Managing Consistency of Business Process Models across Abstraction Levels

by

Moisés Almeida Castelo Branco

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2014

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Process models support the transition from business requirements to IT implementations. An organization that adopts process modeling often maintain several co-existing models of the same business process. These models target different abstraction levels and stakeholder perspectives. Maintaining consistency among these models has become a major challenge for such an organization. For instance, propagating changes requires identifying tacit correspondences among the models, which may be only in the memories of their original creators or may be lost entirely.

Although different tools target specific needs of different roles, we lack appropriate support for checking whether related models maintained by different groups of specialists are still consistent after independent editing. As a result, typical consistency management tasks such as tracing, differencing, comparing, refactoring, merging, conformance checking, change notification, and versioning are frequently done manually, which is time-consuming and error-prone.

This thesis presents the *Shared Model*, a framework designed to improve support for consistency management and impact analysis in process modeling. The framework is designed as a result of a comprehensive industrial study that elicited typical correspondence patterns between Business and IT process models and the meaning of consistency between them.

The framework encompasses three major techniques and contributions: 1) matching heuristics to automatically discover complex correspondences patterns among the models, and to maintain traceability among model parts—elements and fragments; 2) a generator of edit operations to compute the difference between process models; 3) a process model synchronizer, capable of consistently propagating changes made to any model to its counterpart.

We evaluated the Shared Model experimentally. The evaluation shows that the framework can consistently synchronize Business and IT views related by correspondence patterns, after non-simultaneous independent editing.

Acknowledgements

I would like to thank my supervisor, Prof. Krzysztof Czarnecki, for giving me the opportunity to be member of his research team. I was very fortunate to work both on theoretical and practical aspects of model consistency and model synchronization. I am grateful for providing the guidance and direction that has led to the completion of this thesis. I am also thankful for all the assistance that made my life in Canada a great experience.

I also acknowledge Jochen Küster, from Bielefeld University of Applied Sciences, Hagen Völzer, from IBM Research Zurich, and Alex Lau and Phil Coulthard, from IBM Canada Lab. Our collaboration was deeply inspiring and fruitful. Thank you very much!

Thank you to my colleagues in the Generative Software Development Lab and collaborators. A special thanks to Yingfei Xiong, Zinovy Diskin, Michal Antkiewicz, Kacper Bak, Thiago Tonelli, Rafael Lotufo, Leonardo Passos, Javier Troya, István Ráth and Arif Wider. You always found the time to discuss my research and were full of ideas. I admire you folks, because of your dedication, patience, and structured thinking. Your commitment to detail and work ethics deeply influenced the way I think and act today.

I would like to thank my external examiner, Prof. Dragan Gašević, for finding time to visit Waterloo and for his comments on this thesis. I am very grateful to my internal thesis committee, Prof. Daniel Berry, Prof. Lin Tan, and Prof. Paul Ward, for helping me shape this research.

Thank you to my family in Canada: Marcílio Mendonça, Laércio de Oliveira, Nelson Multari, Toacy Oliveira, Gabriel Caridade, Márcio Juliato and Hermênio Lima. Your continuous support and all the funny moments we had together were essential to balance my life.

Thank you very much to all my friends in the Bank of Northeast of Brazil. Many thanks to my great friends: Paulo Jucá, Jesuíno Freitas, Renato Helônio, Roberto Cysne, Ricardo Menezes, Alan Bandeira, and Airton Fernandes. Our daily email messages were an invaluable source of fun and happiness. It would be difficult to finish PhD abroad without you! My special thanks to Rafael Fonseca, who gave me a great support on my empirical studies. My sincere gratitude to my former managers, Cláudio Reginaldo and Stélio Gama, who believed in me and gave me all support that I needed.

My deepest thanks to my wife, Mirela! Your love and dedication were absolutely indispensable to this achievement! Love you!

Finally, a big thanks to all my relatives and friends. My brothers, Oswaldo and Zuila, were always very supportive.

E a todos os amigos brasileiros no Canadá com quem tive o privilégio de conviver nessa fantástica jornada, meu muito obrigado!

To my parents, Oswaldo and Clara, my best source of inspiration.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Business Process Modeling (BPM) is increasingly used by enterprises to improve their agility and operational performance by better aligning their IT infrastructure with their business needs. Typically, a BPM-driven development process involves the participation and collaboration of many stakeholders (e.g., business analysts, systems analysts, IT architects and developers). These roles and responsibilities may be organizationally defined, be the result of the adopted development process, or simply reflect the different competencies and capabilities of the people involved. The distribution of responsibilities and roles usually results in the creation of different models of the same business process. These models vary from business-oriented ones, which are technology-independent and easily understandable by business people, to IT-oriented ones, which are constructed by taking into consideration technicalities of existing systems. Specialized modeling languages have been developed to represent such models, including *Business Process Modeling Notation* (BPMN) [95] for business-level models and *Web Services Business Process Execution Language* (BPEL) [101] for IT-level executable models. Since its 2.0 version, BPMN can also express executable models [102].

The multitude and heterogeneity of models created to describe a business process at different levels of abstraction and from different stakeholder perspectives lead often to inconsistencies among the models. Inconsistencies arise because the models overlap—for example, they contain elements that refer to common aspects of systems and other enterprise resources, such as organizational structure and flow of communication, and make assertions about these aspects that may be contradictory or not satisfiable under certain conditions. On the positive side, inconsistencies highlight different perceptions and goals of the stakeholders involved in the development process

and they can be intentionally introduced to indicate aspects of a process which deserve additional information elicitation and further development. On the negative side, inconsistencies can cause development delays, increased costs, and operational and audit failures.

To manage consistency of multiple business process models, researchers have proposed different approaches [27, 28, 65, 79, 88, 131], each targeting a sub-problem of consistency management. In practice, companies also employ their own processes to manage the consistency among multiple models. It is not clear to what extent the academic approaches are adopted by industry and what remaining challenges are still faced by practitioners. Conversely, many academic approaches are based on assumptions of how models are handled in practice, and some assumptions are even contradictory. For example, Zerguini [139] and Soffer [112] assume that models at different levels of abstraction are related in a strict top-down fashion via hierarchical refinements, whereas Weidlich et al. [128] propose that non-hierarchical refinements should also be considered. It is not clear which of these assumptions are true in practice. Thus, our research starts by collecting empirical evidence to derive requirements for consistency management of business process models.

1.2 Shared Model Overview

The Shared Process Model approach has the capability to synchronize process model views that reside on different abstraction levels. The concept is illustrated by Fig. 1.1. The Shared Process Model provides two different views, a business view and an IT view, and maintains the consistency between them. A current view can be obtained at any time by the corresponding stakeholder by the *get* operation. A view may also be changed by the corresponding stakeholder. With a *put* operation, the changed view can be checked into the Shared 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 he is working on the refinement of the IT model, that the model is missing an important business activity, he can insert that activity in the IT model. He can then check the change into the Shared Process Model, designating it as a public change to express that the activity should be inserted in the business view as well. The Shared Process Model then inserts the new activity in the business view automatically at the right position, i.e., every new business view henceforth obtained from the Shared Process Model will contain the new activity. If the IT architect designated the activity insertion as a private change, then the business view will not be updated and the new activity will henceforth be treated by the Shared Process Model as an IT-only activity.

Figure 1.1 also illustrates the main three status conditions of a Shared Process Model: business conformance, IT conformance and business-IT consistency. The business view is business conformant if it is approved by the business analyst, i.e., if it reflects the business requirements. This should include that the business view passes basic validity checks of the business modeling tool. The IT view is IT conformant if it is approved by the IT architect, i.e., if it meets the IT requirements. This should include that the IT view passes all validity checks of the IT modeling tool and the execution engine. Business-IT consistency means that the business view faithfully reflects the IT view, and equivalently, that the IT model faithfully implements the business view.

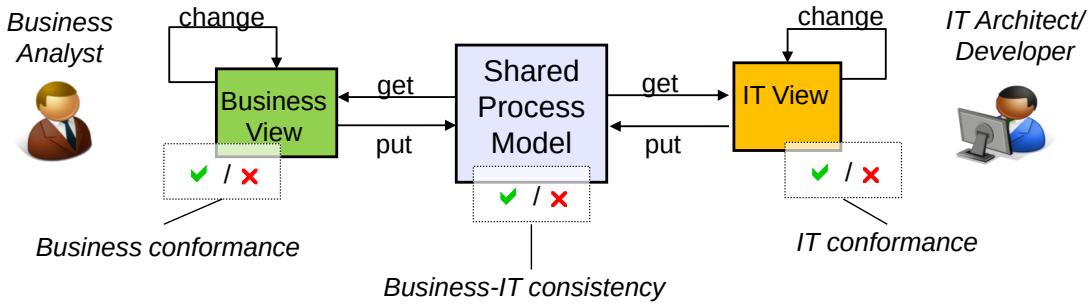


Figure 1.1: Process View Synchronization via a Shared Process Model

1.3 Research Contributions

The overarching goal of this work is to improve support for consistency management and impact analysis in business-process modeling.

This thesis proposes a practical framework for managing consistency of process models, which encompasses five main contributions:

1. *A detailed account of the relationship between models that target different levels of abstraction and how to characterize consistency among them.* We conduct an in-depth empirical study of a business-driven engineering process deployed at a large company in the banking sector (see Chapter 4 and [17]). We analyzed more than 70 business process models developed by the company, including their change history, with over 1000 change requests. We also interviewed 9 business and IT practitioners and surveyed 23 such practitioners to understand concrete difficulties in consistency management, the rationales for the specification-to-implementation refinements found in the models, strategies that the

practitioners use to detect and fix inconsistencies, and how tools could help with these tasks. Our contribution is a set of empirical findings that provide empirical evidence of 1) how business process models are created and maintained, including a set of recurrent patterns used to refine business-level process specifications into IT-level models; 2) what types of inconsistencies occur; how they are introduced; and what problems they cause; and 3) what stakeholders expect from tools to support consistency management.

2. *A technique to discover correspondences via matching and support traceability among the models*. We present a heuristic method for determining correspondences between process models (see Chapter 6 and [14]). A correspondence establishes which activities in one model correspond to which activities in another model. The heuristic is based on the aforementioned empirical study, which revealed frequent correspondence patterns between models spanning multiple abstraction levels. The heuristic has two phases: first, establishing correspondences based on similarity of model element attributes such as types and names, and then, refining the result based on the structure of the models. Compared to previous work, our algorithm can recover complex correspondences relating whole process fragments rather than just individual activities.

3. *A technique to generate edit operations based on the matching and support process model synchronization* (see Chapter 7 and [15]). We leverage the matching and present a practical approach for generating bidirectional model transformations in BPM based on edit operations. The operations are automatically generated and used by the framework to propagate changes and synchronize the models.

4. *A process model synchronizer* (see Chapter 8 and [85]). We present a practical approach—(*The Shared Process Model*)—that combines all the aforementioned techniques to automatically manage traceability links between Business and IT views and synchronize changes made to any view. A conceptual description of the approach is presented in the Chapter 5.

5. *A comprehensive evaluation of the framework on real-world modeling scenarios* (see Chapter 9 and [86]). We perform a comprehensive evaluation of the framework on actual BPM projects from an industry partner. We present recommended best practices to obtain the maximum benefit from the framework and also discuss its current limitations.

Figure 1.2 shows an overview of our proposed framework for consistency management in business process modeling. All the techniques we have developed are combined to follow a step-by-step process that we briefly describe here:

- First, Business and IT process models (BM, IT) are matched using appropriate heuristics to deal with common correspondence patterns. The matched models are augmented with structural information, by parsing them into Process Structure Trees (PSTs) [124]. BM^+ and IT^+ are the models enriched with the new information. The hooked arrows, $\hookrightarrow$ and $\hookleftarrow$, in Fig 1.2, represent inclusion mappings; the double arrow, $\leftrightarrow$, represents the correspondence mapping between the PSTs.

- Second, the *deltas*—i.e., changes made to either process model ($\mu_{BM}, \mu_{IT}, \mu_{BM^+}, \mu_{IT^+}$)—are represented as a set of edit operations; these operations can be either recorded or computed. The down arrows, $\Downarrow$, represent the deltas.

- Third, users select which edit operations need to be propagated ($bPpg$, $fPpg$) and the models are synchronized via model transformation (symmetric delta *Lens*).

- Finally, the new—i.e., *consistent*—versions of the models are updated (get_{BM}, get_{IT}) on both sides (BM', IT').

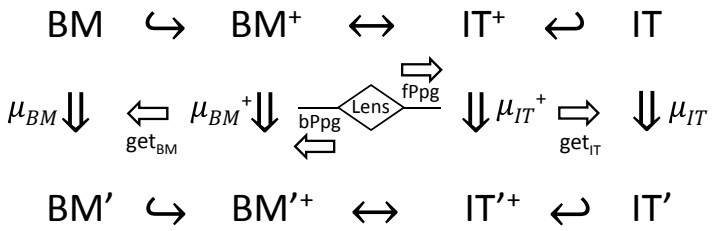


Figure 1.2: Framework Overview

Our proposed framework targets consistency management of process models that describe the same business intent in different abstraction levels. The framework comprises two major parts. The first one is the conceptual architecture, which describes consistency in terms of correspondence patterns and bi-directional model synchronization. The second part is a practical

realization of the framework implemented in the Java programming language, which can be leveraged by developers to extend and improve the techniques. The framework can be gradually improved as future work.

1.4 Research Method

The research was carried out in an iterative, example-driven, and experimental fashion. Work towards contribution 1 consisted of a series of empirical studies, including study of real-world business process models, questionnaires and interviews. By combining the artifact studies, questionnaires and interviews with the artifact creators and users, we were able to obtain a rich picture of the requirements and challenges related to creating and using business process models. Work towards contributions 2, 3, and 4 involved coming up with the theoretical underpinnings of concepts and methods to be created, conceptual design, prototyping, and experimental evaluation of the prototypes. Towards evaluating the framework (contribution 5), we simulated the evolution of the models by replaying the real history of model changes using BPM projects from The Bank of Northeast of Brazil (BNB)—the bank that provided our case study. Additionally, we obtained expert feedback from BNB engineers.

1.5 Outline of the Thesis

The remainder of the thesis is structured as follows. Chapter 2 provides background on BPM and describes the running example, the models of an *Automated Teller Machine* (ATM), which we use throughout the thesis. Chapter 3 discusses related work on model consistency management and other important topics for the thesis. Chapter 4 describes the empirical study design, presenting details about the organization, the analyzed projects and artifacts, conducted interviews and survey, and also presents the most relevant findings for this research proposal, including the refinement patterns catalog. Chapter 5 shows a conceptual view of the framework and its synchronization operations based on the formal theory of *symmetric delta lenses*. Chapter 6 presents our method to match process models at different levels of abstraction. Chapter 7 presents our method to generate edit lenses for model synchronization based on the automatic matching. Chapter 8 presents the concept of the shared model in detail. Chapter 9 evaluates our framework in practice, by means of a proof of concept with the shared model prototype. Finally, Chapter 10 summarizes the contributions of the thesis and highlights additional directions we envision to improve the framework.

1.6 Publications

This thesis contains material from the following publications:

- Moisés Castelo Branco, Javier Troya, Krzysztof Czarnecki, Jochen Küster, and Hagen Völzer. Matching Business Process Workflows Across Abstraction Levels. In Proceedings of 15th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2012. ACM/IEEE, 2012.

- Moisés Castelo Branco and Arif Wider. Generating Preliminary Edit Lenses from Automatic Pattern Discovery in Business Process Modeling. In Proceedings of the CAiSE 2013 Forum at the 25th International Conference on Advanced Information Systems Engineering (CAiSE), 2013.

- Moisés Castelo Branco, Yingfei Xiong, Krzysztof Czarnecki, Janette Wong, and Alex Lau. Effective Collaboration and Consistency Management in Business Process Modeling. In Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research, CASCON 2010, Riverton, NJ, USA, 2010. IBM Corp.

- Moisés Castelo Branco, Yingfei Xiong, Krzysztof Czarnecki, Jochen Küster, and Hagen Völzer. A Case Study on Consistency Management of Business and IT Process Models in Banking. Software and Systems Modeling (SoSyM), Special Issue on Enterprise Modeling, 2013.

- Weidlich, M., E. Sheetrit, M. C. Branco, and A. Gal. Matching Business Process Models Using Positional Language Models. In Proceedings of 32nd International Conference on Conceptual Modeling, ER 2013, Hong Kong, 2013.

- Küster, J., F. Cedric, H. Völzer, M. C. Branco, and K. Czarnecki. Supporting Different Process Views through a Shared Process Model. In Proceedings of 9th European Conference on Modelling Foundations and Applications, ECMFA 2013, 2013.

Chapter 2

Business Process Modeling: Background and a Running Example

2.1 Overview

A business process is a collection of related, structured or ad-hoc activities (tasks) that produce a specific output, such as service or product, for a particular customer or market [22]. Structured processes, which our study focuses on, are usually modeled as workflows, i.e., flows of activities. Typical examples of business processes are *Purchasing*, *Manufacturing*, *Marketing*, and *Sales*. A business process begins with a mission objective and normally ends with achievement of the objective. The activities of a process interact with IT assets to capture, transform, or report business data. As with processes, the data may be structured, such as a new order conforming to some well-defined schema, or ad-hoc (unstructured) data, such as an e-mail message [108].

