Supporting Different Process Views through a Shared Process Model

Jochen Küster[1], Hagen Völzer[1], Cédric Favre[1], Moises Castelo Branco[2], and Krzysztof Czarnecki[2]

[1] IBM Research — Zurich, Switzerland
[2] Generative Software Development Laboratory, University of Waterloo, Canada

Abstract. Different stakeholders in the Business Process Management (BPM) life cycle benefit from having different views onto a particular process model. Each view can show, and offer to change, the details relevant to the particular stakeholder, leaving out the irrelevant ones. However, introducing different views on a process model entails the problem to synchronize changes in case that one view evolves. This problem is especially relevant and challenging for views at different abstraction levels. In this paper, we propose a *Shared Process Model* that provides different stakeholder views at different abstraction levels and that synchronizes changes made to any view. We present detailed requirements and a solution design for the Shared Process Model in this paper. Moreover, we also present an overview of our prototypical implementation to demonstrate the feasibility of the approach.

1 Introduction

A central point in the value proposition of BPM suites is that a business process model can be used by different stakeholders for different purposes in the BPM life cycle. It can be used by a business analyst to document, analyze or communicate a process. Technical architects and developers can use a process model to implement the business process on a particular process engine. These are perhaps the two most prominent uses of a process model, but a process model can also be used by a business analyst to visualize monitoring data from the live system, or by an end user of the system, i.e., a process participant, to understand the context of his or her participation in the process.

These different stakeholders would ideally share a single process model to collaborate and to communicate to each other their interests regarding a particular business process. For example, a business analyst and a technical architect could negotiate process changes through the shared model. The business analyst could initiate process changes motivated by new business requirements, which can then be immediately seen by the technical architect and forms the basis for him to evaluate and implement the necessary changes to the IT system. The technical architect may revise the change because it is not implementable in the proposed form on the existing architecture. Vice versa, a technical architect can also initiate and communicate process changes motivated from technical requirements, e.g., new security regulations, revised performance requirements, etc. In this way, a truly shared process model can increase the agility of the enterprise.

This appealing vision of a single process model that is shared between stakeholders is difficult to achieve in practice. One practical problem is that, in some enterprises, different stakeholders use different metamodels and/or different tools to maintain their version of the process model. This problem makes it technically difficult to conceptually share 'the' process model between the stakeholders (the BPMN -BPEL *roundtripping problem* is a known example). This technical problem disappears with modern BPM suites and the introduction of BPMN 2, as this single notation now supports modeling both business and IT-level concerns.

However, there is also an essential conceptual problem. We argue that different stakeholders intrinsically want different *views* onto the same process because of their different concerns and their different levels of abstraction. This is even true for parts that all stakeholders are interested in, e.g., the main behavior of the process. Therefore we argue that we need separate, stakeholder-specific views of the process that are kept *consistent* with respect to each other. Current tools do not address this problem. Either different stakeholders use different models of the same process, which then quickly become inconsistent, or they use the same process model, which then cannot reflect the needs of all stakeholders.

This problem is a variation of the coupled evolution problem [11] and the model synchronization problem [10]. Coupled evolution has been studied between metamodels and models but not for process models at different abstraction levels and in the area of model synchronization various techniques have been proposed. Put into this context, our research question is how process views at different abstraction levels can be kept consistent and changes can be propagated in both directions automatically in a way that is aligned with existing studies of requirements from practice. In this paper, we address this problem, present detailed requirements and a design to synchronize process views on different abstraction levels. The challenge for a solution arises from an interplay of a variety of possible interdependent process model changes and their translation between the abstraction levels. We also report on an implementation to substantiate that a solution is indeed technically feasible. An extended version of this paper is available as a technical report [14].

2 The Business-IT Gap Problem

In this section, we motivate our Shared Process Model concept. First we argue why we think that a single process model view is often not adequate for different stakeholders and we discuss how different views differ. We illustrate this issue by example of two prominent stakeholder views of a process: the business analysts view used for documentation, analysis and communicating requirements to IT and the IT view of a process that is used directly for execution. Then, we briefly argue that, with multiple views, we need a dedicated effort to keep them consistent.

2.1 Why we want different views

Since BPMN 2 can be used for both documentation and execution, why can't we use a single BPMN 2 model that is shared between business and IT? To study this question,

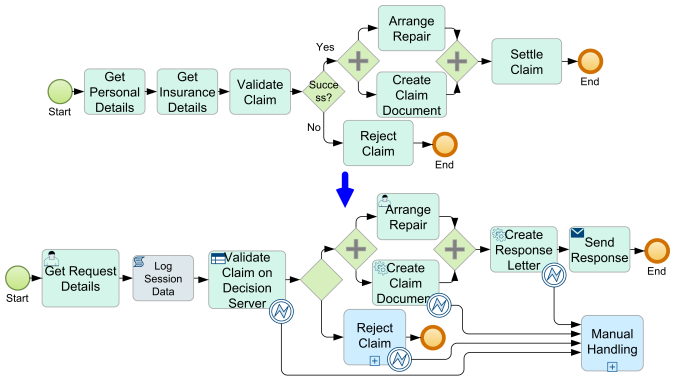

Fig. 1. Illustration of some refinements often made going from the business to the IT model

we analyzed the range of differences between a process model created by a business analyst and the corresponding process model that was finally used to drive the execution on a BPM execution engine. We built on our earlier study *et al.*[2], which analyzed more than 70 model pairs from the financial domain, and we also investigated additional model pairs from other domains. Additionally we talked to BPM architects from companies using process models to collect further differences. We summarize our findings here.

