RZ 3703 (# 99713) 02/18/08
Computer Science 15 pages

Research Report

Process Merging in Business-Driven Development

Jochen M. Küster[1], Christian Gerth[1,2], Alexander Förster[2], and Gregor Engels[2]

[1]IBM Zurich Research Laboratory, Säumerstr. 4, 8803 Rüschlikon, Switzerland
{jku,cge}@zurich.ibm.com

[2]Department of Computer Science, University of Paderborn, Germany
{gerth,alfo,engels}@upb.de

Process Merging in Business-Driven Development

Jochen M. Küster[1], Christian Gerth[1,2], Alexander Förster[2], and Gregor Engels[2]

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

Abstract. Business-driven development favors the construction of process models at different abstraction levels and by different people. As a consequence, there is a demand for consolidating different versions of process models by merging them. Process merging can be considered as a special case of model composition. However, in order to be applicable by a business user, process merging has to fulfill specific requirements such as user-friendliness and minimal manual intervention. In this paper, we present our approach to process merging. It is based on calculating differences and ordering them according to the underlying structure of process models. This allows to resolve differences in a particular user-friendly way by e.g. automating reconnection of inserted process elements.

1 Introduction

The field of business process modeling has a long standing tradition. Recently, new requirements and opportunities have been identified which allow the tighter coupling of business process models to its underlying IT implementation: In Business-Driven Development (BDD) [13], business process models are iteratively refined, from high-level business process models into models that can be directly implemented. Business process models can therefore be used for creating a link between the business needs and the IT implementation. As a consequence, business process models are a key artifact in BDD and advanced techniques for consolidating different versions of a process model are needed.

In general, such techniques for consolidating and merging process models have to provide means for identifying differences between versions of process models and resolving these by merging parts of process models. Specific techniques for process merging heavily depend on the underlying modeling environment. Our approach is directed towards a situation where no change log describing process model changes exists. This is a common situation in process modeling tools such as the IBM WebSphere Business Modeler [1]. In such a case, detection of differences between process models and their user-friendly visualization represent key requirements for process merging.

Existing work on process change management has focused mainly on the question of dynamic process changes where changes are made on already running processes [3, 18]. Solutions include techniques for migrating process instances to a new process schema and for identifying those cases where this is not possible. In these approaches, process changes are usually captured in a change log which is maintained by the process-aware information system [4].

Merging of process models has similarities to model composition [7] in model-driven development. It includes a structural and a behavioral aspect: With regards to structure, new process model elements must be integrated into the process structure. Here, one has to decide where in the hierarchy of a process model the new elements shall reside. With regards to behavior, the new elements must be integrated into the control or data flow of the existing business process model. Although this can be done manually, this involves quite some overhead when applied for a large number of model elements. In addition, often the new elements have certain ordering requirements that should be ensured when inserting them into the flow.

In this paper, we present our approach to process merging which consists of detection and resolution of differences. Detection makes use of the concept of correspondences, well-known from model merging and model composition, but enriched with the technique of Single-Entry-Single Exit fragments (SESE fragments) for detecting different categories of differences. The result of the detection is a list of differences which can be considered as a change log: Each difference is associated with a difference resolution operation that resolves this particular difference. The paper is structured as follows: Section 2 introduces process merging in BDD and describes the key requirements. Then, in Section 3, we discuss the foundations for our approach, correspondences and SESE fragments. In Section 4, we present our approach for difference detection and in Section 5 our approach to difference resolution is described. A prototype realized as a plug-in for IBM WebSphere Business Modeler is presented in Section 6. We conclude with a discussion of related and future work.

2 A Process Merging Scenario in Business-Driven Development

Business-driven development provides a model-driven approach to business-IT alignment. We distinguish between *analysis* and *design models* of business processes [11]—a distinction which is also made in object-oriented modeling. An analysis model describes what the process is doing. It shows the initial partitioning of the process into subprocesses and actions with the main flow of control and, optionally, of data. It completely abstracts from IT-related aspects, but can be used for simulation and discussion with business analysts. A design model contains a refined partitioning of the process that reflects existing application systems and shows an IT-based flow of data and control and it describes how the process is realized using hardware, software, and people.

Within business-driven development, process models are the central modeling artifacts. In this context, business process models are manipulated in a team environment and multiple versions of a shared process model need to be consolidated at some point in time. A basic scenario is obtained when a process model V_1 is copied and then changed into a process model V_2, possibly by another person. After completion, only some of the changes shall be applied to the original model V_1 to create a consolidated process model. Figure 1 shows an example process model V_1 that has been changed into a process model V_2.

Both models describe the handling of a claim request by an insurance company. V_1 starts with an *InitialNode* followed by the *actions "Check Claim"* and *"Record Claim"*. Then, in the *Decision*, it is decided whether the claim is covered by the insurance contract or not. In the case of a positive result the claim is settled. In the other case the claim is rejected and closed, represented by the *actions "Reject Claim"* and *"Close Claim"*.