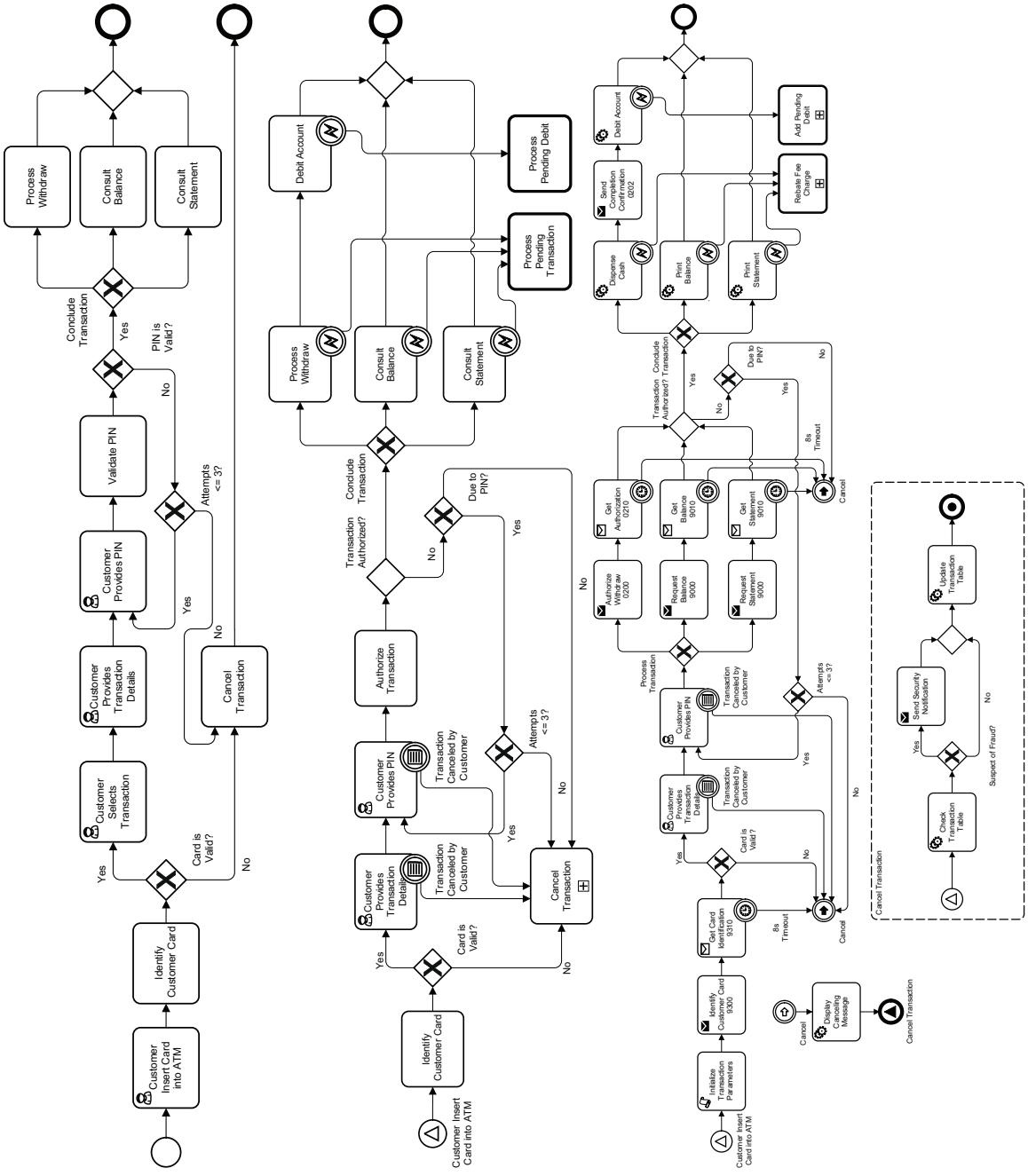
In practice, a range of business to IT-oriented stakeholders create and use business process models for specific purposes, including requirements elicitation, documentation, simulation, and execution [11]. Each model must be appropriate for its target audience and purpose—having adequate level of detail, focusing on relevant aspects, and neglecting irrelevant ones [65]. This goal can be achieved by creating either several separate models—each focused on particular set of stakeholders and purposes—or a single model with multiple views [12].

2.2 Running Example

Figure 2.1 shows three models, each representing the process of using an *Automated Teller Machine* (ATM) system at different level of abstraction. We will use these models, which are versions of real process models from one of the studied projects (project *P4*, Sect. 4.3), as our running example. The two specifications are the same as the original models, except that it has its labels translated from Portuguese to English. The IT model is 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 [102]. The control flow of the original models was entirely preserved in these examples. 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.

The first model (Fig. 2.1.a) represents a business-level process specification, which is created and maintained by Business Analysts. The second one (Fig. 2.1.b) is a refinement of the first one, created and maintained by IT Systems Analysts. These stakeholders use such models to align the modeled process with the existing service infrastructure; specify how the process interacts with IT assets; and ensure that the process is sound and free of design flaws, such as incomplete data objects and deadlocks. The third model (Fig. 2.1.c), created by IT Architects and Developers, refines the second model and represents the executable process implementation that goes into production. The executable process is implemented on top of an ISO8583 service infrastructure [73] and the codes that appear in the names of some tasks, such as 0200 and 9010, are types of messages of this protocol. Note that the final refinement (Fig. 2.1.c) consists of multiple, modularized executable models. These models orchestrate the actual services provided by the IT service infrastructure.

The models in Fig. 2.1 are expressed in BPMN. The notation represents activities by rounded rectangles, events by circles, gateways by diamonds (rhombi), and sequence flows by arrows (see Appendix A for legend of BPMN symbols used in the example). Each model has a start, usually modeled by an start event (e.g., *Customer insert Card into ATM*), and a flow of activities that is governed by decisions (e.g., *Card is Valid?*) and exceptions (e.g., *8s Timeout*). Each model also has an end point, which represents the achievement of the process: either a value delivered to an user or the termination of the process because of an error or a user decision (e.g., *Cancel Transaction*).



(a) Business Specification (b) Technical Specification (c) Executable Process

Figure 2.1: ATM Process Models

Chapter 3

Related Work

We discuss related work in five groups. First, we discuss the general problem of Business-IT alignment in BPM. Second, we introduce the general area of consistency management and discuss related work addressing specific consistency management tasks. Third, we turn to work on consistency management of business process models. Then, we discuss related work on bidirectional transformation frameworks. Finally, we review empirical work related to our thesis.

3.1 Business-IT Alignment in BPM

Bridging the gap between business and IT abstraction levels is a standard topic in enterprise systems engineering [20, 135]. Bieberstein et al. define business-IT alignment as "a dynamic state in which a business organization is able to use information technology (IT) effectively to achieve business objectives—typically improved financial performance or marketplace competitiveness." [11]

A key aspect of the business-IT alignment is establishing a correlation between business specifications and IT implementations. We discuss some of the existing approaches below. All approaches are based on assumptions on how specifications and implementations are created and how traceability between them is established and managed. Some approaches deal with issues of transforming business processes into executable models.

Buchwald et al. [19] present an approach which allows for a transfer of business requirements into executable processes. Their approach provides a three-level modeling method that automatically maintains an intermediate model called *Business-IT-Mapping Model* (BIMM) to

describe how activities from the business process are transferred into activities of the system process. A BIMM manages correspondences between model activities by means of transformation operations such as *rename, insert, remove, merge,* and *split*. A limitation of the approach is that it only considers consistency in terms of coverage, i.e., whether or not corresponding business-relevant activities are correctly mapped between the models. In practice, consistency involves other aspects, such as control flow (see Sect. 4.6.4).

Tran et at. [120] present a modeling framework realized as a view-based reverse engineering tool-chain. The framework maps process descriptions onto appropriate high-level or low-level views. The framework can be extended with support for different modeling languages, including BPMN and BPEL. Although the approach supports representing process structures at different levels of abstraction, it does not support consistency management among these views when they are independently edited (see Sect. 4.6.3).

Delgado et al. [26] provide a methodology for incremental development of business processes, based on the joint application of Model Driven Development and Service Oriented Computing paradigms. Their proposed methodology recognizes the need of integrating business and IT people into the development life-cycle and conveying the right level of detail as output of each development stage. Our work confirms such a need, by providing evidence on how business and IT people collaborate to create process models throughout the development process (see Sect 4.6.1).

Decker [24] proposes patterns for introducing a process support layer that solves incompatibilities between business- and IT-level process models. The work assumes that a single process model for business and IT is inherently undesirable and that both perspectives are hierarchically related to each other.

3.2 Consistency Management

Consistency management is a set of methods and tools for establishing and maintaining consistency among software artifacts, such as models, code, documentation, and test cases, which are usually created and used by multiple stakeholders [80, 113]. Existing works divide consistency management into a set of tasks [45, 80, 100]. The remainder of this subsection introduces these tasks and the corresponding related work in general; the next subsection discusses the related work specific to BPM.

- ***Defining consistency properties:*** Assuming a set of software models and a set of correspondence relations among their elements, consistency is a property of these models and

their correspondences [34, 42]. Such a property is typically defined as a consistency rule, expressed in some logic and interpreted in a knowledge domain. Knowledge domains range from well-formedness of language constructs to industry- and organization-specific policies, such as legal regulations and organization-specific IT standards [42]. For example, a reasonable policy is to require that every business-relevant task in an executable model (e.g., *Identify Customer Card 9300* in Fig. 2.1.c) is reflected in its business-level specification (*Identify Customer Card* in Fig. 2.1.a); conversely, a purely technical task (*Initialize Transaction Parameters* in Fig. 2.1.c) should not be reflected in the specification.

- ***Matching the models:*** This task deals with finding correspondence relations among elements of different models. For example, *Identify Customer Card* in Fig. 2.1.a corresponds to *Identify Customer Card* in Fig. 2.1.b, and to both *Identify Customer Card 9300* and *Get Card Identification 9310* in Fig 2.1.c. As we discuss in Sect. 4.6.3, process model matching is often challenging because identifying correspondences may require uncovering tacit knowledge, which may be only in the memories of the original creators of the models or may be lost entirely. Unless the correspondences have been recorded (e.g. via unique IDs), model alignment requires matching the models using domain- or organization-specific heuristics (e.g., by name and model structure). Examples of approaches that match different types of artifacts include document to code traceability recovery [90] and generic, graph-based matching [136]. A related area is schema integration, and in particular, schema matching, which deals with establishing correspondences among database schemas (see surveys on this topic [43, 107]). It is not clear how the existing techniques can be tailored to the problem of aligning process models. Our work presents evidence on how business and IT process models are related and how the maintenance process is done. This evidence will help developing appropriate alignment techniques.

- ***Checking consistency:*** Once the models are aligned, consistency is checked by evaluating the consistency rules. Spanoudakis and Zisman distinguish four types of approaches to consistency checking: logic-based approaches, model checking, specialized model analyses, and human-centered collaborative exploration [113]. The adopted consistency management policy is subjective and specifies the circumstances that will trigger the checks.

- ***Diagnosing causes of inconsistencies:*** This task identifies the source, the cause, and the impact of an inconsistency [113]. The source of an inconsistency is the set of elements of software models that violate a consistency rule [100]. The cause of an inconsistency could be conflicting stakeholder goals or just a mistake in one or more of the conflicting models. The impact of an inconsistency are the consequences that the inconsistency has on

the modeled system. Spanoudakis and Zisman include a survey of diagnosis approaches in their paper [113].

- ***Fixing inconsistencies:*** The final task is to fix inconsistencies. Ideally one or more fixes should be automatically proposed to the user. For example, Nentwich et al. [98] give approach that generates abstract fixes from first-order logic rules. An abstract fix specifies only the locations to be changed and the user needs to complete the edits. Egyed et al. [38] present an approach that generates concrete fixes for UML models, based on predefined inconsistency rules. Ameluxen et al. [3] propose an approach in which models are checked and corrected using graph transformation rules. Pinna et al. propose using an automated planning system, which does not require defining operations manually [105].

3.3 Consistency Management of Process Models

We summarize work on consistency management in the context of BPM.

- ***Defining consistency properties:*** Weidlich et al. categorize differences among related process models that can cause inconsistencies into the following types [128]:
  - *Model coverage differences* are differences of what the related models describe in terms of functionality. For example, a particular task can exist in one model, but may be missing in the other.
  - *Behavioral differences* are differences in how a particular functionality is implemented in each of the models. For instance, the execution sequence of corresponding tasks might differ.
  - *Information density differences* are differences in the level of detail. For example, one model might have two or more tasks that decompose a single corresponding task from another model.

  We used and confirmed the above categories to investigate how they affect consistency (see Sect. 4.6.3). Behavioral consistency typically involves some notion of behavioral equivalence, such as trace equivalence or bisimulation. For example, Küster [82] provides a behavioral consistency notion for object-oriented behavioral models. In contrast, Weidlich et al. view the consistency of two process models as a degree of consistency rather than a strict binary criterion [128, 129]. An example of such notion are *behavioral profiles* [132];

they replace strict criteria such as trace equivalence with less strict degree of trace similarity. They build on properties of free-choice Petri nets and give a numeric degree of consistency ranging from 0 (inconsistent models) to 1.0 (consistent models).

- ***Matching the models:*** Effective matching techniques applied to business process models require heuristics that are notation and application specific [29, 122, 130]. Discovery of effective heuristics usually requires studying the differences among such models. In this context, for example, Dijkman [27] presents another classification of frequently occurring differences between similar business processes in general, such changing names and types of activities and modifying the flow structure. Zerguini [139], Soffer [112] and Dijkman [30] present solutions for matching hierarchically related process models. Our study provides an in-depth analysis of differences between process models targeting different levels of abstraction and shows that non-hierarchical correspondences need to be taken into account (see Sect. 4.6.2). Based on our findings, we have recently presented an algorithm to automatically detect correspondences between BMPN process models across levels of abstraction [14]. The algorithm combines lexical and structural correspondences over the Process Structure Trees (PSTs) [124] of the input models in two phases. The first phase matches the PST nodes using region and model element matching criteria adapted from previous work on matching ASTs [46]. The second phase establishes additional correspondences based on the position of the nodes in the PSTs.

- ***Checking consistency:*** Checking consistency of business process models may involve checking simple structural rules, such as that each business relevant task in the executable models is reflected in the business level specification, or analyzing behavioral properties using model checking or specialized algorithms (e.g., [132]). Two special representations of process models are used in model comparison: process structure trees [123] and process model terms [48]. The first representation represents the essential structure of processes as trees, allowing their easy matching and structural comparison. The second representation gives a canonical representation of process models and allows efficiently checking for a particular relaxed form of behavioural equivalence. Weidlich et al. [128, 129, 132] propose generic frameworks for checking consistency of process models, based on task ordering. Our findings reveal that consistency checking should actually take into account subjective project- and domain-specific differences among the models.

- ***Diagnosing causes of inconsistencies:*** The process model differences classified by Weidlich et al. [128] represent potential causes of inconsistencies. Establishing the actual root causes of the inconsistencies, such as the conflicting goals of stakeholders, usually requires additional knowledge that is not present in the models. We are not aware of of any work investigating how diagnosis of inconsistencies among process models is done in practice.

- ***Fixing inconsistencies:*** Weidlich et al. [132] propose the concept of *behavioural profile* and present a method for computing them in cubic time, when the process models are translated to sound free-choice Petri nets w.r.t. their number of places and transitions. They propose a numeric measure of consistency, which can be used by the users to spot and assess potential inconsistencies. Our findings show that stakeholders prefer immediate notification and editing quick-fixes, integrated to the modeling tools, instead of an offline approach. Hegedüs et al. recently proposed an approach to fix model inconsistencies based on state-space exploration and evaluated it on BPMN models [58]. Küster et al. also discuss the change management and inconsistency resolution in BPM [83, 84].

3.4 Bidirectional Transformation Frameworks

Bidirectional transformation frameworks originate from the lens framework proposed by Foster et al. [47]. Lenses consider the asymmetric synchronization: one model is a view of the other, and define a state-based framework for asymmetric synchronization. "State-based" means that the synchronizer takes the states of models before and after update as input, and produces new states of models as output. Inspired by the lens framework, several researchers propose state-based framework for symmetric synchronization [69, 114]. As a more general case, symmetric synchronization allows neither of the model to be a view of the other. However, as Diskin et al. [33] point out, state-based bidirectional transformations actually mix two different operations—delta (correspondence relations between models or between different versions of a model) discovery and delta propagation, leading to several semantic problems. To fix these problems, several researchers [31, 33, 36, 70] propose delta-based frameworks, where deltas are taken as input and output. Typical delta-based frameworks include delta lens [33, 35] for the asymmetric cases, and symmetric delta lens [36] and edit lens [70] for the symmetric cases.

3.5 Empirical Research

We are not aware of any empirical research on consistency management in BPM, yet empirical studies exist in related areas.

Hutchinson et al. [72] address the relative absence of empirical studies of industrial model driven engineering (MDE) practices by describing lessons learned from three case studies. They applied a combination of research methods, such as interviews and questionnaire surveys for collecting data and deriving lessons learned from MDE practices adopted by three companies.

Compared to their work which focuses on MDE in general, our work focuses on BPM and consistency management.

Zapf and Heinzl [138] present an empirical study of process refinement patterns in the call center domain. They compare different process partitioning strategies as typical design patterns in call centers. The analysis provides insight to the question under which circumstances a specific pattern is used. Our study provides empirical evidence of how process refinement patterns are applied in the domain of banking applications.

Chapter 4

Empirical Study

4.1 Research Methods

The study was designed to answer the following, broadly-scoped research question:

How do people manage consistency of related business- and IT-level process models in practice?

We initially left our problem statement open so that we could discover which facts about this subject really matter to the practice of BPM. We also decided to first focus on understanding the emergent *consistency management process* used at BNB, both in terms of the prescribed procedures and how the participants actually perform the tasks, in the context of the overall development process.

To answer this question, we adopted a structured combination of three research methods: 1) artifact study, 2) semi-structured interviews and 3) electronic survey. The combination allowed us to gradually refine our understanding of how consistency is managed and to triangulate multiple sources to improve confidence in our findings. We now briefly summarize each of the methods.

First, we analyzed business-level and IT-level models to understand the correspondences between them. We were interested in discovering the degree to which these models differ, the refinement patterns applied, and the type of information represented in each model.

Second, we interviewed relevant stakeholders at the studied organization to understand details about the development process, collaboration patterns among the professionals involved, reasons for applying the refinements we found, when and how the consistency among the models is maintained, and the challenges faced during consistency maintenance.

Third, based on the artifact analysis and the interviews, we created an electronic survey with questions to disambiguate unclear points and to solidify our initial findings. We collected responses to this survey from a larger set of stakeholder than those interviewed.

The following sections give more details about the studied organization and the applied methods.

4.2 The Organization

The Bank of Northeast of Brazil (BNB) is a major financial institution in Brazil. It is controlled by the federal government and oriented towards regional development. The IT area of BNB contains over 300 professionals, responsible for maintaining more than 200 information systems in operation. Joining these numbers are five external software development companies, adding up to a virtual workforce of 1500 professionals responsible for the development and maintenance of these systems. The systems are developed using a broad range of technologies, including conventional mainframe transactions and Web-based services. Since 2007, BNB has used *Business Process Management* based on the WebSphere family of products from IBM, including Business Modeler, Integration Developer, Business Monitor, and Process Server. The development process is based on the Rational Unified Process (RUP), extended to include business process modeling. The first version of the development process was customized by BNB with consulting provided by IBM.