We identified the following categories of changes that were applied in producing an execution model from a business model. Fig. 1 illustrates some of these changes in a simplified claim handling process. We refer to our earlier study by Branco *et al.*[2] for a larger, more realistic example. Note that the following categorization of changes, based on a larger study, is a new contribution of this paper.

- *Complementary implementation detail.* Detail that is needed for execution is merely added to the business model, i.e., the part of the model that was specified by the business analyst does not change. Such details include data flow and -transformation, service interfaces and communication detail. For example, to specify the data input for an activity in BPMN 2, one sets a specific attribute of the activity that was previously undefined. The activity itself, its containment in a subprocess hierarchy and its connection with sequence flow does not change.
- *Formalization and renaming.* Some parts of the model need to be formalized further to be interpreted by an execution engine, including routing conditions, specialization of tasks (into service task, human task etc., see Fig. 1) and typing subprocesses (transaction, call) and typing events. Furthermore, activities are sometimes renamed by IT to better reflect some technical aspects of the activity. These are local, non-structural changes to existing model elements that do not alter the flow.
- *Behavioral refinement and refactoring.* The flow of the process is changed in a way that does not essentially change the behavior. This includes
  - *Hierarchical refinement/subsumption.* A high-level activity is refined into a sequence of low-level activities or more generally, into a subprocess with the same input/output behavior. For example, ‘Settle Claim’ in Fig. 1 is refined into ‘Create Response Letter’ and ‘Send Response’. The refining subprocess may or may not be explicitly enclosed in a separate scope (subprocess or call activity). If it is not enclosed in a separate scope, it

is represented as a subgraph which has, in most cases, a single entry and a single exit of sequence flow. We call such a subgraph a *fragment* in this paper.
On the other hand, multiple tasks on the business level may be subsumed in a single service call or a single human task to map the required business steps to the existing services and sub-engines (human task, business rules). For example, in Fig. 1, 'Get Personal Details' and 'Get Insurance Details' got subsumed into a single call 'Get Request Details' of the human task engine.

- *Hierarchical refactoring.* Existing process parts are separated into a subprocess or call activity or they may be outsourced into a separate process that is called by a message or event. Besides better readability and reuse, there are several other IT-architectural reasons motivating such changes. For example, performance, dependability and security requirements may require executing certain process parts in a separate environment. In particular, long-running processes are often significantly refactored under performance constraints. A long-running process creates more load on the engine than a short running process because each change need to be persisted. Therefore, short-running parts of long-running process are extracted to make the long-running process leaner.
- *Task removal and addition.* Sometimes, a business task is not implemented on the BPM engine. It may be not subject to the automation or it may already be partly automated outside the BPM system. On the other hand, some tasks are added on the IT level, that are not considered to be a part of an implementation of a specific business task. For example, a script task retrieving, transforming or persisting data or a task that is merely used for debugging purposes (e.g. 'Log Session Data' in Fig. 1).

- *Additional behavior.* Business-level process models are often incomplete in the sense that they do not specify all possible behavior. Apart from exceptions on the business-level that may have been forgotten, there are usually many technical exceptions that may occur that require error handling or compensation. This error handling creates additional behavior on the process execution level. In Fig. 1, some fault handling has been added to the IT model to catch failing service calls.
- *Correction and revision of the flow.* Some business-level process models would not pass syntactical and semantical validation checks on the engine. They may contain modeling errors in the control- or data flow that need to be corrected before execution. Sometimes activties also need to be reordered to take previously unconsidered data and service dependencies into account. These changes generally alter the behavior of the process. A special case is the possible parallelization of activities through IT, which may or may not be considered a behavioral change.

Different changes that occur in the IT implementation phase relate differently to the shared process model idea. Complementary detail could be easily handled by a single model through a *progressive disclosure* of the process model, i.e., showing one graphical layer to business and two layers to IT stakeholders.

However, the decision which model elements are 'business relevant' depends on the project and should not be statically fixed (as in the BPMN 2 conformance classes). Therefore, an implementation of progressive disclosure requires extensions that specify which element belongs to which layer. Additional behavior can be handled through progressive disclosure in a similar way as long as there are no dependencies to the business layer. For example, according to the BPMN 2 metamodel, if we add an error boundary event to a task with subsequent sequence flow specifying the error handling, then this creates no syntactical dependencies from the business elements to this addition. However, if we merge the error handling back to the normal flow through a new gateway

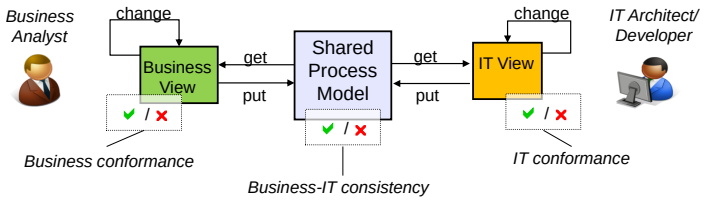

Fig. 2. Process view synchronization via a Shared Process Model

or if we branch off the additional behavior by a new gateway in the first place, then the business elements need to be changed, which would substantially complicate any implementation of a progressive disclosure. In this case, it would be easier to maintain two separate views. Also the changes in the categories *behavioral refinement and refactoring* as well as *formalization and renaming* clearly suggest to maintain two separate views.

