concept found in literature is the concept of *goal* (or objective) (a definition for the term is provided in section 3). Further, we also may find some *other related concepts*, such as softgoals [4] and strategies [7]. In its turn, these concepts may be related by a number of *relationships* such as AND/OR refinement [3] [4] or conflicts [3]. Concerning the integration of the goal domain with the other elements (RQ3), considering that such goal orientation is adopted by many proposals in a large number of areas; we concluded that the relations of the goal domain with the other concepts in the proposals are dictated by the *applicability* of the proposal in each specific area.

2.2 Techniques and Methodologies for Enterprise Architectural Analysis

Once we have understood which information we should capture in our model (goals, softgoals, strategies, etc.), their relations (e.g. AND/OR refinement) as well as the associations with the other elements of the EA, we need a technique/methodology to use this language in order to promote our *goal-driven enterprise analysis*. This leads us to estipulate the forth requirement:

1. **Requirement 4 (RQ4).** We intend to examine the approaches that provide *model-driven* architectural analysis, in particular, *goal-oriented model-driven* enterprise analysis.

Conclusion of literature review. There is a large body of knowledge that addresses *model-driven* techniques. Some of them can be found in the scope of EA such as [10] [11] [1]. However, we have found that none of them incorporate such goal-orientation (although some of them present goal languages as depicted in previous section). Most of the *model-driven* techniques for process analysis are actually included in the scope of BPM (the majority of them also do not have such goal orientation, but exceptions can be found in [12]). For instance, there is a plethora of *model-driven* methods under the BPM umbrella that are generally denominated as **Business Analytics** methods. Among these Business Analytics methods, we may cite the following areas (that address these methods): **Process (Re)design** (or (re)engineering) [13] [14], **Process Maturity** [15], **Process Controlling** [16], **Process Mining** [17], **Business Activity Monitoring** [18] [8] and **Process intelligence** [19].

3 Ongoing and Future Work

In order to promote a *goal-oriented enterprise analysis*, the current literature has been surveyed as means to understand how the related approaches could support this research goal. After this survey, we have noticed that the proposals are fragmented with respect to the issues that must be addressed in order to solve the problem, and none of them addresses these issues in its totality. This section is aimed at discussing some of these issues, proposing research questions that outline these open issues and depicting how the current approaches meet or fail these requirements. The research questions are drawn also in terms of the *language* and *techniques* mentioned in the previous sections.

3.1 Languages for Enterprise Modelling

Support for modelling goal-related concepts. Which concepts are necessary for such approach (such as goals, softgoal, and strategy)? Which are the relations among these concepts (such as AND/OR refinement and conflicts)?

Goal-related concepts. Goals can be defined as statements that declare desired states for the enterprise setting as well as the reasons and motivations (i.e., rationale) for the existence of the components in the other viewpoints [20], describing a desired state or development of the enterprise [21] [22]. The concept must be characterized with respect to the following attributes:

1. **Description.** Represents the description of the goal. In all the surveyed approaches, goals are informally specified in natural language, although a formal specification is required to enable automated analysis;
2. **Level of abstraction.** Since goal definitions may be stated in a broad scope within the organization, ranging from high-level concerns to the declarations of the values that must be operationalized by business processes, this dimension aims at classifying goals in relation to the level of abstraction. In that respect, some proposals [6] [23] [24] [7] present classifications about goal-related concepts such as mission, vision, strategy and its refinements, although a precise criteria for allocation of goal statements into the categories suggested by the proposals are still required;
3. **Ownership.** Given that an EA models are a joint effort involving several stakeholders, we have to be able to specify the goals' owners. These goals' owner can be individuals (agents) [4] [25] or organizations (including the whole enterprise, organization units or roles) [26] [23] [27] [22];
4. **Hardness.** This dimension distinguishes between soft and hard goals. Hardgoals are defined as goals whose satisfaction can be objectively

defined [4], while softgoals have their satisfaction subjectively evaluated. Some approaches do not recognize this distinction, such as [27] [25] [28];

