Softw Syst Model (2016) 15:1207–1233
DOI 10.1007/s10270-015-0453-5

REGULAR PAPER

Supporting different process views through a Shared Process Model

Jochen Küster · Hagen Völzer · Cédric Favre · Moisés Castelo Branco · Krzysztof Czarnecki

Received: 13 March 2014 / Revised: 30 November 2014 / Accepted: 6 January 2015 / Published online: 24 January 2015

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 synchronizes changes made to any view. We present detailed requirements and a solution design for the Shared Process Model. We also present an overview of our prototypical implementation to demonstrate the feasibility of the approach. Finally, we report on a comprehensive evaluation of the approach on real Business–IT modeling scenarios.

Keywords Business process modeling · Business–IT gap · Model synchronization

Communicated by Dr. Pieter Van Gorp.

J. Küster
Bielefeld University of Applied Sciences, Bielefeld, Germany

H. Völzer · C. Favre
IBM Research — Zurich, Zurich, Switzerland

M. C. Branco (✉) · K. Czarnecki
Generative Software Development Laboratory, University of Waterloo, Waterloo, ON, Canada
e-mail: mcbranco@gsd.uwaterloo.ca

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 concerns 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 form the basis 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, the 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 represent and 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 [1]). 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. Several existing works show that different stakeholders intrinsically desire different *views* onto the same process because of their different concerns and their different levels of abstraction [2–6]. 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 satisfy the needs of all stakeholders.

This problem is a variation of the coupled evolution problem [7] and the model synchronization problem [8]. Coupled evolution has been studied between metamodels and models. In the area of model synchronization, various techniques have been proposed [9,10]; however, it is not immediately clear how to apply these techniques in the context of views on business process models. Thus, 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 satisfies practical needs and constraints.

In this paper, we address this problem by presenting a tool-supported approach to consistently synchronizing related Business and IT process models,[1] after independent editing. The key novel aspect of the approach is the distinction of private and public changes. Private changes only affect a single view, whereas public ones are propagated back to the other view. Dependencies between private and public parts, such as sequence flows, are taken into account when the changes are propagated between the views. We contribute detailed requirements and a concrete design to synchronize process views. In contrast to previous work (e.g., [11]), our approach supports synchronizing models across levels of abstractions related by not only hierarchical, but also non-hierarchical refinements. We report on an prototypical implementation to substantiate that a solution is technically feasible and performs a comprehensive evaluation of the approach on industrial Business–IT modeling scenarios. The evaluation shows that the solution can be effective, provided that certain practices, such as frequent synchronizations, are followed. The evaluation also identifies challenges remaining for future work, such as dealing with parallel edits. An earlier version of this paper appeared elsewhere [12]; the present paper extends it with the evaluation study and expands on the description of the synchronization algorithm.

[1] Throughout the text, we indistinctly use 'model' or 'view' when referring to a particular perspective—Business or IT.

The paper is structured as follows. In Sect. 2, we explain in detail why synchronizing different process models views is needed. Although the problem is recognized in academic research, it is not universally accepted in practice after the arrival of BPMN 2. Based on this motivation, we present detailed requirements for a Shared Process Model in Sect. 3 . We also explain several usage scenarios of the Shared Process Model and discuss key aspects of the consistency between different views. In Sect. 4, we introduce our technical realization of the Shared Process Model and report on a prototype. In Sect. 5, we perform a comprehensive evaluation of the approach on real-world data. Related work is discussed in Sect. 6. Finally, Sect. 7 concludes the paper.

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 that is 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 we not use a single BPMN 2 model that is shared between business and IT? To study this question, 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 in this section.

Before delving into details, let us first introduce our running example. Figure 1 illustrates two corresponding Business- and IT-level models, in a simplified claim handling process. The top part of the figure shows the high-level Business specification, and the bottom one shows the refined,

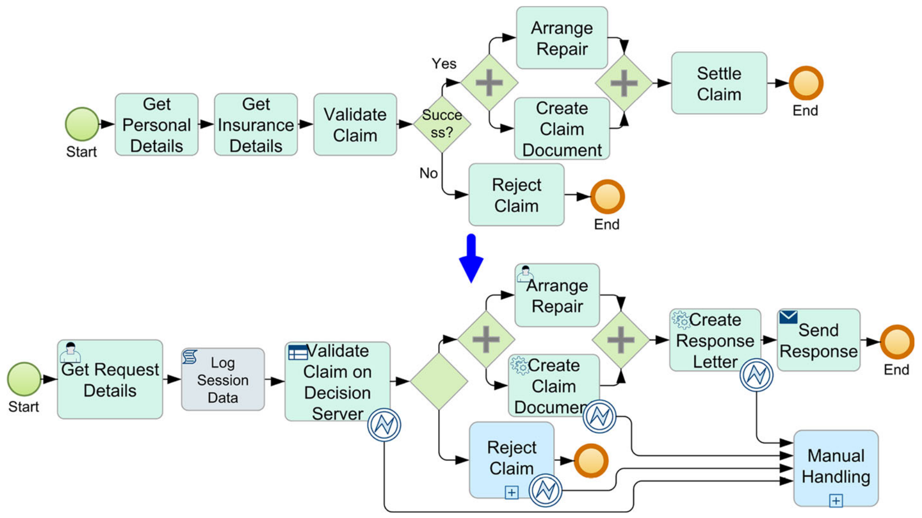


Fig. 1 Illustration of some refinements often made going from the business to the IT model

executable, IT model. Some IT-specific changes were applied to the first to produce the second, for example, grouping tasks ('Get Personal Details', 'Get Insurance Details') into a single one ('Get Request Details'), adding script tasks ('Log Session Data') and adding exception handling events and flows ('Manual Handling'). A larger, more realistic example is shown in the "Appendix," together with a catalog of common refinement patterns identified in the early study [2]. Note that the following categorization of changes, based on the additional data, is a new contribution of this work, compared to [2].

We identified the following categories of changes that were applied in producing an execution model from a business model.

- *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 its transformation, service interfaces, and communication detail. For example, Fig. 1 shows the addition of BPMN 2 task stereotypes in the IT model (the stereotypes appear as small icons in the upper left corner of the decorated tasks): user task ('Get Request Detail'), script task ('Log Session Data'), service task ('Create Response Letter'), and send task ('Send Response').
- *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), typing of subprocesses (transaction, call) and typing of 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, which 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. These types of changes include
  - *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 implemented in a single service call or a single human task to map the required business steps to existing services and sub-engines (human task, business rules). For example, in Fig. 1, 'Get Personal Details' and 'Get Insurance Details' are subsumed by a single call 'Get Request Details' to 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 needs 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 that are not considered to be a part of an implementation of a specific business task are added on the IT level, 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 be specified in accompanying and more detailed use case documents, 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 flow or data flow that need to be corrected before execution. Sometimes activities 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 this addition back to the business elements. However, if we merge the error handling back to the normal flow through a new gateway 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 keep multiple versions of a process model to reflect the different views of the stakeholder (cf., e.g., [2,3,13]). However, because today's tools do not have any support for synchronizing them, they typically become inconsistent over time. That is, the views disagree about which business tasks are executed and in which order. This can lead to costly business disruptions or to audit failures [2].

There are various reasons why business and IT models become inconsistent over time. We explained above in Sect. 2.1 (see *Correction and revision of the flow*) that, already in the initial implementation of a process, the flow may need to be corrected or revised. If these updates are only done on the IT model and not on the business model, then the models become inconsistent in the initial implementation phase. Respondents in our earlier survey [2] have agreed that inconsistency arises already in that phase because the initial business model is incomplete (frequently), contains modeling errors (occasionally to frequently), the business model contradicts some IT requirements (occasionally), and the business model does not faithfully represent the actual business process (rarely).

Furthermore, more inconsistencies arise when business requirements change, which are then often applied to only the IT model because of time pressure, while neglecting a simultaneous update of the corresponding business model. Likewise, changing IT requirements, e.g., an IT infrastructure change, may affect some business-relevant aspects of the IT model, which leads to further inconsistencies between the business model and the IT model.

Thus, while different views onto a process are needed by different stakeholders, different views quickly become inconsistent if not synchronized. Inconsistencies in turn can create business disruptions, audit failures, maintenance problems or delays in the implementation of new requirements. They can also lead to a business analyst misinterpreting process monitoring data.

3 Requirements for a Shared Process Model

3.1 The Shared Process Model concept

The *Shared Process Model*that 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

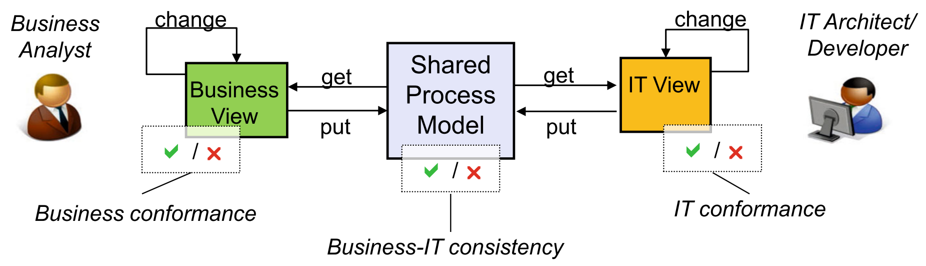


Fig. 2 Process view synchronization via a Shared Process Model

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 Process 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 working on the refinement of the IT model, that the model is missing an important business activity, the architect can insert the missing activity in the IT model. The architect 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 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. For example, in Fig. 1, adding the task 'Log Session Data' would be a private change to the IT view, since it only makes sense for execution purposes. On the other hand, adding a task such as 'Arrange Repair' to the IT model would be a public change, because it is also relevant to the Business view.

Figure 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 condition also implies 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. Similarly, this condition also implies 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 or her 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 that 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 in 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 who modifies a 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 business-relevant 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 it, 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, whereas 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 human-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 in 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 in 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 in the IT view that was induced by a public change in 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. We detail how we have designed and implemented them.