Why different views need to be synchronized. In fact, many organizations today keep multiple versions of a process model to reflect the different views of the stakeholder (cf., e.g., [2, 18, 19]). However, because today's tools do not have any support for synchronizing them, they typically become inconsistent over time. That is, they disagree about which business tasks are executed and in which order. This can lead to costly business disruptions or to audit failures [2]. In the technical report version of this paper [14], we elaborate this point in more detail.

3 Requirements for a Shared Process Model

3.1 The Shared Process Model Concept

The *Shared Process Model*, which we now present, has the capability to synchronize process model views that reside on different abstraction levels. The concept is illustrated by Fig. 2. The Shared Process Model provides two different views, a business view and an IT view, and maintains the consistency between them. A current view can be obtained at any time by the corresponding stakeholder by the 'get' operation. A view may also be changed by the corresponding stakeholder. With a 'put' operation, the changed view can be checked into the Shared Proess Model, which synchronizes the changed view with the other view.

Each view change can be either designated as a *public* or a *private* change. A *public* change is a change that needs to be reflected in the other view whereas a *private* change is one that does not need to be reflected. For example, if an IT architect realizes, while he is working on the refinement of the IT model, that the model is missing an important business activity, he can insert that activity in the IT model. He can then check the change into the Shared Process Model, designating it as a public change to express that the activity should be inserted in the business view as well. The Shared Process Model then inserts the new activity in the business view automatically at the right position, i.e., every new business view henceforth obtained from the Shared Process Model will contain the new activity. If the IT architect designated the activity insertion as a private

change, then the business view will not be updated and the new activity will henceforth be treated by the Shared Process Model as an 'IT-only' activity.

Fig. 2 also illustrates the main three status conditions of a Shared Process Model: *business conformance, IT conformance and Business-IT consistency*. The business view is *business conformant* if it is approved by the business analyst, i.e., if it reflects the business requirements. This should include that the business view passes basic validity checks of the business modeling tool. The IT view is *IT conformant* if it is approved by the IT architect, i.e., if it meets the IT requirements. This should include that the IT view passes all validity checks of the IT modeling tool and the execution engine. *Business-IT consistency* means that the business view faithfully reflects the IT view, or equivalently, that the IT model faithfully implements the business view.

In the remainder of this section, we discuss the requirements and capabilities of the Shared Process Model in more detail.

3.2 Usage Scenarios and Requirements

We distinguish the following usage scenarios for the Shared Process Model. In the *presentation* scenario, either the business or IT stakeholder can, at any time, obtain a current state of his view with the 'get' operation. The view must reflect all previous updates, which may have been caused by either stakeholder.

The Shared Process Model is *initialized* with a single process model (the initial business view), i.e., business and IT views are initially identical. Henceforth, both views may evolve differently through *view change scenarios*, which are discussed below. For simplicity, we assume here that changes to different views do not happen concurrently. Concurrent updates can be handled on top of the Shared Process Model using known concurrency control techniques. That is, either a pessimistic approach is chosen and a locking mechanism prevents concurrent updates, which, we believe, is sufficient in most situations. Or an optimistic approach is chosen and different updates to the Shared Model may occur concurrently—but atomically, i.e., each update creates a separate new consistent version of the Shared Model. Parallel versions of the Shared Model must then be reconciled through a horizontal compare/merge technique on the Shared Model. Such a horizontal technique would be orthogonal to the vertical synchronization we consider here and out of scope of this paper.

In the *view change* scenario, one view is changed by a stakeholder and checked into the Shared Process Model with the 'put' operation to update the other view. A view change may contain many separate individual changes such as insertions, deletions, mutations or rearrangement of modeling elements. Each individual change must be designated as either private or public. We envision that often a new view is checked into the Shared Process Model which contains either only private or only public individual changes. These special cases simplify the designation of the changes. For example, during the initial IT implementation phase, most changes are private IT changes.

A private change only takes effect in one view while the other remains unchanged. Any public change on one view must be propagated to the other view in an automated way. We describe in more detail in Sect. 4, in what way a particular public change in one view is supposed to affect the other view. An appropriate translation of the change

is needed in general. User intervention should only be requested when absolutely necessary for disambiguation in the translation process. We will present an example of such a case in Sect. 4.

The designation of whether a change is private or public is in principle a deliberate choice of the stakeholder that changes his view. However, we imagine that governance rules are implemented that disallow certain changes to be private. For example, private changes should not introduce inconsistencies between the views, e.g., IT should not change the order of two tasks and hide that as a private change. Therefore, the business-IT consistency status need to be checked upon such changes.

The key function of the Shared Process Model is to maintain the consistency between business and IT view. Business-IT consistency can be thought of as a Boolean condition (consistent or inconsistent) or a measure representing a degree of inconsistency. According to our earlier study [2], the most important aspect is *coverage*, which means that (i) every element (e.g. activities and events) in the business view should be implemented by the IT view, and (ii) only the elements in the business view are implemented by the IT view.

The second important aspect of business-IT consistency is *preservation of behavior*. The activities and events should be executed in the order specified by the business view. The concrete selection of a consistency notion and its enforcement policy should be configurable on a per-project basis. A concrete notion should be defined in a way that users can easily understand, to make it as easy as possible for them to fix consistency violations. Common IT refinements as discussed in Sect. 2.1 should be compatible with the consistency notion, i.e., should not introduce inconsistencies, wheras changes that cannot be considered refinements should create consistency violations. Checking consistency should be efficient in order to be able to detect violations immediately after a change.

