RZ 3767 (# 99777) 02/25/2010
Computer Science 18 pages

Research Report

Detection of Semantically Equivalent Fragments for Business Process Model Change Management

Christian Gerth,[1,2] Markus Luckey,[1] Jochen M. Küster,[2] and Gregor Engels[1]

[1]Department of Computer Science
University of Paderborn
Germany

{gerth,luckey,engels}@upb.de

[2]IBM Research – Zurich
8803 Rüschlikon
Switzerland

{cge,jku}@zurich.ibm.com

Detection of Semantically Equivalent Fragments for Business Process Model Change Management

Christian Gerth[1,2], Markus Luckey[1], Jochen M. Küster[2], and Gregor Engels[1]

[1] Department of Computer Science, University of Paderborn, Germany
{gerth,luckey,engels}@upb.de
[2] IBM Research - Zurich, Säumerstr. 4
8803 Rüschlikon, Switzerland {cge,jku}@zurich.ibm.com

Abstract. Modern business process modeling environments support distributed development by means of model version control, i. e. comparison and merging of two different model versions. This is a challenging task since most modeling languages support an almost arbitrary creation of process models. Thus, in multi-developer environments, process models or parts of them are often syntactically very different but semantically equivalent. Hence, the comparison of business process models must be performed on a semantic level rather then on a syntactic level. For the domain of business process modeling, this problem is yet unsolved. This paper describes an approach that allows the semantic comparison of different business process models using a normal form. For that purpose, the process models are fully automatically translated into process model terms and normalized using a term rewriting system. The resulting normal forms can be efficiently compared. Our approach enables the semantic comparison of business process models ignoring syntactic redundancies.

1 Introduction

In model-driven development approaches, business process models are used at different levels in the development process. For instance, they are used to specify the behavior of involved components or to describe the choreography of services in Service-Oriented Architectures (SOA) [7]. In addition, business process models help to obtain a closer alignment of business and IT requirements [12].

Similar to other software artifacts, business process models are created and refined by many different business modelers and software architects. This results in different versions of models that have to be compared and merged with each other at some point in time to obtain a integrated version [13].

Comparing and merging different process model versions requires the knowledge whether two models or individual parts of them are equivalent. In general, there are two different ways to decide their equivalence. First, two models can be compared syntactically. Syntactic comparison has the benefit of being hardly time-consuming. The

C. Gerth is funded by the International Graduate School of Dynamic Intelligent Systems at the University of Paderborn

second approach is the semantic comparison, e.g. by using a notion of equivalence such as trace equivalence [24]. While these semantic approaches are too time-consuming due to the need to check all possible paths, syntactic comparison seems to be too weak to detect equivalences between different models.

This problem results from the fact that well-established modeling languages such as the Business Process Modeling Notation (BPMN) [17] or Activity Diagrams (UML-AD) [18] generally allow the user to connect elements such as Actions and Gateways in an arbitrary way. Together with modeling tools that support free-hand editing, this favors the construction of syntactically very different process models, which are at the same time semantically equivalent regarding their execution logic and execution order.

Using current syntactic comparison techniques, two of those syntactically different business process models would incorrectly not be shown to be equal. The target of our approach is to reduce this kind of false-positives in the results of the comparison of structured and unstructured business process models.

In this paper, we propose an approach for deciding equivalence of structured and unstructured business process models and individual parts of them. Our approach combines the benefits from both the syntactic and semantic comparison. Individual parts of business processes are transformed into a semantically enriched term representation, which is normalized using a term rewriting system. The rules of the term rewriting systems reduce syntactically different but semantically equivalent terms of two process models into their normal form by preserving their behavior. Based on the normal forms, we can decide whether the two parts of a process models are semantically equivalent by comparing their syntactical structure. The approach is in particular useful in distributed process modeling scenarios, to answer the question whether independently applied changes result in semantically equivalent structures in the process models or give rise to conflicts. In addition, the normal form can directly be adopted in the merged version of the process model to achieve a simple to understand and reduced process model.

The remainder of this paper is structured as follows: In Section 2 we introduce a typical scenario for business process model change management. We present a term representation for process models in Section 3. In Section 4 we develop a term rewriting system and consider its correct functional behavior. Then, we present a case study and normalize two process model terms to decide equivalence in Section 5. Finally, we conclude with a discussion of related work and give an outlook on future work.

2 Business Process Model Change Management

Our scenario is inspired by the business process model change management in the IBM WebSphere Business Modeler [9]. Figure 1 shows an example business process model M from the insurance domain. In the example, a claim is checked. Depending on whether the claim is covered by the insurance it is either settled or rejected. In a distributed modeling scenario, the process model M might have been created as an initial process model, which is then independently refined into the versions M_1 and M_2 by different users. Afterwards, the need arises to integrate the different versions of the models to obtain a merged version M_M. Typically, this includes the detection of differences, dependencies and conflicts between the versions. In general, difference detection between business process models requires a matching to identify corresponding model elements, e.g., *'Check Claim'* in model M_1 corresponds to *'Check Claim'* in M_2. Model matching is an area of research in its own right with a lot of interest [4, 11, 10] and is not in the scope of this paper. For the remainder, we assume that model elements with equal names correspond to each other and thus no further matching is needed.

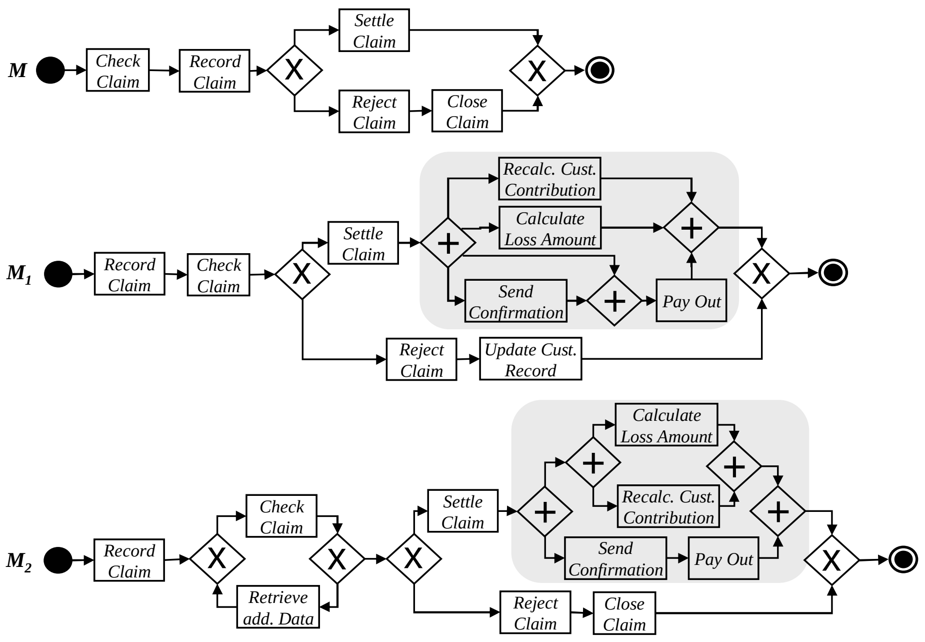
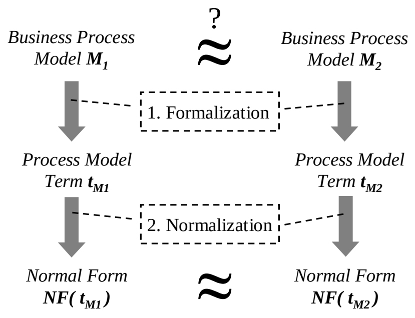