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. The upper part shows the process model for business; the lower part shows 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. Correspondences are also known as traceability links in requirements engineering [14] and in model transformation [15].

For example, in Fig. 3, task 'Validate Claim' of the business level corresponds to task 'Validate Claim on Decision Server' of the IT level, which is an example for a one-to-one correspondence. Similarly, task 'Reject Claim' of the business level corresponds to subprocess 'Reject Claim' of the IT level. Further, tasks 'Get Personal Details' and 'Get Insurance Details' correspond to the (human) task 'Get Request Details' of the IT level, which is an example for a many-to-one correspondence. Many-to-many correspondences are technically possible, but we have not 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.

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 decided to keep both views separate in the shared models to afford us more flexibility during the development of the prototype.

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

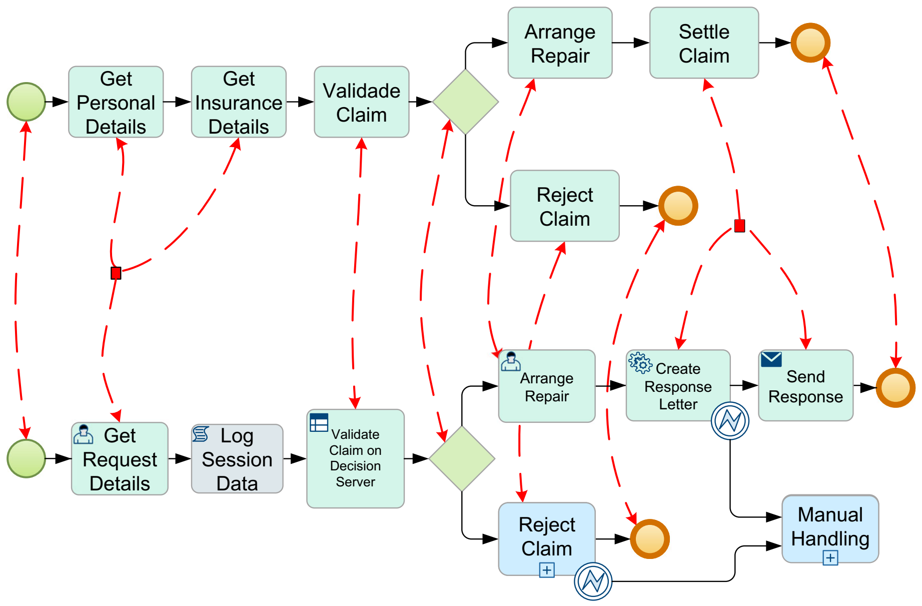


Fig. 3 The Shared Process Model as a combination of two individual models, coupled by correspondences

- 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: The views could be arranged in a chain. For example, some organizations refine a business process view into an IT architecture view, which is then refined into an IT implementation view [2]. Such a chain can be realized with three internal models and two sets of correspondences. A change in one view can then be successively propagated to the other two. In principle, this schema can be used as long the arrangement of the views does not create cycles. A more general and scalable approach is to use a star structure, where a central model contains all synchronizable information and all other models (the views) only connect to this central model by correspondences. Each change to a view then is propagated to a change in the central model, which in turn is propagated to all other views.

The following subsections detail the typical flow of steps when maintaining corresponding views. Section 4.6 discusses the implementation of the prototype.

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 the 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. Our current tool prototype supports this initialization method. Another possible initialization is with a pair of initial business and IT views where the two views are not identical. For example, 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 use process matching techniques to achieve a higher degree of automation [16].

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. For example, in Fig. 3, the task 'Settle Claim,' in the Business view, corresponds to two other tasks in the IT view: 'Create Response Letter' and 'Send Response.'

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. Naturally, wizards offer more automation but may have limited repertoire of transformations, and the user may not be familiar with all the available ones. Manual edits, on the other hand, are more flexible, such as using other editors, but may slow down the development process. Currently, the tool supports free-hand editing and four refactoring wizards (See Sect. 5.4).

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; similarly, all private IT tasks could be marked by the IT architect. The Shared Process Model then remembers these designations. A governance rule may restrict who can do these designations.

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.' Such business-irrelevant behavior in the IT view should be marked as private.

We use the following concretizations of strong and weak consistency.

- 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 are allowed, but deleting them while leaving the flows they established should lead to strongly consistent models. 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. In essence, the private elements introduce new control flow that is not present in the business model. Without the boundary events and without activity Y, the model would be strongly consistent. Figure 4 shows examples for violating strong consistency.

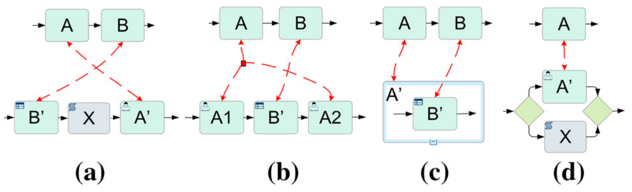


Fig. 4 Examples of inconsistencies. Private tasks are shown in *gray*

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 have not yet seen a strong need for them. The Shared Process Model in Fig. 3 is weakly consistent. On the other hand, the examples in Fig. 4 violate weak consistency:
  (a) Sequence flow out of order: Execution of 'A' and 'B' is reversed in the IT model;
  (b) Overlapping execution via interleaving: Execution of 'A' and 'B' overlap in the IT model;
  (c) Overlapping execution via subprocess: 'B' executed as part of 'A' in the IT model; and
  (d) Potentially missing execution: 'A' may or may not be executed in the IT model.

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 have not yet seen, within our usage scenarios, the need for more complex notions based on behavioral equivalences such as trace equivalence [17] or bisimulation [18]. Further, this work only considers behavior generated by the abstract control flow, i.e., we do not take into account how data influences behavior. We leave data flow consistency for future work.

Strong and weak consistency can be efficiently checked. For strong consistency, the algorithm consists of the following three steps: (1) remove each private task from each of the two views and reconnect the incident elements; (2) check for each one-to-many correspondence, i.e., where a task in one view corresponds to a set T of multiple tasks in the other view, that the set T forms a *single-entry-single-exit fragment* [19], that is, a connected subgraph with a single-entry and a single-exit edge such that the set of tasks in that subgraph is T. Such a check is straightforward and can be implemented in linear time. If the check succeeds, replace each single-entry-single-exit fragment with a single task. The replacement reduces each one-to-many correspondence to a one-to-one correspondence. (3) Check whether the two obtained graphs are isomorphic via the correspondences. To do so, it suffices to check for each pair (x, y) of corresponding elements, whether (i) the graph predecessors of x correspond to the predecessors of y and that (ii) the graph successors of x correspond to the successors of y. Also this check can be implemented to run in linear time. To check weak consistency, first remove each IT-private error boundary event and all private model elements of the IT view that are connected to the public elements only through that error boundary event. Then check strong consistency as described above.

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 model versions

If the Shared Process Model evolves by changes in 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 [20]. We use these compound operations because they minimize the number of changes and represent changes in a higher level of abstraction. This is in contrast to other approaches for comparing and merging models, which focus on computing changes in each model element.

Table 1 shows the change operations that we use for representing changes: InsertActivity, DeleteActivity, and MoveActivity, respectively, insert, delete, and move activities or other elements such as events and subprocesses. InsertFragment, DeleteFragment, and MoveFragment are used for, respectively, inserting, deleting, and moving fragments that 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 [20].

As an example for an evolution scenario of the Shared Process Model, consider Fig. 5. The left-hand side shows a part of the initial state of the Shared Process Model in our scenario (fragment of Fig. 3), which contains a 2-to-1 correspondence and a private IT task. Thus, some IT refinements have been done already. Assume now, that during IT refinement, the IT 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. The architect is wondering why this is not done in this process and checks that with the business analyst, who in turn confirms that this activity 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. 5. 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. 5.

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. 5. In the particular example, we compute three compound change operations: the insertion of a new empty fragment f containing the two XOR gateways and the loop ($InsertFragment$, see Fig. 7), the insertion of a new activity ($InsertActivity$), and the move of an activity ($MoveActivity$), illustrated by the change script in Fig. 5. In the next section, we explain how we use our approach to realizing the evolution of the Shared Process Model.

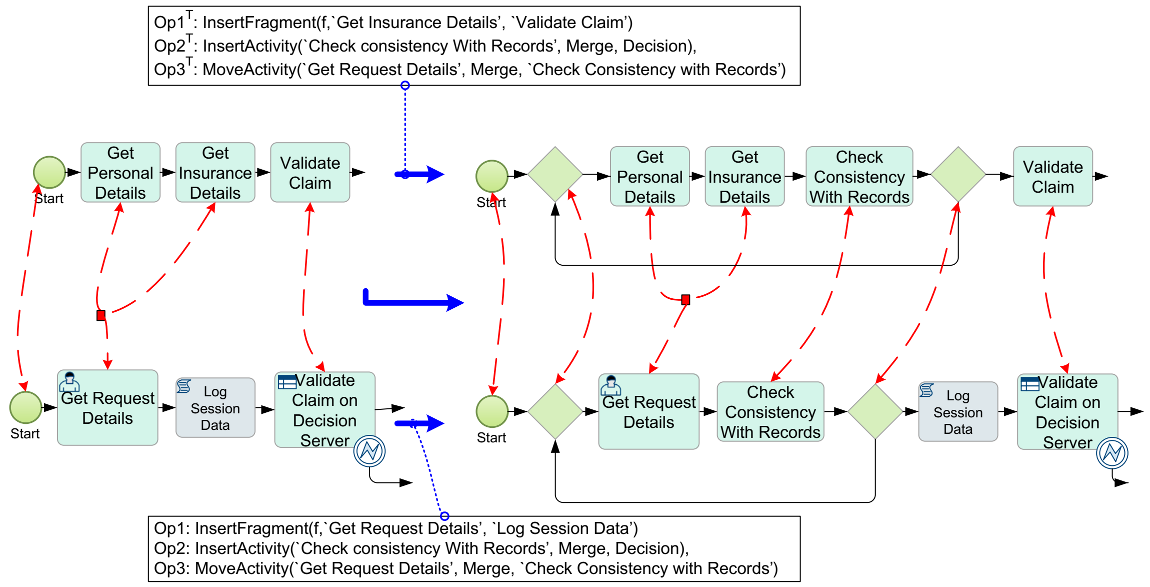
4.5 Evolution of the Shared Process Model