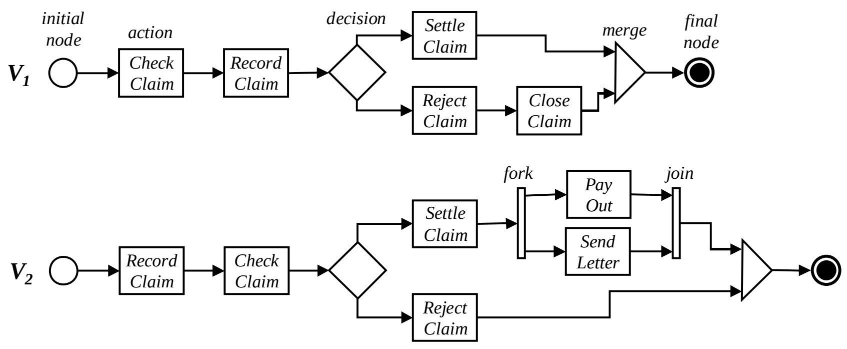


Fig. 1. Versions V_1 and V_2 of a business process model

Although process models V_1 and V_2 are similar at the first sight, there are some differences between the versions. The following differences can be detected:

- The positions of the *actions "Record Claim"* and *"Check Claim"* are changed.
- *Action "Close Claim"* does not exist in V_2.
- A new parallel structure (*Fork* and *Join*) is inserted in V_2 together with two *actions "Pay Out"* and *"Send Letter"*.

Process merging typically depends on the modeling language as well as on constraints of the modeling environment. In our case, the modeling language is given by the WebSphere Business Modeler which provides a language similar to UML 2.0 Activity Diagrams [15]. In our modeling environment, no syntax-directed editing of process models is performed and, as a consequence, also no change log is available. As such, in contrast to databases and existing approaches in process-aware information systems, there is no information about the performed changes on a process model. In the following, we describe the key requirements that a solution to process merging should fulfill:

- The solution must provide a technique to re-construct one possible change log which represents the transformation steps for transforming one process model into the other process model.
- The user should have the opportunity to select only some of the changes and apply it to the original model in order to obtain a new third model which can be considered as the merged process model.
- When applying changes, the user should not be restricted by prescribing a certain order whenever possible.
- Dependencies between change operations should be made explicit and taken into account when applying the changes. For example, when inserting a *Fork*, the corresponding *Join* should also be inserted in order to obtain a correct process model.

– The solution should provide user-friendly resolution of changes in the way that it reconnects inserted elements whenever possible and offers a possibility to perform related changes together at one time.
– Instead of displaying a list of changes, the changes should be arranged to the structure of the process model.

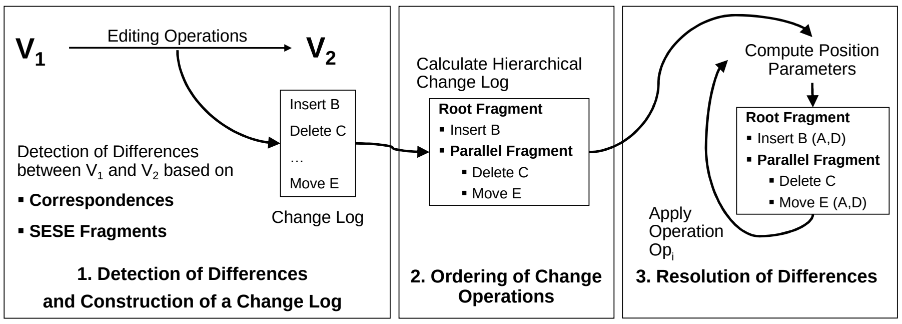


Fig. 2. Overview of our process merging approach

Figure 2 provides an overview of the approach that we developed based on these requirements: The first step is to detect differences between the two process models. This detection makes use of SESE fragments for detecting related differences. In the second step, the detected differences are ordered according to the structure of the process models. For each difference, a resolution transformation is generated which resolves the difference between the two models. The third step is then to resolve differences between the process models in an iterative way, based on the modeler's preferences.

3 Background: Correspondences and SESE Fragments

In this section, we first define business process models and introduce Single-Entry-Single-Exit fragments. Then we introduce correspondences.

3.1 Process Models and SESE Fragments

For our process merging approach we use process models in a notation similar to activity diagrams in UML. For the following discussions, we assume a business process model $V = (N, E)$ consisting of a finite set N of nodes and a relation E representing control flow. N is partitioned into sets of *Actions* and *ControlNodes*. *ControlNodes* contain Decision and Merge, Fork and Join, InitialNodes and FinalNodes. Note that we do not support data flow in our models. In addition, we assume that the following constraints hold:

1. *Actions* have exactly one incoming and one outgoing edge.
2. Nodes are connected in such a way that each node is on a path from the InitialNode to the FinalNode.

3. Control flow splits and joins are modeled explicitly with the appropriate *ControlNode*, e.g. *Fork*, *Join*, *Decision*, or *Merge*. *ControlNodes* have either exactly one incoming and at least two outgoing edges (*Fork*, *Decision*) or at least two incoming and exactly one outgoing edge (*Join*, *Merge*).
4. An *InitialNode* has no incoming edge and at most one outgoing edge and a *FinalNode* has at most one incoming edge and no outgoing edge.

A process model can be decomposed into SESE fragments and then a process structure tree can be constructed [21]. SESE fragments have been used successfully for checking soundness [20, 16] of process models but they are also beneficial for detection of differences between process models. Formally, given a process model $V = (N, E)$, a SESE fragment $F = (N', E')$ is a non-empty subgraph of V such that there exists edges $e, e' \in E$ with $E \cap ((N \setminus N') \times N') = \{e\}$ and $E \cap (N' \times (N \setminus N')) = \{e'\}$ [21]. In addition, a fragment which surrounds the entire process model is also considered as a SESE fragment. To this unique fragment we refer to as *root fragment*.