Fig. 1. Change Management Scenario

Although the different process model versions (M_1 and M_2) look similar, a purely syntactical comparison of these models results in lots of differences and even conflicts, among them many false positives. For instance, the highlighted parts in Figure 1 are semantically equivalent, i. e. the same set of actions can be applied in the same order. However, since their syntactical representation is different, a syntax-based comparison results in differences and conflicts that need to be resolved manually to obtain a merged version M_M. In addition, the matching of the Actions (e.g., *'Pay Out'*) in M_1 and M_2 is difficult, since it is unclear whether they are inserted at the same position. To avoid these problems, the comparison of business process models must consider the semantics of the models.

A widely used approach to decide semantic equivalence of business process models is by using trace equivalence [24]. Computing trace equivalence can be complex, in particular for concurrent structures or cyclic structures in process models and has exponential complexity in the general case. Even if traces are computed only for parts of a process model the number of traces that need to be compared can be high. E.g., in the small example in Figure 1 the four activities in the highlighted parts of the process models result in 12 different traces. These sets of traces need to be compared to decide the equivalence of the highlighted parts. In the case, that two process models are not trace equivalent, sets of different traces are returned that are hardly understandable to the user who merges the business process models. Thus, these different traces need to be analyzed in order to identify the actual differences between the two models.

Based on our recent work [8] and a decomposition of process models into their hierarchical structure [25], we propose a more efficient way to decide equivalence between process models that overcomes the shortcomings of trace equivalence. Our approach is illustrated in Figure 2. To decide equivalence between two business process models (or parts of the models), we compute process structure trees from the process models that reflect the models hierarchical structure [8, 25]. Using a tree walk algorithm, the process structure tree is transformed into a term notation. The resulting process model terms capture semantical information concerning the execution order and the execution logic (such as AND, OR, XOR) of the business process models. In a second step, the process model terms are normalized into their normal form using a semantic preserving term rewriting system. Based on a comparison of the normal forms, we can efficiently decide the equivalence of the business process models.


?
Business Process Model M_1
≈
Business Process Model M_2
1. Formalization
Process Model Term t_{M1}
Process Model Term t_{M2}
2. Normalization
Normal Form $NF(t_{M1})$
≈
Normal Form $NF(t_{M2})$
Fig. 2. Approach Overview

In addition, the normal form can directly be used to adopt changes in the integrated process model M_M. This potentially results in fewer changes that need to be applied to merge two process models and improves the understandability of the merged model. A detailed example can be found in Section 5.

In the following, we introduce the term notation for business process models. Then we present the term rewriting system in Section 4.

3 Process Model Terms

In this section we first introduce process models and its decomposition into fragments. Then we develop a process model term notation and show how process models are transformed into terms that can be compared using the term rewriting system from Section 4.

3.1 Process Models and the Process Structure Tree

For the following discussions, we assume a business process model M to be expressed using the basic workflow graphs described in [21, 25] and define the semantics of process models similar to the semantics of Petri nets [16] in terms of token flows. In our recent work [8], we have shown how concrete BPMN models can be abstracted to such process models and described the behavior of particular process model nodes in detail. Figure 3 (a) shows a simple process model.

To enable a comparison of individual parts of two process models, they are decomposed into single-entry-single-exit fragments [25]. Such a fragment is a non-empty subgraph in the process model with a single entry and a single exit edge. Based on the decomposition into fragments we are able to decide equivalence between pairs of fragments instead of relying on the complete process model.

Among all possible fragments, we consider only so called canonical fragments which are not overlapping and denote them with $\mathcal{F}(M)$ for a process model M. The canonical fragments of a process model M can be organized into a corresponding PST representing the hierarchical structure of the fragments.

Figure 3 gives an example. The canonical fragments of the process model M_1 in Figure 3 (a) are visualized by a surrounding of dotted lines and its corresponding PST is given in Figure 3 (b).

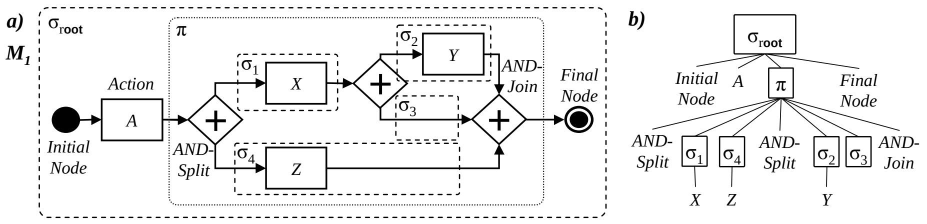


Fig. 3. (a) Process Model M_1 decomposed into Fragments and (b) its Process Structure Tree

Given a fragment $f \in \mathcal{F}(M)$, we distinguish by $type(f)$ the following fragment types (similar to [25]):

- a *sequence* σ has no AND/XOR/Undefined-ControlNodes as children. Further, a sequence is maximal, i.e., neither a preceding nor a succeeding model element can be added to the sequences. For example, in Figure 3 (a), σ_{root}, σ_1, σ_2, σ_3, and σ_4 are sequences.
- a *parallel fragment* π does not contain any cycles and has no XOR/Undefined-ControlNodes as children. For example, in Figure 3 (a), π is a parallel fragment.
- an *alternative fragment* α and an alternative loop λ have no AND/Undefined-ControlNodes as children.
- a *complex fragment* ι is any other fragment that is none of the above.

Further, a fragment is considered as being *structured* if it consists of matching pairs of nodes that split and join the control flow. Otherwise, it is considered as being *unstructured*. In the following we introduce process model terms and show how process models and their corresponding PST can be formalized.

3.2 Transformation of Process Models into Terms

A term representation of process models needs to capture precise information about the contained model elements, their hierarchical structure in terms of fragments, as well as information about the execution order of the elements.

Figure 4 summarizes the translation of PST elements of process models to their corresponding term representation. ControlNodes are not added to process model terms, since they are already represented by their surrounding fragments. Figure 5 shows the grammar for valid terms.