4.3 Artifact Analysis

We analyzed five BPM projects, containing more than 70 models in total (see Table 4.1). The development process at BNB entails iterative and multi-staged model refinement, resulting in three types of models: business specifications, technical specifications, and executable implementations (cf. Fig. 2.1). Table 4.1 lists the number of models of each type. It is important to mention that the project *P1* was the first one developed at BNB (pilot project), and its initial development was conducted with IBM consultancy. BNB took advantage of the pilot project to create 23 generic and reusable IT level service processes, e.g., for logging and auditing. As they belong to *P1*, they count as implementation models in this project. That explains the large number, 29, of implementation models as part of this project. Table 4.2 gives the model sizes in number of elements of different types.

We analyzed the models by manually inspecting and identifying corresponding elements and model fragments—typically single-entry and single-exit regions [124]—based on names and

Table 4.1: BPM Projects

Project	Domain	Number of Models		
		Business	Technical	Implementation
P1	Customer Registration	2	2	29
P2	Credit Backoffice	6	6	6
P3	Credit Risk Assessment	2	2	4
P4	ATM	1	1	3
P5	Procurement	3	3	4

structural similarity. The analysis relied on the domain knowledge of the author; we clarified any unclear cases with the creators of the models. As a last step, we classified the correspondences into recurring refinement patterns presented in Sect. 4.6.2.

Table 4.2: Model Sizes

		Number of Model Elements				
		Pools	Tasks	Gateways	Events	Flows
P1	Business Spec.	11	59	38	25	149
	Technical Spec.	11	78	46	36	164
	Implementation	11	123	56	43	186
P2	Business Spec.	6	47	46	18	128
	Technical Spec.	6	95	48	23	142
	Implementation	6	107	52	31	154
P3	Business Spec.	4	17	8	6	19
	Technical Spec.	4	19	10	8	21
	Implementation	4	22	6	9	23
P4	Business Spec.	1	10	5	3	21
	Technical Spec.	1	11	6	8	27
	Implementation	1	18	9	14	51
P5	Business Spec.	8	13	10	11	31
	Technical Spec.	8	18	12	15	43
	Implementation	8	25	14	17	57

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 an 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, De-

velopers) 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.

We recovered from the ClearQuest database the change log of all projects and also the textual descriptions associated with every change request. Our objective was to find the reasons for changing the artifacts in each project we analyzed. Our first step was matching the textual description of each change request with the actual artifacts changed. The aim of this process was to discover how inconsistencies were introduced by regular maintenance. For example, by finding a particular change in August 2009 that had affected only the business model of the project *P1*, we realized from the description of the request that this change had *re-established* the consistency between the business specification and the production process (implementation). A new project was being started on the business side requiring an updated specification to build on. Then, we recorded any such cases to clarify with the people involved. In total, we manually inspected more than 1000 change requests, as shown in Table 4.3.

Table 4.3: Change Requests

Project	Change Requests Analyzed
P1	388
P2	234
P3	176
P4	78
P5	207
	Total 1083

4.4 Interviews

We used semi-structured in-depth interviews. The durations ranged from one to three hours, and the interviews were informal: although organized around a number of themes, we allowed each respondent to follow her own interest. The themes ranged from respondent's background, current role and experience, to practical working scenarios with BPM and personal feelings on how the tools should be improved.

The interviewees' roles were selected from those having personal responsibility in editing BPM models. An IT Manager was also interviewed because of his experience in several projects. These roles served as a representative sample of a larger population of professionals who later

answered the survey. Statistics showing the roles involved, their experiences with BPM, and the interview durations are shown in Table 4.4. Section 4.6.1 provides more details about the responsibilities of each role and the artifacts they produce.

We created transcripts of each interview and submitted them for approval of the respondents. Subsequently, we classified and categorized recurrent facts mentioned in the interviews, such as what consistency aspects are relevant; when and how inconsistencies are detected and fixed; and which tool support would help to perform these tasks. Sample questions asked are the following:

What is your current role? What types of tasks do you perform? How much experience do you have with BPM?

What are the roles involved in creating and maintaining business- and IT-level models?

What tools and architecture- and company-specific guidelines and methodologies impact the content and form of these models?

What collaborations exist between the different roles?

How do different roles coordinate and communicate when they make changes?

Are there examples where inconsistencies were detected?

Are there examples where inconsistencies had undesirable consequences?

Table 4.4: Interviews

Interview	Role	Num. Projects	Duration (h)
1	IT Systems Analyst	2	1:45
2	IT Systems Analyst	2	1:32
3	IT Systems Analyst	3	1:40
4	IT Manager	6	1:10
5	IT Architect	4	3:01
6	IT Developer	2	2:34
7	Business Analyst	4	1:25
8	IT Architect	12	2:10
9	IT Architect	8	1:52
			Total 17:09

4.5 Survey

We created a questionnaire to identify and then resolve conflicting and overlapping facts from the interviews. For example, during the interviews some respondents mentioned that *task ordering* affects consistency, whereas others mentioned that it may not be important. Then we included the following question in the survey: *Corresponding tasks must obey exactly the same relative order*, and the respondents could chose between four answers: *Necessary all the times*; *Important, but not always*; *May be important sometimes*; and *Irrelevant*. We also added open fields, so that the respondents could provide comments and examples supporting their answers. The questionnaire was divided into six groups of questions: *Alignment of Business and IT Models, Tool Customization, Refinement, Change Management, Consistency Checking, and Fixing Actions*. In total, 23 professionals answered it as a web survey. Figure 4.1 shows the distribution of answers per professional role.

It is important to mention that BPM is a relatively recently adopted technology in BNB. At the time of this study, there were few BPM projects in production—they include the ones we used in this study—plus 5 new projects in early phases of development (i.e., the business models were under discussion). Around 30 professionals — including business and IT oriented ones — had been conducting these projects. Our survey collected 23 answers from this population, i.e., 76% of participation. The other professionals in the bank work with several other technologies and programming languages, ranging from traditional mainframe to web-based platforms.

The complete report of the survey and the comments made by the respondents are available online at our web site[1].

From the survey and the data we collected in the previous two phases, we found that our main research question can be divided in the following sub-questions:

1. *What development process is used for creating business- and IT-level process models?*
2. *How business- and IT-level process models are related and how do they differ?*
3. *How do business- and IT-level process models evolve over time?*
4. *How do differences between business- and IT-level process models affect consistency?*
5. *Can inconsistencies cause problems in practice?*
6. *How do BPM stakeholders define consistency between business- and IT-level process models?*

[1] http://gsd.uwaterloo.ca/empiricalstudybpm

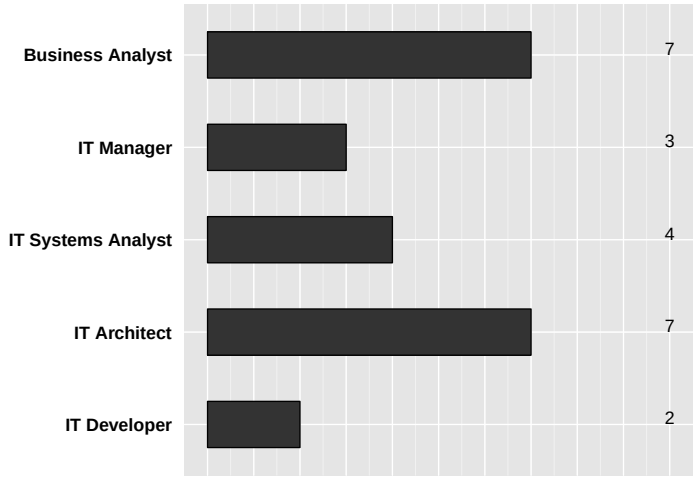


Figure 4.1: Survey Answers per Professional Role

7. *Are the BPM stakeholders satisfied with the development process they currently employ?*
8. *How are inconsistencies dealt with?*

Guided by this sub-questions, we distill the most relevant findings for this research proposal in the next section. The full list of findings is given in [16].

4.6 Main Findings

4.6.1 Processes are developed and maintained in several levels of abstraction

Summary The state of the art recognizes the need for specialized models (or specialized views) in business process modeling (e.g., [26, 79]), such that specific needs of the stakeholders are respected in terms of concepts, modeling notation, and level of detail. Our work confirms such a need in the analyzed case study, by providing evidence on how process models are created, ranging from business- to IT-oriented ones.

The development process adopted by BNB starts by *Business Analysts* producing a *Business Specification* (Fig. 2.1.a) which focuses on the concepts and rules relevant to the business level.

The business specification is refined by *IT Systems Analysts* to create a *Technical Specification* (Fig. 2.1.b). The technical specification has two objectives: a) to ensure that the process is sound and free of design flaws, such as incomplete data objects, deadlocks, and lack of synchronization; and b) to adapt the specification to the existing service infrastructure, making it clear and understandable to developers and outsourcers. The business and technical specifications are written in BPMN. The technical specification is subsequently refined by *IT Architects* and *IT Developers* to implement the executable process (Fig. 2.1.c). Executable processes are written in BPEL. Naturally, several other artifacts are part of the development process, for instance, glossaries, requirement documents, use cases, architecture documents, business rules descriptions, and test cases. IT Managers are also involved in negotiating deadlines, assigning IT professionals, and contracting external services and manhours. Below are descriptions of the main roles involved in developing a BPM project in BNB:

- *Business Analyst:* Define and simulate the business process in terms of organizational structure (lanes, pools), business items (information to flow), resources (e.g., people who interact with the process), tasks (human and automated), business rules and *Key Performance Indicators* (time, costs, etc.). The business process is created in BPMN.
- *IT Manager:* Produce contracts for meeting the business requests. Assign IT personnel to projects and contract outsourcers.
- *IT Systems Analyst:* Provide technical support for Business Analysts; correct and adjust the BPMN model; clarify business items and rules; detail tasks and flows; specify *Use Cases* for each task, gateway, conditional flow, and event.
- *IT Architect:* Create a BPEL model out of the BPMN. Refine the BPEL. Describe service interfaces, integration methods (queue manager, message broker, service bus), design human tasks, produce an *Architecture Document*, *Technical Use Cases*, *Design Models*, and *Deployment Plan*.
- *IT Developer:* Produce code (BPEL, Java, other languages). Create testable builds.

The consequence of this development process is that three different process models for each project are created and maintained. This is considered suitable by BNB to effectively separate concerns and to convey the right information to diverse stakeholders. The common use cases for consistency management throughout the development process are the following:

- *Change propagation*: By applying a development process based on *RUP*, requirements are created or updated by carrying out the business modeling discipline. Business-level

process specifications are updated and the changes should be propagated across related IT-level models. Similarly, due to incident resolution or time constraints it is possible that a process running into production is modified before updating its specification. Later the specification needs to be updated.

- *Validation*: Audits often require checking production processes against high-level specifications and control points of legal reference models, such as Basel II and Sarbanes-Oxley.

It is important to note that the actual workflow is defined by the business-level model *together* with business rules. In particular, detailed cases, such as when withdrawals are authorized, are specified in the business rules document, and might not be visible in the workflow of the process model. These points are more effectively captured as rules, and adding them to the diagram would result in visual clutter. The business-level model is intended to give an overall, high-level flow, and the stakeholders know that they also need to review the business rules document in addition to the process models, in order to cover all the business-relevant details.

The stakeholders complain about the poor tool support for managing traceability and correspondence links among this multiplicity of models. This is particularly critical when specifications are updated and given to outsourcers. From time to time, the correspondences need to be reestablished and described using textual artifacts and model annotations, which is time-consuming when maintained manually. Another important aspect of correspondence links among process models is that they are domain- and project-specific. For example, we found correspondences that can be understood only by having knowledge on the existing systems used in BNB. Then, automatic techniques for deriving correspondences should have means to include specific human domain knowledge as part of the matching methods.

We thus confirm the challenges of establishing correspondences among process models presented in [128]. Since establishing correspondences may require uncovering tacit knowledge—present only in the original model creators' memories or lost entirely—a fully automatic approach with high recall does not seem realistic. More research is necessary to understand the trade-offs between automatic and manual efforts to establish and manage correspondence links, integrated into the development process.

The following quote provides a summary of the development process obtained in an interview from an *IT Systems Analyst*:

The development is done in several iterations for accomplishing the project milestones. This is managed by the project manager following the same methodology used for any other software project. The objective of the inception phase is to clarify what should be done, then all the requirements should be clear at the end of this phase. Most of the collaboration is performed by

business analysts and system analysts, although the architect is also involved in some meetings to anticipate possible integration issues, such as data replications and unavailable services or application components. The artifacts discussed in the inception phase are mainly BPMN models, use cases for tasks, and business rules. In the elaboration phase the objective is to eliminate all the architectural risks and know how the project should be implemented. The main artifacts are the integration model (BPEL), the architecture document and the technical use cases. Most of the collaboration is done by the architect and the developers. Systems analysts still collaborate with architects and developers in the elaboration and construction phases when a business rule or a use case is not well understood. The main problem with our BPM development is maintaining traceability among such models and artifacts. This often requires considerable rework and is specially critical when outsourcers are involved. I say that we could have much better tool support for managing this.

4.6.2 Business and IT process models are related by hierarchical and non-hierarchical refinement patterns

Summary Existing works argue the need for *non-hierarchical* refinements when deriving IT process models from their business specifications (e.g., [128]), while other works propose the transition from business to IT process models in a strictly hierarchical fashion (e.g., [30, 112, 139]). Our study provides evidence of the need for both types of refinements (hierarchical and non-hierarchical).

Using the ATM running-example case study, *P4* in Chapter 2, we now present the refinement patterns we identified. We chose *P4* as the illustration because it is the smallest one and it also contains concrete instances of all the patterns we found in the other projects. Although the executable model is implemented in BPEL, for simplicity, we remodeled here a simpler version in BPMN 2.0 [102] that preserves the salient refinements patterns applied in the real project. Naturally, mismatches that stem from using different languages pose further complications; however, the problem of managing consistency of related process models is generic and independent of any specific language [128].

Table 4.5 shows statistics of the pattern occurrences across the models.

4.6.2.1 Add properties

Description Parameters for grounding the executable model on top of the underlying IT infrastructure are added during the implementation.

Table 4.5: Refinement Occurrences

	Occurrences				
Refinement Pattern	P1	P2	P3	P4	P5
Add properties	27	32	12	8	6
Add script task	21	13	4	1	4
Add protocol task	31	16	2	5	4
Add boundary event	34	9	9	6	6
Add technical exception flow	15	14	3	2	4
Change activity name	14	5	2	3	2
Change activity type	12	3	11	4	2
Refactor gateway	6	8	-	1	3
Split task into block	28	24	4	1	2
Split workflow	25	3	4	5	4
Suppress specification activity	11	7	5	1	6

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. 2.1 (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.

4.6.2.2 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.

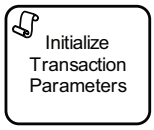
Example A credit process can contain a task to send a hard copy of a contract to the customer, via postal service.

4.6.2.3 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 4.2 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.



(a) Executable

Figure 4.2: Add Script Task

4.6.2.4 Add protocol task

Description An asynchronous service can be implemented by a connection-less request or reply protocol.

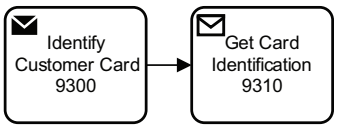
Motivation It is common to implement a business task by using an asynchronous connection-less service. In such cases, the protocol needs to compose and send a message and, after that, wait for a response.

Example Figure 4.3 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).

4.6.2.5 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.

(a) Business and Technical Specifications



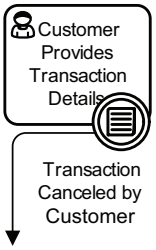
(b) Executable

Figure 4.3: Add Protocol 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 4.4 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. 2.1, 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.

(a) Business Specification



(b) Technical and Executable

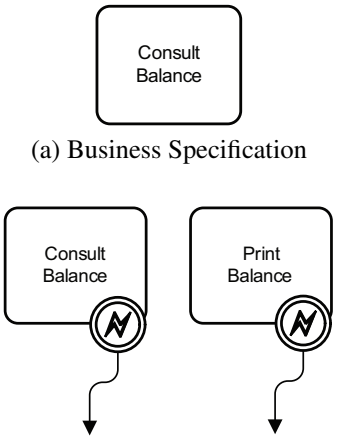
Figure 4.4: Add Boundary Event

4.6.2.6 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-functional requirements elicited during the *elaboration* phase of the development process.

Example Figure 4.5 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.



(a) Business Specification

(b) Technical and Executable

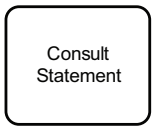
Figure 4.5: Add Technical Exception Flow

4.6.2.7 Change activity name

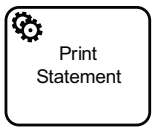
Description The name of a business activity can be changed to facilitate the identification of an IT service that has a similar but different name.

Motivation IT specialists can decide to use technical names in model elements for facilitating maintenance.

Example Figure 4.6 shows an example.



(a) Business and Technical Specifications



(b) Executable

Figure 4.6: Change Activity Name

4.6.2.8 Change activity type

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

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 4.7 shows an example were a human task represented in the business model was implemented by an event.

(a) Business Specification

(b) Technical and Executable

Figure 4.7: Change Activity Type

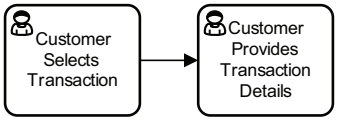
4.6.2.9 Suppress specification activity

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 examples for applying this refinement 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 4.8 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.



(a) Business Specification

(b) Technical and Executable

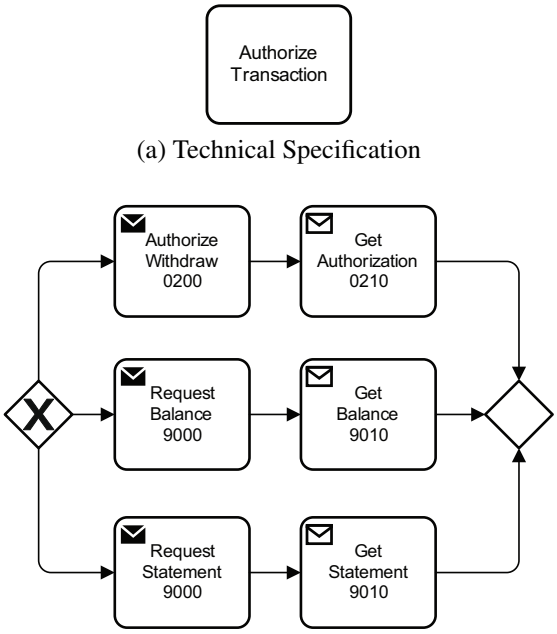
Figure 4.8: Suppress Specification Activity

4.6.2.10 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 4.9 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.