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

Table 1 Change operations according to [20]

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 reconnection of control flow



Op1^T: InsertFragment(f,`Get Insurance Details', `Validate Claim')
Op2^T: InsertActivity(`Check consistency With Records', Merge, Decision),
Op3^T: MoveActivity(`Get Request Details', Merge, `Check Consistency with Records')
Start
Get Personal Details
Get Insurance Details
Validate Claim
Check Consistency With Records
Get Request Details
Log Session Data
Validate Claim on Decision Server
Op1: InsertFragment(f,`Get Request Details', `Log Session Data')
Op2: InsertActivity(`Check consistency With Records', Merge, Decision),
Op3: MoveActivity(`Get Request Details', Merge, `Check Consistency with Records')

Fig. 5 Example of a change script on the IT level that is propagated to the business level

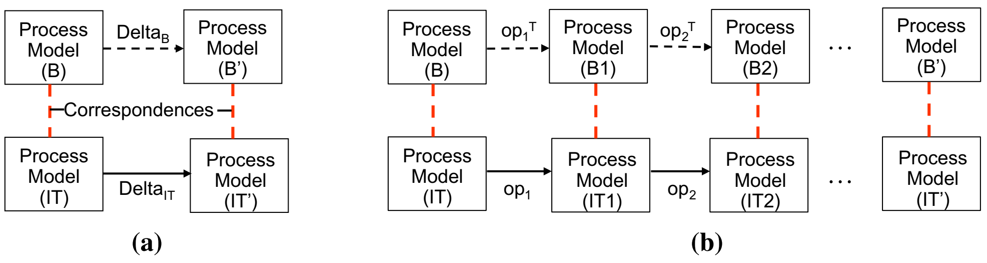


Fig. 6 Delta computation for propagating changes

$Delta_{IT}$, see Fig. 6a. The change script is expressed in terms of the change operations introduced above, i.e., $Delta_{IT} = \langle op_1, \ldots, op_n\rangle$ 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 process models for the IT level. Overall, we thereby achieve a synchronous evolution of the two process models, illustrated in Fig. 6b.

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 2 the change operation as well as the translated change operation is applied to produce new models B_i and IT_i, as illustrated in Fig. 6b. Afterwards, Step 3 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 Fig. 5, 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 is 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