PST Element	**Term**
InitialNode, FinalNode, Action ν	ν
Sequence σ	$\sigma(...)$
Parallel Fragment π	$\pi(...)$
Alternative Fragment α	$\alpha(...)$
Alternative Loop Fragment λ	$\lambda(...)$
Complex Fragment ι	$\iota(...)$

Fig. 4. Transformation of Process Models into Terms

$$\begin{array}{lll}
t \in \mathcal{T} & ::= & Frag \quad \mathcal{O} \\
Frag & ::= & \pi(Seqs) \mid \alpha(Seqs) \mid \lambda(Seqs) \mid \iota(Seqs) \mid Seq \\
Frags & ::= & Frag\ ,\ Frags \mid Frag \\
Seq & ::= & \sigma(Frags) \mid \sigma(Nodes) \\
Seqs & ::= & Seq\ ,\ Seqs \mid Seq \\
Node & ::= & \nu \mid Frag \mid \epsilon \\
Nodes & ::= & Node\ ,\ Nodes \mid Node \\
\mathcal{O} & ::= & Seq \times Seq
\end{array}$$

Fig. 5. Grammar for Process Model Terms

$$\begin{aligned}
t_{M1} = \sigma(&'InitialNode',' A', \\
&\pi(\sigma_1('X'), \sigma_2('Y'), \\
&\sigma_3(), \sigma_4('Z')), \\
&'FinalNode') \\
\mathcal{O}_{M1} = \{&\sigma_1 < \sigma_2, \sigma_1 < \sigma_3\}
\end{aligned}$$

Fig. 6. Example

The transformation of process models into terms is straight-forward by traversing its PST using the infix approach. We define $\mathcal{O}$ to be the partial order set over the carrier set *Seq*, i. e. the set of sequences. $\mathcal{O}$ specifies the execution order of branches within fragments that is necessary to decide equivalence of unstructured parallel and alternative fragments. For instance, within the parallel fragment π in Figure 3 (a), the sequence σ_1 is executed before sequences σ_2 and σ_3, resulting in a partial order $\mathcal{O} = \{\sigma_1 < \sigma_2, \sigma_1 < \sigma_3\}$. We do not explicitly specify the execution order of elements within sequences, since it is given implicitly. The partial order within a fragment can be obtained easily by traversing all branches of each fragment using the depth first approach.

Let $s \in \mathcal{F}(M)$ be a sequence of the process model M and $\sigma \in Seq$ be its corresponding sequence in the process model term t_M. We define $\mathcal{O}_{Pre}(\sigma) = \{(\sigma_i, \sigma) \in \mathcal{O}\}$ to be the preset of σ containing tuples $\sigma_i < \sigma$, i.e., the sequences σ_i that are executed directly before σ, and $\mathcal{O}^*_{Pre}(\sigma)$ contain all preceding sequences reachable from σ in its parent fragment. Analogously, we define $\mathcal{O}_{Post}(\sigma) = \{(\sigma, \sigma_j) \in \mathcal{O}\}$ to be the postset of σ containing tuples $\sigma < \sigma_j$, i.e., the sequences σ_j that are executed directly after σ, and $\mathcal{O}^*_{Post}(\sigma)$ contain all succeeding sequences reachable from σ in its parent fragment. We denote the union of both sets as $\mathcal{O}(\sigma) = \mathcal{O}_{Pre}(\sigma) \cup \mathcal{O}_{Post}(\sigma)$.

The business process model M_1 and its PST introduced in Figure 3 (b) result in the process model term t_{M1} and execution order $\mathcal{O}_{M1}$ shown in Figure 6. We use indices for sequences for short. For instance, we use $\sigma_1('X')$ in the process model term and refer to this particular sequence using σ_1 in the partial order. However, the sequence indices are not part of the original grammar but are only used for the sake of brevity. The variable v ranges over the nodes in the process models.

Process model terms can be syntactically compared very efficiently. However, to compare the semantic equivalence of syntactically different models, their corresponding terms have to be transformed into a normal form to be considered equivalent, since syntactically different models result in different terms regardless their semantic meaning. The term rewriting system for this transformation is introduced in the next section.

4 Term Rewriting System for Process Model Terms

In this section we introduce a term rewriting system [1] for process model terms to enable semantic comparison of business process models. The system consists of a set of rules, which reduce process model terms to a normal form. Then, we consider the correct functional behavior of the rewriting system. Finally, we define equivalence of process models based on the normal form of process model terms.

4.1 Term Rewriting System

The term rewriting system consists of rules for the reduction of sequences σ, parallel fragments π, and alternative fragments α in structured and unstructured form. Overall the rule system consists of 16 reduction rules. Some of the rules are inspired by the rules presented in [23, 15, 3, 5]. We do not intent the set of reduction rules to be complete, in the sense that by using the rules, equivalence can be decided for all fragments of process model. However, fragments whose equivalence cannot be decided using our approach are known in advance. For these cases, trace equivalence is leveraged that additionally is speeded up, since traces only need to be computed for fragments, which are significantly smaller compared to the whole process model.

In the following, we present the rules and give graphical examples for clarification. Further, we briefly consider their correctness.

Rule Par 1 *(Elimination of Empty Sequences) Given a parallel fragment π containing at least two sequences. If one of the contained sequences is empty σ_ϵ it is removed and the partial execution order is aligned in such a way that all $\sigma_i < \sigma_\epsilon$ and $\sigma_\epsilon < \sigma_j$ are replaced by new restrictions $\sigma_i < \sigma_j$.*

$$\text{(par1)}\ \frac{\pi(\sigma_1, \ldots, \sigma_\epsilon, \ldots, \sigma_n) \quad \mathcal{O}}{\pi(\sigma_1, \ldots, \sigma_n) \quad ((\mathcal{O} \setminus \mathcal{P}) \setminus \mathcal{S}) \cup \mathcal{N}}$$

$$\begin{aligned} \textbf{\textit{where}}\ \mathcal{P} &= \{(\sigma_i, \sigma_\epsilon) \in \mathcal{O} \mid \sigma_i < \sigma_\epsilon\} \\ \mathcal{S} &= \{(\sigma_\epsilon, \sigma_j) \in \mathcal{O} \mid \sigma_\epsilon < \sigma_j\} \\ \mathcal{N} &= \{(\sigma_i, \sigma_j) \mid (\sigma_i, \_) \in \mathcal{S} \wedge (\_, \sigma_j) \in \mathcal{P}\} \end{aligned}$$

Rule Par 1 removes one empty sequence of a parallel fragment at a time and is applied until all empty sequences are eliminated. Figure 7 gives an example, where the empty sequence σ_3 is removed by applying Rule Par 1. The removal of an empty sequence in a parallel fragment is semantic preserving, since traces are not changed and empty sequences can be neglected in process models.