5. **Priority.** Stipulates an order for the achievement of goals [3] [22] [28];
6. **Deadline.** Represents the maximum point in time that the goal can be achieved [22] [28];
7. **Evaluation type.** Specifies how the satisfaction of the goal must be checked for a given interval of time. In [22], goals have *goal patterns* that are properties that can be checked for a given state/time point or interval in order to evaluate if the goal is satisfied or not (this pattern have types, namely: *achieve*/*cease*, *maintain*/*avoid*, *optimized* (*maximized*/*minimized*/*approximated*)). This proposal builds its definition on [3];
8. **Measurement.** The satisfaction of goals needs to be quantitatively evaluated. This is usually achieved by associating goals with *Key Performance Indicators* (KPIs) [27] [22] [28] [24].

Goal-related relations. Goals can be related through some types of relations. The survey revealed that there are the following types of goal relations: *AND*/*OR decomposition, conflict, influence* [27], *(positive/negative) contribution* and *means-ends* [4].

Alignment of goal-related concepts with the viewpoints of EA. Which are the relations between the goal domain and the other domains of the EA, such as business process, organizational structure domains, etc?

With our analysis of the literature, we have observed that the response of such question is related with the intended *applicability* of the model in the several areas. For instance, in RE and EA, goals are aimed at capturing stakeholders' *requirements* for a target computational system (RE) or an architecture yet-to-be constructed or redesigned (EA), what lead them to be associated with *agents*/*roles*/*stakeholders* [4] [3] [21] [25] or even with organizational units [23] or communities [26]. In these areas, goals statements can also be defined on the basis of *objects/resources* [3] [4] since these resources can be used by the stakeholders in the achievement of goals.

The only two approaches that consider goals as being linked to the normative aspect (*rules*) are the BMM model [23] and the Business Motivation Ontology [24]. Possibly, this can be accounted by the fact that in EA, norms may constrain the achievement of goals within the enterprise setting.

Finally, the majority of the approaches recognize *business processes* as the most important asset responsible for the achievement of goals in organizations such as [5] [4] [29] [30] [26] [23] [21]. Some works in the discipline of BPM have been inspired by this goal-orientation [9] [6] [12], by adding goal-related concepts in order to *overcome the semantic gap* between high-level enterprise's goals and the business processes which are responsible for implementing these goals. Other approaches are intended to provide additional support in *business process reengineering* activities [7] [31] [32]. Furthermore, some proposals appear in the context of BPM using ontologies [28] [24] [33] to promote *semantic interoperability of business processes* at the conceptual level with the other viewpoints of the enterprise.

Concerning this problem of identifying the set of concepts in each viewpoint that have associations with goals (and the nature of these relations), we have already started an effort [34]. We observed this connection is far from trivial and not addressed by any of the aforementioned approaches, requiring us to consider the semantics of goals, the semantics of many other enterprise elements as well as the nature of the relation between goals and these other enterprise elements. As a consequence, we tackled the problem using an *ontological approach* [35].

3.2 Techniques and Methodologies for Enterprise Architectural Analysis

Use of BPM approaches. How BPM methods can be adapted to perform architectural analysis?

Within the BPM approaches, processes can be evaluated with respect to their *structural properties* or the *execution characteristics*. Within the field of **Business Process Reengineering**, the approaches are concerned about guiding the (re)design of processes so that they contain only activities that generate value for the organization (*structural properties* of business processes). They commonly comprise recommended best practices [14] and other informal methods like "classic" reengineering view [13]. Concerning the *execution characteristics* of business processes, three types of analysis can be made [16]: *past analysis* to evaluate what happened in the past (**Process Controlling** [16]), *real-time analysis* to monitor the currently active business processes (**Business Activity Monitoring** [18] [8]) and *predictive analysis* to predict what may happen in the future (**Process intelligence** [19]).

We can argue that BPM methods concentrate in the analysis and optimization of business process models (process viewpoint of the EA). These methods can be considered of great value in our approach, since we may adapt the optimization techniques in the process viewpoint taking the goal viewpoint into consideration.