(a) Technical Specification

(b) Executable

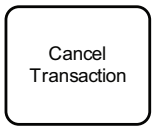
Figure 4.9: Split Task into Block

4.6.2.11 Split workflow

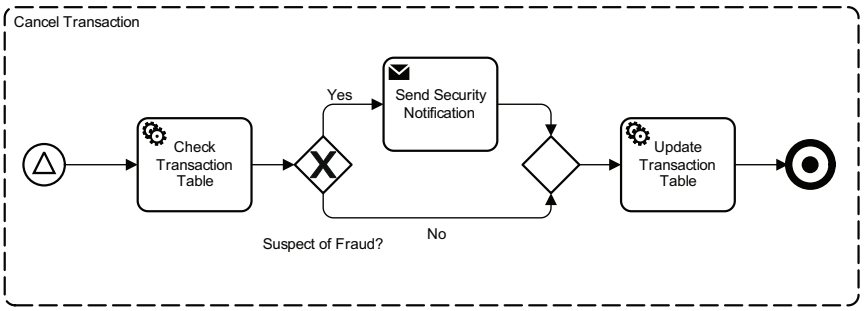
Description The specification workflow can be split into smaller workflows that should be orchestrated by a main flow.

Motivation The typical reason for this pattern is the creation specialized and reusable workflows, such as for logging and auditing purposes.

Example In Fig. 4.10 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.



(a) Business and Technical Specifications



(b) Executable

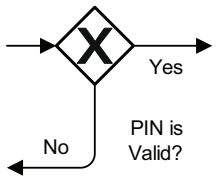
Figure 4.10: Split Workflow

4.6.2.12 Refactor gateway

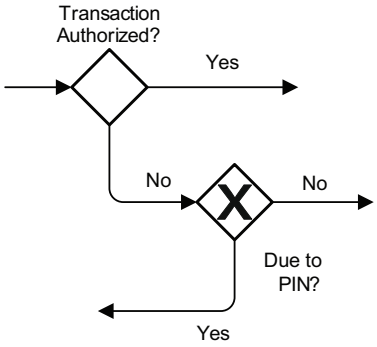
Description A business level gateway may need to be refined to take into account the technical behavior of the services involved.

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 4.11 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 (see Sect. 4.6.1). 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.



(a) Business Specification



(b) Technical and Executable

Figure 4.11: Refactor Gateway

4.6.2.13 Hierarchical and non-hierarchical refinements

A refinement pattern is hierarchical if it is possible to fit the refined model elements into a collapsed subprocess that preserves the original number of incoming and outgoing sequence flows, otherwise it is non-hierarchical. The pattern *Split task into block* (see Fig. 4.9) is an example of hierarchical refinement whereas *Refactor gateway* (see Fig. 4.11) is an example of non-hierarchical one. Other examples of non-hierarchical refinements can be seen in Fig. 2.1, where flows were added to divert the main workflow in case of *timeouts*.

Interestingly, several approaches for aligning business and IT perspectives are based on the assumption that hierarchical refinements are sufficient in practice [30, 112, 139]. Our case study clearly shows that non-hierarchical refinements occur and are relevant in practice. One could argue that non-hierarchical refinements are present in the case study since the tools used there have not enforced hierarchical refinements, as in, for example, the approach by Dijkman et al. [30]. However, the majority of surveyed stakeholders express the need to support both hierarchical and non-hierarchical refinements in practice, as shown in Table 4.6.

We believe though that settling the question of whether hierarchical transformation can be

Table 4.6: Refinement Patterns Needed by Stakeholder

	Strictly hierarchical	Any type of refinement	Role does not need to apply refinements
Business Analyst	13%	87%	0%
IT Systems Analyst	13%	87%	0%
IT Architect	9%	91%	0%
IT Developer	9%	78%	13%

expressive enough to create models for different perspectives requires further research.

4.6.3 Models undergo parallel maintenance

Summary Existing works recognize that process modeling needs support for parallel maintenance of models or views that target different levels of abstraction and stakeholder perspectives (e.g., [19]). Our study provides evidence on such a need, by analyzing the change history of five real-world projects. The histories cover both the development prior production and also changes after the projects went into production. Their analysis shows that, while the majority of changes affect only artifacts other than the related business- and IT-level models (i.e., use case descriptions, databases, services, and application components), about 10% of changes affect both models simultaneously, about 20% affect only IT-level models, and about 5% affect only business-level models. The number of changes varies according to the projects' life-cycles, with the majority of business-model-only changes occurring at the project's start. The analysis also revealed (i) cases where the business models were changed in response to earlier IT model changes, in order to eliminate inconsistencies considered as undesired by the stakeholders; and (ii) cases where changes first specified in the business models where later implemented in the IT models; these cases were viewed as acceptable, controlled inconsistencies, by the stakeholders.

As announced in Sect. 4.3, we report on a substantial dataset featuring in total 5 BPM projects, 74 models (business and IT ones) and more than 1000 changes made on these models throughout their life-cycles. A project at BNB contains the versions and baselines of all the artifacts of a system, including models, textual documents—such as use cases and business rules—and source code. We inspected the change history of each project to identify when inconsistencies were introduced by day-to-day maintenance and when they were found and fixed.

We classified each change request with respect to the model types being affected. For example, *Only Business* means that a request has changed solely the business model but not the IT

model, whereas *None* means that the request has changed neither the business model nor the IT model, but other resources such as databases, web services, and application components. The changes to the artifacts ranged from adding or modifying a single model element (e.g., a task or flow) to applying multiple patterns in multiple places.

Figure 4.12 shows the distribution of the changes per type in each project and Table 4.7 shows the corresponding percentage numbers.

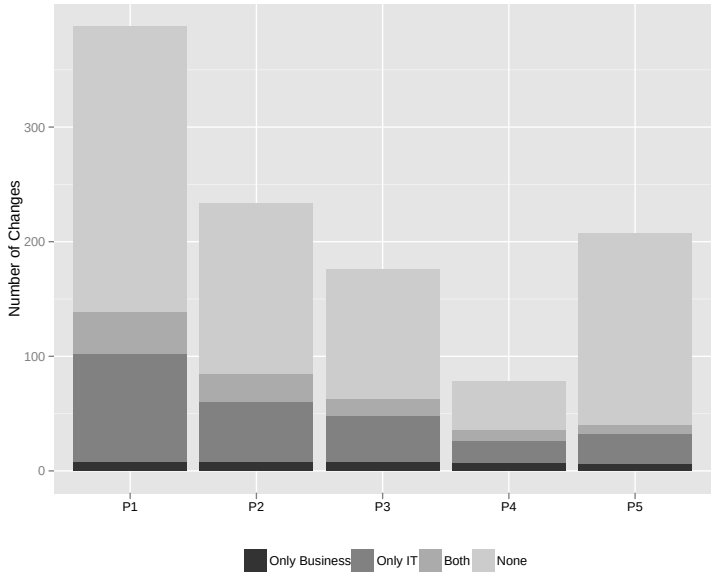


Figure 4.12: Distribution of Changes per Type

Table 4.7: Percentage of Changes per Project

	Change Type			
Project	Only Business	Only IT	Both	None
P1	2%	24%	10%	64%
P2	3%	22%	11%	64%
P3	4%	23%	9%	64%
P4	9%	24%	13%	54%
P5	3%	13%	3%	81%

Figure 4.13 shows the temporal distribution of 388 changes made on project *P1* throughout

its three first years. Figure 4.14 shows the first-year *stacked* distribution of changes for the other projects.

The analysis of the change history revealed that inconsistencies were introduced mainly due to parallel maintenance. The following two cases summarize situations where the inconsistencies were detected and fixed:

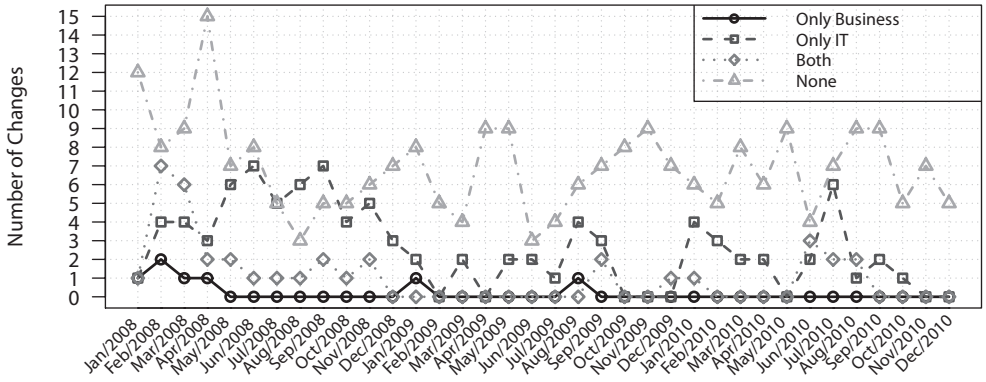


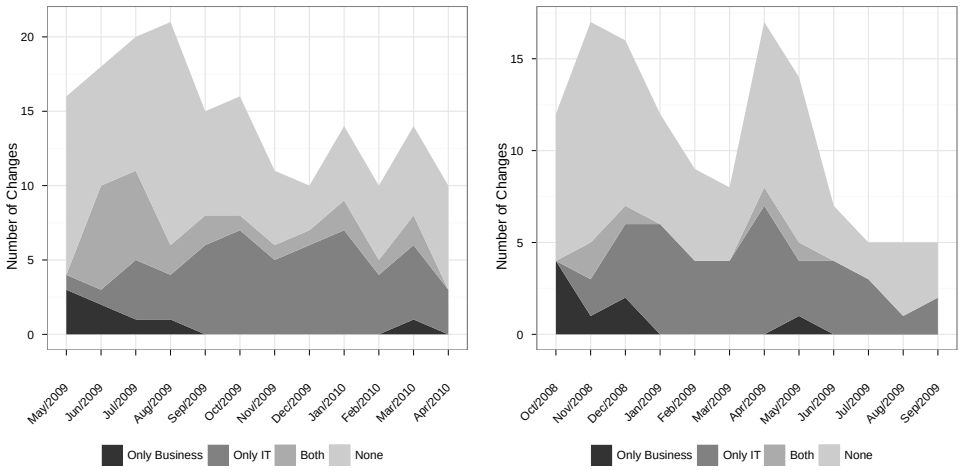
Figure 4.13: P1 Change History

- *Case 1*: Update only the business model because at least one previous maintenance request that should have affected both the business and the IT model has been made only in the IT model. This is considered an *undesirable inconsistency* by the stakeholders, and the business model is being updated to address it. Audits often motivate this type of maintenance. Another reason is when a new project requires an accurate business-level model for AS-IS analysis.

- *Case 2*: Update only the IT model to reflect, e.g., a planned process optimization that has previously been made only in the business model. This is considered a *controlled inconsistency* by the stakeholders.

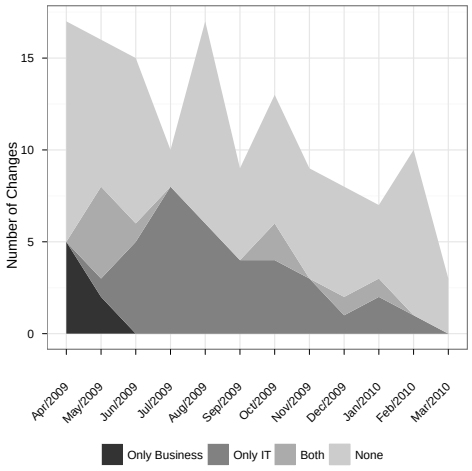
For project *P1* we identified changes made only in the business model because of *Case 1* in January 2009 and August 2009. Also, we identified requests because of *Case 1* in projects *P2* and *P5* in March 2010 and July 2010, respectively. In project *P3*, we identified a process optimization made initially in the business model because of *Case 2* in May 2009.

Some stakeholders complain that the current tool support to plan and manage future changes–i.e., *controlled inconsistencies*–is deficient, although they mitigate the issue by using resources

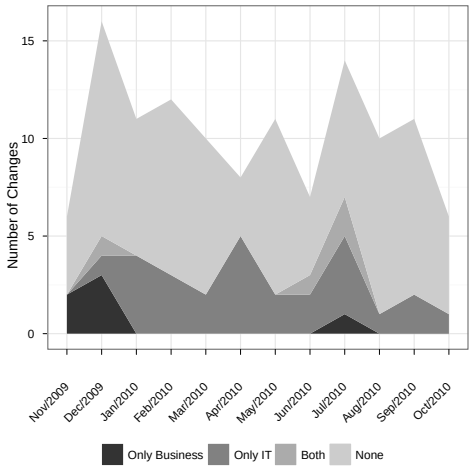


(a) P2

(b) P3



(c) P4



(d) P5

Figure 4.14: First Year Change History

of the artifact repository. The main complaint is the lack of tool features for easily comparing, differencing, and merging process models, as they are widely available for textual source code.

The stakeholders also complain about the tool support for managing traceability and correspondence links among this multiplicity of models. This is particularly critical when specifications are updated and given to outsourcers. From time to time, the correspondences need

to be reestablished and described using textual artifacts and model annotations, which is time-consuming when maintained manually. Another important aspect of correspondence links among process models is that they are domain- and project-specific. For example, we found correspondences that can be understood only by having knowledge of existing systems used in BNB. Then, automatic techniques for deriving correspondences should have means to include specific domain knowledge as part of the matching methods.

Our analysis confirms the challenges of establishing correspondences among process models identified in [128]. Since establishing correspondences may require uncovering tacit knowledge—present only in the original model creators' memories or lost entirely—a fully automatic approach with high recall does not seem realistic. More research is necessary to understand the trade-offs between automatic and manual efforts to establish and manage correspondence links, integrated into the development process.

The following quote was made by an *IT Systems Analyst*:

"The main problem with our BPM development is maintaining traceability among such models and artifacts over time - this often requires considerable rework and is specially critical when outsourcers are involved. I say that we could have much better tool support for managing this."

4.6.4 Coverage and behavioral differences affect consistency most

Summary The state-of-the art discusses types of differences among process models that may affect the consistency among them [128]. Our work provides results of a survey where practitioners were asked to evaluate to what extent such types of differences impact their notion of consistency.

From the artifact analysis, we prepared examples of the three types of model differences defined in [128] (briefly explained in Chapter 3) and asked the respondents to answer two questions:

Please indicate to what extent the following types of differences affect the notion of consistency between Business and IT models and

Please indicate to what extent the following types of differences may be tolerated or ignored when checking consistency between Business and IT models

The answers to these two questions are shown in Table 4.8 and Table 4.9, respectively. We also asked the respondents to rank a set of consistency aspects that were frequently mentioned in the interviews. The results are shown in Table 4.10. 86% of the respondents support that a difference in coverage always affects consistency, 68% support that a difference in behaviour sometimes affects consistency, and 68% support that a difference in density does not affect consistency. 74% support that corresponding tasks between the models must obey the same relative

order. We also collected open answers from the respondents explaining their understanding on these types of differences.

Our analysis leads us to propose the notion of *Business Relevance*, which seems to be crucial whenever stakeholders check consistency. If a mismatch is considered relevant to the business it should be fixed, otherwise it is ignored. Although this definition is subjective, we noticed that typically differences that are considered technical details of implementation are ignored. For example, Fig. 4.2 shows a case of *Coverage* mismatch that is not business relevant: the added script task is essentially a detail of implementation and does not have any correspondence in the business-level model. Similarly, Fig. 4.5 and Fig. 4.9 show respectively examples of *Behaviour* and *Density* differences that are also not considered business relevant: both are details of implementation.

Our analysis confirms the postulates presented by Weidlich et al. [128] on how differences affect consistency. We extend the postulates by adding the concept of business relevance, which can be summarized as follows:

- Difference in *Coverage* is what most affects consistency, as long as it is business relevant.
- Difference in *Behavior* is relevant when it affects task ordering.
- Difference in *Density* does not seem to affect consistency. It is generally considered an implementation detail and thus not relevant to business.

Table 4.8: How Differences Affect Consistency

Type of Mismatch	Always Affects	Sometimes Affects	Does not Affect
Coverage (There is a difference between WHAT is modeled)	86%	14%	0%
Behavior (There is a difference in HOW a certain scenario is implemented)	14%	68%	18%
Density (There is a difference in the LEVEL OF DETAIL a certain scenario is implemented)	0%	32%	68%

We believe that more investigation is necessary to understand how one can allow the stakeholders to define and manage these types of differences individually in each project. The same type of mismatch can be considered to affect or not to affect consistency, depending on its *business relevance*. This is why the answers to these previous two questions were subjective, consistently with results from the interviews.

Table 4.9: How Differences are Tolerated

Type of Mismatch	Never Tolerated	Sometimes Tolerated	Always Tolerated
Coverage (There is a difference between WHAT is modeled)	50%	50%	0%
Behaviour (There is a difference in HOW a certain scenario is implemented)	14%	77%	9%
Density (There is a difference in the LEVEL OF DETAIL a certain scenario is implemented)	0%	59%	41%

Table 4.10: Consistency Aspects Mentioned in the Interviews

Consistency Aspect	Necessary at all times	Important, not always	Sometimes irrelevant	Irrelevant
Corresponding model elements have the same names	30%	70%	0%	0%
Corresponding tasks must obey the same relative order	74%	26%	0%	0%
Corresponding tasks have the same types (service, human etc.)	22%	61%	13%	4%
Corresponding gateways have the same number of incoming and outgoing flows	9%	52%	30%	9%
Corresponding business objects must have exactly the same fields	13%	52%	30%	4%
Every task in the business model has at least one corresponding task in the IT model	9%	70%	22%	0%
Every gateway in the business model has at least one corresponding gateway in the IT model	13%	70%	17%	0%
Every event in the business model has at least one corresponding event in the IT model	9%	70%	22%	0%

4.6.5 Inconsistencies can cause severe problems

Summary It is natural to expect that undesired inconsistencies in software artifacts may cause problems, for example, by delaying a new version of a system (e.g., Spanoudakis et al. [113]

and Nuseibeh et al. [100]). We provide examples of unnoticed inconsistencies that: (i) delayed business and impacted customers (production process instances needed to be canceled and recreated) and (ii) affected compliance regulations, subjecting the company to fines. Such problems can cause serious financial and corporate image losses. Our motivation to present such cases is reinforcing the need for better support in process modeling for dealing with multiple abstraction levels.