Rule Par 2 *(Concatenation of Sequences) Given a parallel fragment π containing two sequences $\sigma_1(a, .., n)$ and $\sigma_2(m, .., z)$ with the execution order $\sigma_1 < \sigma_2$ and no other restriction of the form $\sigma_1 < \sigma_i$ and $\sigma_i < \sigma_2$. Then $\sigma_1(a, .., n)$ and $\sigma_2(m, .., z)$ are concatenated into $\sigma_\star(a, .., n, m, .., z)$ and the partial execution order is aligned in such a way that all $\sigma_i < \sigma_1$ and $\sigma_2 < \sigma_j$ are replaced by new restrictions $\sigma_i < \sigma_\star$ and $\sigma_\star < \sigma_j$.*

$$\textit{(par2)}\ \frac{\pi(x, \sigma_1(a, \dots, n), \sigma_2(m, \dots, z), y) \quad \mathcal{O}}{\pi(x, \sigma_\star(a, \dots, n, m, \dots, z), y) \quad \mathcal{O}'}$$

$$\begin{aligned} \textbf{\textit{where }} \mathcal{O}' &= \mathcal{O} \setminus (\{(\sigma_1, \sigma_2)\} \cup \mathcal{O}_{Pre}(\sigma_1) \cup \mathcal{O}_{Post}(\sigma_2)) \cup \mathcal{P} \cup \mathcal{S} \\ \mathcal{P} &= \{(\sigma_i, \sigma_\star) \mid (\sigma_i, \sigma_1) \in \mathcal{O}_{Pre}(\sigma_1)\} \\ \mathcal{S} &= \{(\sigma_\star, \sigma_j) \mid (\sigma_2, \sigma_j) \in \mathcal{O}_{Post}(\sigma_2)\} \end{aligned}$$

Figure 7 gives an example. The unstructured parallel fragment π contains four sequences $\sigma_1 \dots \sigma_4$. After the deletion of sequence σ_3 (by the application of Rule Par 1), Rule Par 2 becomes applicable, which concatenates the sequences σ_1 and σ_2 into $\sigma_\star$. Rule Par 2 preserves the behavior of the process model since all actions are obviously executed in the same order.

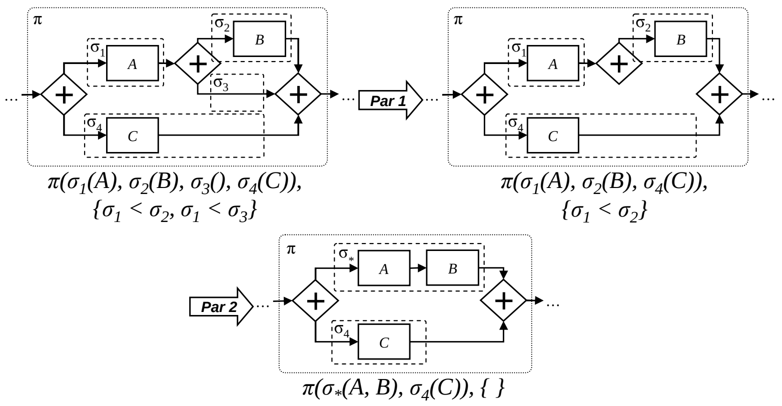


Fig. 7. Example for the Application of Rule Par 1 and Par 2

Rule Par 3 *(Resolution of Empty Parallel Fragments) Given a parallel fragment π that contains one single sequence σ. Then the parallel fragment can be dropped regardless of any given partial order.*

$$\textit{(par3)}\ \frac{\pi(\sigma)\quad O}{\sigma\quad O}$$

Figure 8 gives an example. Similar to the former rule, after the removal of the empty sequence σ_2 by Rule Par 1, Rule Par 3 becomes applicable that removes the parallel fragment π and integrates σ_1 into the enclosing sequence σ. Also Rule Par 1 is behavior preserving since the contained sequence is still executed at the same position.

Rule Seq 1 *(Resolution of Nested Sequences) Given a sequence that contains another sequence. Then the inner sequence may be dropped and its components are inserted into the outer sequence.*

$$\textit{(seq1)}\ \frac{\sigma(x, \sigma(a \dots n), y)\quad O}{\sigma(x, a, \dots, n, y)\quad O}$$

The application of Rule Seq 1 is shown in Figure 8. Here, the inner sequence σ_1 is removed and its contained elements are integrated into the enclosing sequence σ. Since Rule Seq 1 changes only the fragment hierarchy of the model and not the model itself, it is semantic preserving.

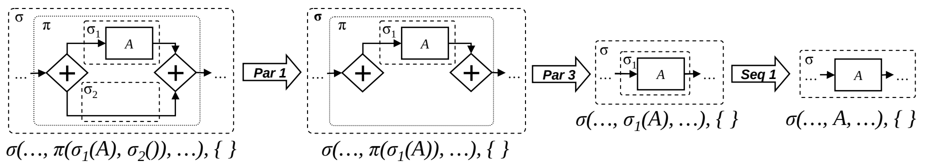


Fig. 8. Example for the Application of Rule Par 1, Par 3, and Seq 1

Rule Seq 2 *(Extraction of Sequences from parallel Fragments) Given a parallel fragment π and a set $\mathcal{S}$ comprising all sequences σ directly contained in π. We assume that a non empty sequence $\sigma_i \in \mathcal{S}$ exists that is executed **after** all other sequences $\sigma_j \in \mathcal{S}$, i.e. $\forall \sigma_j \in \mathcal{S} \setminus \{\sigma_i\} : \exists(\sigma_j, \sigma_i) \in O$. We denote this set with $\mathcal{P}$. Then σ_i is extracted from π and inserted directly after π in the surrounding parent fragment σ.*

*Analogously, Rule Seq 2(b) is applicable in cases, where σ_i is executed **before** all other sequences $\sigma_j \in \mathcal{S}$, i.e., $\forall \sigma_j \in \mathcal{S} \setminus \{\sigma_i\} : \exists(\sigma_i, \sigma_j) \in O$. Again, we denote this set with $\mathcal{P}$.*

$$\textit{(seq2a)}\ \frac{\sigma(x, \pi(\sigma_1, .., \sigma_i, .., \sigma_n), y)\quad O}{\sigma(x, \pi(\sigma_1, .., \sigma_n), \sigma_i, y)\quad O \setminus \mathcal{P}} \qquad \textit{(seq2b)}\ \frac{\sigma(x, \pi(\sigma_1, .., \sigma_i, .., \sigma_n), y)\quad O}{\sigma(x, \sigma_i, \pi(\sigma_1, .., \sigma_n), y)\quad O \setminus \mathcal{P}}$$

Figure 9 gives an example. The sequence σ_4 is removed by Rule Par 1. Then Rule Seq 2(b) becomes applicable, since sequence σ_1 is executed before all other sequences in π. : The Rule Seq 2 is semantic preserving since all traces in the origin process model execute the moved action in the end of the parallel fragment which is obviously the case after extracting the action from the parallel fragment.