On top of the previous scenarios, support for change management is desirable to facilitate collaboration between different stakeholders through the Shared Process Model. The change management should support approving or rejecting public changes. In particular, public changes made by IT should be subject to approval by business. Only a subset of the proposed public changes may be approved. The tool supporting the approval of individual changes should make sure that the set of approved changes that is finally applied to the Shared Process Model leads to a valid model. The Shared Process Model should be updated automatically to reflect only the approved changes. The change management requires that one party can see all the changes done by the other party in a consumable way. In particular, it should be possible for an IT stakeholder to understand the necessary implementation steps that arise from a business view change.

If a process is in production, all three conditions, business conformance, IT conformance and business-IT consistency, should be met. Upon a public change of the IT view, the business view changes and hence the Shared Process Model must show that the current business view is not approved. Conversely, a public change on the business view changes the IT view and the Shared Process Model must indicate that the current IT view is not approved by IT. Note that a change of the IT view that was induced by a public change of the business view is likely to affect the validity of the IT view with respect to executability on a BPM engine.

4 A Technical Realization of the Shared Process Model

In this section, we present parts of a technical realization of the concepts and requirements from Sect. 3 as we have designed and implemented them.

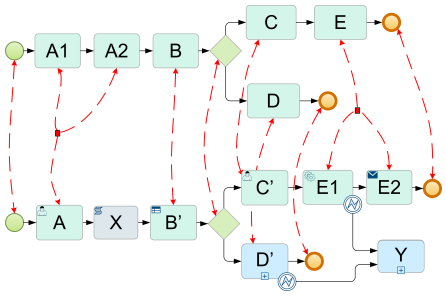

Fig. 3. The Shared Process Model as a combination of two individual models, coupled by correspondences

4.1 Basic Solution Design

We represent the Shared Process Model by maintaining two process models, one for each view, together with *correspondences* between their model elements, as illustrated by Fig. 3. In the upper part, the process model for business is shown, in the lower part the process model for IT. A *correspondence*, shown by red dashed lines, is a bidirectional relation between one or more elements of one model and one or more elements of the other model.

For example, in Fig. 3, task B of the business layer corresponds to task B' of the IT layer which is an example for a one-to-one correspondence. Similarly, task D of the business layer corresponds to subprocess D' of the IT layer and tasks A_1 and A_2 correspond to the (human) task A of the IT layer which is an example for a many-to-one correspondence. Many-to-many correspondences are technically possible but we haven't found a need for them so far. We only relate the main flow elements of the model, i.e., activities, events and gateways, but sequence flow is not linked. Each element is contained in at most one correspondence. An element that is contained in a correspondence is called a *shared* element, otherwise it is a *private* element.

Alternatively, we could have chosen to represent the Shared Process Model differently by merging the business and IT views into one common model with overlapping parts being represented only once. This ultimately results in an equivalent representation, but we felt that we stay more flexible with our decision above in order to be able to easily adapt the precise relationship between business and IT views during further development.

Furthermore, with our realization of the Shared Process Model we can easily support the following:

- Import/export to/from the Shared Process Model: From the Shared Process Model, a process model must be created (e.g. business view) that can be shown by an editor. This is straight-forward in our representation. We use BPMN 2 internally in the Shared Process Model, which can be easily consumed outside by existing editors. Likewise, other tools working on BPMN 2 can be leveraged for the Shared Process Model prototype easily.
- Generalization to a Shared Process Model with more than two process models is easier to realize with correspondences rather than with a merged metamodel. This

includes generalization to three or more stakeholder views, but also when one business model is implemented by a composition of multiple models (see Sect. 2.1) or when a business model should be traced to multiple alternative implementations.

The technical challenges occur in our realization if one of the process models evolves under changes because then the other process model and the correspondences have to be updated in an appropriate way.

4.2 Establishing and Maintaining Correspondences

A possible initialization of the Shared Process Model is with a single process model, which can be thought of the initial business view. This model is then internally duplicated to serve as initially identical business and IT models. This creates one-to-one correspondences between all main elements of the process models for business and IT. The creation of such correspondences is completely automatic because in this case a correspondence is created between elements with the same universal identifier during the duplication process. Another possible initialization is with a pair of initial business and IT views where the two views are not identical, e.g. they might be taken from an existing project situation where the processes at different abstraction levels already exist. In such a case, the user would need to specify the correspondences manually or process matching techniques can be applied to achieve a higher degree of automation [1].

A one-to-many or many-to-one correspondence can be introduced through an editing wizard. For example, if an IT architect decides that one business activity is implemented by a series of IT activities, he uses a dedicated wizard to specify this refinement. The wizard forces the user to specify which activity is replaced with which set of activities, hence the wizard can establish the one-to-many correspondence.

The Shared Model evolves either through such wizards, in which case the wizard takes care of the correspondences, or through free-hand editing operations, such as deletion and insertion of tasks. When such changes are checked into the Shared Model as public changes, the correspondences need to be updated accordingly. For example, if an IT architect introduces several new activities that are business-relevant and therefore designated as public changes, the propagation to the business level must also include the introduction of new one-to-one correspondences. Similarly, if an IT architect deletes a shared element on the IT level, a correspondence connected to this shared element must be removed when propagating this change.