A process model with N nodes can have $O(N^2)$ SESE fragments. We are interested in so-called canonical fragments that are not overlapping [21]. Given a process model V, we denote the set of canonical fragments of V with $SESE(V)$. Figure 3 shows an example of a SESE decomposition into canonical fragments.

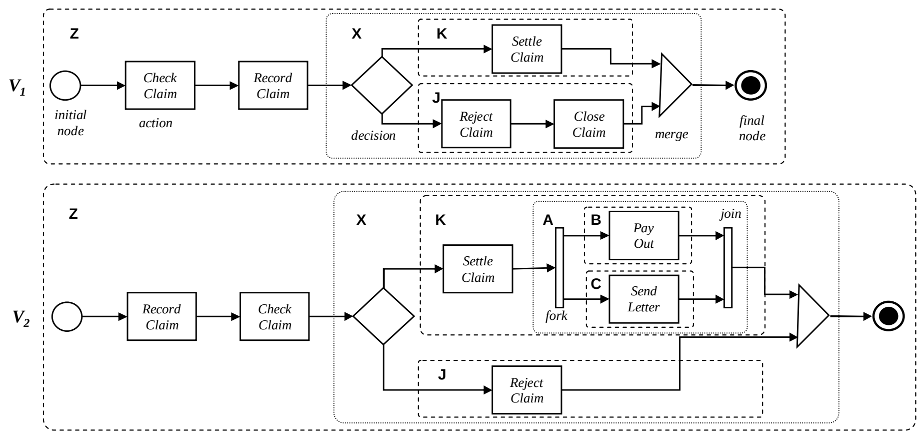


Fig. 3. Versions V_1 and V_2 decomposed into canonical SESE fragments

The canonical fragments of a process model V can be organized into a SESE tree, denoted by $SESE_{TREE}(V)$, according to the composition hierarchy of the fragments. Fragments of $SESE(V)$, which contain no other fragments, are leaf nodes in the tree and the root of the tree is the root fragment.

3.2 Correspondences

Correspondences form one key technique in process merging because they provide the link between elements in different process models. We assume process models $V_1 = (N_1, E_1)$ and $V_2 = (N_2, E_2)$ and $x \in V_1$ and $y \in V_2$ as given. A correspondence is used to express that a model element x has a counterpart y with the same functionality in the other version. In such a case, we introduce a 1-to-1 correspondence between them. In the case that a model element x does not have a counterpart with the same functionality, we speak of a 1-to-0 correspondence. In case that y does not have a counterpart, we speak of a 0-to-1 correspondence. In addition, refinement of an element into a set of elements would give rise to a 1-to-many correspondence and abstraction of a set of elements into one element would give rise to a many-to-1 correspondence. The last two types are not considered further in this paper.

We express a 1-to-1 correspondence by inserting the tuple (x, y) into the set of correspondences $Corr(V_1, V_2) \subseteq V_1 \times V_2$. We further introduce the set of elements in V_1 which do not have a counterpart and denote this set by $Corr_{1-0}(V_1, V_2)$. Similarly, we denote the set of elements in V_2 without counterparts as $Corr_{0-1}(V_1, V_2)$.

Correspondences can be computed in the process merging scenario in a straightforward way by first establishing 1-to-1 correspondences between all process model elements when copying process model V_1 to create V_2. After changing V_2, all 1-to-1 correspondences have to be inspected and 1-to-0 or 0-to-1 correspondences are created if elements have been deleted or added. In addition, for new elements in V_2, additional 0-to-1 correspondences have to be created. In a similar way, correspondences can also be established for SESE fragments.

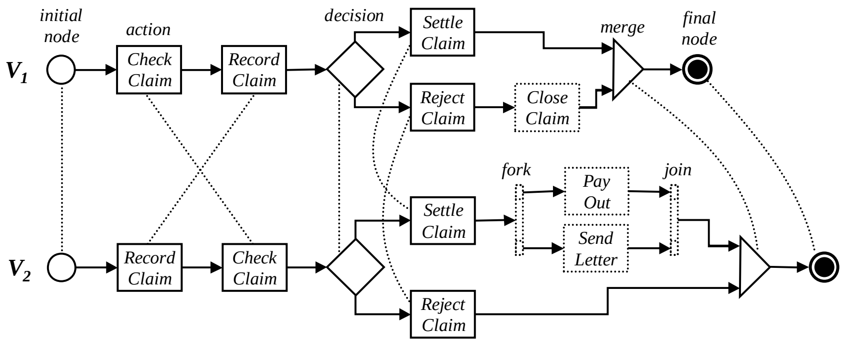


Fig. 4. Correspondences between V_1 and V_2

Fig. 4 shows correspondences between versions V_1 and V_2 of the process model introduced earlier in this paper. A dotted line represents 1-to-1 correspondences and connects model elements with the same functionality between V_1 and V_2. Dotted model elements have no counterpart with the same functionality in the other version. 1-to-0 correspondences are visualized by dotted elements in V_1 and 0-to-1 correspondences are visualized by dotted elements in V_2.