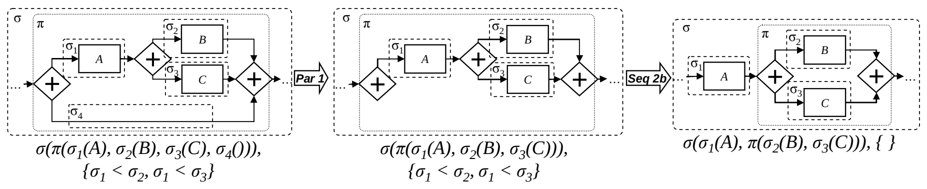

σ
π
σ1
σ2
σ3
σ4
A
B
C
Par 1
Seq 2b
σ(π(σ1(A), σ2(B), σ3(C), σ4())), {σ1 < σ2, σ1 < σ3}
σ(π(σ1(A), σ2(B), σ3(C))), {σ1 < σ2, σ1 < σ3}
σ(σ1(A), π(σ2(B), σ3(C))), { }

Fig. 9. Example for the Application of Rule Par 1 and Seq 2(b)

Rule Par 4 *(Resolution of Nested Parallel Fragments) Given a sequence σ in a parallel fragment π_1. If σ contains only a well-formed*[1] *parallel fragment π_2 then the sequence σ and π_2 are dropped and the contained sequences of π_2 are inserted into the outer parallel fragment π_1.*

Rule Alt 1 *(Resolution of Nested Alternative Fragments) Analogously, Rule Alt 1 aligns nested alternative fragments.*

$$\textit{(par4)}\ \frac{\pi_1(x,\sigma(\pi_2(z)),y) \quad O}{\pi_1(x,z,y) \quad O} \qquad\qquad \textit{(alt1)}\ \frac{\alpha_1(x,\sigma(\alpha_2(z)),y) \quad O}{\alpha_1(x,z,y) \quad O}$$

$$\textbf{\textit{where}}\ \forall\sigma \in \pi_2 : O(\sigma) = \emptyset \qquad\qquad \textbf{\textit{where}}\ \forall\sigma \in \alpha_2 : O(\sigma) = \emptyset$$

An example for Rule Par 4 can be found in Figure 10. There, a structured parallel fragment π_2 is enclosed by a sequence σ_1, which is otherwise empty. Since, σ_1 is in turn located in a parallel fragment, Rule Par 4 can be applied. The application removes π_2 and replaces σ_1 with the contained sequences σ_2 and σ_3 of π_2. Nested parallel and alternative fragments that are structured are resolved by Rule Par 4 and Rule Alt 1 in a semantic preserving way, since the process model is not changed at all, but only its representation in the PST.

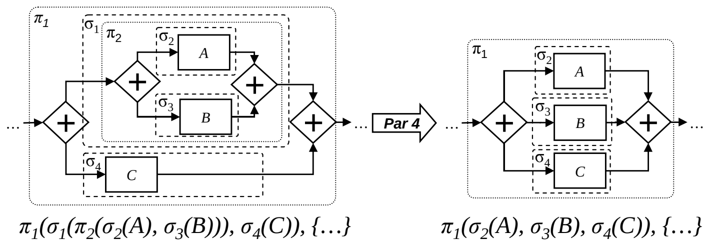

π1
σ1
π2
σ2
σ3
σ4
A
B
C
Par 4
π1(σ1(π2(σ2(A), σ3(B))), σ4(C)), {...}
π1(σ2(A), σ3(B), σ4(C)), {...}

Fig. 10. Example for the Application of Rule Par 4

[1] Note that structured fragments f are indicated by empty partial order set, i.e., $O(f) = \emptyset$.

Rule Alt 2 *(Elimination of Doubled Sequences) Given an alternative fragment α that contains two sequences $\sigma_1(z)$ and $\sigma_2(z)$ that equal each other (e. g. they are empty). Furthermore, they need to be in the exact same ordering relation, that is, if σ_1 is executed before some σ_s and after some σ_p, then σ_2 is executed before σ_s and after σ_p, too. Then one of them may be dropped.*

$$\textit{(alt2)}\ \frac{\alpha(x, \sigma_1(z), \sigma_2(z), y) \quad O}{\alpha(x, \sigma_1(z), y) \quad O \setminus O(\sigma_2)} \qquad \textbf{\textit{where}}\ O(\sigma_1) = O(\sigma_2)$$

Rule Alt 3 *(Elimination of Empty Sequences in Alternative Fragments) Let an alternative fragment α containing an empty sequences σ_ϵ be given. If either the preset $O_{Pre}(\sigma_\epsilon)$ or the postset $PO_{Post}(\sigma_\epsilon)$ of σ_ϵ are not empty. Then, σ_ϵ can be removed and the partial order relation O is aligned as follows:*

$$\textit{(alt3)}\ \frac{\alpha(x, \sigma_\epsilon, y) \quad O}{\alpha(x, y) \quad O'} \qquad \begin{aligned} \textbf{\textit{where}}\ O' &= O \setminus O_{Pre}(\sigma_\epsilon)\textbf{\textit{, if}}\ O_{Post}(\sigma_\epsilon) = \emptyset \\ O' &= O \setminus O_{Post}(\sigma_\epsilon)\textbf{\textit{, if}}\ O_{Pre}(\sigma_\epsilon) = \emptyset \\ O' &= (O \setminus O(\sigma_\epsilon)) \cup \mathcal{S}\textbf{\textit{, otherwise}} \\ \mathcal{S} &= \{(\sigma_i, \sigma_j) \mid (\sigma_i, \sigma_\epsilon) \in O_{Pre}(\sigma_\epsilon) \\ &\quad \wedge (\sigma_\epsilon, \sigma_j) \in O_{Post}(\sigma_\epsilon)\} \end{aligned}$$

Figure 11 shows two examples for the application of Rule Alt 3. In both cases, sequence σ_4 is removed.

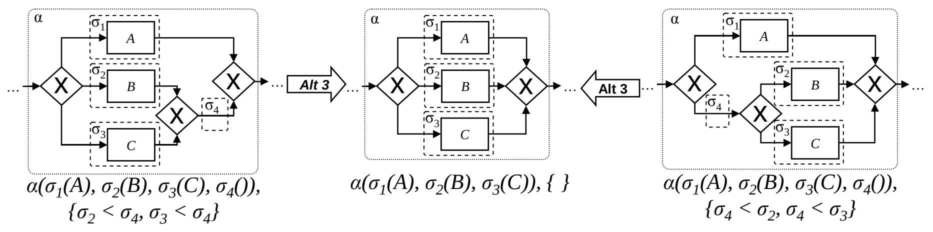

α
σ1
A
σ2
B
σ3
C
σ4
X
Alt 3
α(σ1(A), σ2(B), σ3(C), σ4()),
{σ2 < σ4, σ3 < σ4}
α(σ1(A), σ2(B), σ3(C)), { }
α(σ1(A), σ2(B), σ3(C), σ4()),
{σ4 < σ2, σ4 < σ3}

Fig. 11. Two Examples for the Application of Rule Alt 3

Rule Alt 4 *(Extraction of Activities out of Alternative Fragments) Let an alternative fragment α containing sequences σ_1 to σ_i be given. If all of the sequences start with the same activity A and their presets are equal ($O_{Pre}(\sigma_1) = ... = O_{Pre}(\sigma_i)$). Then, the activity A can be extracted from the sequences σ_1 to σ_i and is inserted in the preceding sequence σ.*

$$\textit{(alt4a)}\ \frac{\sigma(\alpha(\sigma_1(A, ...), ..., \sigma_i(A, ...))) \quad O}{\sigma(A, \alpha(\sigma_1(...), ..., \sigma_i(...))) \quad O}$$

$$\textbf{\textit{where}}\ O_{Pre}(\sigma_1) = O_{Pre}(\sigma_2) = ... = O_{Pre}(\sigma_i)$$

In the top part of Figure 12 an examples for the application of Rule Alt 4 is shown. There, the Activity A is extracted from the sequences σ_1 to σ_i that succeed the entry *XOR-Split* of the alternative fragment α. Afterwards, A is inserted in the preceding sequence σ.