We identified two particular cases in which inconsistencies caused troubles. The first case was caused by an incomplete technical-level process specification—a problem of business-relevant *coverage* mismatch, see Sect. 4.6.4. An inconsistent technical-level process specification, the corresponding IT model and several other artifacts, such as use cases, architecture document etc., were sent to an outsourcer as part of a maintenance project. When updating the IT model, a developer inadvertently removed the functionality shown in Fig. 4.15. The developer was new to the team and thought that this functionality should be deleted from the IT model: the developer did not see any *reasonable* correspondence in the specification, and also there was no reference to it in the architecture document. As a result, the problem passed unnoticed during the tests and the phase of user approval, and was discovered very late when the project went into production.

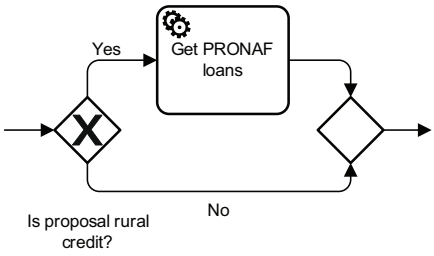


Figure 4.15: Functionality Inadvertently Removed*

PRONAF is a business acronym in BNB that means a credit line for *family agriculture*.

This was considered a severe problem, because some running instances of the process had to be canceled and recreated, delaying business. In outsourcing, the communication throughout a project usually observes a rigid schedule and the external developers do not have direct access to talk to business or systems analysts: double-checking the understanding of a specification is not as simple as in an internal development. Although the test cases were improved after the incident, similar incidents can still happen if the business and IT-level models are incomplete, as there are no specified tests to capture every possible issue.

The second case was similar, but this time it was discovered by a regular audit procedure, where projects and their artifacts are inspected for consistency. Unclear points are marked to

be explained. It turned out that the specification was outdated, and a notification was issued to correct the problem. This is also a severe problem, because business specifications are used for satisfying regulation purposes. Whenever a compliance issue is reported by an audit procedure the company is subject to fines and the managers are subject to legal responsibility.

One of the *Business Analysts* made the following comment about such incidents:

"It is somehow frustrating that BPM has not solved our problem of reliably communicating with outsourcers by using process models as specifications. In practice the technology is preferable for internal development, where the communication between business and IT is straightforward. There is always a risk of something is missing in one or another model, or some correspondence not being completely understood. We have to maintain heavy textual documents describing the correspondences between specifications and implementations, which is cumbersome and time-consuming. Today the quality team is spending a huge effort to guarantee that such problems of misunderstanding the models do not require to cancel production process instances. This may affect customers and negatively impact the image of the company."

4.6.6 The majority of the surveyed practitioners would prefer a single model for Business and IT

Summary It is generally accepted that a single model for business and IT is undesirable [24]. Mixing business and IT concepts may produce cluttered models that are hard to understand and maintain, specially on the business side. Curiously, most of the specialists from BNB would prefer a single model for both business and IT. We actually do not interpret this finding as strictly contradicting the state-of-the-art. We rather interpret this result as a twofold message: (i) people want a single model, i.e., a single source of truth, for the overlapping part of related business and IT models, and (ii) they are dissatisfied with the current tool support for managing consistency of multiple, independently edited, process models.

We asked the practitioners to answer which development method they consider more effective for keeping Business and IT perspectives consistent. We ultimately wanted to find out how satisfied they are with the current development process and tool support for consistency management. The results are shown in Fig. 4.16.

We examined the answers with respect to the roles of the respondents—the first question of the survey asked respondents to provide their roles. Curiously, we noticed that all the respondents who answered *Create wrappers on top of existing IT systems in order to enact a pure Business Model* were Business Analysts. They manifest that pure business models are the ones

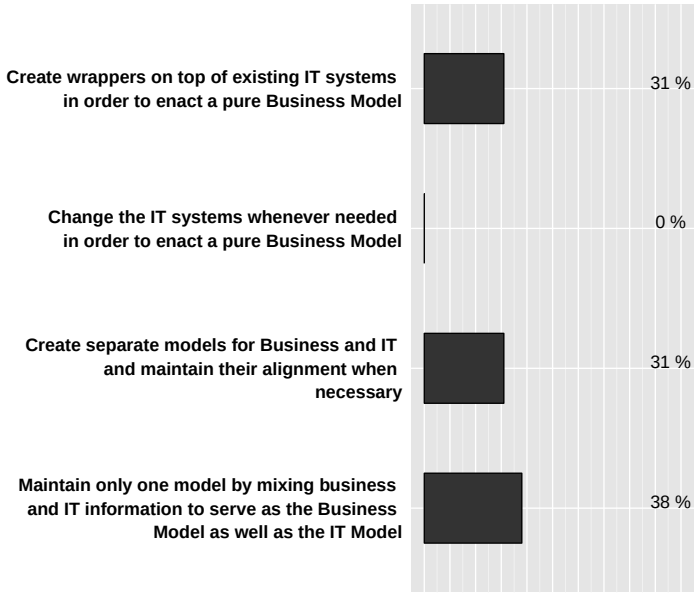


Figure 4.16: Preferred Approach to Enforce Consistency

really needed by the company and that the IT department should do anything necessary to enact them directly. Dealing with some pollution of information in a single model is even considered preferable by business people over the burden and the risk of losing consistency between different models. None of the IT specialists chose that answer. Regarding the remaining answers, most of the IT stakeholders (38% of the answers) are skeptical whether it would be actually possible to enforce consistency by maintaining different models for Business and IT.

Surprisingly, having a single model for Business and IT is generally considered undesirable by existing works [24]. When maintaining a single model, the company might run into the problem that business analysts and managers could no longer understand the resulting model. They might not recognize how their business is reflected in the resulting model. Another problem of this approach occurs when the business model is used to satisfy compliance check points. Mixing Business and IT concepts can force changing the terminology or the level of granularity of business concepts, making the model less clear and less useful for fulfilling the regulations.

Collating the answers and comments from the practitioners, given in both the interviews and the survey, we interpret this result as a twofold message:

- People want a single model for their *common* (overlapping) aspects, i.e., a single source

of truth—that is the same as consistency. Nevertheless, there may be disjoint parts for business- and IT-specific concepts, which could live in separate models. For example, all the IT aspects that are not *business relevant* (see Sect. 4.6.4) could be maintained in a different place, other than the 'single' model. The same applies for business concepts that are not system-supported, such as manual tasks. This does not completely contradicts the literature, which mainly refers to the specific parts.

- There is a dissatisfaction of the users with the current tool support for managing consistency of multiple process models. More research is necessary to understand whether a single model for the common part would be feasible as solution in practice. An open problem is that the existing process metamodels do not reflect the situation of two overlapping models for business and IT. That is, alternatively to synchronizing two separate models, a new metamodel could be developed to reflect the multiple views use case better. A possible direction is to allow custom views on a shared model [81].

In addition, the use of a single (i.e., unique) model for both common and specific parts may not be technically possible, as some respondents pointed out:

IT Systems Analyst 1: I sympathise with the idea of having a single process model, as it would eliminate this burden of synchronizing business and IT processes. However, I still have some unclear points in my mind on how this would work: 1) If the language is the same, most probably the mechanism of having modeling perspectives is critical, since the business roles should stick with their basic building blocks, while on the IT side we have full modeling capability. How would this work in practice? By hiding or showing things, like model elements? Is it really possible to do this? What if by adding transaction scopes and controls we need to split the original process and thus drastically change the business view? It is not clear for me whether you can just hide or show things. 2) It seems that you expect improving the collaboration between business and IT, but what exactly do you expect that tools would do for improving collaboration? For me the collaboration today is already good with the current tools, although there is a lack of automated support for change propagation and synchronization. However, I do think that the tools also lack a better integration with the development process, such as iteration planning and fine-grained change traceability.

IT Systems Analyst 2: I believe in a single model only if we can still have specialized views for business and IT—I do not know how this would differ from having different models, since in practice we may implement the business model only partially, or split it into several pieces. On the other hand, if the tool enforces a unique model for both business and IT and does not give any freedom of changing it in parallel for particular users, I am afraid that people would create two different models anyway.

IT Architect:For me a single model is viable and ideal when you have a highly mature IT service infrastructure, with several business services already available and aligned with the business objectives. In case you need to implement many things from scratch, it is almost inevitable having the business model only as a reference and the executable model more similar to the reality of existing systems.

Business Analyst: The ideal solution is having only the business model, because it is in the end the consumable asset of the company. With a single model, the alignment will be enforced by the technology, which is good. In the case of technical issues preventing the enactment of a pure business model, it should be possible to solve that by other means instead of changing the model itself.

4.6.7 Inconsistencies and fixes should be presented as they occur

Summary Existing approaches propose *quick fixes*, generated during the editing of the models [58], other works propose off-line reports where the practitioners can assess the degree of consistency of related process models [132]. The surveyed practitioners prefer instantaneous fixing actions, integrated to the modeling tools.

We asked the respondents about their preferences on how to check whether the models are consistent and how potential inconsistencies should be automatically presented by the tools. Figure 4.17 shows that the respondents seem to prefer looking at concrete model differences, which may be grouped into high-level model changes, rather than metric measures associated with a degree of consistency as proposed in [132].

With respect to fixing actions, most of the respondents would prefer having quick fixes, automatically generated by the tools during the modeling task, as shown in Table 4.11.

Table 4.11: How Fixing Actions Should be Presented

	Instantly, during the modeling task	As an offline report, when required
For Business Stakeholders	86%	14%
For IT Stakeholders	95%	5%

An *IT Architect* has made this comment in the survey:

I think that one of the main reasons for the lack of alignment between business and IT is not related to how we create business and IT models or related to what contents they should have

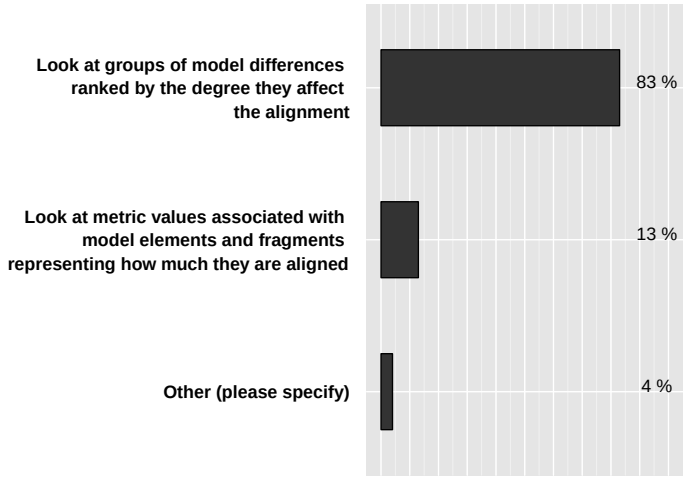


Figure 4.17: Preferred Method for Aligning Models

or not. I believe that the development process plays an important role in this: today we try to minimize the lack of alignment by enforcing a close relationship between the technical modeller and the business analyst. This is good for new projects, but it often fails in day-to-day for several reasons: in practice many changes are minor, which leads to accumulating some inconsistencies considered not critical until a big change is necessary. Usually most of the change requests made by a business role are described only textually and the business model is not even touched—the problem here is that the business analyst believes that only the production process should be updated and its documentation does not need updating. It is hard to enforce a policy requiring the business analyst to always update the business model, because the one who knows when the documentation should be updated is the business analyst anyway. There are long periods of maintenance that affect primarily the executable model, so during the life cycles of small projects you accumulate several small waterfalls of textual requirements in the sense that the business model, as it should also be part of the requirements, is forgotten. I think that the best way to address this is by showing potential inconsistencies immediately, whenever the models are changed. This would make people aware to keep the models always consistent to a sufficient level. We can also manage the inconsistencies by planning when they should be resolved in future projects.

Chapter 5

General Concept of the Shared Model Approach

5.1 Overview

Figure 5.1 shows a conceptual view of our proposed framework for consistency management in business process modeling. The framework assumes the existence of two independent views (i.e., models) for Business and IT—respectively, BM and IT—that can be obtained anytime from a single Shared Model (SM)—highlighted by the dashed box. In a nutshell, the Shared Model works as described below:

1. Business and IT process domain models (BM, IT) are matched using appropriate heuristics to deal with common correspondence patterns (Chapter 6).

2. Domain models to be matched are augmented with structural information (BM^+, IT^+)—e.g., implicit regions, transitive flow edges—by parsing them into Process Structure Trees [124]. The hooked arrows, $\hookrightarrow$ and $\hookleftarrow$, in Fig 1.2, represent inclusion mappings.

3. Each augmented domain model is aligned via correspondences to the shared model. The double arrow, $\leftrightarrow$, represents the correspondence mapping between the PSTs.

4. The shared model contains all public and private information. Private information is private to a given role; for example, the IT model contains private tasks that are not mapped to the Business model, and vice versa. Public information is reflected to other roles; that is, public tasks in the IT model are mapped to public tasks in the Business model and

vice versa. Visibility is controlled by annotations, *public* or *private*, associated with every model element, i.e., activities such as tasks, events, gateways etc., and sequence edges;

5. The shared model transforms changes via *diff* into edit operations, μ_{BM}, μ_{IT}, μ_{BM^+}, μ_{IT^+}, and propagates them as graph transformation rules: $bPpg$, $fPpg$. This pair of operations constitutes a single symmetric delta lens [36]. The down arrows, $\Downarrow$, represent the deltas.

6. Public changes made to any domain model are propagated to the shared model as well as to the other domain model;

7. Private changes made to any domain model are propagated only to the shared model;

8. Public or Private changes are defined by the user. Each edit operation can be public or private (see Sect. 5.2).

9. Finally, the new—i.e., *consistent*—versions of the models are obtained from the shared model, get_{BM}, get_{IT}, and both views get updated: BM', IT'.

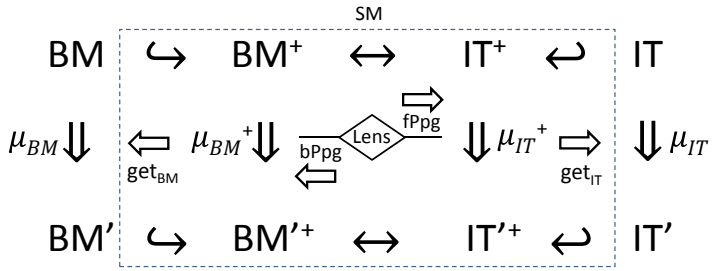


Figure 5.1: Framework Overview

5.2 Edit Operations

The shared model allows the users to perform changes to any view using predefined edit operations. Each edit operation is defined by the user as being *Public*—both views and the shared model are to be updated— or *Private*—a single view and the shared model are to be updated.

As we presented in our earlier study [17], the following basic operations are sufficient to perform any types of changes, according to the common correspondence patterns used to derive IT processes out of their business specifications. In Chapter 9 we discuss the relations between the correspondence patterns elicited in the study and the conceptual edit operations.

Double arrow dashed lines show the correspondence (traceability) links maintained by the shared model. Single arrow dashed lines show the result of a *get* over the shared model onto a particular view. The attributes *Vis*, *Hid*, *Pub* and *Prv* mean, respectively, that a model element is *Visible*, *Hidden*, *Public* or *Private*. Only public elements are mapped across the Business and IT views. Hidden elements are parts of the enhanced models that are not in the original view, such as regions. They are there just to allow expressing correspondence and update mappings. Only the visible model elements are shown in a particular view.

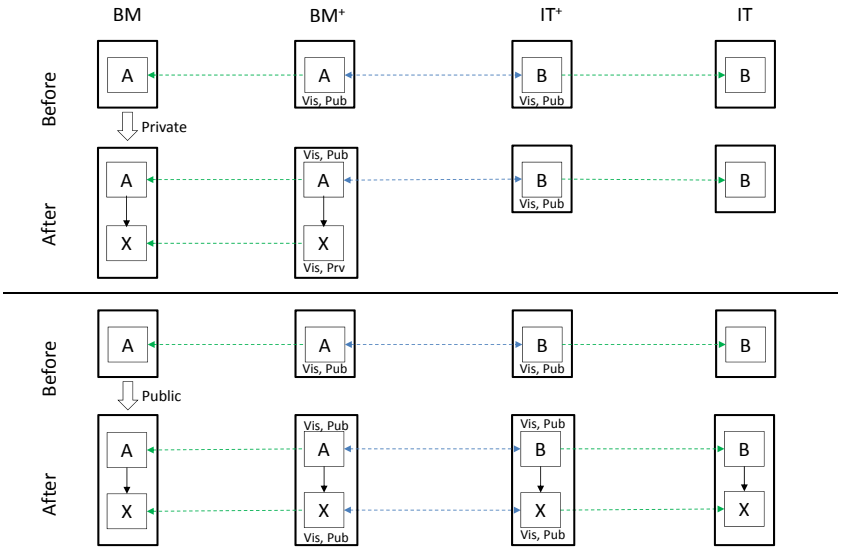


Figure 5.2: Add

5.2.1 Add

This operation is used to add model elements to a view. Figure 5.2 shows public and private *Add* operations. Although it is natural that an *add* operation can be performed in an initially empty model, we depicted some context to show that it needs to be taken into account. In the example, a new task (X) is being added as successor of an existing one (A). Attributes and links are updated accordingly.