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
if op==InsertActivity(x,a,b) then
    Search backward from a until an element a' with correspondences
        is reached
    Search forward from b until an element b' with correspondences
        is reached
    $op^T$ := InsertActivity(x,a',b')
else if op==InsertFragment(f,a,b) then
    Search backward from a until an element a' with correspondences
        is reached
    Search forward from b until an element b' with correspondences
        is reached
    $op^T$ := InsertFragment(f,a',b')
else if op==MoveActivity(x,a,b) then
    Search backward from a until an element a' with correspondences
        is reached
    Search forward from b until an element b' with correspondences
        is reached
    $op^T$ := MoveActivity(x,a',b')
else if op==MoveFragment(f,a,b) then
    Search backward from entry of f until an element a' with corre-
        spondences is reached
    Search forward from exit of f until an element b' with correspon-
        dences is reached
    $op^T$ := MoveFragment(f,a',b')
else if op==DeleteActivity(x) then
    $op^T$ := DeleteActivity(x')
end if
Step 2: apply $op^T$ to $B$, apply op to $IT$
if $op^T$==InsertActivity(x,a',b') or $op^T$=MoveActivity(x,a',b') or
    $op^T$=MoveFragment(f,a',b') then
    if connected(a',b) then
        Apply $op^T$ to $B$
        Apply op to $IT$
    else
        Choose Insertion Point ip on Path from a' to b'
        Apply $op^T$ to $B$ at insertion point ip
        Apply op to $IT$
    end if
end if
Step 3: update correspondences between $B$ and $IT$
if op==DeleteActivity(x) then
    Remove all correspondences involving x
else if op==InsertActivitiy(x,a,b) then
    Insert correspondence (x,x')
else if op==InsertFragment(f,a,b) then
    Insert correspondences between all fragment elements of f and f'
end if
```

particular case, new correspondences are created, e.g., between the control structures of the inserted fragments. The result is shown in Fig. 7.

- 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. 5, 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, many scenarios require that before changes are propagated, they are approved by the stakeholders. In order to support this capability, 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 originating 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 a 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 publicly available [21]. 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, allowing marking individual operations as private or public, this assumption can be removed. Then, the change script computed for the pair of IT views or business views would be presented to the user who could 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

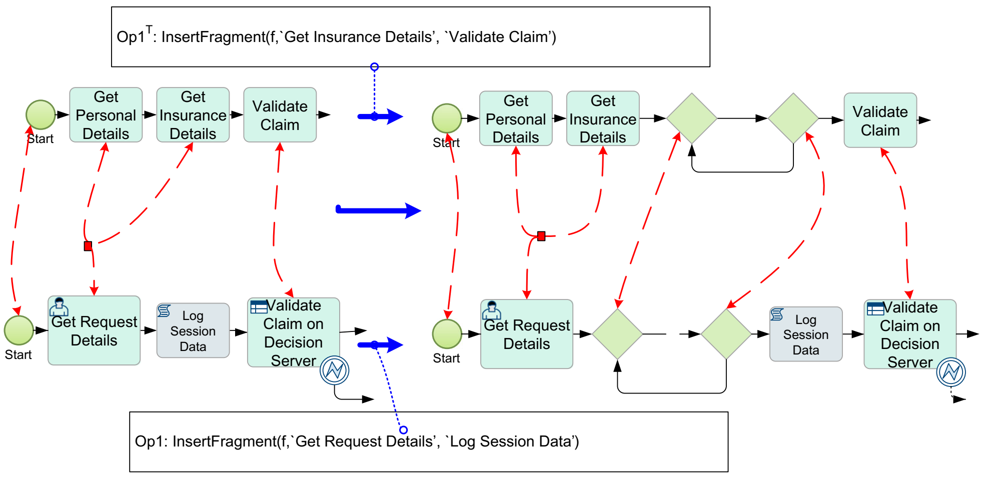

Op1^T: InsertFragment(f,`Get Insurance Details', `Validate Claim')
Start
Get Personal Details
Get Insurance Details
Validate Claim
Start
Get Request Details
Log Session Data
Validate Claim on Decision Server
Op1: InsertFragment(f,`Get Request Details', `Log Session Data')

Fig. 7 First step of evolution

subset of operations to a model. Therefore, a compare/merge component may not support separating 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 Evaluation

5.1 Objectives

We conducted an evaluation of the shared model tool using real-world data. The evaluation was carried out by means of several process modeling experiments, counting on help and feedback from industry practitioners. Our overarching goal was to observe the adherence of the tool behavior to the design requirements previously elicited in the development process of the same company. More specifically, the evaluation aimed at answering the following questions:

Q1 *How successfully can the tool synchronize typical Business-to-IT process modeling edit patterns?* [2]

We wanted to know how successfully the tool deals with edit patterns—instances of typical *correspondence patterns*—used by practitioners to build IT executable models based on their high-level business specifications. We applied the tool by replaying several concrete modeling scenarios and obtained feedback from practitioners who created and maintained the real models. A summary of typical correspondence patterns employed by the company and their rationales is shown in the "Appendix."

Q2 *How successfully can the tool synchronize scenarios composed of multiple edit patterns?*

In practice, an update to a model may lag far behind updates to its counterpart. We wanted to know how the tool would deal with synchronizing larger chunks of change, composed of multiple and mixed (private and public) edits at once. Insights from such scenarios also suggest new tool features and future work.

Q3 *Are there recommended best practices in using the tool, such that they could ensure consistency between Business and IT views?*

We wanted to know the most effective ways of using the tool, such that it ensures consistency between business and IT models.

To answer these questions, we replayed concrete change scenarios in Business and IT views, from a real project, as described in the following.

5.2 Subjects

To study how the tool works in a concrete setting, we remodeled and replayed change history of a BPM project—Credit Backoffice—from the Bank of Northeast of Brazil (BNB),

Table 2 Project size

	Number of model elements				
	Pools	Tasks	Gateways	Events	Flows
CreditBackoffice					
Business	6	47	46	18	128
IT	6	107	52	31	154

our industry partner. Table 2 shows the size of the project, in terms of the number of model elements in each of its Business and IT views (i.e., each of its types of process models).

BNB manages the change of software artifacts using two IBM products—*ClearQuest* (workflow of change requests) and *ClearCase* (artifact repository). Business employees open change requests to the IT department using ClearQuest. Every request has a unique ID, a textual description, and several parameters, such as priority and nature of the change (e.g., legal, evolution). Requests follow a sequence of steps, for example, to group them into projects (when applicable) before they arrive to IT. IT Managers assign IT professionals (Project Managers, Architects, Developers) to every request. IT technicians only can change artifacts in ClearCase by having an assigned change request. When artifacts are changed, ClearCase stores the change request ID in the change log. With their current tool support, BNB specialists perform synchronizations between Business and IT models *by hand*, i.e., by looking at the changes in one model and propagating them to the other, when applicable.

We recovered the change log of the case study project from the ClearQuest database, including the textual descriptions associated with every change request. We had the following objectives in collecting this data:

1. Select a *snapshot* of the project from the past, containing *consistent* Business and IT views. We relied on domain knowledge from BNB specialists to find a consistent pair of those views. The snapshot selected was from February of 2010, just before a new business evolution was about to start. As the models from BNB are implemented using commercial versions of IBM tools (Business in *Websphere Business Modeler*, and IT in *Websphere Integration Developer*), we needed to remodel them using the shared model tool—the prototype uses an open source BPMN 2.0 modeler, based on Eclipse.
2. Identify a set of concrete changes that were made in Business and IT views, along the project's life cycle. By analyzing 160 change requests, we found 23 of them (instances of common Business-to-IT correspondence patterns, see "Appendix") that specifically affected the process models. Most of the changes do not affect the process models themselves, but other resources such as databases, documentation and external services.

This dataset includes changes that cover different synchronization scenarios between Business and IT views. Based on it, we were able to: (i) replay the changes using the shared model tool, (ii) apply the synchronization mechanisms available, and (iii) compare the results with the actual consistent versions of the models, counting on knowledge from BNB domain experts.

5.3 Correspondence patterns versus edit patterns

It is important to distinguish between *correspondence* and *edit* patterns. The first represent typical correspondences to derive an IT-level process out of its business-level specification. Business and IT models are considered consistent if it is possible to establish correspondence among all their models elements that is consistent with the patterns.

Edit patterns, on the other hand, are particular ways of implementing correspondence patterns. For example, one can implement the correspondence pattern ***Split task into block*** (see "Appendix") by splitting a task in the IT model or merging tasks in the business model. Table 3 summarizes the relations between correspondence and edit patterns that occurred in the case study. The edit patterns specify where the changes occur and in which direction they are propagated (when they are propagated), as explained in the next section.

5.4 Method

First, we used the shared model tool to recreate the aforementioned version of the project from BNB (i.e., the Business and IT process models from February 2010). The two new process models were created identically as the original ones, except that the IT model was translated from BPEL to BPMN. For checking consistency, we focus on the control flow of the process models. BPMN and BPEL control flow constructs are similar in the sense that each can be mapped into the other, according to the OMG specification of BPMN 2.0 [22]. The control flow of the original models was entirely preserved for the evaluation. Note that the original models, as represented in their respective modeling tools, also have detailed information as attributes of nodes and flows, such as the communication protocols and the addresses of the services used. Some of those properties and parameters were ignored in the remodeling effort, since they were not relevant for the evaluation.

Second, for each one of the 23 real changes, we compared the two adjacent versions of the BNB models (i.e., before and after each change), and manually computed the *diff* between the versions. As a result, for each change we recorded which model elements (such as tasks, flows, gateways and events)

Table 3 Correspondence versus edit patterns

Correspondence pattern (see "Appendix")	Edit pattern
Add boundary event (CP5)	Add boundary event (IT only)
Add script task (CP3)	Add business-relevant task to business (B ⇒ IT)
Add script task (CP3)	Add business-relevant task to IT (IT ⇒ B)
Add manual task (CP2)	Add manual task (Business only)
Add properties (CP1)	Add properties (IT only)