In a similar way, variants of this rule extract activities at the end of alternative fragments (**Rule Alt 4b**) and from several sequences within an alternative fragment into their preceding sequence (**Rule Alt 5a/b**) as shown in the bottom of Figure 12.

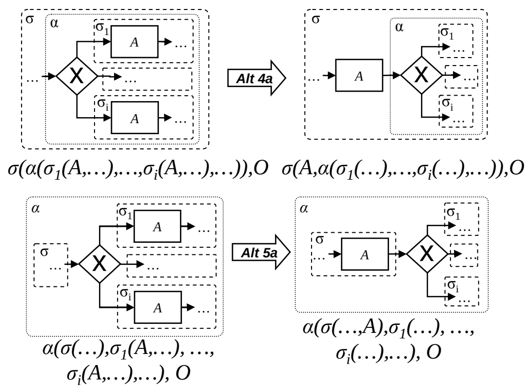


Fig. 12. Two Examples for the Extraction of an Activity by Rule Alt 4 and Rule Alt 5

The following commutativity rules reorder sequences within concurrent and alternative fragments.

Rule Com 1 *(Reordering of Sequences) Given a parallel fragment π and two contained sequences σ_1 and σ_2. Further, let σ_1 and σ_2 have the same set of preceding and succeeding sequences in the partial order relation, i.e., $\forall(\sigma_i,\sigma_1) \in \mathcal{O} : \exists(\sigma_i,\sigma_2) \in \mathcal{O}$ and $\forall(\sigma_1,\sigma_j) \in \mathcal{O} : \exists(\sigma_2,\sigma_j) \in \mathcal{O}$ and vice versa. Then, the two sequences can swap their positions within their parent fragment π.*

Analogously, Rule Com 1b reorders sequences contained in alternative fragments.

$$\text{(com1a)}\ \frac{\pi(x,\sigma_1(u),\sigma_2(v),y) \quad \mathcal{O}}{\pi(x,\sigma_2(v),\sigma_1(u),y) \quad \mathcal{O}} \qquad \text{(com1b)}\ \frac{\alpha(x,\sigma_1(u),\sigma_2(v),y) \quad \mathcal{O}}{\alpha(x,\sigma_2(v),\sigma_1(u),y) \quad \mathcal{O}}$$

Rules Par, Alt, and Seq constitute the rule system which is used to transform process model terms into a normal form. Rules Com 1(a) and (b) are applied after the normalization to align process model terms for a syntactical comparison. In the next section, we consider the correct functional behavior of the term rewriting system.

4.2 Functional Behavior of the Term Rewriting System

The term rewriting system can be considered as an algorithm that reduces a given process model term into its normal form. This algorithm has a correct functional behavior if it terminates (Termination) and results in a unique normal form (Confluence) for a process model. Functional behavior can be achieved by ensuring a set of criteria, well-known from the theory of abstract reduction systems [1]. In the following we examine termination and confluence for the rewriting system for process model terms.

Termination Concerning termination, we have to guarantee that no rule is applied infinitely often. One way to show termination is by giving a so-called monotone measure function [1]. This function shows that the application of the rules reduces a certain value which is limited from below.

In the case of our term rewriting system, a potential candidate for the monotone reduction function is the number of fragments in a process model term, which is limited by the number of fragments contained in a process model. Rules Par, Alt, and Seq 1 reduce a process model term t by exactly one fragment, thus the maximum number of applicable rules is limited by the number of fragments within a process model term t. Rules Seq 2(a) and (b) do not reduce the number of fragments. However, the application of these rules is also limited by the number of fragments, since each application moves a sequence from a parallel fragment into the surrounding sequence. Rules Com 1(a) and (b) are applied after the normalization to reorder sequences within fragments and can theoretically be applied infinitely often. However, by logging the reordered sequences, we take care that each sequence is reordered only once.

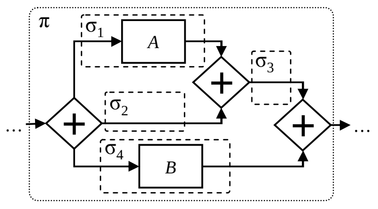
Confluence Confluence ensures that in cases where multiple rules are applicable the choice of the rule does not matter. For terminating rewriting systems confluence follows from the weaker local confluence[2]. This requires if there are two direct rules $t_1 \xleftarrow{r1} t \xrightarrow{r2} t_2$, t_1 and t_2 can be joined again, i.e., they have a common successor. In such scenarios, where multiple rules are applicable on a term t, it can happen that the application of rules overlap, and the application of one rule turns the other one inapplicable. All of these so-called critical pairs need to be considered for confluence by analysing whether they have a common successor, i.e. they are harmless.

In case of our term rewriting system for process models we first overlap each pair of the left-hand sides of the rules to identify critical pairs. Then, for each critical pair we show that it is harmless.

Figure 13 provides an example for a critical pair. The fragment is transformed into the initial term t on the right-hand side. First, the empty sequence σ_2 is removed by applying Rule Par 1. Then, two rules are applicable, either Rule Par 1 on the empty sequence σ_3 or Rule Par 2 to concatenate the sequences σ_1 and σ_3. If one rule is applied the other one is no longer applicable. However, since the resulting terms are equal ($\sigma_1 = \sigma_{13*}$) the critical pair is harmless. Further harmless critical pairs can be obtained by overlapping Rule Par 1 and Rule Seq 2.

In this section we have examined the correct functional behavior of our term rewriting system for process model terms using the existing theory for abstract reduction systems.