4 Detection of Differences and Computation of a Change Log

In this section, we describe an approach to detect differences between process models, based on the existence of correspondences and SESE fragments. We first present a classification of possible difference types and then discuss how each type can be systematically detected.

4.1 Element and Control Flow Differences

When comparing the sets of elements in both process models, based on their correspondences, one can define element differences as follows:

Definition 1 (Element difference). *Given two business process models V_1, V_2 and sets of correspondences $Corr_{1-0}(V_1, V_2)$ and $Corr_{0-1}(V_1, V_2)$, an element difference is defined as an element $x \in Corr_{1-0}(V_1, V_2) \cup Corr_{0-1}(V_1, V_2)$.*

In addition to element differences, different versions of process models can also be constructed by exchanging the order of actions or by introducing or deleting control nodes. For defining this, we assume predecessor and successor relations between elements of a process model that can be obtained from the control flow. This gives rise to control flow differences as follows:

Definition 2 (Simple Control Flow Difference). *Given two business process models V_1 and V_2 and sets of correspondences $Corr(V_1, V_2)$, a simple control flow difference is defined as a tuple of elements $(x, x', y, y') \in Corr(V_1, V_2) \times Corr(V_1, V_2)$, such that*

- *x is predecessor of y in V_1 and x' is successor of y' or unordered to y' in V_2, or*
- *y is successor of x in V_1 and y' is predecessor of x' or unordered to x' in V_2, or*
- *x is unordered to y in V_1 and x' is predecessor or successor of y' in V_2.*

We can distinguish between two types of simple control flow differences: *Intra-fragment* control flow differences are those where elements belong to corresponding SESE fragments. *Inter-fragment* control flow differences are those where elements have been moved between SESE fragments. The detection of inter-fragment control flow differences can be done by iterating over all 1-to-1 correspondences and checking whether the surrounding SESE fragments are also in a 1-to-1 correspondence. If this is not the case, then the element has been moved and is considered as an inter-fragment control flow difference. The detection of intra-fragment control flow differences has to compare all elements within a fragment with the elements in the corresponding fragment and identify changes in the order of elements.

In contrast to simple control flow differences, complex control flow differences are those that occur due to the insertion or deletion of SESE fragments:

Definition 3 (Complex Control Flow Difference). *Given two business process models V_1 and V_2 and a set of correspondences, a complex control flow difference is defined as either a SESE fragment $f_1 \in SESE(V_1)$ or a SESE fragment $f_2 \in SESE(V_2)$ that does not have a correspondence.*

Complex control flow differences can be detected by identifying fragments in the two process models that do not have corresponding fragments. For identifying these, one has to iterate over all fragments and identify those without a counterpart. Note that when copying process model V_1 to create V_2 also these correspondences between SESE fragments can be established. While editing V_2 and changing it, it can happen that either new fragments are created or existing fragments are deleted. For example, in Figure 3, a *Fork* and *Join* have been added in V_2 and thereby a new SESE fragment has been created without a corresponding fragment in V_1.

4.2 Change Operations derived from Differences

Based on the differences described above, we can define operations that resolve each type of difference. Each difference in the sets of elements can be directly converted into a suitable *Insert* or *Delete* operation. Each complex control flow difference can be converted into a *InsertFragment* or *DeleteFragment* operation. A simple control flow difference gives rise to either a *Move* operation within a fragment or between fragments. An overview of the operations and their effect on a process model is given in Figure 5. The benefit of using explicit operations for fragments such as *InsertFragment* is that we can ensure that always a correct process model is created. For example, it cannot happen that a *Fork* is inserted without its corresponding *Join*.

Change Operation applied on V	Effects on Process Model V
Insert(V,X,A,B)	Insertion of element **X** between two succeeding elements **A** and **B** in process model **V**.
InsertParallelFragment(V,A,B,M)	Insertion of a parallel fragment (Fork and Join) between two succeeding elements **A** and **B** in process model **V.** Mapping **M** specifies the number and positions of branches in the parallel fragment.
InsertAlternativeFragment(V,A,B,M)	Insertion of an alternative fragment (Decision and Merge) between two succeeding elements **A** and **B** in process model **V.** Mapping **M** specifies the number and positions of branches in the alternative fragment.
Delete(V,X)	Deletion of element **X** from process model **V**.
DeleteParallelFragment(V)	Deletion of a parallel region (Fork and Join) from process model **V**.
DeleteAlternativeFragment(V)	Deletion of an alternative region (Decision and Merge) from process model **V**.
Move(V,X,A,B)	Movement of element **X** between two succeeding elements **A** and **B** in process model **V**.

Fig. 5. Overview of change operations

In the case of versions V_1 and V_2 of the example introduced earlier in this paper, the differences give rise to the following change operations:

- Move(V_1,"Check Claim", _, _)
- InsertParallelFragment(V_1, _, _)
- Insert(V_1, "Pay Out", _, _)
- Insert(V_1, "Send Letter", _, _)
- Delete(V_1, "Close Claim")

The *Insert* and *Move* operations given here are only incompletely specified. The missing parameters comprise information where the underlying model element shall be inserted or moved to in the process model V_1. These parameters are computed later based on unchanged model elements in order to achieve an arbitrary resolution of differences.

4.3 Definition of Hierarchical Change Log

A set of change operations contains all change operations for two business process models V_1 and V_2 and is denoted by $Changes(V_1, V_2)$. In the following, we assume that $Changes(V_1, V_2)$ contain the changes that have to be applied to V_1 for obtaining V_2.

