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

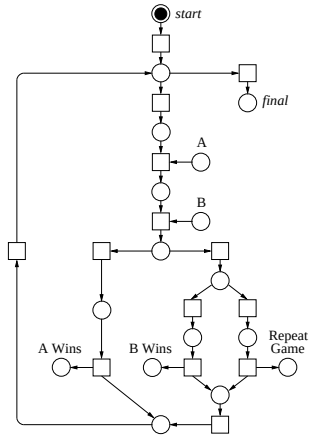
For illustration, we will use the example shown in Figure 1. On its upper left-hand side, a (business) process is shown in a textual format. The process models the game *Rock-paper-scissors*, where two partners (A and B) play against each other. The idea is, that each player decides whether to take one 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.

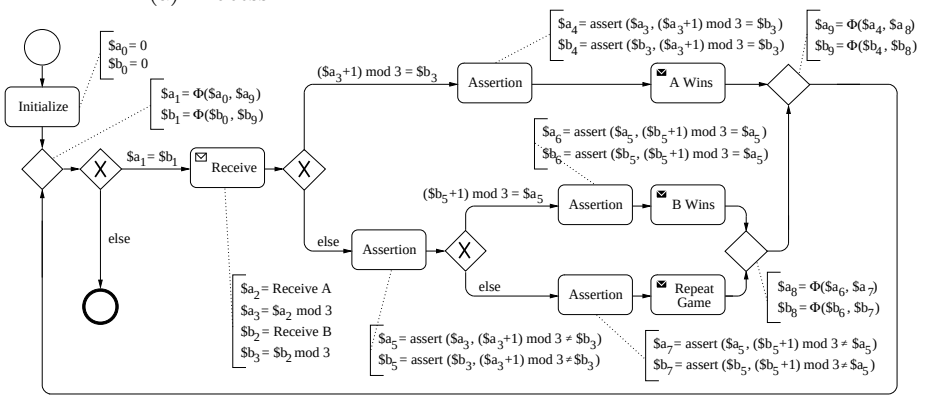
When verifying this process with regard to *controllability* [4], i.e., whether partners A and B exist for which the process will always be able to terminate its execution, the process is mapped to a Petri-net-based process model first. The application of a conventional mapping, e.g., using the pattern-based 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.

```
$a = 0
$b = 0
WHILE ($a = $b) DO
    $a = RECEIVE A
    $a = $a mod 3
    $b = RECEIVE B
    $b = $b mod 3
    IF (($a + 1) mod 3 = $b) THEN
        REPLY A WINS
    ELSE
        IF (($b + 1) mod 3 = $a) THEN
            REPLY B WINS
        ELSE REPLY REPEAT GAME
        END
    END
END
```

(a) Process



(b) Petri Net Model



(c) Extended Workflow Graph

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 $\$a_1$:

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

and for variable $\$b_1$:

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

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:

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

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

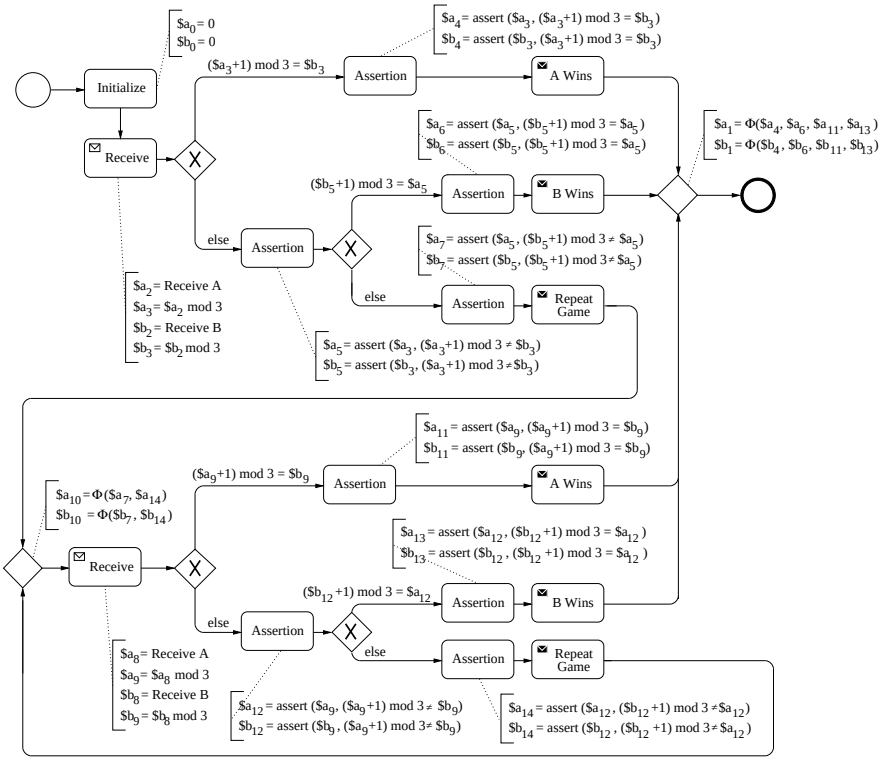


Fig. 2. Extended workflow graph with unfolded loop

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

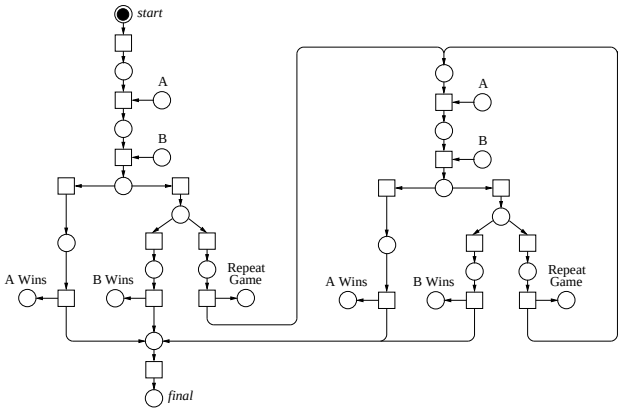


Fig. 3. Refined Petri net model

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)