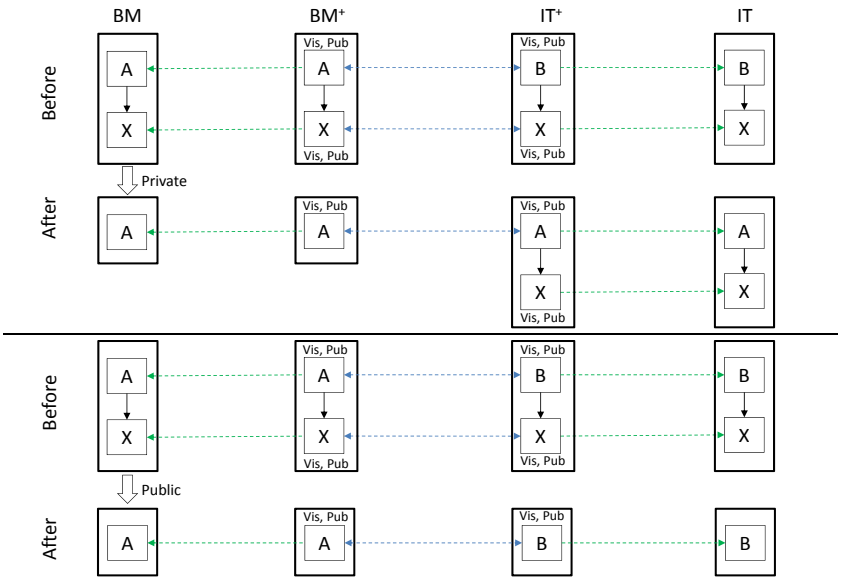


Figure 5.3: Delete

5.2.2 Delete

This operation is used to delete model elements from a view. Figure 5.3 shows public and private *Delete* operations. In the example, an existing task (*X*) is being removed as successor of another one (*A*). Attributes and correspondence links are updated accordingly.

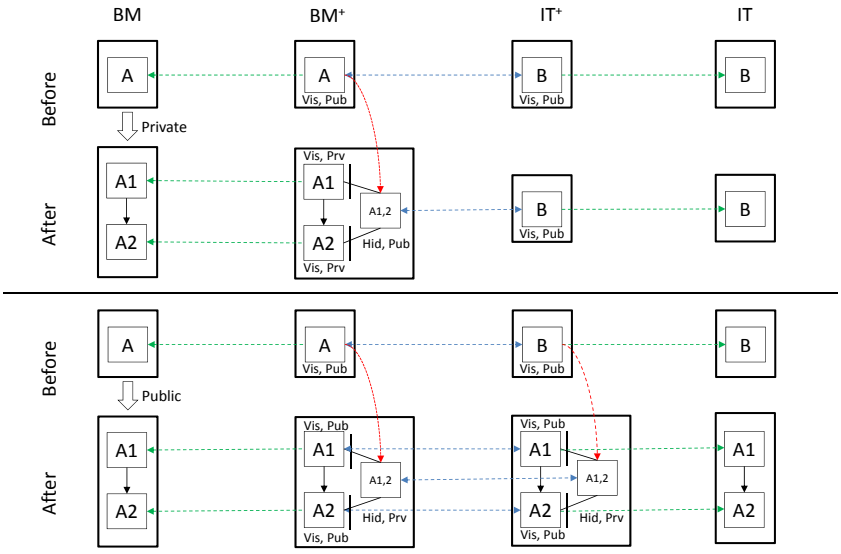


Figure 5.4: Split

5.2.3 Split

This operation is used to split a particular task into a fragment of other tasks. Figure 5.4 shows public and private *Split* operations. In the example, an existing task (*A*) is being split into a sequence of two other tasks (*A1* and *A2*). *A1,2* is a region automatically created by the shared model as a container of the fragment, such that it can keep track of the correct correspondences. Attributes and links are updated accordingly.

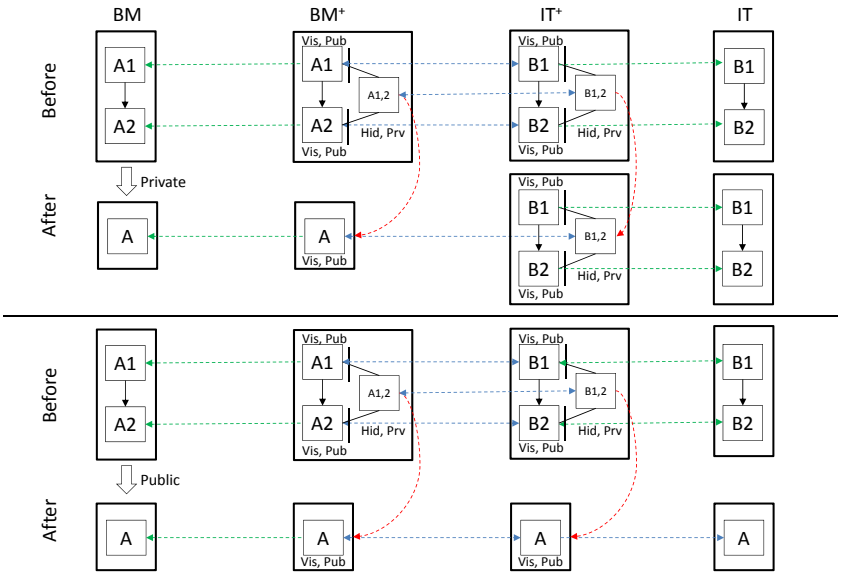


Figure 5.5: Collapse

5.2.4 Collapse

This operation is the inverse of split and it is used to subsume a a fragment of tasks into a single task. Figure 5.5 shows public and private *Split* operations. In the example, a sequence of tasks (*A1* and *A2*) is being collapsed into a single one (*A*). *A1,2* and *B1,2* are regions automatically created by the shared model as containers of the fragments, such that it can keep track of the correct correspondences. Attributes and links are updated accordingly.

Figure 5.6: Attribute Assign

5.2.5 Attribute Assign

This operation is used to change model elements properties. Figure 5.6 shows public and private *Attribute Assign* operations. In the example, the name of a task (*A*) is being changed to (*X*). Attributes and links are updated accordingly.

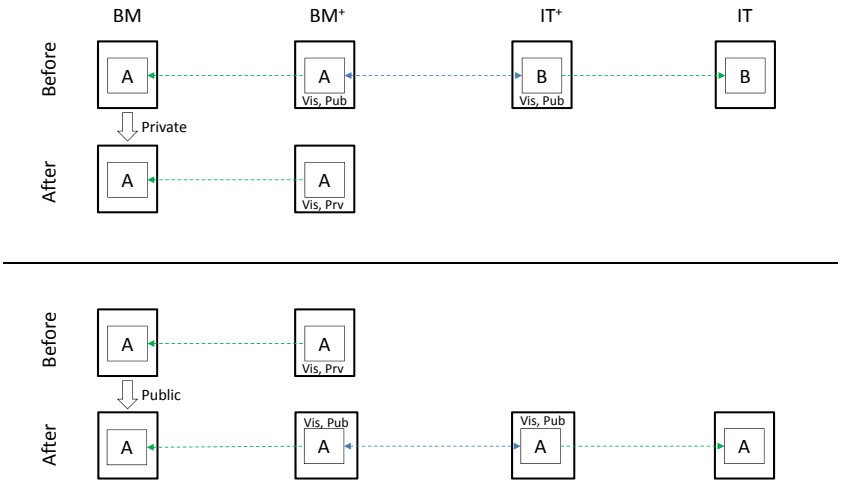


Figure 5.7: Change Visibility

5.2.6 Change Visibility

This operation is used to change the visibility of a model element. Figure 5.7 shows public and private *Change Visibility* operations. In the example, the visibility of a task (*A*) is being changed (from *Public* to *Private* and vice versa). Attributes and links are updated accordingly.

5.3 Framework Implementation

In the following chapters, implementation designs and individual evaluations for each one of the major framework components are discussed. Chapter 6 discusses the matching component. Chapter 7 discusses the diff generator component. Chapter 8 discusses the process model synchronizer and the combination of all components together, i.e., the Shared Model Approach. Finally, chapter 9 evaluates the Shared Model in action, against real process modeling scenarios.

Chapter 6

Matching Process Models Across Abstraction Levels

6.1 Overview

This chapter presents a technique for matching business process models. Figure 6.1 highlights the matching component inside the framework.

6.2 BPMN, SESE, and PST

This work assumes that the models to be matched are expressed in BPMN 2.0 [102]. BPMN 2.0 allows businesses to represent their internal business procedures in a graphical notation and communicate them in a standard way for both documentation and execution. Models expressed in other languages, such as BPMN 1.0 and BPEL, can be translated into BPMN 2.0 without adversely impacting the information used by our algorithm (cf. Sect. 6.9.1). BPMN inherits and combines elements from a number of previously proposed notations, including the Activity Diagrams component of the Unified Modeling Notation (UML).

Figure 6.2 shows two simplified BPMN process models. We added shorter names in parentheses (e.g., *(AC)*) only to later facilitate concisely representing correspondences between the models—the method uses the original names. The notation displays activities by rounded rectangles, events by circles, gateways by diamonds, and sequence flows by arrows. Each model has a start, usually modeled by a start event (e.g., *Customer inserts Card into ATM*), a flow of

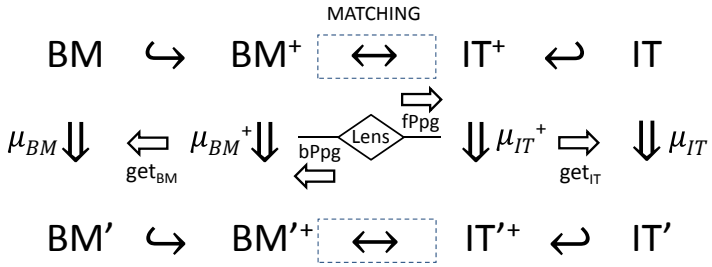
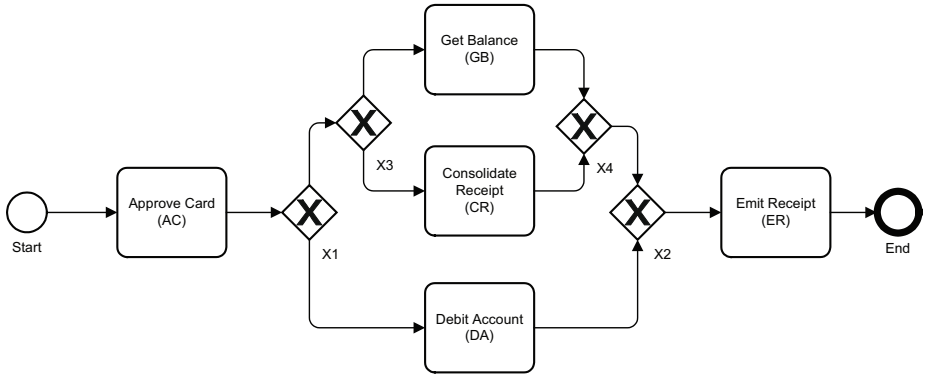


Figure 6.1: Matching Component

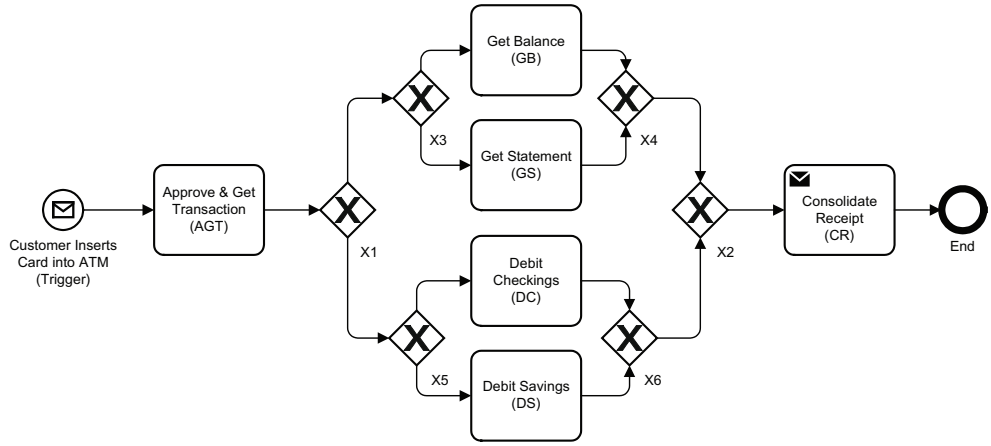
activities governed by decisions (e.g., *X1*), and an end point. A larger, realistic example is given in Fig. 2.1.

Any workflow graph (a BPMN process model in our case) can be uniquely decomposed into single-entry single-exit (SESE) regions [124]. Let $G = (N, E)$ be a workflow graph, where N is the set of nodes and E the set of edges. A SESE region $R = (N', E')$ is a nonempty subgraph of G, i.e., $N' \subseteq N$ and $E' = E \cap (N' \times N')$ such that there exist edges $e, e' \in E$ with $E \cap ((N \backslash N') \times N') = \{e\}$ and $E \cap (N' \times (N \backslash N')) = \{e'\}$; e and e' are called the *entry* and the *exit* edge of R, respectively. According to the formal definition, a SESE region is any region in the workflow graph that has a single entry at the beginning and a single exit at the end. In this way, an activity itself is a SESE region, and so is the whole workflow graph.

The Process Structure Tree (PST) for a BPMN process model is a tree representing the decomposition of the model into SESE regions [124], similar to the much older notion of a program structure tree [76]. Figure 6.3 shows the PSTs corresponding to the BPMN process models. There is a unique PST for each BPMN model. The root represents the whole process model since a process model is a SESE itself. Leaves represent model elements, i.e., activities, gateways and events. Inner nodes represent SESE regions. In particular, the parent of a region R is the smallest region R' that contains R.



(a) Business Specification



(b) Technical Specification

Figure 6.2: BPMN Models

6.3 Differences between Business and IT process models

Our target scenario involves matching business-level models specified by business analysts and the corresponding IT-level models implemented by IT specialists. IT specialists usually refine the original specification to meet technical requirements of the underlying IT infrastructure, such as invoking existing and new services, adding exception treatment, and changing the control flow to satisfy application protocols and optimize the execution. In Chapter 4 we describe a catalog of 11 recurrent patterns used to refine business-level models into IT-level models. These patterns include (i) adding or modifying properties of model elements, such as changing the name or

type of an activity or adding service call details, and (ii) changing the flow structure. The latter category includes behavioral refinement and refactoring and adding additional behavior, such as technical exception flow. An example from category (i) is the renaming and retyping of the empty start event *Start* (Fig. 6.2.a) into the message-driven event *Customer inserts card into ATM* (Fig. 6.2.b). An example from category (ii) is the refinement of the task *Debit Account* (Fig. 6.2.a) into the block consisting of the gateways *X5* and *X6* and two other tasks *Debit Checkings* and *Debit Savings* (Fig. 6.2.b). Examples of other patterns are given in Chapter 4.

6.4 Matching Algorithm

We assume that the models to be matched represent the same process, but at different levels of abstraction, as described in Sect. 6.3. We also assume that, although the models are intended to be consistent, inconsistencies can occur during their evolution. Thus, the models may include inconsistencies, such as order of activities switched during refinement or business-relevant activities added to the IT-level model but not reflected in the business-level model (see [16] for other examples).

The algorithm identifies a correspondence between two models residing at different abstraction levels. The algorithm operates on the PST representations of the models. As stated in Sect. 6.2, leaves in a PST represent *model elements*; inner nodes represent SESE regions, or *regions*, for short. The algorithm computes a *(model) correspondence*, which is a set of *correspondence links* among PST nodes; each link connects a single node in the PST of the first model with a single node of the PST of the other model. Thus, our algorithm is able to identify correspondence links of different cardinality with respect to model elements:

- *1:1* link among two model elements or two regions with only one model element each;
- *1:n* link between a region with one model element in the first PST and a region with more than one model elements in the second PST;
- *m:n* link between regions with more than one model element each.

Our algorithm has two phases: *attribute matching* and *structure matching*. The first phase deals with the search of correspondence links based on the attributes of model elements such as names and types; the second phase tries to find correspondence links based purely on the structures of the PSTs and the links established in the first phase. Note that the first phase also considers the structure of the PSTs since it matches both model elements and regions. The next

section presents the similarity measures for model elements and regions. The following two sections explain the two matching phases using the running example from Fig. 6.2. The pseudo-code of the algorithm is discussed in the Appendix B.

6.5 Matching Criteria for Model Elements and Regions

Our algorithm uses two attribute matching criteria for PSTs: one for matching individual model elements and another for matching regions. We adapted them from previous work on matching source code represented as abstract syntax trees (ASTs) [46]. The original criteria use *bigram string similarity* to match the values of AST leaves and inner nodes. Fluri et al. [46] achieved better results for source code matching using Dice Coefficient with bigrams as string similarity compared to other measures such as the Levenshtein Distance [87]. In particular, the bigram-based similarity tolerates word re-orderings, which also occur in process refinement (e.g., ApproveCard vs. CardApproval).

We have adapted the original matching criteria by Fluri et al. to the process matching context, by using the information available in PSTs and refining the criteria based on experiments with sample models. In particular, we require exact matches for model elements and use bigram similarity only for inner nodes (regions). The reason is that process model elements have often relatively short names, and the names can be very similar, although representing completely different functions (e.g., ApproveCredit, ApproveContract, CreditAccount). The resulting criteria are as follows:

Matching criterion for model elements

$$match_e(n, m) \triangleq (type(n) = type(m)) \wedge (name(n) = name(m))$$

Matching criterion for regions

$$match_r(r, s) \triangleq (\frac{common(r, s)}{max(r, s)} \geq l) \wedge (sim_{2g}(value(r), value(s)) \geq f)$$

where

type returns the type of the model element as a numeric code, such as 0 for start event, 1 for task, 2 for exclusive gateway, etc.

name returns the name of the model element, for example: *Get Balance*, *Debit Savings*, etc.

sim$_{2g}$ calculates the bigram-based similarity of two strings [46]; it returns a numeric value between *0* and *1*, where *1* means that the strings are equal.

value returns the string formed by the concatenation of the names and types of all model elements of a region. Thus, similarity of names is emphasized, since types are short numeric codes and names are typically complete words.

common returns the number of pairs of model elements of the two regions that match exactly (i.e. $match_e$ is *true*).

max returns the maximum number of distinct pairs that could be matched (i.e., the number of all model elements in the smaller region).

f and ***l*** are thresholds controlling the algorithm. We obtained the best results in our evaluation with *0.6* and *0.4*, respectively.

6.6 Attribute Matching

Let us explain the first phase by applying it to the PSTs in Fig. 6.3 obtained from the models in Fig. 6.2.