Further, although there is little support (or inexistent) in BPM methods to address optimizations in other viewpoints of the EA, the Business Process Maturity Model from OMG [15] could be used as an instrument of enterprise analysis, since it enables description of "as-is" enterprise's state, from the perspective of process management maturity [36]. This enterprise description will enable us to gain understanding of the current situation of the enterprise, what ultimately represents our objective of *enterprise analysis*.

Enhancement of enterprise modeling techniques with goal-oriented analysis. How enterprise analysis techniques can be enhanced with goal-oriented analysis?

Current enterprise architectural techniques [1] are able to perform some types of analysis in EA models, such as *functional analysis* and *quantitative analysis*. Although these techniques are very useful for performing an enterprise-wide analysis, they still do not incorporate goal-oriented concepts to perform such analysis and may be used as a starting point in our work.

Adaptation of current goal-oriented analysis techniques. How current goal-oriented techniques from other areas can be used in enterprise analysis?

A first effort into the incorporation of goal-oriented techniques for enterprise analysis is proposed in [37]. The work proposes a quantitative-reasoning based approach to model and simulate feedback loops of goal influences relations in the ArchiMate Motivational Extension language [21] [27]. Although the proposal is very useful in the scope of evaluating goal satisfaction, it still lacks an evaluation of goal satisfaction considering values that come from enterprise architectural analysis. This is an open challenge that may be addressed in the context of our future work.

To summarize our discussion, after addressing the issues of *language* and *techniques* for enterprise analysis, we also envision that methodological guidelines for producing models using this language must be developed and the resulting techniques must be validated through real-world case studies with the purpose of validating them in practice.

References

[1] M. Lankhorst, M. E. Iacob, H. Jonkers, L. van der Torre, H. A. Proper, F. Arbab, F. de Boer, M. Bonsangue and W. Janssen, Enterprise Architecture at Work - Modelling, Communication, and Analysis, Springer-Verlag, 3rd edition, 2012.

[2] J. Zachman, "A Framework for Information Systems Architecture," *IBM Systems Journal,* pp. 276-292, 1987.

[3] A. Dardenne, A. v. Lamsweerde and S. Fickas, *Goal-directed Requirements Acquisition,* vol. 20, Amsterdam, The Netherlands, Elsevier Science Publishers, 1993, pp. 3-50.

[4] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos and A. Perini, "Tropos: An Agent-Oriented Software Development Methodology," *Journal of Autonomous Agents and Multi-Agent Systems,* p. 203–236, 2004.

[5] A.-W. Scheer, ARIS – Business Process Modeling, Springer, 2000.

[6] D. Neiger and L. Churilov, *Goal-Oriented Business Process Modeling with EPCs and Value-Focused Thinking,* 2004b, pp. 98-115.

[7] S. Nurcan, A. Etien, R. Kaab and I. Zouka, *A Strategy-Driven Business Process Modelling Approach,* 6 ed., vol. 11, 2005, pp. 628-649.

[8] H. Kwan Hee, C. Sang Hyun, J. G. Kang and G. Lee, *Performance-centric business activity monitoring framework for continuous process improvement,* United Kingdom: World Scientific and Engineering Academy and Society (WSEAS), 2010, pp. 40-45.

[9] P. Kueng and P. Kawalek, "Goal-based business process models: creation and evaluation," *In Business Process Management Journal 3,* pp. pp. 17-38, 1997.

[10] P. Johnson, E. Johansson and T. Sommestad, *A Tool for Enterprise Architecture*

Analysis, Annapolis USA, 2007.

[11] M.-E. Iacob and H. Jonkers, *Quantitative Analysis of Enterprise Architectures,* Geneva, Switzerland, 2005, pp. 239-252.

[12] P. Soffer and Y. Wand, *On the Notion of Softgoals in Business Process Modeling,* vol. 11, Emerald Group Publishing Limited, 2005, p. 663– 679.

[13] M. Hammer and J. Champy, Reengineering the Corporation: A Manifesto for Business Revolution, London, England: Nicholas Brealey Publishing, 1993.

[14] S. Mansar and H. Reijers, *Best Practices in Business Process Redesign: Use and Impact,* vol. 13, 2007, pp. 193-213.