[2] This result is usually known as Newman's Lemma [1].



$$\begin{tikzcd}
 & \pi(\sigma_1(\text{‘A’}),\sigma_2(\,),\sigma_3(\,),\sigma_4(\text{‘B’})), O \arrow[d, "\textbf{\textit{Rule Par 1 on }}\boldsymbol{\sigma_2}"] & \\
 & \pi(\sigma_1(\text{‘A’}),\sigma_3(\,),\sigma_4(\text{‘B’})), O \arrow[dl, "\textbf{\textit{Rule Par 1 on }}\boldsymbol{\sigma_3}"'] \arrow[dr, "\textbf{\textit{Rule Par 2 on }}\boldsymbol{\sigma_1}\textbf{\textit{ and }}\boldsymbol{\sigma_3}"] & \\
\pi(\sigma_1(\text{‘A’}),\sigma_4(\text{‘B’})), O & & \pi(\sigma_{13*}(\text{‘A’}),\sigma_4(\text{‘B’})), O
\end{tikzcd}$$

Fig. 13. Example for a Critical Pair obtained by overlapping Rule Par 1 and Rule Par 2

4.3 Equivalence of Process Models and Fragments

Based on the normalized process model terms we define equivalence of process models as follows.

Definition 1 ***(Process Model Equivalence)*** *Given two process models M_1 and M_2 and their representation in process model terms t_{M1} and t_{M2}, M_1 and M_2 are considered to be equivalent, if each fragment f_1 in the normalized term t_{M1} has an equivalent fragment f_2 in the normalized term t_{M2} and vice versa.*

Definition 2 ***(Fragment Equivalence)*** *Two fragments $f_1 \in t_{M1}$ and $f_2 \in t_{M2}$ are considered to be equivalent, if they have the same type ($type(f_1) = type(f_2)$), their contained elements (nodes or fragments) correspond to each other, and they have the same execution order specified in their partial order relations ($\mathcal{O}(f_1) = \mathcal{O}(f_2)$).*

This definition of equivalence relies on a matching of the process models to identify corresponding model elements. As mentioned earlier, we assume that model elements with equal names correspond to each other and corresponding fragments are determined based on their type and their contained model elements as well as sub-fragments.

In the next section, we will apply our term rewriting system to our example introduced above to decide equivalence.

5 Detection of Semantically Equivalent Fragments

The top of Figure 14 shows the highlighted structures of the process model in Figure 1 which would be shown to be different using the current approach without term rewriting. Our approach allows the efficient comparison of these two model fragments while considering their semantics.

Using a simple traversal algorithm, the two models M_1 and M_2 are transformed into their corresponding term representation T_{M1} and T_{M2}. The complexity of this step in our approach is linear to the size of the process model.[3] The resulting process model terms are exact representations of the corresponding process models regarding their syntax. Applying our rule system to both terms results in their corresponding normalizations, i. e. a canonical version of the original term that has not been changed regarding its

[3] Cf. the complexity of an infix tree (PST) traversal and a depth first graph (PM) search.

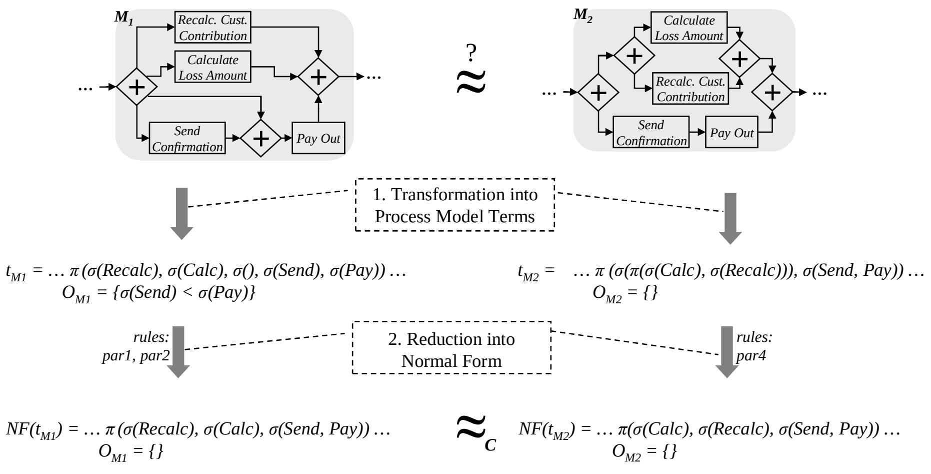

M1
Recalc. Cust. Contribution
Calculate Loss Amount
Send Confirmation
Pay Out
?
≈
M2
Calculate Loss Amount
Recalc. Cust. Contribution
Send Confirmation
Pay Out
1. Transformation into Process Model Terms
t_M1 = … π (σ(Recalc), σ(Calc), σ(), σ(Send), σ(Pay)) …
O_M1 = {σ(Send) < σ(Pay)}
t_M2 = … π (σ(π(σ(Calc), σ(Recalc))), σ(Send, Pay)) …
O_M2 = {}
rules: par1, par2
2. Reduction into Normal Form
rules: par4
NF(t_M1) = … π (σ(Recalc), σ(Calc), σ(Send, Pay)) …
O_M1 = {}
≈_C
NF(t_M2) = … π(σ(Calc), σ(Recalc), σ(Send, Pay)) …
O_M2 = {}

Fig. 14. Deciding Equivalence of Process Models

semantics. The term normalization is linear to the length of the term, and thus, linear to the size of the corresponding process model, since every application of a rule decreases the length of the term.

The last step in our approach is the comparison of two normal forms (see bottom of Figure 14). According to Definition 1, we decide equivalence of the process model terms based on the equivalence of their contained fragments. The comparison of sequences is straight-forward. The comparison of parallel and alternative fragments has to be performed under the consideration of the commutativity Rule COM 1.

In the case of our example, the parallel fragments in Figure 14, turned out to be semantically equivalent, since they have the same normal form.

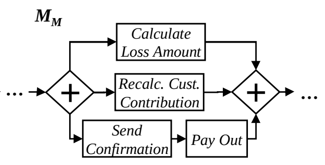


Fig. 15. Visual Representation of the Normal Form

This information can be used to resolve the conflict between the two fragments. In addition, the visualization of the normalized process model term (shown in Figure 15) can directly be adopted in the consolidated version M_M of the process model. The fragment in Figure 15 results in less changes compared to the highlighted fragments in Figure 14 and is more readable. Whether the normal form is always more understandable and can be adopted with fewer (or at least equal) changes than the non reduced fragment is part of future work.

6 Related Work

For the comparison of process models, several different approaches exist. In [14], Li et al. measure distance and similarity of process models based on change operations. Bae et al. [2] measure the similarity between process models using a tree representation and compare its block similarity. Eder et al. [5] provide an equivalence definition for workflow graphs and describe a set of structural modifications to workflow graphs that are semantic preserving. In contrast to our work, they focus on block-structured models.

Van der Aalst et al. [20] compute a quantified equivalence. Weidlich et al. [26] compare process models based on so called behaviour profiles, which reflect different relations between nodes (strict order, exclusiveness and concurrency). Both approaches rely on the existence of execution traces of a process model.

Ehrig et al. [6] compute a combined similarity value consisting of syntactic, linguistic, and structural similarity of elements in process models. The execution order of the elements is not considered.