In order to enable user-friendly resolution of changes, change operations can be visualized according to the structure of the two process models. This structure of a process model is given by the decomposition of a process model into its SESE fragments. Given such a fragment decomposition, each operation can be associated to the fragment in which it occurs. This enables the user to detect operations applicable to a certain area of a process model and also allows to find dependencies between different operations.

In the following, we first define a joint SESE tree as a basis of such a hierarchical change log. Given two SESE trees $SESE_{TREE}(V_1)$, $SESE_{TREE}(V_2)$ and correspondences between their nodes, then the joint SESE tree is denoted as $SESE_{TREE}(V_1, V_2)$. We also assume a function $father : V \longrightarrow V$ which maps a node $v \in V$ to its father in the SESE tree. The joint SESE tree can be constructed as follows:

- for a node $v_1 \in SESE_{TREE}(V_1)$ that has a corresponding node $v_2 \in SESE_{TREE}(V_2)$ a node v_3 is inserted into $SESE_{TREE}(V_1, V_2)$ with $father(v_3) = father(v_2)$.
- for a node $v_1 \in SESE_{TREE}(V_1)$ that does not have a corresponding node, a node v_3 is inserted into $SESE_{TREE}(V_1, V_2)$ with $father(v_3) = father(v_1)$,
- for a node $v_2 \in SESE_{TREE}(V_2)$ that does not have a corresponding node, a node v_3 is inserted into $SESE_{TREE}(V_1, V_2)$ with $father(v_3) = father(v_2)$.

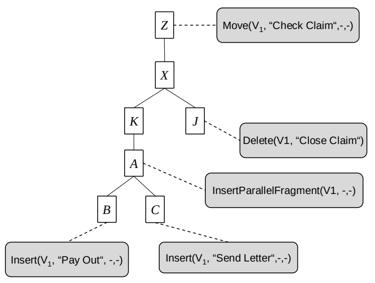
The following definition introduces the concept of a hierarchical change log where change operations are arranged according to the joint SESE tree:

Definition 4 (Hierarchical Change Log). *Given two business process models V_1, V_2, SESE trees $SESE_{TREE}(V_1)$, $SESE_{TREE}(V_2)$, and a set of change operations $Changes(V_1, V_2)$, a hierarchical change log for transforming V_1 into V_2 is the joint SESE structure tree $SESE_{TREE}(V_1, V_2)$ where the nodes are enriched with change operations as follows:*

- *if $op = Insert$ then op is associated to the node representing the fragment in which op takes place,*
- *if $op = Delete$ then op is associated to the node representing the fragment in which op takes place,*
- *if $op = Move$ then op is associated to the node representing the fragment into which the element is moved and to the node representing the fragment out of which the element is moved.*
- *if $op = InsertFragment$ or $op = DeleteFragment$, then op is associated to the node in $SESE_{TREE}(V_1, V_2)$ representing this fragment.*

Given process models V_i, V_j and change operation op, we write $V_i \overset{op}{\Longrightarrow} V_j$ if V_j is obtained after applying op on V_i.

Figure 6 shows a hierarchical change log for the two versions V_1 and V_2 of a process model introduced earlier in this paper. The *Move* operation is associated with the root fragment. The *InsertParallelFragment* occurs in the alternative fragment and is therefore associated to this fragment. Within this parallel fragment, there are two *Insert* operations. Using this hierarchical change log, one can easily identify the areas of the process model that have been manipulated. The hierarchical change log also allows the user to find dependencies between the insertion of the parallel fragment and the insertions of actions into that parallel fragment.


Z
Move(V1, “Check Claim“,-,-)
X
K
J
Delete(V1, “Close Claim“)
A
InsertParallelFragment(V1, -,-)
B
C
Insert(V1, “Pay Out“, -,-)
Insert(V1, “Send Letter“,-,-)
Fig. 6. Hierarchical change log of example

In the next section, we elaborate on the resolution of differences using the change operations introduced.

5 Resolution of Differences

In this section, we first explain how position parameters of change operations are computed. Then we show how the change operations can be used for iteratively resolving differences between two versions.

5.1 Computation of Position Parameters

According to our requirements, differences between two versions of a process models should be resolvable in an arbitrary way. Whether two operations can be applied in an arbitrary order depends on the positions parameters.

Figure 7 shows a simple example for illustrating why position parameters need to be chosen in a careful way. In this example, two actions A_3 and A_4 have been inserted, leading to *insert*$(V_1, A_3, -, -)$ and *insert*$(V_1, A_4, -, -)$. In order to ensure that the user can choose both operations, we compute position parameters to be *insert*(V_1, A_3, A_1, A_2) and *insert*(V_1, A_4, A_1, A_2). If either A_3 or A_4 were position parameters, this would induce a dependency between them, requiring that one of them is applied before the other one. In order to avoid such situations, we express position parameters using *fixpoints* which are defined as follows:

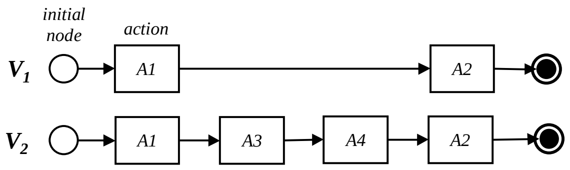


Fig. 7. Simple example

Definition 5 (Fixpoint). *Given two business process models V_1 and V_2 and a set of correspondences $Corr(V_1, V_2)$. Then a tuple (n_1, n_2) is a fixpoint if $(n_1, n_2) \in Corr(V_1, V_2)$ and n_1 and n_2 are not affected by an operation in the change log.*

Given a change operation, the computation of fixpoints is divided into the calculation of a fixpoint predecessor and a fixpoint successor for the underlying model element *n* of an insert or move operation within the fragment of *n*. In order to compute a fixpoint predecessor for *n*, we start with the direct predecessors *p* of *n* in V_2. If *p* is a fixpoint, i.e. *p* is not affected by an operation in the change log, we have found the nearest fixpoint predecessor of *n*. Otherwise, we examine the direct predecessor of *p* and so on until we have found a fixpoint. The computation of a fixpoint successor of *n* is analog to the computation of a predecessor, except that we examine the successors of *n* in control flow order starting with the direct successor *s* of *n*.

Using fixpoints as position parameters also ensures that the insert and move operations can always produce a model that is connected: As the elements specified as position parameters are fixpoints, they exist in both process models and the newly inserted or moved element or fragment can be connected to these elements automatically.

Figure 8 shows the hierarchical change log with computed position parameters. For both insert operations (”Pay Out” and ”Send Letter”) within the newly inserted parallel fragment, position parameters could not be computed, because their surrounding fragment is not existing so far. In such a case, we assume that the operation is not yet applicable.

Root Fragment
- **Move**(V_1,”Check Claim”, ”Record Claim”, Decision)
- Alternative Fragment
  - Branch
    - **InsertParallelFragment**(V_1, ”Settle Claim”, Merge)
      - Branch
        - ***Insert****(V_1, ”Pay Out”, _, _)*
      - Branch
        - ***Insert****(V_1, ”Send Letter”, _, _)*
  - Branch
    - **Delete**(V_1, ”Close Claim”)

Fig. 8. Position parameters of operations

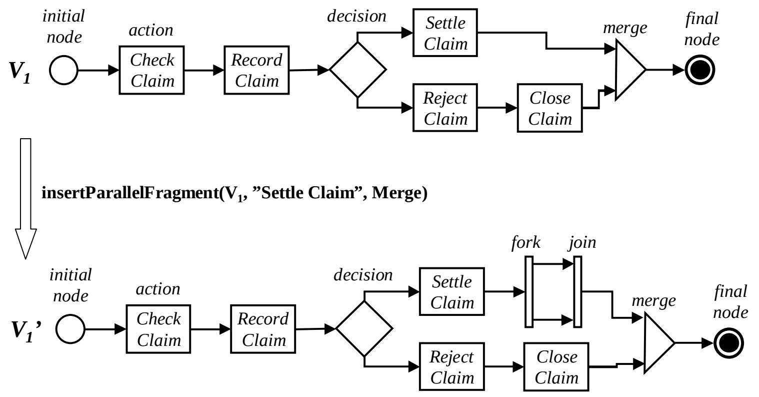


Fig. 9. Applying a change operation

5.2 Application of Operations

The operations in the change log with position parameters are ready for application. Figure 9 shows V_1 and the application of the *InsertParallelFragment* operation, leading to the insertion of the *Fork* and *Join* and the automated reconnection of control flow. Alternatively, the user could have chosen any other operation of the change log that is applicable.

The application of an insert or move operations increases the set of fixpoints between two process models because the underlying model element of the applied operation exists now in both process models. By recomputing the position parameters of the remaining operations we can increase the number of applicable operations and refine existing position parameters. In the example, the position parameters of the *Insert(V_1,"Pay Out",-,-)* and *Insert(V_1,"Send Letter",-,-)* operations can be computed after inserting the parallel fragment operation. This leads to a new change log which is shown in Figure 10.

Root Fragment
- **Move**(V_1',"Check Claim", "Record Claim", Decision)
- Alternative Fragment
  - Branch
    - ParallelFragment
      - Branch
        - ***Insert***(*V_1', "Pay Out", Fork, Join*)
      - Branch
        - ***Insert***(*V_1', "Send Letter", Fork, Join)*
  - Branch
    - **Delete**(V_1', "Close Claim")