Add protocol task (CP4)	Add protocol task (IT only)
Add script task (CP3)	Add script task (IT only)
Add technical exception flow (CP6)	Add technical exception flow (IT only)
Change activity name (CP7)	Change activity name (Business only)
Change activity type (CP8)	Change activity type (IT only)
Refactor gateway (CP12)	Refactor gateway (IT ⇒ B)
Split task into block (CP10)	Refine task into fragment (IT only)
Split workflow (CP11)	Refine task into subprocess (IT only)
Split task into block (CP10)	Simplify selection into task (business only)
Split task into block (CP10)	Split task into block (IT ⇒ B)
Suppress specification activity (CP9)	Suppress specification activity (B ⇒ IT)

were added, removed or updated (e.g., renamed or other properties changed).

Third, we replayed (remodeled) the changes, individually and then combined, using the tool. During the process, we applied one of the synchronization mechanisms available, according to how the propagation actually happened in the revision history:

- *Private (Business Only)* Change affects only the Business view and need to be privately kept on it.
- *Private (IT Only)* Change affects only the IT view and need to be privately kept on it.
- *Public (Business ⇒ IT)* Change is initially made on the Business view and needs to be propagated to the IT view.
- *Public (IT ⇒ Business)* Change is initially made on the IT view and needs to be propagated to the Business view.

For some changes, when applicable, we also applied built-in model refactoring operations provided by tool:

- *Simplify Selection into Task* Several model elements can be selected and collapsed into a single task.
- *Turn into Service Task* A generic task can be changed into an IT service task.
- *Refine Task into a Fragment* A single task can be split into a fragment (subflow) of other model elements—i.e., the inverse of *Simplify Selection into Task*.
- *Refine Task into a Subprocess* A special case of splitting a task, where the resulting fragment is a subprocess.

Finally, we showed and discussed the results of the synchronized views with BNB specialists who created the original models. This way they could help us to assess whether the tool had successfully synchronized the views consistently, according to their domain knowledge and the current consistent versions of the models.

The next two sections describe the results of applying the tool to keep Business and IT views synchronized, by replaying the model changes in two categories:

- *Single* synchronize one edit pattern at a time;
- *Compound* accumulate several edit patterns, respecting their occurrence over time, and synchronize them together.

In sequence, we discuss main lessons learned and threats to their validity.

5.5 Results: single refinement patterns

Table 4 presents the change scenarios of individual edit patterns, in terms of the number of model elements that have been added or removed, as seen in the *diff* between adjacent versions of BNB models. Some patterns, e.g., *Change activity name*, do not change the workflow, only alter model element properties. The synchronization method used to propagate the change is shown by a checkmark (✓). Also, the tool has some predefined refactoring operations, such as *Refine Task into Fragment*. The last column informs which tool-provided operation was used.

After applying the synchronization mechanism for each scenario, the resulting (updated) Business and IT views were captured and later discussed with the BNB specialists who created and maintained the actual project. Table 5 summa-

Table 4 Evaluation scenarios: single refinement patterns

Scenario	Pattern instance	Added				Removed				Synchronization method				Tool refactoring
		Tasks	Flows	Events	Gateways	Tasks	Flows	Events	Gateways	Private (business only)	Private (IT only)	Public (B ⇒ IT)	Public (IT ⇒ B)	
1	Add manual task	2	4	–	–	–	–	–	–	✓				–
2	Change activity name	–	–	–	–	–	–	–	–	✓				–
3	Simplify selection into task	–	–	–	–	–	–	–	–	✓				Simplify selection into task
4	Add properties	–	–	–	–	–	–	–	–		✓			–
5	Add script task	1	2	–	–	–	–	–	–		✓			–
6	Add script task	2	4	–	–	–	–	–	–		✓			–
7	Add protocol task	2	4	–	–	–	–	–	–		✓			–
8	Add protocol task	1	2	–	–	–	–	–	–		✓			–
9	Add boundary event	1	1	1	–	–	–	–	–		✓			–
10	Add technical exception flow	1	1	1	–	–	–	–	–		✓			–
11	Add technical exception flow	2	2	2	–	–	–	–	–		✓			–
12	Change activity type	–	–	–	–	–	–	–	–		✓			Turn into service task
13	Refine task into fragment	2	4	–	2	–	–	–	–		✓			Refine task into fragment
14	Suppress specification activity	–	–	–	–	1	2	–	–		✓			–
15	Split task into block	6	12	–	2	–	–	–	–		✓			–
16	Refine task into subprocess	–	–	–	–	–	–	–	–		✓			Refine task into subprocess
17	Refine task into subprocess	–	–	–	–	–	–	–	–		✓			Refine task into subprocess
18	Refactor gateway	1	2	–	–	–	–	–	–		✓			–
19	Add business-relevant task to Business	1	2	–	–	–	–	–	–			✓		–

Table 4 continued

Scenario	Pattern instance	Added				Removed				Synchronization method				Tool refactoring
		Tasks	Flows	Events	Gateways	Tasks	Flows	Events	Gateways	Private (business only)	Private (IT only)	Public (B ⇒ IT)	Public (IT ⇒ B)	
20	Suppress specification activity	–	–	–	–	2	4	–	2			✓		–
21	Add business-relevant task to IT	1	2	–	–	–	–	–	–				✓	–
22	Split task into block	3	6	–	2	1	2	–	–				✓	–
23	Refactor gateway	1	2	–	–	–	–	–	–				✓	–

rizes the assessment made by the specialists. The tool was capable of correctly synchronizing all the individual edit patterns, with minor layout issues. We discuss these results in the Sect. 5.7.

5.6 Results: compound refinement patterns

Besides the *atomic* (simple) change cases, composed of single edit patterns per synchronization, we also tested the tool on other concrete cases, where one model update (typically on the Business side) lags behind the other. Such situation requires multiple edits to be synchronized at once.

Thus, we created seven extra scenarios (as shown in the Table 6) by combining multiple edit patterns together. The edits were combined according to the change history, i.e., respecting their occurrences over time. We divided the first 49 months of the projects' change history into seven periods of evolution, such that each scenario comprises 7 months of change in the IT process model.

The experiments to synchronize each compound scenario were conducted as follows. First, for each 7-month period, the initial versions of business and IT models were recovered from the repository and remodeled on the tool. Second, all the actual changes were solely made on the IT side, while the business view remained intact. Third, the Shared Model was updated from the IT view (i.e., public and private parts were synchronized). Finally, the resulting (updated) business model was compared to the actual corresponding version on the repository, and also discussed with BNB specialists.

This way we ensured that each extra scenario was a potential concrete case for synchronization. Table 7 summarizes the results we obtained. We discuss them in the next Sect. 5.7.

5.7 Discussion of results

Single Edit Scenarios

All the single edit patterns, commonly used by BNB specialists to create IT process models out of their Business specifications, were successfully synchronized by the shared model tool. The following factors contribute to this result:

- *Single edit patterns produce small impact* the number of model elements affected by a single edit pattern is small (e.g., 1–3 tasks). Changes produced by single edit patterns are well localized and affect few model element dependencies.
- *Some edits are private* a private edit does not (initially) affect the other model. However, it is critical for the synchronization mechanism to keep track of such private parts. This way the tool can correctly propagate a public change, especially when it has dependencies (e.g., flow connections) on private model elements.

Table 5 Evaluation results, single refinements

Scenario	Edit Pattern	Instance of (see "Appendix")	Result	Comments	Issues
1	Add manual task	CP2	Success	Manual task and corresponding incoming and outgoing flows were correctly updated on the Business view only	–
2	Change activity name	CP7	Success	New name was correctly updated on the Business view only	–
3	Simplify selection into task	CP10	Success	Selection of fine-grained changes was correctly simplified into a single task on the Business view	–
4	Add properties	CP1	Success	Specific properties were correctly updated on the IT view only	–
5	Add script task	CP3	Success	Script task and corresponding incoming and outgoing flows were correctly updated on the IT view only	–
6	Add script task	CP3	Success	Script task and corresponding incoming and outgoing flows were correctly updated on the IT view only	–
7	Add protocol task	CP4	Success	Protocol tasks and corresponding incoming and outgoing flows were correctly updated on the IT view only	–
8	Add protocol task	CP4	Success	Protocol task and corresponding incoming and outgoing flows were correctly updated on the IT view only	–
9	Add boundary event	CP5	Success	Event, flow, and task were correctly updated on the IT view only	–
10	Add technical exception flow	CP6	Success	Event, flow, and task were correctly updated on the IT view only	–
11	Add technical exception flow	CP6	Success	Event, flow, and task were correctly updated on the IT view only	–
12	Change activity type	CP8	Success	New type was correctly updated on the Business view only	–
13	Refine task into fragment	CP10	Success	Task was correctly refined into a fragment of other activities, using the refactoring method provided by the tool	–
14	Suppress specification activity	CP9	Success	Task and flows were correctly removed from the IT view only	–
15	Split task into block	CP10	Success	Selection of activities was correctly simplified into a single task on the Business view	Layout
16	Refine task into subprocess	CP11	Success	Task was correctly refined into a subprocess of other activities, using the refactoring method provided by the tool	–
17	Refine task into subprocess	CP11	Success	Task was correctly refined into a fragment of other activities, using the refactoring method provided by the tool	–
18	Refactor gateway	CP12	Success	New activities added and modified flows were correctly updated on the IT view	–
19	Add business-relevant task to Business	CP3	Success	Business-relevant task and its corresponding incoming and outgoing flows were correctly updated on the Business view, and propagated to the IT view as well	Layout
20	Suppress specification activity	CP9	Success	Gateway, tasks, and flows were correctly removed on the IT view only	–
21	Add business-relevant task to IT	CP3	Success	Business-relevant task and its corresponding incoming and outgoing flows were correctly updated on the IT view, and correctly propagated to the Business view as well	Layout