In [22], van Dongen et al. related process elements with their directly preceding and succeeding elements which they call footprints to measure the similarity between EPC processes. In contrast to our work, their approach does not support the notion of equivalence for parts of a process models.

For the purpose of process model verification, reduction rules are used in different scenarios. E.g., in [19], Sadiq et al. check soundness properties of process models (such as deadlock and lack of synchronization) by iteratively removing sound structures until the model is completely reduced. In [23, 15, 3], reduction rules are applied to EPCs that remove sound functions, events, and connectors in order to reduce the state space and speed up the verification process. However, in our scenario, sound structures cannot be removed, since they are essential to decide equivalence.

7 Conclusion and Outlook

In this paper we have shown a formalism to detect equivalent business process models based on the detection of equivalent fragments contained in the models. First, we have transformed business process models into a process model term. We presented a term rewriting system consisting of several rules that transform process model terms into a normal form. Finally, we have compared the normalized terms to identify equivalent fragments and process models.

Our initial results have shown that a comparison of business process models in their normal form reduce the number of false-positive differences and conflicts in model management.

We intent to integrate the term rewriting system for business process models into our existing tool support for model merging in the IBM WebSphere Business Modeler [9]. Future work also includes to establish a new similarity value for business process models based on the semantical equivalence of process models.

References

1. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, Cambridge, 1998.
2. J. Bae, J. Caverlee, L. Liu, and H. Yan. Process Mining by Measuring Process Block Similarity. In J. Eder and S. Dustdar, editors, *Business Process Management Workshops*, volume 4103 of *LNCS*, pages 141–152. Springer, 2006.

3. R. M. Dijkman. Diagnosing Differences between Business Process Models. In Marlon Dumas, Manfred Reichert, and Ming-Chien Shan, editors, *BPM*, volume 5240 of *Lecture Notes in Computer Science*, pages 261–277. Springer, 2008.
4. Eclipse Foundation. EMF Compare. http://www.eclipse.org/modeling/emft/?project=compare.
5. J. Eder, W. Gruber, and H. Pichler. Transforming Workflow Graphs. In *INTEROP-ESA'05*, pages 203–214, Genf, Switzerland, 2 2005. Springer London.
6. M. Ehrig, A. Koschmider, and A. Oberweis. Measuring Similarity between Semantic Business Process Models. In J. F. Roddick and A. Hinze, editors, *APCCM*, volume 67 of *CRPIT*, pages 71–80. Australian Computer Society, 2007.
7. T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.
8. C. Gerth, J. M. Küster, and G. Engels. Language-Independent Change Management of Process Models. In A. Schürr and B. Selic, editors, *MODELS'09*, volume 5795 of *LNCS*, pages 152–166. Springer, 2009.
9. International Business Machines Corp. (IBM). IBM WebSphere Business Modeler. http://www.ibm.com/software/integration/wbimodeler/.
10. G. Kappel, E. Kapsammer, H. Kargl, G. Kramler, T. Reiter, W. Retschitzegger, W. Schwinger, and M. Wimmer. Lifting Metamodels to Ontologies: A Step to the Semantic Integration of Modeling Languages. In O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, editors, *MoDELS*, volume 4199 of *LNCS*, pages 528–542. Springer, 2006.
11. U. Kelter, J. Wehren, and J. Niere. A Generic Difference Algorithm for UML Models. In P. Liggesmeyer, K. Pohl, and M. Goedicke, editors, *Software Engineering 2005*, volume 64 of *LNI*, pages 105–116. GI, 2005.
12. J. Koehler, R. Hauser, J. Küster, K. Ryndina, J. Vanhatalo, and M. Wahler. The Role of Visual Modeling and Model Transformations in Business-Driven Development. In *Proceedings of GT-VMT'06*, pages 1–12, 2006.
13. J. M. Küster, C. Gerth, A. Förster, and G. Engels. Detecting and Resolving Process Model Differences in the Absence of a Change Log. In M. Dumas and M. Reichert, editors, *BPM'08*, volume 5240 of *LNCS*, pages 244–260. Springer-Verlag, 2008.
14. C. Li, M. Reichert, and A. Wombacher. On Measuring Process Model Similarity Based on High-Level Change Operations. In Q. Li, S. Spaccapietra, E. S. K. Yu, and A. Olivé, editors, *ER*, volume 5231 of *Lecture Notes in Computer Science*, pages 248–264. Springer, 2008.
15. J. Mendling and W. M. P. van der Aalst. Formalization and Verification of EPCs with OR-Joins Based on State and Context. In J. Krogstie, A. L. Opdahl, and G. Sindre, editors, *CAiSE*, volume 4495 of *LNCS*, pages 439–453. Springer, 2007.
16. T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.
17. Object Management Group (OMG). Business Process Modeling Notation (BPMN). http://www.omg.org/spec/BPMN/1.2.
18. Object Management Group (OMG). Unified Modeling Language (UML): Superstructure. http://www.uml.org, 2005.
19. W. Sadiq and M. E. Orlowska. Analyzing Process Models Using Graph Reduction Techniques. *Inf. Syst.*, 25(2):117–134, 2000.
20. W. M. P. van der Aalst, A. K. A. de Medeiros, and A. J. M. M. Weijters. Process Equivalence: Comparing Two Process Models Based on Observed Behavior. In S. Dustdar, J. L. Fiadeiro, and A. P. Sheth, editors, *BPM'09*, volume 4102 of *LNCS*, pages 129–144. Springer, 2006.
21. W. M. P. van der Aalst, A. Hirnschall, and H. M. W. Verbeek. An Alternative Way to Analyze Workflow Graphs. In A. Banks Pidduck, J. Mylopoulos, C. C. Woo, and M. Tamer Özsu, editors, *CAiSE*, volume 2348 of *LNCS*, pages 535–552. Springer, 2002.

22. B. F. van Dongen, R. M. Dijkman, and J. Mendling. Measuring Similarity between Business Process Models. In Z. Bellahsene and M. Léonard, editors, *CAiSE*, volume 5074 of *Lecture Notes in Computer Science*, pages 450–464. Springer, 2008.
23. B. F. van Dongen, W. M. P. van der Aalst, and H. M. W. Verbeek. Verification of EPCs: Using Reduction Rules and Petri Nets. In Oscar Pastor and João Falcão e Cunha, editors, *CAiSE*, volume 3520 of *Lecture Notes in Computer Science*, pages 372–386. Springer, 2005.
24. R.J. van Glabbeek. The Linear Time-Branching Time Spectrum I - The Semantics of Concrete, Sequential Processes. In *Handbook of Process Algebra, Chapter 1*, pages 3–99. Elsevier.
25. J. Vanhatalo, H. Völzer, and F. Leymann. Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition. In *ICSOC 2007*, volume 4749 of *LNCS*, pages 43–55. Springer, 2007.
26. M. Weidlich, M. Weske, and J. Mendling. Change Propagation in Process Models Using Behavioural Profiles. In *IEEE SCC*, pages 33–40. IEEE Computer Society, 2009.