Fig. 10. Recomputed change log after applying an operation

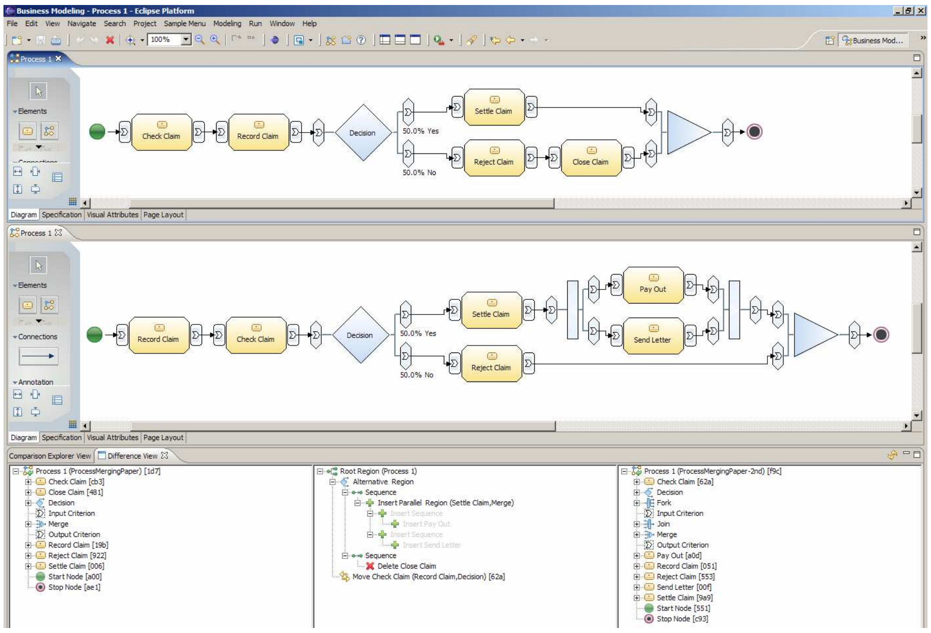
6 Tool Support

As proof of concept, we have implemented a prototype as an extension to the IBM WebSphere Business Modeler (see Fig. 11). This prototype currently supports the following functionality [6]: copying of business process models, initial creation and update of correspondences, decomposition of process models into SESE fragments and detection of differences between two versions of a process model. In addition, the prototype provides several views that allow to visualize and resolve differences as well as to manipulate correspondences.

Fig. 11 shows versions V_1 and V_2 of the business process model introduced earlier in this paper. The lower third of Fig. 11 illustrates the Difference View, which is divided into three columns. The left and right hand columns show versions V_1 and V_2 of the process model in a tree view, which abstracts from control flow details of the process and focuses only on model elements of the process. The middle column of the difference view displays the hierarchical change log as introduced previously. Using our prototype, differences between the two versions can be iteratively resolved using the change operations introduced in this paper.

7 Related Work

Within the workflow community, the problem of migrating existing workflow instances to a new schema [3, 17] has received considerable attention: Given a process schema



```
Process 1 (ProcessMergingPaper) [1d7]
    Check Claim [cb3]
    Close Claim [481]
    Decision
    Input Criterion
    Merge
    Output Criterion
    Record Claim [19b]
    Reject Claim [922]
    Settle Claim [006]
    Start Node [a00]
    Stop Node [ae1]
```

```
Root Region (Process 1)
    Alternative Region
        Sequence
            Insert Parallel Region (Settle Claim,Merge)
                Insert Sequence
                    Insert Pay Out
                Insert Sequence
                    Insert Send Letter
        Sequence
            Delete Close Claim
    Move Check Claim (Record Claim,Decision) [62a]
```

```
Process 1 (ProcessMergingPaper-2nd) [f9c]
    Check Claim [62a]
    Decision
    Fork
    Input Criterion
    Join
    Merge
    Output Criterion
    Pay Out [a0d]
    Record Claim [051]
    Reject Claim [553]
    Send Letter [00f]
    Settle Claim [9a9]
    Start Node [551]
    Stop Node [c93]
```

Fig. 11. Business Process Merging Prototype in the IBM WebSphere Business Modeler

change, it can be distinguished between process instances that can be migrated and those that cannot be migrated [18]. Rinderle et al. [18] describe migration policies for the situation that both the process instance as well as the process type has been changed. They introduce a selection of change operations and examine when two changes are commutative, disjoint or overlapping. Recent work by Weber et al. [22] provides a comprehensive overview of possible change patterns that can occur when process models or process instances are modified. These change patterns are used for evaluating different process-aware information systems [4] with respect to change support. Grossmann et al. [8] show how two business processes can be integrated using model transformations after relationships have been established. Both the change operations by Rinderle et al. [18] and the change patterns for inserting, deleting and moving a process fragment are similar to our change operations. In contrast to the existing work, we describe an approach how to identify change operations in the case that no change log is available and how to use SESE fragments for ordering discovered change operations.

Nejati et al. [14] present an approach to matching and merging of statecharts. Matching comprises static matching of state names and behavioral matching based on the semantics of states and transitions and computes a correspondence relation. This correspondence relation is used for constructing a merged statechart by constructing the union of individual statecharts, taking into account corresponding elements. We believe that merging statecharts could also benefit from the techniques presented in this paper with regards to user-friendly visualization of differences.

In model-driven engineering, generic approaches for detecting model differences and resolving them have been studied [2, 12, 9, 10]. Alanen und Porres [2] present an algorithm that calculates differences of two models based on the assumption of unique identifiers. The computation of differences has similarities to our reconstruction of change operations, however, in our work, we aim at difference resolution for process models with minimal user interaction.

View integration has also been studied in the graph transformation domain. Engels et al. [5] specify views using graphs and use graph transformations for describing behavior of operations. A system model can then be obtained by integrating different views, based on view relations. Sabetzadeh and Easterbrook [19] propose an algebraic framework for merging views. In their approach, independently modeled views are first coupled using a connector view and then integrated based on the connector view. Although process merging can be considered as a special case of view integration, we believe that approaches fulfilling specific requirements as discussed in this paper are needed for applicability in practice.