22	Split task into block	CP10	Success	Deleted task and other activities added were updated on the IT view and changes were correctly propagated to the Business view as well	Layout
23	Refactor gateway	CP12	Success	New activities added and modified flows were correctly updated on the IT view, and propagated to the Business view as well	Layout

Table 6 Evaluation scenarios: compound refinement patterns

Scenario	Period	Number of changes (IT)	Private (%)	Public (%)	Added				Removed			
					Tasks	Flows	Events	Gateways	Tasks	Flows	Events	Gateways
24	May/09–Nov/09	27	30	70	83	116	19	41	–	–	–	–
25	Dec/09–Jun/10	23	48	52	12	22	8	6	–	–	–	–
26	Jul/10–Jan/11	15	67	33	10	8	–	–	3	16	5	2
27	Feb/11–Aug/11	10	70	30	5	14	–	–	5	18	4	2
28	Sep/11–Mar/12	19	73	27	14	30	–	2	–	–	2	–
29	Apr/12–Oct/12	8	60	40	7	14	–	–	–	–	–	–
30	Nov/12–May/13	6	83	17	3	6	–	–	1	3	1	–

Table 7 Evaluation results, compound refinements

Scenario	Result	Comments	Issues
24	Success	All changes were correctly synchronized. A consistent business view was generated	Layout. Some sequence flows were broken
25	Success	All changes were correctly synchronized. A consistent business view was generated	Layout. Some sequence flows were broken
26	Success	All changes were correctly synchronized. A consistent business view was generated	Layout. Some sequence flows were broken
27	Success	All changes were correctly synchronized. A consistent business view was generated	Layout. Some sequence flows were broken
28	Success	All changes were correctly synchronized. A consistent business view was generated	Layout
29	Success	All changes were correctly synchronized. A consistent business view was generated	Layout. Some sequence flows were broken
30	Success	All changes were correctly synchronized. A consistent business view was generated	Layout

Some edit synchronizations (15, 19, 21, 22 and 23) caused broken *layout* of the synchronized views, such as new model elements overlapping pre-existing ones and sequence flows being entangled. Such cases require manual adjustments on the views to make them visually clean. Although BNB specialists considered this a minor issue, they pondered that this may become a tedious task in practice.

Compound Edit Scenarios

To deal with the scenarios combining multiple edit patterns, we needed to divide the synchronization in two parts: one collecting all public changes, and another collecting all the private ones.

For each change period shown in Table 6 we performed the following steps (Fig. 8 shows an overview):

1. Recovered the initial and final versions of both business and IT models;

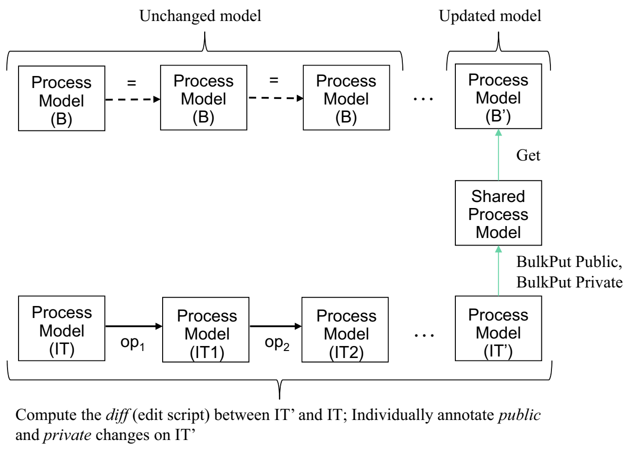


Fig. 8 Synchronization of compound edits

2. Computed the *diff* (i.e., an *edit script* showing all *inserted*, *deleted*, or *updated* model elements) between the two versions of the IT model [23];
3. Manually decided which parts were *private* IT changes and which were *public* (IT and business changes). We

counted on help from BNB specialists to recover such information from the change request log, documentation, and their own expertise regarding the models.
4. Replayed all the public changes in the IT model and pushed the updates to the shared model;
5. Replayed all the private changes in the IT model and pushed the updated to the shared model;
6. Obtained an updated view of the business model;
7. Compared the updated view with the current (final) version of the business model on the repository.

The third step (above) was the most laborious and time-consuming. We needed to count on domain knowledge from BNB specialists to precisely distinguish which individual edits were public or private. That distinction was only possible by also inspecting the change request log, which contains textual descriptions of each change, and also the project's documentation.

All compound scenarios were correctly synchronized, and a consistent business view was eventually produced, as shown in the Table 7. Some issues were occasionally observed in the generated business view: broken layout (as the aforementioned entangled flows and overlapping model elements) and missing sequence flows. The later does not represent a problem with the synchronization mechanism, but rather requires improving the current heuristic implemented by the prototype to infer graph dependencies (sequence flows) between public and private parts. Figure 9 shows an example of inferred sequence flow between *Task X* and *Task Z* on the business side, after synchronizing all public and private changes in the IT side.

Concurrent Changes

As explained in the Sect. 3.2, the current prototype does not deal with cases where both views are changed concurrently. Such cases would need comparing and merging different instances of the shared model, as shown in the Fig. 10. We plan to further study this problem as future work.

We conclude the discussion by answering our initial evaluation questions:

Q1 *How successfully can the tool synchronize typical Business-to-IT process modeling edit patterns?*

The tool was able to deal well with all the evaluated scenarios from BNB, when synchronizing after each edit pattern. However, since the evaluation relies on data from a single company, additional experiments with data from other organizations is needed to strengthen this result.

Q2 *How successfully can the tool synchronize scenarios composed of multiple edit patterns?*

In its current version, the tool can deal with scenarios where multiple edits need to be synchronized at once, as long as it is possible to distinguish between public and private individual edits. Manually performing such task is, however, a painstaking and time-consuming effort. More research is necessary to understand how such mechanism for dealing with intertwined types of changes, during the development process, could be built.

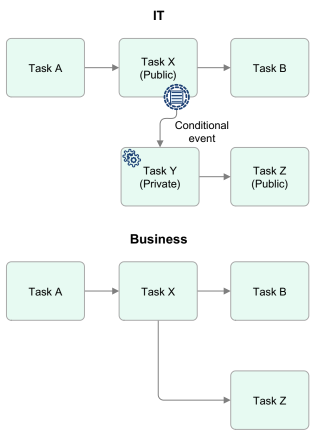


Fig. 9 Public and private synchronization dependencies

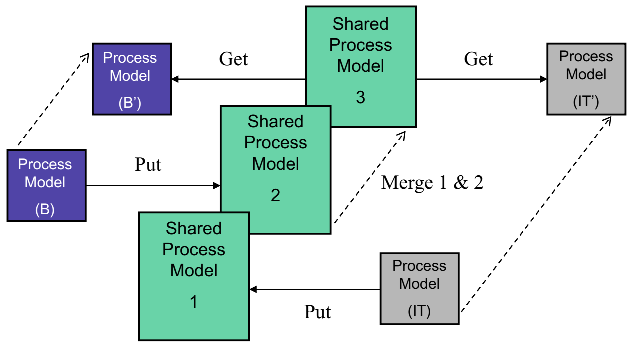


Fig. 10 Synchronization of concurrent changes

Q3 *Are there recommended best practices in using the tool, such that they could ensure consistency between Business and IT views?*

Yes. The tool can ensure consistent Business and IT views if the development process enforces that each occurrence of a refinement pattern performed in either model is synchronized as soon as it occurs. This approach avoids the problem discussed in the previous question. However, assessing the feasibility of this approach in industrial practice requires further research.

5.8 Threats to validity and lessons learned

Many tool evaluations suffer from limitations, such as the number of subjects not being representative of the entire population, the differences between development methods employed across different organizations, and so on. This evaluation is subject to three main limitations:

- *The limited number of evaluation scenarios* It is difficult to obtain data to drive research in the domain of process modeling. For example—to the best of our knowledge—there are no available open source projects featuring business-level specifications and their IT-level implementations. Also, companies which adopt such technologies usually consider process artifacts extremely sensitive and confidential. We obtained access to people and artifacts from BNB. However, mining the artifact repository, the change log, and remodeling the project and change scenarios was a laborious and time-consuming effort. Such tasks were only possible with help from several BNB technicians and managers.
- *The artifacts come from a single company (domain)* Clearly, different development processes and organizational cultures will likely lead to different results.
- *The evaluation relied on subjective and relatively quick assessments of the BNB specialists* While the correspondence and edit patterns are grounded in the studied artifacts, assessments of consistency are based on subjective perceptions of the participating specialists.

Although the observed results are very promising, it is important to note that (in practice) there may exist many other types of organization-, domain-, or even project-specific edit patterns. Clearly, there may also exist many test cases where the context (dependencies) of the model elements affected by an edit may lead to synchronization failures. We do not intend to claim that the tool (in its current version) can successfully synchronize all types of changes.

6 Related work

We build on prior work [20] 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 [24]. 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. [25]. In recent years, various techniques have been developed for synchronization of models. Popular approaches are based on triple graph grammars (e.g., Giese et al. [8]). 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. [7] and Cicchetti et al. [26,27]). 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 in 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. However, the immediate applicability and effectiveness of the more general approaches in this more specific context are not clear. In fact, such general approaches would need to be refined with many of the topics we discussed, such as the different types of consistency and change patterns.

On an even more general level, (in)consistency management of different views has been extensively studied in recent years by many authors (e.g., Finkelstein et al. [28], Egyed et al. [29]). The goal of these works is to define and manage consistency of different views where views can be diverse software artefacts including models. As stated earlier, 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. [30] 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. [31] 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. [13] 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. [32] 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. [33] 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., [34]. A process view can be recomputed whenever the underlying process model changes. Recently, Kolb et al. [11] have taken the idea further to allow changes in 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.

7 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 practitioners. 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 report on a substantial evaluation using an industrial case study, which investigated the tool in 23 single edit pattern scenarios and seven compound ones. The evaluation revealed strengths and current limitations of the approach.

Suggestions for future work include improving the synchronization of compound scenarios, as well as handling concurrent changes. Such improvements require further research and case studies. 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 work.