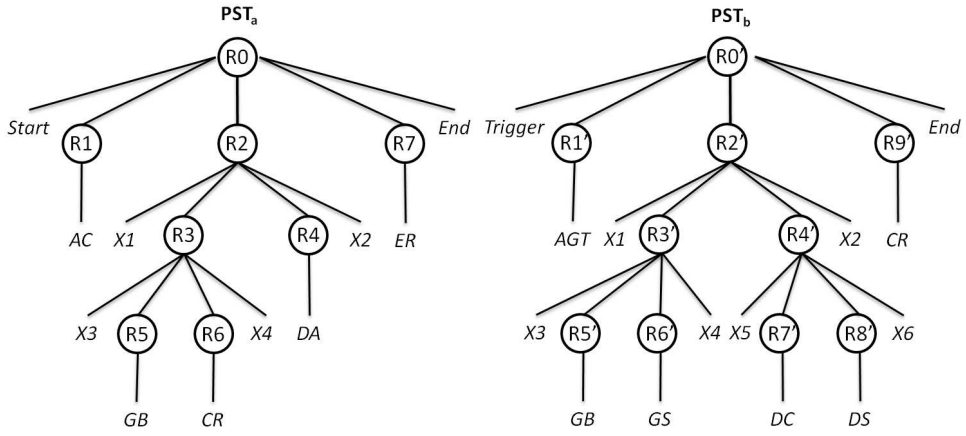


Figure 6.3: PSTs Representation of the Business Process Models

First, the algorithm assumes that the roots of both PSTs correspond to each other. Then, the algorithm performs a depth-first traversal in one of the PSTs in order to establish correspondence links with the second PST. Starting with region *R1* in PST_a, it tries to find a corresponding region in PST_b. According to the matching criterion for regions (cf. Sect. 6.5), a necessary condition for a match is to satisfy the formula $\frac{common(R1,X)}{max(R1,X)} \geq l$ with any region X in PST_b. Since $R1$ has only one child (a model element), satisfying the formula requires finding a region in PST_b containing a model element with exactly the same name and type (matching criterion for model elements) as the activity *Approve Card*. Since there is none, the algorithm proceeds to region $R2$.

For $R2$, the algorithm finds $R2'$ in PST_b to satisfy the above formula ($\frac{5}{6} \geq 0.4$). The algorithm also checks that $sim_{2g}(value(R2), value(R2')) \geq f$ is satisfied. Assuming abbreviations, $value(R2)$ returns *X12X32GB1CR1X42DA1X22*; $value(R2')$ returns *X12X32GB1GS1X42X52DC 1DS1X62X22*. Both strings have a similarity of around 0.65 (assuming full names). The algorithm then keeps on searching more matches for $R2$ in PST_b. The formula $\frac{common(R2,R3')}{max(R2,R3')} \geq l$ is also satisfied, returning $\frac{3}{4}$; however, the value obtained from the string comparison, 0.51, is smaller that f, so $R3$ is discarded as a match (see left figure in Fig. 6.4, where the top link is selected and the bottom one is discarded). No other region in PST_b satisfies the matching criterion with $R2$; however, if there were several matching regions in PST_b, the correspondence link would be established with the region with the highest string similarity to $R2$. If there are more than one region with the same highest string similarity to $R2$ (unlikely though, because copies are uncommon in process modeling), one of them is chosen arbitrarily.

The algorithm keeps traversing PST_a and establishes a correspondence link between $R3$ and $R3'$, since the string similarity value is 0.79 (right figure in Fig. 6.4). $R5'$ in PST_b corresponds to $R5$, since the string similarity is 1. The same applies to $R6$ and $R9'$. There are no correspondence links for $R4$ and $R7$. Finally, the algorithm establishes correspondence links among model elements. In our example, correspondence links from $X1, X2, X3, X4, GB, CR$ and *End* in PST_a to the model elements with the same name in PST_b are created.

Figure 6.5 shows the complete set of correspondence links based on attribute matching, also indicating their model element cardinality. To avoid clutter, the links among model elements with the same name are not shown.

6.7 Structure Matching

The second phase of the algorithm aims to match nodes that have not been matched in the first phase due to their different content. It does so by considering the location of the unmatched nodes in the PSTs and the correspondence links established so far. For example, consider regions $R4$

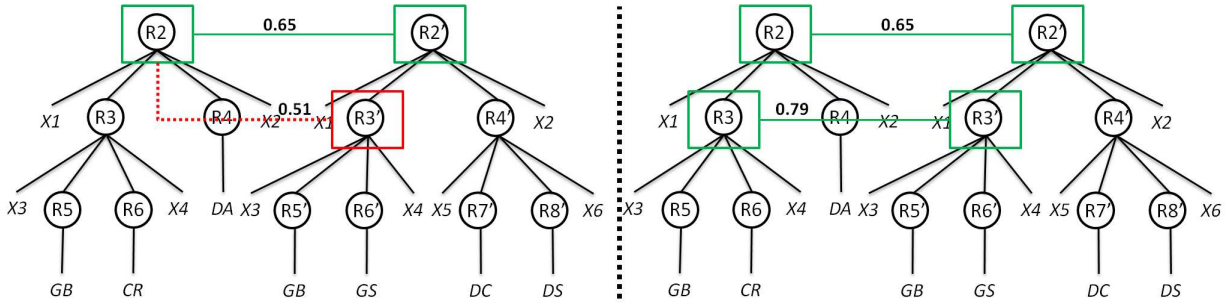


Figure 6.4: Attribute Matching Phase Step by Step for $R2$ and $R3$

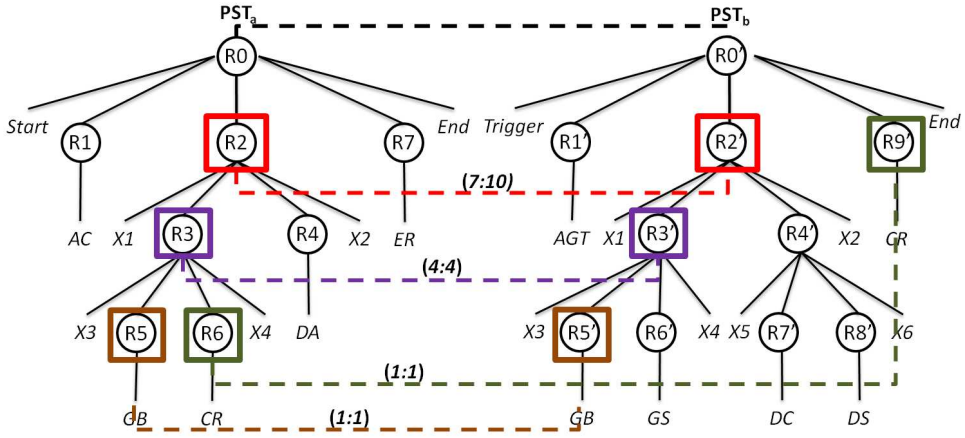


Figure 6.5: Correspondence Links for the Attribute Matching Phase

and $R4'$ in Fig. 6.5. Although they are dissimilar, it is likely, given the correspondence links so far, that they should be linked. The task of this procedure is to find such pairs of nodes and link them. The rule for finding node pairs to link is as follows. Let n_a and n_b be a pair of unmatched nodes. If the parents of n_a and n_b are linked, and if at least one sibling (the left or right one) of n_a and n_b are linked, n_a and n_b should be linked, too. If none of the siblings are linked (possibly because they do not exist), we will also link the nodes if both n_a and n_b are the last or first node in the child list. According to these rules, the aforementioned regions $R4$ and $R4'$ should be matched, as their parents ($R2$ and $R2'$) and their left and right siblings ($R3$ with $R3'$ and $X2$ with $X2$) match. The same happens with $R1$ and $R1'$ since $R0$ matches with $R0'$ and $R2$ with $R2'$. This newly created correspondence link allows us to link the $Start$ and $Trigger$ events, too. All correspondence links established by both phases of the algorithm are shown in Fig. 6.6. As previously, correspondence links between model elements with the same name are not shown.

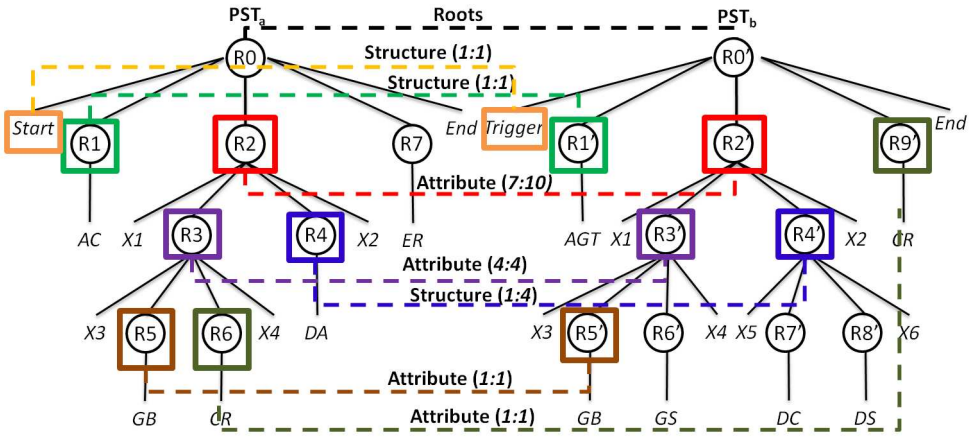


Figure 6.6: Correspondence Links from Both Phases

6.8 Complexity

Assume $n = max(|PST_a|, |PST_b|)$, where $|PST|$ is the number of regions. The cost of comparing the attributes of two regions is denoted by c and the cost of checking their structure similarity is s. The matching of all regions is in $O(n^2(c + s))$, that is, $O(n^2)$, since the algorithm compares each possible region pair.

6.9 Evaluation

We are interested in knowing the *precision* and *recall* of the presented algorithm when establishing correspondences among pairs of real-world process models across different levels of abstraction. Precision tells us whether the recovered correspondence links are correct and recall tells how large of a portion of the links the algorithm can recover. The following subsections present the method we have followed and the results.

6.9.1 Method

We want to evaluate the precision and recall of our algorithm. We define these measures for model correspondences (sets of correspondence links) between the PSTs. We refer to a model correspondence established by the domain experts as a *reference correspondence* (*RC*) and to a

model correspondence established by our algorithm as a *computed correspondence* (CC). Given these sets, precision (P) and recall (R) are defined, respectively, as $P = \frac{|RC \bigcap CC|}{|CC|}$ and $R = \frac{|RC \bigcap CC|}{|RC|}$.

High recall means few *false negatives*, whereas high precision means few *false positives* [10]. False positives are those correspondence links that our algorithm finds but do not belong to the set of reference correspondence links. False negatives are those correspondence links included in the reference correspondence that our algorithm is unable to detect. For most process model matching tools, it is hard to achieve each of high recall and high precision, and it is even harder to achieve both high recall and high precision. As Berry et al. point out [10], a model matching tool should favour recall over precision because errors of commission are generally much easier to correct than errors of omission. Compared to existing approaches for process model matching, our method achieves both high precision and high recall (see Sect. 6.10).

6.9.1.1 Subject data

We used business process models taken from the Bank of Northeast of Brazil (BNB), a major financial institution in Brazil that is controlled by the federal government and oriented towards regional development. BNB has been using Business Process Modeling since 2007 in a development process based on the *Rational Unified Process*. The development process entails iterative and multi-staged model refinement, resulting in three types of process models (from higher to lower level of abstraction): business specifications, technical specifications, and executable processes. We had access to several projects developed as a result of this process and used them for evaluating the algorithm.

Table 6.1: BPM Projects

		Number of Models		
Project	Domain	Business	Technical	Implementation
P1	Customer Registration	2	2	2
P2	Credit Backoffice	6	6	6
P3	Credit Risk Assessment	2	2	2
P4	Procurement	3	3	3

We obtained four real BPM projects, containing 39 models in total. Table 6.1 shows, for each project, the number of models defined in each stage. Our target is to determine the correspondences between each corresponding pair of business and technical specifications and between

the latter and executable implementations. Table 6.2 gives the total number of model elements for each level of abstraction.

Table 6.2: Model Sizes

		Total Numbers		
		Tasks	Gateways	Events
P1	Business Spec.	59	38	25
	Technical Spec.	78	46	36
	Implementation	123	56	43
P2	Business Spec.	47	46	18
	Technical Spec.	95	48	23
	Implementation	107	52	31
P3	Business Spec.	17	8	6
	Technical Spec.	19	10	8
	Implementation	22	6	9
P4	Business Spec.	13	10	11
	Technical Spec.	18	12	15
	Implementation	25	14	17

6.9.1.2 Reference correspondences

As reference correspondences, we use the correspondence links established manually by the domain experts (the bank's employees) who created and maintain the models. The reference correspondences in one of the projects was already established for auditing and regulatory compliance purposes, and reused here. The correspondences for the other projects were established as part of this research.

6.9.1.3 Algorithm implementation

We have implemented the algorithm in Java as an Eclipse feature, on top of the SOA Tools Platform BPMN Modeler [111]. Since the original models from BNB were created using IBM's WebSphere Process Modeler, we needed to recreate them to run our tool.

6.9.2 Results

Table 6.3 shows the results of our evaluation. We matched pairs of models at different levels of abstraction from each project. Concretely, we compared business and technical models, and the latter and IT implementation models. Column "Pair Type" indicates the type of models compared in each row. Column "Corresp - RC" gives the total number of correspondence links identified by the domain experts. Column "Corresp Type" shows the numbers obtained in each phase of the algorithm."Total" represents the net result of the two phases. Notice that the correspondence links do not overlap between the phases. Column "Correct" specifies the number and the cardinality of correspondence links that our algorithm was able to identify, in each phase, from those in the reference correspondence, including their cardinalities. Columns "FP" and "FN" give the number of false positives and negatives, respectively. In each phase, "FP" and "FN" are computed with respect to the complete reference. Finally, "Prec" gives precision, followed by column "Recall".

If we consider the correspondence links all together—as if they had been extracted from only one pair of models—we have 622 reference links found manually by the domain experts. Out of these 622, our algorithm was able to correctly identify 438, with 32 false positives and 184 false negatives, yielding overall recall of 70% and precision of 93% Among the reference links, 117 had cardinality type *1:n* and 89 had the cardinality type *m:n*. From these, the algorithm identified correctly 14 (12%) and 24 (27%), respectively.

The overall precision ranges between 87%-96%. None of the false positives is obtained in the attribute matching phase. This is very positive since a large portion of the reference correspondence links is recovered in this phase. The number of false positives in the structure matching phase is quite large compared to the number of reference correspondence links of purely structural nature. This would be a serious problem in a situation where models have many such purely structural correspondences. We identified two causes for having so many false positives. The principal cause is the presence of non-hierarchical refinement patterns [16]. For example, in one B-T pair of the Project 2, there is an activity in the business specification that corresponds to 3 activities in the technical specification. Each of the 3 activities belongs to a different region in the technical specification. The algorithm cannot identify such kind of correspondence; the second phase matched the region containing the business activity to an incorrect region in the technical specification. Another cause is matching nodes that are the last or first node in the child list. Although this is reasonable in many cases, it also leads to incorrect matches. For example, this rule produced a false positive in one T-IT pair of the Project 3. Unfortunately, without extra information (e.g., IDs or annotations) it is likely not possible to decide whether or not to match the regions in many of such cases.

The relatively high number of false negatives —20%-40%—is caused mainly also by the

Table 6.3: Correspondences among Models across Different Abstraction Levels. B: Business; T: Technical; IT: Information Technology; Corresp - RC: Reference Correspondence; FP: False Positives; FN: False Negatives; Prec: Precision

Project	Pair Type	Corresp - RC	Corresp Type	Correct (*1:1*; *1:n*; *m:n*)	FP	FN	Prec	Recall
1	B–T	30	Attribute	16 (15;0;1)	0	14	100%	53%
			Structure	4 (1;2;1)	2	26	67%	13%
			Total	20 (16;2;2)	2	10	91%	67%
1	T–IT	42	Attribute	28 (26;0;2)	0	14	100%	67%
			Structure	3 (2;1;0)	2	39	60%	7%
			Total	31 (28;1;2)	2	11	94%	74%
2	B–T	138	Attribute	95 (90;0;5)	0	43	100%	69%
			Structure	8 (6;2;0)	4	130	67%	6%
			Total	103 (96;2;5)	4	35	96%	75%
2	T–IT	240	Attribute	136 (127;0;9)	0	104	100%	57%
			Structure	18 (10;5;3)	12	222	60%	8%
			Total	154 (137;5;12)	12	86	93%	64%
3	B–T	32	Attribute	22 (21;0;1)	0	10	100%	69%
			Structure	4 (4;0;0)	2	28	67%	13%
			Total	26 (25;0;1)	2	6	93%	81%
3	T–IT	44	Attribute	32 (32;0;0)	0	12	100%	72%
			Structure	2 (2;0;0)	5	42	29%	5%
			Total	34 (34;0;0)	5	10	87%	77%
4	B–T	42	Attribute	24 (23;0;1)	0	18	100%	57%
			Structure	6 (3;3;0)	3	36	67%	14%
			Total	30 (26;3;1)	3	12	91%	71%
4	T–IT	54	Attribute	36 (36;0;0)	0	18	100%	67%
			Structure	4 (2;1;1)	2	50	67%	7%
			Total	40 (38;1;1)	2	14	95%	74%

presence of non-hierarchical refinements, which occurred in all the projects. We believe that these numbers can be reduced by applying a pattern matching technique for describing and finding instances of well-known or organization-specific non-hierarchical refinements patterns, which we leave for future work.

6.9.3 Threats to validity

This section summarizes the potential threats that may impact the internal and external validity [37] of the empirical results.

6.9.3.1 Threats to external validity

A potential threat to external validity is that the models used in the evaluation may not be representative of those occurring in other realistic settings. While the models used here come from real-world projects, the algorithm should be tested additionally on models from other organizations and domains.

6.9.3.2 Threats to internal validity

The main threat is the re-modeling of the BNB's business process models to be processed by our tool. BNB applies IBM tools that use an extension of BPMN. Some features of the BNB models that are not covered in BPMN had to be omitted during the translation. This threat was minimized by checking with the domain experts that the BMPN models obtained after the simplification were largely equivalent to the original models.

6.10 Comparison

Matching of models is a standard topic in MDD. For example, UMLDiff is a prominent approach for matching UML models [137]. However, effective matching requires heuristics that are usually notation and application specific. Our method focuses on finding such heuristics for matching business process models across levels of abstraction. Discovery of effective heuristics usually requires studying the differences among such models. In this context, Dijkman [27] presents a classification of frequently occurring differences between similar business processes in general.

As in our approach, the work by Dijkman et al. [29] aims to realize business process models alignment based on lexical matching (similar to our attribute matching) and structural matching. They report recall of 60% and precision of 89% for their approach. However, their algorithm only captures 1:1 correspondences between model elements. Our algorithm also identifies correspondences between SESE regions, which is necessary for matching models at different levels of abstraction.