4.3 Business-IT Consistency

As described in Sect. 3.2, we distinguish coverage and preservation of behavior. Coverage can be easily checked by help of the correspondences. Every private element, i.e., every element that is not contained in a correspondence must be accounted for. For example, all private business tasks, if any, could be marked once by the business analyst and all private IT tasks by the IT architect. The Shared Process Model then remembers these designations. A governance rule implemented on top may restrict who can do these designations. All private tasks that are not accounted for violate coverage.

For preservation of behavior, we distinguish *strong and weak consistency* according to the IT refinement patterns discussed in Sect. 2.1. If business and IT views are strongly consistent, then they generate the same behavior. If they are weakly consistent, then every behavior of the IT view is a behavior of the business view, but the IT view may have additional behavior, for example, to capture additional exceptional behavior. As with coverage, additional behavior in the IT view should be explicitly reviewed to check that it is indeed considered technical exception behavior and not-considered 'business-relevant'.

We use the following concretizations of strong and weak consistency here. At this stage, we only consider behavior generated by the abstract control flow, i.e., we do not yet take into account how data influences behavior.

- We define the Shared Process Model to be *strongly consistent* if the IT view can be derived from the business view by applying only operations from the first three categories in Sect. 2.1: complementary implementation detail, formalization and renaming, and behavioral refinement and refactoring. Private tasks in either view are compatible with consistency only if they are connected to shared elements by a path of sequence flow. The operations from the first three categories all preserve the behavior. The Shared Process Model in Fig. 3 is not strongly consistent because the IT view contains private boundary events. Without the boundary events and without activity *Y*, the model would be strongly consistent. Fig. 4 shows examples for violating strong consistency.
  An initial Shared Process Model with two identical views is strongly consistent. To preserve strong consistency, all flow rearrangements on one view, i.e., moving activities, rearranging sequence flow or gateways must be propagated to the other view as public changes.
- For *weak consistency*, we currently additionally allow only IT-private error boundary events leading to IT private exception handling. Technically we could also allow additional IT-private gateways and additional branches on shared gateways here, but we haven't yet seen a strong need for them. The Shared Process Model in Fig. 3 is weakly consistent. The examples in Fig. 4 also violate weak consistency.

We have used the simplest notion(s) of consistency such that all the refinement patterns we have encountered so far can be dealt with. We haven't yet seen, within our usage scenarios, the need for more complex notions based on behavioral equivalences such as trace equivalence or bisimulation.

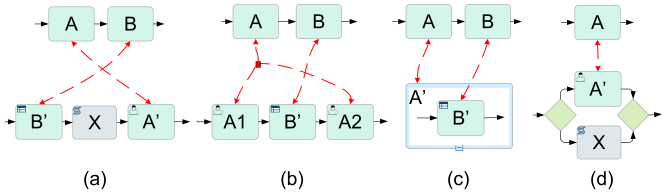

Fig. 4. Examples of inconsistencies

Strong and weak consistency can be efficiently checked but the necessary algorithms and also the formalization of these consistency notions are beyond the scope of this paper[3]. The automatic propagation of public changes, which we will describe in the following sections, rests on the Shared Process Model being at least weakly consistent.

4.4 Computing Changes between Process Models

If the Shared Process Model evolves by changes on the business or IT view, then such changes must be potentially propagated from one view to the other. As a basis for our technical realization of the Shared Process Model, an approach for *compare and merge* of process models is used [15]. We use these compound operations because they minimize the number of changes and represent changes on a higher level of abstraction. This is in contrast to other approaches for comparing and merging models which focus on computing changes on each model element.

Figure 5 shows the change operations that we use for computing changes. InsertActivity, DeleteActivity and MoveActivity insert, delete and move activities or other elements such as events and subprocesses. InsertFragment, DeleteFragment and MoveFragment is used for inserting, deleting and moving fragments which represent control structures. The computation of a *change script* consisting of such compound operations is based on comparing two process models and their Process Structure Trees. For more details of the comparison algorithm, the reader is referred to [15].

Change Operation op	Effects on Process Model V
InsertActivity(x, a, b)	Insertion of a new activity x between two succeeding elements a and b in process model V and reconnection of control flow.
DeleteActivity(x)	Deletion of activity x and reconnection of control flow.
MoveActivity(x, a, b)	Movement of activity x from its old position into its new position between two succeeding elements a and b in process model V and reconnection of control flow.
InsertFragment(f, a, b)	Insertion of a new fragment f between two succeeding elements a and b in process model V and reconnection of control flow.
MoveFragement(f, a, b)	Movement of a fragment f from its old position to its new position.
DeleteFragment(f, c, d)	Deletion of fragment f between c and d from process model V and

Fig. 5. Change operations according to [15]

As an example for an evolution scenario of the Shared Process Model, consider Figure 6. The left hand side shows a part of the initial state of the Shared Process Model in our scenario, which contains a 2-to-1 correspondence and a private IT task. So, some IT refinements have been done already. Assume now, that during IT refinement, the IT

[3] For strong consistency, one has to essentially check that the correspondences define a *continuous* mapping between the graphs as known in graph theory.