Acknowledgments We would like to thank the Bank of the Northeast of Brazil (Banco do Nordeste – BNB) for granting us full access to people and artifacts, fundamental for conducting the evaluation of the approach. This work was partially supported by an IBM Ph.D. CAS Fellow Scholarship, the Ontario Research Fund's Research Excellence Project on Model-Integrated Software Service Engineering.

Appendix

In this appendix, we summarize the correspondence patterns (***CPs***) described in previous work [2]. All correspondence patterns are illustrated by an ATM system example shown in Fig. 11. More details can also be found in a technical report [35].

CP1: add properties

Description Parameters for grounding the executable model on top of the underlying IT infrastructure are added during the implementation.
Motivation Several properties of tasks, gateways, flows, events, etc., are added to the implementation-level model, such as application or service URLs, protocol types (e.g., http or https), transactional behavior (e.g., commit before, commit after, participates, etc.). Such properties do not change the workflow and may be tool or platform-specific.
Example Each ISO8583 sending or receiving task shown in Fig. 11 (e.g., *Identify Customer Card 9300* and *Get Card Identification 9310*) has parameters that include the message queue, authentication method, security protocol, and message encoding.

CP2: add manual task

Description Some tasks on the business side are not subject to automation.
Motivation Manual tasks are used for non-automated (typically human-performed) actions of a process, such as transporting assets via postal service, stowing retrieval, visual inspection, etc. Manual tasks are commonly used solely on the business view, since they have no counterpart on the company's IT infrastructure.
Example A credit process can contain a task to send a hard copy of a contract to the customer, via postal service.

CP3: add script task

Description Script tasks are used to initialize variables and implement business rules and non-functional requirements that access or transform business objects data, e.g., logging steps of the workflow.
Motivation This type of task is frequently used because it has significantly better performance than calling external services.
Example Figure 12 shows a task created in the ATM application for initializing several parameters of a *transaction* object, which controls user actions across the workflow. Such kind of task in the IT model does not have any correspondence in the business model.

CP4: add protocol task

Description An asynchronous service can be implemented by a connectionless request or reply protocol.

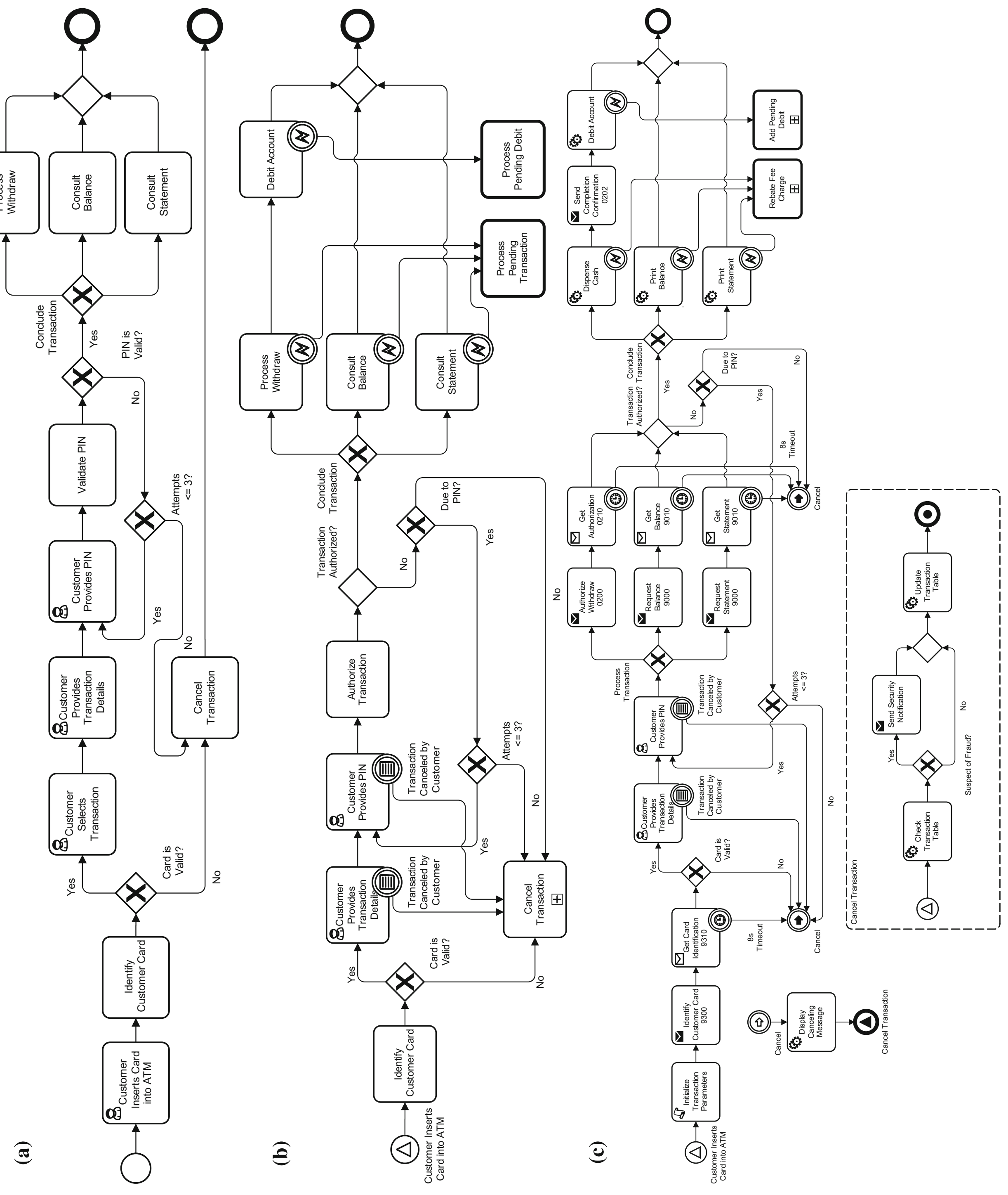


Fig. 11 ATM process models. **a** Business specification, **b** technical specification, **c** executable process

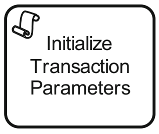


Fig. 12 Add script task

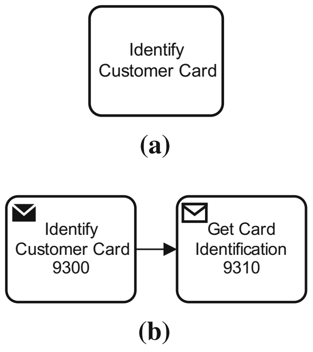


Fig. 13 Add protocol task. **a** Business and technical specifications, **b** executable

Motivation It is common to implement a business task by using an asynchronous connectionless service. In such cases, the protocol needs to compose and send a message and, after that, wait for a response.
Example Figure 13 shows an example where the business task *Identify Customer Card* is implemented on top of the ISO8583 protocol by sending a identification request message (9300) and waiting for a validation message (9310).

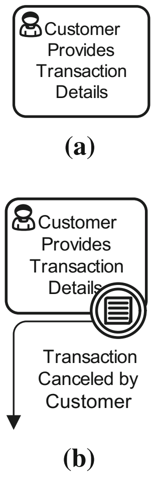
CP5: add boundary event

Description Boundary events are used to divert the normal flow under special conditions, for example, because of a particular action performed by the operator on a human task.
Motivation The reason to divert the flow can be merely technical or too low level to be represented in the business model. Such conditions can be implemented as result of requirements and use cases that describe a human task in detail.
Example Figure 14 depicts an example of boundary event added to human tasks to capture the customer's decision to cancel the transaction at any time. Another example can be seen in Fig. 11, where boundary events were added to asynchronous receiving tasks (e.g., *Get Statement 9010*) to cancel the transaction in the case of a timeout of 8s.

CP6: add technical exception flow

Description Technical exception flows are included to divert the flow in case of technical exceptions, such as an unavailable service or a permission denied.
Motivation Technical exceptions are not expected to be represented in the business model, because they implement non-

Fig. 14 Add boundary event. **a** Business specification, **b** technical and executable


Customer Provides Transaction Details
(a)
Customer Provides Transaction Details
Transaction Canceled by Customer
(b)

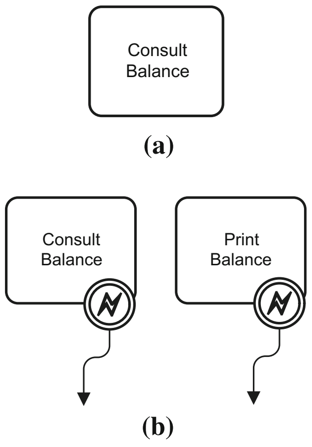


Fig. 15 Add technical exception flow. **a** Business specification, **b** technical and executable

functional requirements elicited during the *elaboration* phase of the development process.
Example Figure 15 shows examples of technical exceptions flows added for dealing with service errors, in which the transaction parameters are saved and the system administrator is notified to complete the transaction later.

CP7: change activity name

Description The name of a business activity can be changed to facilitate the identification of an IT service that has a similar but different name.
Motivation IT specialists can decide to use technical names in model elements for facilitating maintenance.
Example Figure 16 shows an example.

CP8: change activity type

Description The type of a model element can be changed because of an implementation decision.

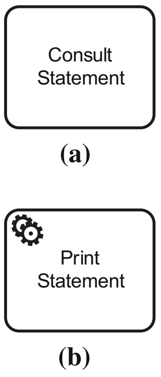


Fig. 16 Change activity name. **a** Business and technical specifications, **b** executable

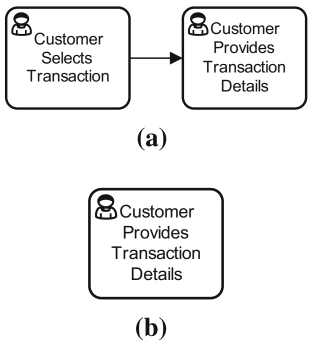

Customer Selects Transaction
Customer Provides Transaction Details
(a)
Customer Provides Transaction Details
(b)

Fig. 18 Suppress specification activity. **a** Business specification, **b** technical and executable

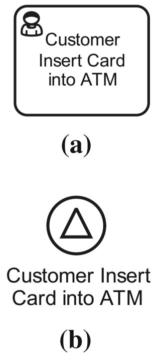


Fig. 17 Change activity type. **a** Business specification, **b** technical and executable

Motivation It is easier for business people to stick with basic modeling constructs (such as plain tasks and gateways), while other types of model elements are more suitable to implement the business intent.
Example Figure 17 shows an example were a human task represented in the business model was implemented by an event.

CP9: suppress specification activity

Description Business elements can be suppressed during the implementation.
Motivation Some elements of the business specification may be considered redundant, not subject to automation, or subsumed by a particular task at the implementation level. Typical cases for observing this pattern are:

- Combine several business tasks into a single service call (the service provided is coarser than the business steps described),
- Combine human tasks into a single human task, with the separate steps of the human task being described elsewhere as a screenflow, for example.
- Ignore manual business tasks, for example, "Send contract to the post office."

Example Figure 18 shows a case where the two human tasks described in the business model were collapsed into a single human task in the technical and implementation levels.

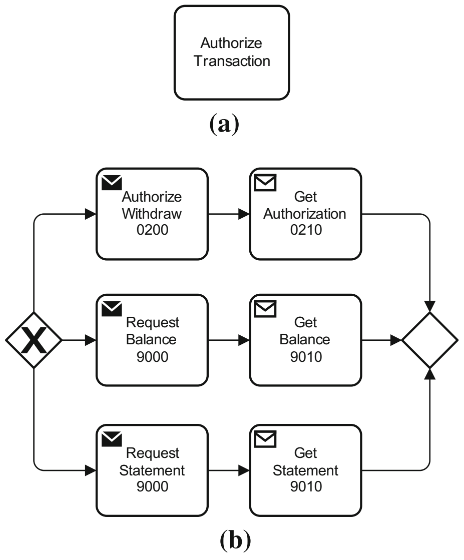


Fig. 19 Split task into block. **a** Technical specification, **b** executable

CP10: split task into block

Description A single business task can be implemented by a combination of services.
Motivation To implement a specification task, it may be necessary to combine several existing services, including additional control flow logic to organize the way the services should be called to achieve the specified functionality.
Example Figure 19 illustrates such scenario, where a technical specification task, *Authorize Transaction*, is split into a block of ISO8583 service calls, organized as an exclusive gateway that controls the type of authorization required for each transaction type.

CP11: split workflow

Description The specification workflow can be split into smaller workflows that should be orchestrated by a main flow.

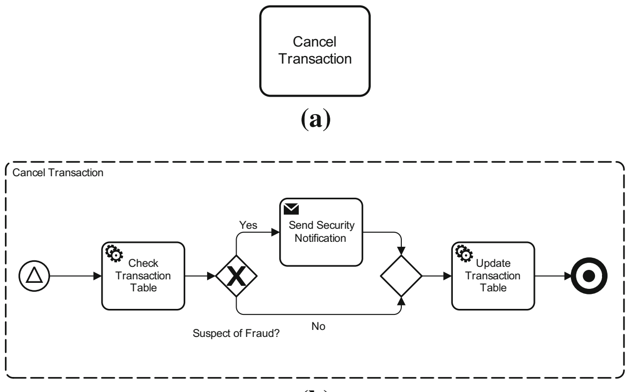


Fig. 20 Split workflow. **a** Business and technical specifications, **b** executable

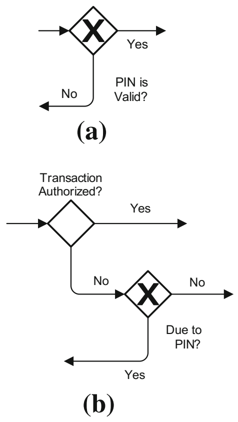


Fig. 21 Refactor gateway. **a** Business specification, **b** technical and executable

Motivation The typical reason for this pattern is the creation of specialized and reusable workflows, such as for logging and auditing purposes.
Example In Fig. 20, the task *Cancel Transaction* was implemented by a specialized subflow that includes fraud control and is reused by other projects. It is common to use web service interfaces or event triggering for calling the subflows.

CP12: refactor gateway

Description A business-level gateway may need to be refined to take into account the technical behavior of the services involved.
Motivation IT services may impose constraints on the control flow. For example, the business model may specify tasks executing in parallel; however, in the implementation the corresponding IT services are called in sequence to avoid deadlocks.
Example Figure 21 shows an example where the business specification has a rule for checking the maximum number of times that a customer can enter a wrong PIN. In the actual implementation, checking the validity of the PIN is a particular result of the transaction authorization. In this particular project, some of the other cases where the transaction is not authorized are also relevant to the business (e.g., insufficient funding). However, since the business analysts did not know how the systems were implemented, they specified such cases as part of business rules of three business tasks: *Process Withdraw*, *Consult Balance*, and *Consult Statement*. Business rules documents are produced together with business process models. The business analysts did not consider necessary to change the business model to approximate it to the actual system, at which point the workflows became different.

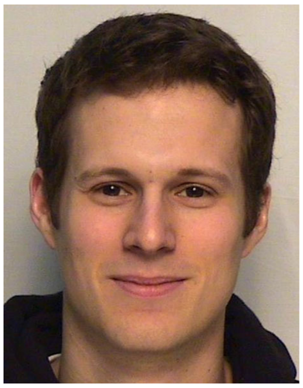
References

1. Recker, J., Mendling, J.: On the translation between bpmn and bpel: conceptual mismatch between process modeling languages. In: Proceedings of 11th International Workshop on Exploring Modeling Methods in Systems Analysis and Design, 2006
2. Branco, M.C., Xiong, Y., Czarnecki, K., Küster, J., Völzer, H.: A case study on consistency management of business and IT process models in banking. Software and Systems Modeling, March 2013
3. Weidlich, M., Barros, A.P., Mendling, J., Weske, M.: Vertical alignment of process models—how can we get there? In: Proceedings of Enterprise, Business-Process and Information Systems Modeling, 10th International Workshop, BPMDS 2009, ser. Lecture Notes in Business Information Processing, vol. 29, pp. 71–84. Springer, Berlin (2009)