8 Conclusion and Future Work

User-friendly process merging is a key technique for practical business-driven development. In this paper, we have first studied a basic scenario of process merging in BDD and established key requirements. We have then introduced our approach which comprises detection and resolution of differences. The detection of differences is based on correspondences between process models and also makes use of the concept of a SESE fragment decomposition of process models. This SESE decomposition enables the visualization of differences according to the structure of process models. The resolution of differences is performed in an iterative way, by applying change operations that automatically reconnect the control flow.

There are several directions for future work. The change operations are intended to preserve well-formedness and soundness of the process model which needs to be formally proven. Such a formal foundation would also enable us to reason about minimality of the computed change operations. In addition, our approach can be adapted to cover other process modeling languages. Future work will also include the elaboration of our approach for merging process models in a distributed environment. In those scenarios, the concept of conflict becomes important because one resolution can turn the other resolution non-applicable.

Acknowledgements: We would like to thank Jana Koehler and Olaf Zimmermann for their valuable feedback on an earlier version of this paper.

References

1. IBM WebSphere Business Modeler. http://www.ibm.com/software/integration/wbimodeler/.
2. M. Alanen and I. Porres. Difference and Union of Models. In P. Stevens, J. Whittle, and G. Booch, editors, *UML 2003*, volume 2863 of *LNCS*, pages 2–17. Springer, 2003.
3. F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow evolution. *Data Knowl. Eng.*, 24(3):211–238, 1998.
4. M. Dumas, W. M. P. van der Aalst, and A. H. M. ter Hofstede. *Process-Aware Information Systems*. Wiley, 2005.
5. G. Engels, R. Heckel, G. Taentzer, and H. Ehrig. A View-Oriented Approach to System Modelling Based on Graph Transformation. In M. Jazayeri and H. Schauer, editors, *ESEC/FSE '97*, volume 1301 of *LNCS*, pages 327–343. Springer, 1997.

6. C. Gerth. *Business Process Merging - An Approach based on Single-Entry-Single-Exit Regions*. Diplomarbeit, Universität Paderborn, October 2007.
7. M. Gervais, K. Engel, D. Kolovos, D. Touzet, Y. Shaham-Gafni, R. Paige, and J. Aagedal. MODELWARE Delivery 1.5 Model Composition: Definition of Model Composition Properties. http://www.modelware-ist.org/.
8. G. Grossmann, Y. Ren, M. Schrefl, and M. Stumptner. Behavior Based Integration of Composite Business Processes. In W. M. P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, *BPM 2005*, volume 3649 of *LNCS*, pages 186–204. Springer, 2005.
9. C. Herrmann, H. Krahn, B. Rumpe, M. Schindler, and S. Völkel. An Algebraic View on the Semantics of Model Composition. In D. H. Akehurst, R. Vogel, and R. F. Paige, editors, *ECMDA-FA 2007*, volume 4530 of *LNCS*. Springer, 2007.
10. 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, Fachtagung des GI-Fachbereichs Softwaretechnik, 8.-11.3.2005 in Essen*, volume 64 of *LNI*, pages 105–116. GI, 2005.
11. J. Koehler, R. Hauser, J. Küster, K. Ryndina, J. Vanhatalo, and M. Wahler. The Role of Visual Modeleling and Model Transformations in Business-Driven Development. In *Proceedings of the 5th International Workshop on Graph Transformations and Visual Modeling Techniques*, pages 1–12, 2006.
12. D. S. Kolovos, R. Paige, and F. Polack. Merging Models with the Epsilon Merging Language (EML). In O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, editors, *MoDELS 2006*, volume 4199 of *LNCS*, pages 215–229. Springer, 2006.
13. T. Mitra. Business-driven development. IBM developerWorks article, http://www.ibm.com/developerworks/webservices/library/ws-bdd, IBM, 2005.
14. S. Nejati, M. Sabetzadeh, M. Chechik, S. M. Easterbrook, and P. Zave. Matching and Merging of Statecharts Specifications. In *ICSE 2007*, pages 54–64. IEEE Computer Society, 2007.
15. Object Management Group (OMG). *UML 2.0 Superstructure Final Adopted Specification. OMG document pts/03-08-02*, August 2003.
16. F. Puhlmann and M. Weske. Investigations on Soundness Regarding Lazy Activities. In S. Dustdar, J. Fiadeiro, and A. Sheth, editors, *BPM 2006*, volume 4102 of *LNCS*, pages 145–160. Springer, 2006.
17. M. Reichert and P. Dadam. ADEPT$_{\text{flex}}$-Supporting Dynamic Changes of Workflows Without Losing Control. *J. Intell. Inf. Syst.*, 10(2):93–129, 1998.
18. S. Rinderle, M. Reichert, and P. Dadam. Disjoint and Overlapping Process Changes: Challenges, Solutions, Applications. In R. Meersman and Z. Tari, editors, *CoopIS'04*, volume 3290 of *LNCS*, pages 101–120. Springer, 2004.
19. M. Sabetzadeh and S. Easterbrook. An Algebraic Framework for Merging Incomplete and Inconsistent Views. In *13th IEEE International Requirements Engineering Conference*, pages 306–318. IEEE Computer Society, September 2005.
20. W. M. P. van der Aalst. The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.
21. 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.
22. B. Weber, S. Rinderle, and M. Reichert. Change Patterns and Change Support Features in Process-Aware Information Systems. In J. Krogstie, A. L. Opdahl, and G. Sindre, editors, *CAiSE'07*, volume 4495 of *LNCS*, pages 574–588. Springer, 2007.