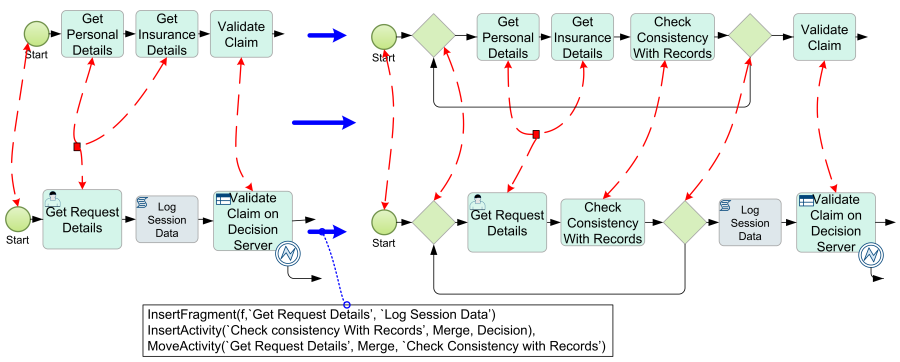

Fig. 6. Example of a change script on the IT level that is propagated to the business level

architect realizes that, in a similar process that he has implemented previously, there was an additional activity that checks the provided customer details against existing records. He is wondering why this is not done in this process and checks that with the business analyst, who in turn confirms that this was just forgotten. Consequently, the IT architect now adds this activity together with a new loop to the IT view, resulting in a new IT view shown in the lower right quadrant of Fig. 6. Upon checking this into the Shared Process Model as a public change, the business view should be automatically updated to the model shown in the upper right quadrant of Fig. 6.

To propagate the changes, one key step is to compute change operations between process models in order to obtain a *change script* as illustrated in Fig. 6. In the particular example, we compute three compound change operations: the insertion of a new empty fragment containing the two XOR gateways and the loop (*InsertFragment*), the insertion of a new activity (*InsertActivity*) and the move of an activity (*MoveActivity*), illustrated by the change script in Figure 6. In the next section, we explain how we use our approach to realize the evolution of the Shared Process Model.

4.5 Evolution of the Shared Process Model

For private changes, only the model in which the private changes occured is updated. In the following, we explain how public changes are propagated from IT to business, the case from business to IT is analogous.

When a new IT view is checked into the Shared Process Model, we first compute all changes between the old model IT and the new model IT', giving rise to a change script $Delta_{IT}$, see Figure 7 (a). The change script is expressed in terms of the change operations introduced above, i.e., $Delta_{IT} = \langle op_1, ..., op_n \rangle$ where each op_i is a change operation. In order to propagate the changes to the business level, $Delta_{IT}$ is translated into a change script $Delta_B$ for the business-level. This is done by translating each individual change operation op_i into an operation op_i^T and then applying it to the business-level. Likewise, we also apply each change operation on the IT-level to produce intermediate

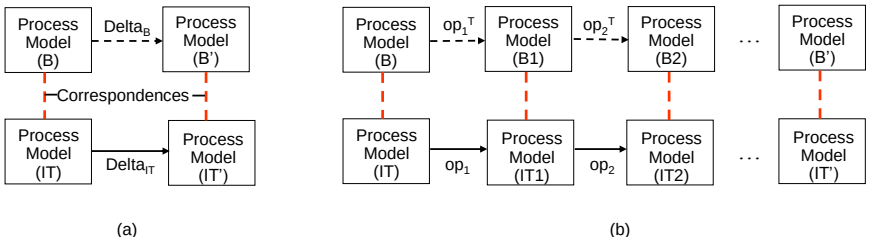

Fig. 7. Delta computation for propagating changes

process models for the IT level. Overall, we thereby achieve a synchronous evolution of the two process models, illustrated in Figure 7 (b).

Algorithm 1 Translation of a compound operation op from process model IT to Business model B

Step 1: compute corresponding parameters of the operation op
Step 2: replace parameters of op **with corresponding parameters to obtain** op^T
Step 3: apply op^T **to** B**, apply** op **to** IT
Step 4: update correspondences between B **and** IT

Algorithm 1 describes in pseudo-code the algorithm for translating a compound operation from IT to business. The algorithm for translation from business to IT can be obtained by swapping business and IT. Overall, one key step is replacing parameters of the operation from the IT model by parameters of the business model according to the correspondences. For example, for translating a change $InsertActivity(x, a, b)$, the parameters a and b are replaced according to their corresponding ones, following the correspondences in the Shared Process Model. In case that a and b are private elements, this replacement of elements requires forward/backward search in the IT model until one reaches the nearest shared element (Step 1 of the algorithm). Similarly, for translating an $InsertFragment(f, a, b)$, the parameters a and b are replaced in the same way. An operation $DeleteActivity(x)$ is translated into $DeleteActivity(x')$ (assuming here that x is related to x' by a one-to-one correspondence). After each translation, in Step 3 the change operation as well as the translated change operation are applied to produce new models B_i and IT_i, as illustrated in Figure 7 (b). Afterwards, Step 4 updates the correspondences between the business and IT model. For example, If x is the source or target of a one-to-many/many-to-one correspondence, then all elements connected to it must be removed.

For the example in Figure 6, the change script $Delta_{IT}$ is translated iteratively and applied as follows:

- The operation *InsertFragment(f, 'Get Request Details', 'Log Session Data')* is translated into *InsertFragment(f, 'Get Insurance Details', 'Validate Claim')*. The operation as well as the translated operation are applied to the IT and business model, respectively, to produce the

models IT_1 and B_1, and also the correspondences are updated. In this particular case, new correspondences are created e.g. between the control structures of the inserted fragments.
- The operation *InsertActivity('Check Consistency with Records', Merge, Decision)* is translated into *InsertActivity('Check Consistency with Records', Merge, Decision)*, where the new parameters now refer to elements of the business model. These operations are then also applied, in this case to IT_1 and B_1, and correspondences are updated.
- The operation *MoveActivity('Get Request Details', Merge, 'Check Consistency with Records')* is translated into *MoveActivity('Get Request Details', Merge, 'Check Consistency with Records')*, where the new parameters now refer to elements of the business model. Again, as in the previous steps, the operations are applied and produce the new Shared Process Model consisting of B' and IT'.