4. Weidlich, M., Decker, G., Weske, M., Barros, A.: Towards vertical alignment of process models—a collection of mismatches. Hasso Plattner Institute, Tech. Rep., 2008. [Online]. http://bpt.hpi.uni-potsdam.de/pub/Public/MatthiasWeidlich/collection_of_mismatches.pdf
5. De Castro, V., Marcos, E., Wieringa, R.: Towards a service-oriented MDA-based approach to the alignment of business processes with it systems: from the business model to a web service composition model. Int. J. Coop. Inf. Syst. **18**(2), 225–260 (2009). [Online]. http://doc.utwente.nl/68172/
6. Luftman, J., Papp, R., Brier, T.: Enablers and inhibitors of business-IT alignment. Commun. AIS, vol. 1, March 1999. [Online]. http://portal.acm.org/citation.cfm?id=374122.374123
7. Herrmannsdoerfer, M., Benz, S., Jürgens, E.: Cope - automating coupled evolution of metamodels and models. In: Drossopoulou, S. (ed.) ECOOP, ser. Lecture Notes in Computer Science, vol. 5653, pp. 52–76. Springer, Berlin (2009)
8. Giese, H., Wagner, R.: From model transformation to incremental bidirectional model synchronization. Softw. Syst. Model. **8**(1), 21–43 (2009)
9. Diskin, Z.: Model synchronization: mappings, tiles, and categories. In: Proceedings of the 3rd international summer school conference on generative and transformational techniques in software engineering III, ser. GTTSE'09, pp. 92–165. Springer, Berlin (2011). [Online]. http://portal.acm.org/citation.cfm?id=1949925.1949929
10. Diskin, Z., Xiong, Y., Czarnecki, K.: Specifying overlaps of heterogeneous models for global consistency checking. In: Proceedings of the first international workshop on model-driven interoperability, ser. MDI '10, pp. 42–51. ACM, New York (2010). [Online]. doi:10.1145/1866272.1866279
11. Kolb, J., Kammerer, K., Reichert, M.: Updatable process views for user-centered adaption of large process models. In: Liu, C., Ludwig, H., Toumani, F., Yu, Q. (eds.) ICSOC, ser. Lecture Notes in Computer Science, vol. 7636, pp. 484–498. Springer, Berlin (2012)
12. Küster, J.M., Völzer, H., Favre, C., Branco, M.C., Czarnecki, K.: Supporting different process views through a shared process model.

In: 9th European Conference on Modelling Foundations and Applications, ECMFA, 2013, pp. 20–36
13. Tran, H., Zdun, U., Dustdar, S.: View-based integration of process-driven SOA models at various abstraction levels. In: Kutsche, R.-D., Milanovic, N. (eds.) 1st International Workshop on Model-Based Software and Data Integration. Springer, Berlin (2008)
14. Winkler, S., von Pilgrim, J.: A survey of traceability in requirements engineering and model-driven development. Softw. Syst. Model. **9**(4), 529–565 (2010)
15. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Syst. J. **45**(3), 621–645 (2006)
16. Branco, M., Troya, J., Czarnecki, K., Küster, J.M., Völzer, H.: Matching business process workflows across abstraction levels. In: France, R. B., Kazmeier, J., Breu, R., Atkinson, C. (eds.) MoDELS, ser. Lecture Notes in Computer Science, vol. 7590, pp. 626–641. Springer, Berlin (2012)
17. Weidlich, M., Mendling, J., Weske, M.: Efficient consistency measurement based on behavioural profiles of process models. IEEE Transactions on Software Engineering, vol. 99, no. PrePrints (2010)
18. Bergstra, J.A.: The Linear time — branching time spectrum I. The semantics of concrete, sequential processes. In: Ponse, A., Smolka, S.A. (eds.) Handbook of Process Algebra, Chap. 1. Elsevier Science Inc., New York (2001)
19. Vanhatalo, J., Völzer, H., Leymann, F.: Faster and more focused control-flow analysis for business process models through SESE decomposition. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007, ser. LNCS, pp. 43–55. Springer, Berlin (2007)
20. Küster, J.M., Gerth, C., Förster, A., Engels, G.: Detecting and resolving process model differences in the absence of a change log. In: Dumas, M., Reichert, M. (eds.) BPM'08, ser. LNCS, vol. 5240, pp. 244–260. Springer, Berlin (2008)
21. Favre, C., Küster, J., Völzer, H.: Recorded demo of shared process model prototype. http://researcher.ibm.com/view_project.php?id=3210
22. OMG, Business process model and notation (BPMN) version 2.0, omg document number dtc/2010-05-03, Tech. Rep., 2010
23. Branco, M.C., Wider, A.: Generating preliminary edit lenses from automatic pattern discovery in business process modeling. In: CAiSE Forum, 2013, pp. 65–72
24. Diskin, Z., Xiong, Y., Czarnecki, K., Ehrig, H., Hermann, F., Orejas, F.: From state- to delta-based bidirectional model transformations: the symmetric case. In: MoDELS, ser. Lecture Notes in Computer Science, vol. 6981, pp. 304–318. Springer, Berlin (2011)
25. Hofmann, M., Pierce, B., Wagner, D.: Symmetric lenses. In: POPL. ACM, pp. 371–384 (2011)
26. Cicchetti, A., Ruscio, D.D., Pierantonio, A.: Managing dependent changes in coupled evolution. In: Paige, R.F. (ed.) ICMT, ser. Lecture Notes in Computer Science, vol. 5563, pp. 35–51. Springer, Berlin (2009)
27. Cicchetti, A., Ciccozzi, F., Leveque, T.: A solution for concurrent versioning of metamodels and models. J. Object Technol. **11**(3), 1–32 (2012)
28. Finkelstein, A., Gabbay, D., Hunter, A., Kramer, J., Nuseibeh, B.: Inconsistency handling in multi-perspective specifications. In: ESEC, ser. Lecture Notes in Computer Science, vol. 717, pp. 84–99. Springer, Berlin (1993)
29. Egyed, A.: Instant consistency checking for the UML. In: ICSE 2006, pp. 381–390. ACM, New York (2006)
30. Weidlich, M., Dijkman, R., Mendling, J.: The ICoP framework: identification of correspondences between process models. In: CAiSE, ser. LNCS, vol. 6051, pp. 483–498. Springer, Berlin (2010)
31. Buchwald, S., Bauer, T., Reichert, M.: Bridging the gap between business process models and service composition specifications. Methods, Trends and Advances, Int'l Handbook on Service Life Cycle Tools and Technologies (2011)
32. Werth, D., Leyking, K., Dreifus, F., Ziemann, J., Martin, A.: Managing SOA through business services—a business-oriented approach to service-oriented architectures. In: ICSOC Workshops, ser. Lecture Notes in Computer Science, vol. 4652, pp. 3–13. Springer, Berlin (2006)
33. Thomas, O., Leyking, K., Dreifus, F.: Using process models for the design of service-oriented architectures: methodology and e-commerce case study. In: HICSS. IEEE Computer Society, 2008, p. 109
34. Schumm, D., Leymann, F., Streule, A.: Process viewing patterns. In: EDOC. IEEE Computer Society, 2010, pp. 89–98
35. Branco, M., Xiong, Y., Czarnecki, K., Küster, J., Völzer, H.: An empirical study on consistency management of business and IT process models", Generative Software Development Laboratory, University of Waterloo, Technical Report GSDLAB-TR 2012–03-22, 2012, accepted for publication in Software and Systems Modeling, draft available at http://gsd.uwaterloo.ca/reportstudybpm

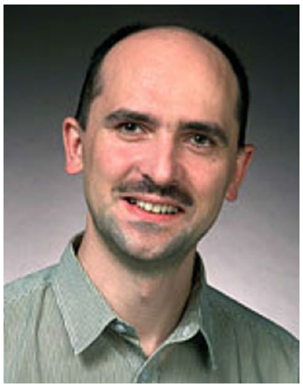
Jochen Küster studied computer science with mathematics at the University of Paderborn, Paderborn, Germany, with stays at University College London, UK, and Carleton University, Canada. He received the degree of Diplom-Informatiker and the Ph.D. degree, from the University of Paderborn in 2000 and March 2004, respectively. From 2004 until 2013, he was with the Zurich Research Laboratory, IBM Research Division, Switzerland. In September 2013, he was appointed Professor of Business Informatics at Bielefeld University of Applied Sciences, Germany. He is the author of more than 50 peer-reviewed publications in the area of model-driven development, model transformations, and business process modeling.

Hagen Völzer received his master (1995) and doctoral (2000) degrees in computer science from Humboldt-University Berlin, Germany. He has been a research fellow at the Software Verification Research Centre at the University of Queensland, Australia (2001–2003), and a senior research and teaching associate at the University of Lübeck, Germany (2003–2007). He joined IBM Research in March 2007. With business process management (BPM) as his current research focus, he is interested in all aspects of process modeling, analysis, refinement, execution, and monitoring as well as in related topics such as consistency-, configuration-, and change management for processes. He has contributed to and co-authored the OMG standard BPMN 2.0.



Cédric Favre received his M.Sc., in Computer Science from the Swiss Federal Institute of Technology in Lausanne (EPFL) in 2008. He then became a Ph.D., student at IBM Research, Zurich, and at the Swiss Federal Institute of Technology in Zurich (ETHZ), where he successfully defended his PhD thesis in September 2014. His research focuses on static analysis techniques and tools allowing a user, with no verification background, to detect, locate, understand, and repair control-flow errors in business process models.

Moisés Castelo Branco received his master degree (1999) in Computer Science from Universidade Federal do Ceará, Brazil, and his Ph.D., in Electrical and Computer Engineering (2014) from University of Waterloo, Canada. He has been working in industry for the past two decades performing major software engineering roles and currently holds a managerial position at Bank of Northeast of Brazil (BNB), leading the Systems Division of Banking Products and Services. While atBNB, he also had the opportunity of leading the IT Architecture Team (2003–2009), when actively participated in the adoption of a SOA/BPM infrastructure. His research interests focus on consistency management of software artefacts, including matching, traceability, and impact analysis among models that express the same intent at different abstraction levels, ranging from high-level business specifications to IT executable models.



Krzysztof Czarnecki is a Professor of Electrical and Computer Engineering at the University of Waterloo. Before coming to Waterloo, he was a researcher at DaimlerChrysler Research (1995–2002), Germany, focusing on improving software development practices and technologies in enterprise, automotive, space and aerospace domains. He co-authored the book on "Generative Programming" (Addison-Wesley, 2000), which deals with automating software component assembly based on domain-specific languages. While at Waterloo, he held the NSERC/Bank of Nova Scotia Industrial Research Chair in Requirements Engineering of Service-oriented Software Systems (2008–2013) and has worked on a range of topics in model-driven software engineering, including software-product lines and variability modeling, consistency management and bi-directional transformations, and example-driven modeling. He received the Premier's Research Excellence Award in 2004 and the British Computing Society in Upper Canada Award for Outstanding Contributions to IT Industry in 2008.