[15] The Object Management Group (OMG), "Business Process Maturity Model (BPMM)," 2008. [Online]. Available: URL http://www.omg.org. http://www.omg.org.

[16] J. vom Brocke and M. (. Rosemann, Handbook on Business Process Management 2 - Strategic Alignment, Governance, People and Culture, Springer, 2010.

[17] W. M. P. van der Aalst, Process Mining - Discovery, Conformance and Enhancement of Business Processes, vol. 1, Springer, 2011.

[18] J. Kolár, *Business Activity Monitoring,* Masaryk University, 2009.

[19] D. Grigori, F. Casati and M. Castellanos, "Business Process Intelligence," *Computers in Industry,* vol. 53, no. 3, pp. 321-343, 2004.

[20] J. Bubenko, A. Persson and J. Stirna, "D3 Appendix B: EKD User Guide," Royal Institute of Technology (KTH) and Stockholm University, Stockholm, Sweden, 2001.

[21] D. Quartel, W. Engelsman, H. Jonkers and M. v. Sinderen, *A Goal-Oriented Requirements Modelling Language for Enterprise,* Auckland, New Zealand: IEEE Computer Society, 2009.

[22] V. Popova and A. Sharpanskykh, *Formal goal-based modeling of organizations,* INSTICC Press, 2008.

[23] Object Management Group (OMG), *Business Motivation Model (BMM),* 2008.

[24] C. Pedrinaci, I. Markovic, F. Hasibether and J. Domingue, *Strategy-Driven Business Process Analysis,* 2009, pp. 169-180.

[25] A. Dardenne, A. van Lamsweerde and S. Fiskas, *Goal Directed Requirements Acquisition,* vol. 20, 1993, pp. 3-50.

[26] ISO - International Organization for Standard, *Information technology — Open Distributed Processing — Use of UML for ODP system specifications,* 2008.

[27] W. Engelsman and R. Wieringa, *Goal-oriented requirements engineering and enterprise architecture: Two case studies and some lessons learned,* vol. 7195 of Lecture Notes in Computer Science, London, UK: Springer Verlag, 2012, p. 306–320.

[28] I. Markovic and M. Kowalkiewicz, *Linking Business Goals to Process Models in Semantic Business Process Modeling,* Munich, Germany, 2008, pp. 332-338.

[29] British Ministry of Defence, "MOD Architecture Framework (MODAF)", 2005.

http://www.mod.uk/DefenceInternet/AboutDefence/WhatWeDo/InformationManagement/MODAF. [Accessed 16 02 2013].

[30] USA Department of Defense, *DoD Architecture Framework version 1.5 Volume I: Definitions and Guidelines,* 2007.

[31] P. Halleux, L. Mathieu and B. Andersson, *A Method to Support the Alignment of Business Models and Goal Models,* Montpellier, France: CEUR Workshop Proceedings, 2008, pp. 120-134.

[32] G. Koliadis, A. Vranesevic, M. Bhuiyan, A. Krishna and A. Ghose, *A Combined Approach for Supporting the Business Process Model Lifecycle,* Kuala Lumpur, Malaysia, 2006a.

[33] Y. Lin, *Semantic Annotation for Process Models: Facilitating Process Knowledge Management via Semantic Interoperability,* Trondheim, Norway, 2008.

[34] E. Cardoso, J. P. A. Almeida and R. Guizzardi, "Analyzing the Relations between Strategic and Operational Aspects of an Enterprise: Towards an Ontology-based Approach," *International Journal of Organizational Design and Engineering (IJODE), 2012,* vol. 2, no. 3, pp. 271 - 294, 2012.

[35] G. Guizzardi, *Ontological Foundations for Structural Conceptual Models,* University of Twente, The Netherlands, 2005.

[36] M. Pesic, *Business process management maturity model and Six Sigma: An integrated approach for easier networking,* Sarajevo: Springer, 2009.

[37] A. Teka, *Analysis of indirect influence relations in goal-oriented requirements engineering,* Electrical Engineering, Mathematics and Computer Science (EEMCS), University of Twente , 2012.

Author Index