In general, when propagating a change operation, it can occur that the insertion point in the other model cannot be uniquely determined. For example, if a business user inserts a new task between the activity *'Get Insurance Details'* and *'Validate Claim'* in Fig. 6, then this activity cannot be propagated to the IT view automatically without user intervention. In this particular case, the user needs to intervene to determine whether the new activity should be inserted before or after the activity *'Log Session Data'*.

In addition to computing changes and propagating them automatically, in many scenarios it is required that before changes are propagated, they are approved from the stakeholders. In order to support this, changes can first be shown to the stakeholders and the stakeholders can approve/disapprove the changes. Only approved changes will then be applied. Disapproved changes are handed back to the other stakeholder. They will then have to be handled on an individual basis. Such a change management can be realized on top of our change propagation.

4.6 Implementation

As proof of concept, we have implemented a prototype as an extension to the IBM Business Process Manager and as an extension to an open source BPMN editor. A recorded demo of our prototype is publically available [8]. Our current prototype implements initialization of a Shared Process Model from a BPMN process model, check-in of private and public changes to either view and change propagation between both views. Furthermore, we have implemented a check for strong consistency, which can be triggered when checking in private changes. We currently assume that the changes between two subsequent IT views (or business views respectively) are either all public or all private.

With an additional component, this assumption can be removed. Then, the change script is presented to the user who can then mark the public changes individually. For this scenario, the compare/merge component needs to meet the following two requirements: (i) the change script must be consumable by a human user and (ii) individual change operations presented to the user must be as independent as possible. Note that the change operations in a change script are in general interdependent, which restricts the ability to apply only an arbitrary subset of operations to a model. Therefore, a compare/merge component may not support to separate all public from all private changes.

In fact, we first experimented with a generic compare/merge component from the EMF Compare Framework, which could be used to generate a change script for two process model based on the process metamodel, i.e., BPMN 2. The change operations

were so fine-grained, e.g. ‘a sequence flow reference was deleted from the list of incoming sequence flows of a task’, such that the change script was very long and not meaningful to a human user without further postprocessing. Furthermore, the BPMN 2 metamodel generates very strong dependencies across the different parts of the model so that separate changes were likely to be dependent in the EMF Compare change script.

For these reasons, we switched to a different approach with compound changes as described above. Note that the change approval scenarios described in Sect. 3.2 generate the same requirements for the compare/merge component: human consumability of the change script and separability of approved changes from rejected changes.

5 Related Work

We used prior work [15] on comparing and merging process models on the same abstraction level. Our work deals with changes of models on different abstraction level and distinguishes between public and private changes.

Synchronizing a pair of models connected by a correspondence relation is an instance of a symmetric delta lens [6]. In a symmetric delta lens, both models share some information, but also have some information private to them. Deltas are propagated by translation, which has to take the original and the updated source including the relation between them and original target model including the correspondence to the original source as a parameter. Symmetric delta lenses generalize the state-based symmetric lenses by Pierce et al. [12]. In recent years, various techniques have been developed for synchronization of models. Popular approaches are based on graph grammars (e.g. Giese et al. [10]). In contrast to these approaches, our idea of explicitly marking private elements is novel.

In the area of model-driven engineering, the problem of a coupled evolution of a meta-model and models is related to our problem. Coupled evolution has recently been studied extensively (compare Herrmannsdoerfer et al. [11] and Cicchetti et al. [5, 4]). The problem of coupled evolution of a meta-model and models has similarities to our problem where two or more models at a different abstraction level evolve. One key difference is that in our application domain we hide private changes and that we allow changes on both levels to occur which then need to be propagated. In contrast to Herrmannsdoerfer et al., we aim at complete automation of the evolution. Due to the application domain, we focus on compound operations and also translate the parameters according to the correspondences. Overall, one could say that our solution tries to solve the problem in a concrete application domain whereas other work puts more emphasis on generic solutions which can be applied to different application domains.

On an even more general level, (in)consistency management of different views has been extensively studied in recent years by many different authors (e.g. Finkelstein et al. [9], Egyed et al. [7]). The goal of these works is to define and manage consistency of different views where views can be diverse software artefacts including models. As indicated in the paper, our problem can be viewed as one instance of a consistency problem. In contrast, we focus on providing a practical solution for a specific application domain which puts specific requirements into place such as usability and hiding of private changes.

In the area of process modeling, Weidlich et al. [20] have studied vertical alignment of process models, which brings models to the same level of abstraction. They also discuss an approach for automatic identification of correspondences between process models. Buchwald et al. [3] study the Business and IT Gap problem in the context of process models and introduce the Business IT Mapping Model (BIMM), which is very similar to our correspondences. However, they do not describe how this BIMM can be automatically maintained during evolution. Tran et al. [18] focus on integration of modeling languages at different abstraction levels in the context of SOA Models but they do not focus on the closing the business IT gap as we do. Werth et al. [21] propose a business service concept in order to bridge the gap between the process layer and the technical layer, however, they do not introduce two abstraction layers of process models. Thomas et al. [17] on the other hand distinguish between different abstraction layers of process models and also recognize the need of synchronizing the layers but they do not provide techniques for achieving the synchronization.

Various authors have proposed different forms of abstractions from a process model, called a *process view*, e.g. [16]. A process view can be recomputed whenever the underlying process model changes. Recently, Kolb et al. [13] have taken the idea further to allow changes on the process view that can be propagated back to the original process model, which can be considered as a model synchronization. They restrict to hierarchical abstractions of control flow in well-formed process models.

6 Conclusion

Different process model views are important to reflect different concerns of different process stakeholders. Because their concerns overlap, a change in one view must be synchronized with all other overlapping views in order to facilitate stakeholder collaboration.

In this paper, we have presented detailed requirements for process model view synchronization between business and IT views that pose a significant technical challenge for its realization. These requirements were derived from a larger industrial case study [2] and additional interviews with BPM practicioners. A central intermediate step was the systematic categorization of changes from business to IT level given in Sect. 2.1. We have also presented our solution design and reported first results of its implementation to demonstrate the feasibility of our approach.

We are currently working on the further elaboration and implementation of the change management scenarios described above, and we are preparing an experimental validation with users in order to further demonstrate the value of our approach. Also, not all elements of the BPMN metamodel are currently synchronized but only the main ones. In particular, the synchronization of the layout information of the models was not yet addressed and requires further study.

References

1. M. Branco, J. Troya, K. Czarnecki, J. M. Küster, and H. Völzer. Matching business process workflows across abstraction levels. In *MoDELS*, *LNCS* 7590, pp 626–641. Springer, 2012.

2. M. C. Branco, Y. Xiong, K. Czarnecki, J. Küster, and H. Völzer. A case study on consistency management of business and IT process models in banking. *Software and Systems Modeling*, March 2013.
3. S. Buchwald, T. Bauer, and M. Reichert. Bridging the gap between business process models and service composition specifications. *Int'l Handbook on Service Life Cycle Tools and Technologies: Methods, Trends and Advances*, 2011.
4. A. Cicchetti, F. Ciccozzi, and T. Leveque. A solution for concurrent versioning of metamodels and models. *Journal of Object Technology*, 11(3):1: 1–32, 2012.
5. A. Cicchetti, D. Di Ruscio, and A. Pierantonio. Managing dependent changes in coupled evolution. In *ICMT*, *LNCS* 5563, pages 35–51. Springer, 2009.
6. Z. Diskin, Y. Xiong, K. Czarnecki, H. Ehrig, F. Hermann, and F. Orejas. From state- to delta-based bidirectional model transformations: The symmetric case. In *MoDELS*, *LNCS* 6981, pages 304–318. Springer, 2011.
7. A. Egyed. Instant consistency checking for the UML. In *ICSE*, pp 381–390. ACM, 2006.
8. C. Favre, J. Küster, and H. Völzer. Recorded demo of shared process model prototype. `http://researcher.ibm.com/view_project.php?id=3210`.
9. A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency handling in multi-perspective specifications. In *ESEC*, *LNCS* 717, pages 84–99. Springer, 1993.
10. H. Giese and R. Wagner. From model transformation to incremental bidirectional model synchronization. *Software and System Modeling*, 8(1):21–43, 2009.
11. M. Herrmannsdoerfer, S. Benz, and E. Jürgens. Cope - automating coupled evolution of metamodels and models. In *ECOOP*, *LNCS* 5653, pp 52–76. Springer, 2009.
12. M. Hofmann, B. Pierce, and D. Wagner. Symmetric lenses. In *POPL*, pp 371–384. ACM, 2011.
13. J. Kolb, K. Kammerer, and M. Reichert. Updatable process views for user-centered adaption of large process models. In *ICSOC*, *LNCS* 7636, pp 484–498. Springer, 2012.
14. J. Küster, H. Völzer, C. Favre, M. C. Branco, and K. Czarnecki. Supporting different process views through a shared process model. Technical Report RZ3823, IBM, 2013.
15. J. M. Küster, C. Gerth, A. Förster, and G. Engels. Detecting and Resolving Process Model Differences in the Absence of a Change Log. *BPM*, *LNCS* 5240, pp 244–260. Springer, 2008.
16. D. Schumm, F. Leymann, and A. Streule. Process viewing patterns. In *EDOC*, pp 89–98. IEEE Computer Society, 2010.
17. O. Thomas, K.Leyking, and F. Dreifus. Using process models for the design of service-oriented architectures: Methodology and e-commerce case study. In *HICSS*, p 109. IEEE Computer Society, 2008.
18. H. Tran und U. Zdun and S. Dustdar. View-based Integration of Process-driven SOA Models at Various Abstraction Levels. In *1st Intl Workshop on Model-Based Software and Data Integration*. Springer, 2008.
19. M. Weidlich, A. P. Barros, J. Mendling, and M. Weske. Vertical alignment of process models - how can we get there? In *BPMDS*, *LNBI* 29, pp 71–84. Springer, 2009.
20. M. Weidlich, R. Dijkman, and J. Mendling. The ICoP framework: Identification of correspondences between process models. In *CAiSE*, *LNCS* 6051, pp 483–498. Springer, 2010.
21. D. Werth, K. Leyking, F. Dreifus, J. Ziemann, and A. Martin. Managing SOA through business services - a business-oriented approach to service-oriented architectures. In *ICSOC Workshops*, *LNCS* 4652, pp 3–13. Springer, 2006.