Weidlich et al. present ICOP in [130], a framework based on matchers to identify correspondences between process models. They represent the models using *Refined Process Structure*

Table 6.4: Related BPM Matching Approaches. + : Feature Provided; – : Feature not Provided; NA : Not Available

	Approach			
Feature	Weidlich et al. [130]	Dijkman et al. [29]	Ehrig et al. [41]	Our Approach
Match Activity Attributes	+	+	+	+
Match Model Structure	+	+	–	+
Match Activity-Activity (1:1)	+	+	+	+
Match Activity-SESE (1:n)	+	–	–	+
Match SESE-SESE (m:n)	–	–	–	+
Do not Require Model Element IDs	+	+	+	+
Support Activity Inserts and Deletes	+	+	+	+
Support Activity Moves	+	+	–	+
Support Activity Renaming	+	+	–	+
Support Activity Copies	+	+	–	–
Overall Precision	80%	89%	NA	93%
Overall Recall	60%	60%	NA	70%

Trees (RPSTs) [106, 123] rather than PSTs. In RPSTs, regions can have more than one entry and more than one exit. The approach by Weidlich et al. deals both with 1:1 and 1:n matches. Our approach additionally relates regions to regions, which are examples of m:n matches. They report recall of 60% and precision of 80%.

Ehrig et al. [41] propose a set of similarity measures for process models, for example, in order to discover existing related process models in repositories. However, the approach does not establish fine grained correspondence links like in our approach. The authors do not discuss recall and precision of the approach.

Several related works deal with comparing process models (e.g., [49, 50]), checking their consistency (e.g., [131]), and their synchronization (e.g., [134]). All these works assume that model correspondences have been previously established.

Table 6.4 summarizes our contribution in the light of the related works.

Chapter 7

Generating Edit Operations from Automatic Correspondence Discovery

7.1 Overview

This chapter presents a technique for generating a *diff* between two process models. Figure 7.1 highlights the *diff* generator component inside the framework.

7.2 Motivation

Synchronizing Business and IT process models means propagating changes in both directions, i.e., from Business to IT and vice versa. Writing such synchronizations usually requires uncovering tacit knowledge, as we discussed on Chapter 4. Without appropriate tool support, this task is very time-consuming and error-prone.

Aiming at mitigating this problem, there is a multitude of frameworks for bidirectional model transformations (BXs) [33,35,36,47,69,70]. Although these approaches provide the foundations of BXs in terms of properties and operations, few concrete implementations of such frameworks are available. It remains unclear how to generate such transformations in many problem domains. In particular, there is a lack of practical BX frameworks tailored to deal with consistency of business process models that target different levels of abstraction.

We leverage the approach presented in the Chapter 6 and present a practical approach for generating a *diff* between two process models, by means of edit operations [15].

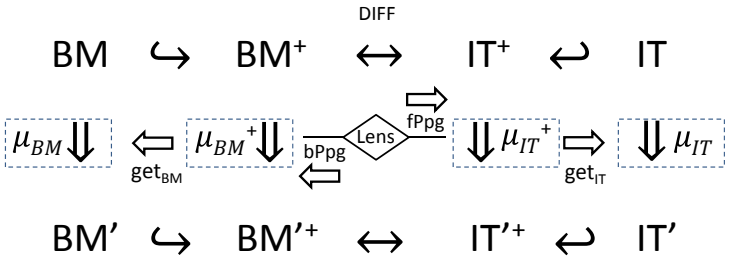


Figure 7.1: Diff Component

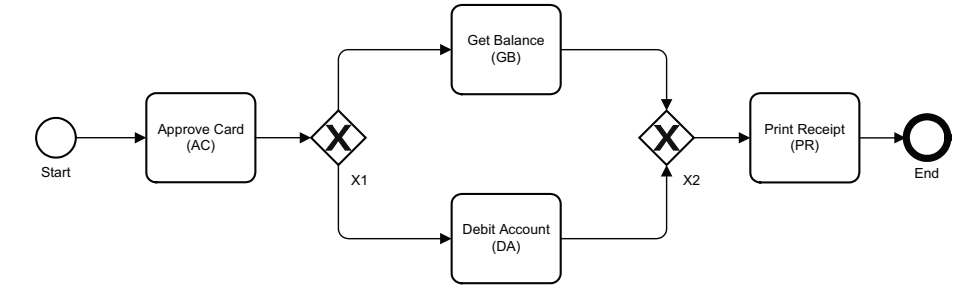
7.3 Running Example

Figure 7.2 shows two models in BPMN 2.0, each representing the process of using an *Automated Teller Machine* (ATM) system at different level of abstraction. We added shorter names in parentheses (e.g., *(AC), (GB)*) to avoid clutter when referring to the models throughout the chapter. The first model (Fig. 7.2.a) represents a business-level process specification. The second one (Fig. 7.2.b) is an IT-level specification.

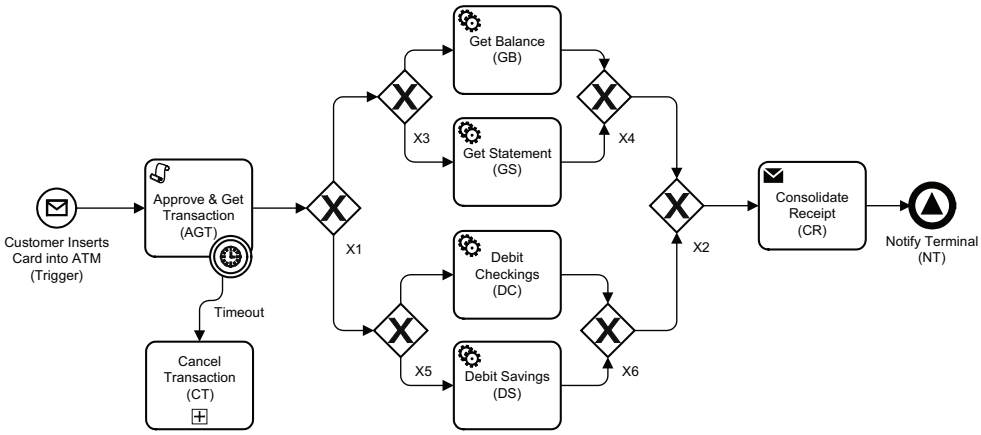
Figure 7.3 shows the PSTs corresponding to the BPMN process models shown in the Figure 7.2. There is a unique PST for each BPMN model. The root represents the whole process model. Leaves represent model elements, i.e., tasks, gateways and events. Inner nodes represent SESE regions.

7.4 Edit Operations

A *lens* is a bidirectional transformation between a pair of connected data structures, X and Y, capable of translating an edit on one structure into an appropriate edit on the other. Each lens is a pair of functions—**to** and **from**—one mapping X updates to Y updates and the other mapping Y updates to X updates.



(a) Business Specification



(b) Technical Specification

Figure 7.2: BPMN Models

Our approach employs *edit operations* to implement symmetric delta lenses for two reasons. First, in business process modeling, changes (including refinement patterns [17]) can be distilled into atomic editing operations, such as *adding, deleting or moving* an activity or *updating* its attributes. Second, edit operations are intuitive. Human users can inspect them easily to review, add, discard or select specific operations before synchronizing the models.

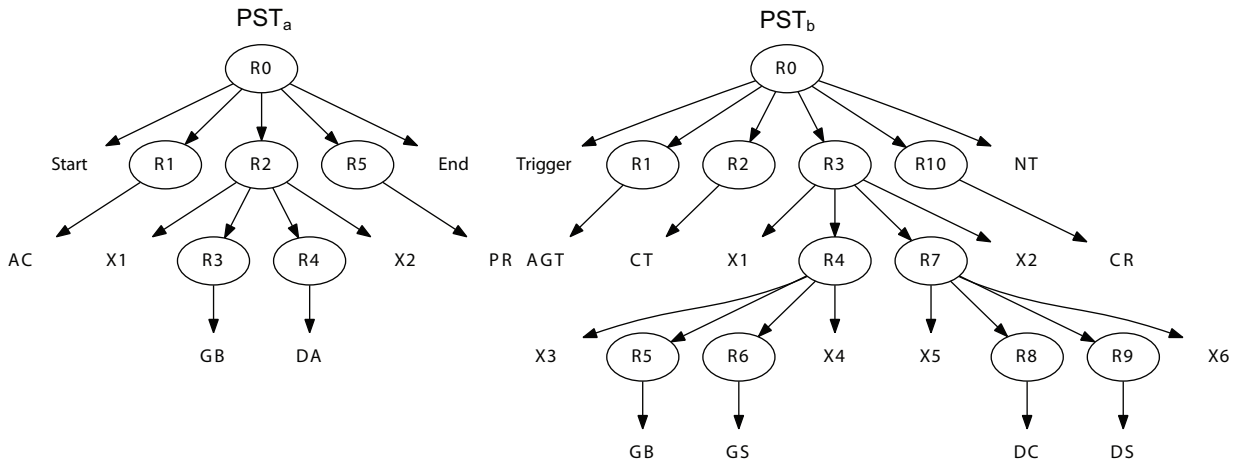


Figure 7.3: PSTs representation of the business process models

7.5 Generating Edit Operations from Correspondences between Process Models

In Chapter 6, we present an algorithm to automatically detect non-trivial correspondence patterns between BMPN process models across levels of abstraction (see Chapter 4). The algorithm identifies attribute and structural correspondences over the PSTs of the input models. Table 7.1 shows the correspondences identified by the matching algorithm on the PSTs shown in the Fig. 7.3.

Table 7.1: Correspondences

i	PST_a.R0	$\triangleq$	PST_b.R0	(Root)
ii	PST_a.Start	$\triangleq$	PST_b.Trigger	(Structure)
iii	PST_a.R1	$\triangleq$	PST_b.R1	(Structure)
iv	PST_a.AC	$\triangleq$	PST_b.AGT	(Structure)
v	PST_a.R2	$\triangleq$	PST_b.R3	(Attribute)
vi	PST_a.R3	$\triangleq$	PST_b.R5	(Attribute)
vii	PST_a.GB	$\triangleq$	PST_b.GB	(Attribute)
viii	PST_a.R4	$\triangleq$	PST_b.R7	(Structure)
ix	PST_a.R5	$\triangleq$	PST_b.R10	(Structure)
x	PST_a.PR	$\triangleq$	PST_b.CR	(Structure)
xi	PST_a.End	$\triangleq$	PST_b.NT	(Structure)

For each correspondence, the lenses generator produces a pair of functions, **to** and **from**,

composed of edit operations for bidirectional transformations in both directions: *business* $\hookrightarrow$ *IT* (*to*) and *business* $\hookleftarrow$ *IT* (*from*). Regions and model elements without correspondences, such as PST_a.DA and PST_b.GS, are treated as individual *adds* or *deletes*. The edit operations are generated according to the following heuristic:

- ***add(l,y,k)***; add a new PST node l as the kth child of node y. For example, in Fig. 7.3, the region PST_b.R6 is added as the 3rd child of PST_b.R4: *add*(PST_b.R6,PST_b.R4,3). Section 5.2.1 describes this operation conceptually.
- ***delete(x)***; delete PST node x from its parent. In the example, PST_a.DA is deleted from PST_a.R4: *delete*(PST_a.DA). Section 5.2.2 describes this operation conceptually.
- ***move(x,y,k)***; node x becomes the kth child of y. In the example, the task PST_a.GB is moved from PST_a.R3 to PST_b.R5: *move*(PST_a.GB,PST_b.R5,1). Move is a combination of the aforementioned *delete* and *add* operations.
- ***update(x,v)***; update value of x with v. For example, the value of the node PST_a.PR was updated to PST_b.CR: *update*(PST_a.PR,PST_b.CR). Section 5.2.5 describes this operation conceptually.
- ***refine(r,s)***; the region r is refined into the region s: the atomic changes are also presented by *adds* and *deletes*. In the example, the region PST_a.R4 is refined into the region PST_b.R7. The corresponding *adds* and *deletes* are presented hierarchically, such as *delete*(PST_a.DA), *add*(PST_b.R8,PST_b.R7,2), and so on. Refine can represent both operations: Split (see Sect. 5.2.3) and Collapse (see Sect. 5.2.4).

Node positions are shown as parameters using absolute paths. A path like "/0/4/3/0" means: the unique child of the 3rd child of the 4th child of the root node. The output of the lenses generator for the correspondences *vii* and *viii* previously shown are as follows:

vii PST_a.GB $\triangleq$ PST_b.GB

to (*business* $\hookrightarrow$ *IT*)

```
move PSTNode="GB" destination="/0/4/2/2" origin="/0/3/2"
```

from (*business* $\hookleftarrow$ *IT*)

```
move PSTNode="GB" destination="/0/3/2" origin="/0/4/2/2"
```

viii PST_a.R4 $\triangleq$ PST_b.R7

to (*business* $\hookrightarrow$ *IT*)

```
refine PSTNode="R4(DA..DA) to R7(X5..X6)" composed of:
  delete PSTNode="DA" source="/0/3/3/0"
  add PSTNode="X5" destination="/0/4/3/0"
  add PSTNode="X6" destination="/0/4/3/1"
  add PSTNode="R8(DC..DC)" destination="/0/4/3/2"
  add PSTNode="R9(DS..DS)" destination="/0/4/3/3"
  add PSTNode="DC" destination="/0/4/3/2/0"
  add PSTNode="DS" destination="/0/4/3/3/0"
```

from (*business* $\hookleftarrow$ *IT*)

```
refine PSTNode="R7(X5..X6) to R4(DA..DA)" composed of:
  delete PSTNode="X5" source="/0/4/3/0"
  delete PSTNode="X6" source="/0/4/3/1"
  delete PSTBranch="R8(DC..DC)" source="/0/4/3/2"
  delete PSTBranch="R9(DS..DS)" source="/0/4/3/3"
  add PSTNode="DA" destination="/0/3/3/0"
```

Users can review this preliminary set of edit operations to add, remove, group and discard specific ones. The final lens (revised by the user) is executed when something is changed in either of the two models to generate a consistent version of the other model. The synchronization is fully automatic: our implementation provides a module on top of VIATRA2 [44] to synchronize any number of individual operations selected by the user. Transformations are first performed on the PSTs, and afterwards they are transformed back into BPMN. Occasional malformed BPMN models after the transformations (e.g., an added task without incoming or outgoing flows) need to be fixed manually by the user. To deal with this issue, we can leverage previous work on generating *quick fixes* that guide a user in fixing post-synchronized models [58].

7.6 Evaluation

7.6.1 Implementation

We have implemented the edit script generator framework in Java as an Eclipse feature, on top of the SOA Tools Platform BPMN Modeler.

7.6.2 Results

The quality of the generated edit operations directly depends on the quality of the matching algorithm, whose recall varies between 40-70% [14]. Thus, the users always need to review the initial lenses to capture the correct refinement patterns and create a baseline of operations that ensure proper business-IT synchronization. We wanted to know how much manual work is needed to update baselines of operations over time, in the presence of typical model changes. We inspected 48 real changes made in three BPM projects over a period of one year, and counted how many individual operations would need to be manually changed in each baseline to meet those changes. The results are shown in the Table 7.2.

Table 7.2: Evaluation

	Number of		
Project	Baseline Operations	Changes	Operations Revised
Customer Registration	273	23	106 (39%)
Credit Backoffice	356	16	163 (46%)
Procurement	161	9	83 (52%)

A large number of operations need to be manually revised to cope with changes. Nevertheless, we believe that keeping the baselines of lenses is useful, instead of the burden of periodically rebuilding all synchronizations from scratch. Approximately 50% of the work over the analysed period would be saved by basically maintaining the lenses incrementally, in pace with the changes.

7.7 Conclusions

We have developed a practical approach that generates edit operations in BPM, tailored to maintaining consistency of process models at different abstraction levels. A prototype tool is imple-

mented on top of Eclipse and SOA Tools Platform. We performed a preliminary evaluation of the tool based on real-world models. As for future work, we aim to perform a qualitative assessment of the approach, obtaining feedback from BPM practitioners in industry. We also hope that this work may encourage developers to evolve the approach and provide tool support to more elaborated BX scenarios in BPM.

Chapter 8

The Shared Model Approach

8.1 Overview

This chapter joins all the pieces of the framework together and shows the Shared Model in practice. The process model synchronizer component—highlighted in the Fig. 8.1—is discussed in detail.

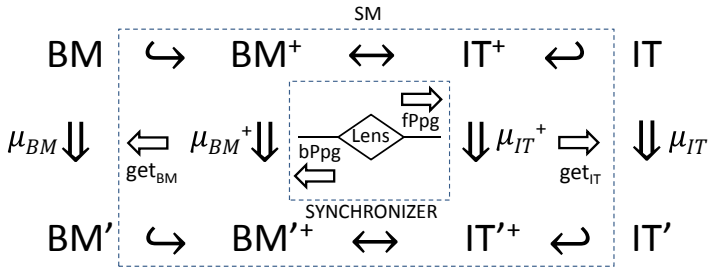


Figure 8.1: Shared Model

8.2 Motivation for a Shared Model

In this section, we motivate our Shared Process Model concept. First we reinforce why we think that a single process model view is often not adequate for different stakeholders and we discuss extra examples on 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.

8.2.1 Why we want different views

Since BPMN 2 can be used for both documentation and execution, why cannot we 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 [17] (see Chapter 4), which analyzed more than 70 model pairs from the financial domain, and we also investigated additional model pairs from other domains. Additionally we talked to BPM architects from companies using process models to collect further differences. We summarize our findings here.

We identified the following categories of changes that were applied in producing an execution model from a business model. Fig. 8.2 illustrates some of these changes in a simplified claim handling process. A larger, more realistic example, is shown in the Appendix, together with a catalog of common refinement patterns identified in the early study [17]. Note that the following categorization of changes, based on the additional data, is a new contribution of this thesis compared to [17].

- *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, to specify the data input for an activity in BPMN 2, one sets a specific attribute of the activity that was previously undefined. The activity itself, its containment in a subprocess hierarchy and its connection with sequence flow do not change.
- *Formalization and renaming.* Some parts of the model need to be formalized further to be interpreted by an execution engine, including routing conditions, specialization of tasks (into service task, human task, etc.; see Fig. 8.2), typing of subprocesses (transaction, call), and typing of events. Furthermore, activities are sometimes renamed by IT to better reflect some technical aspects of the

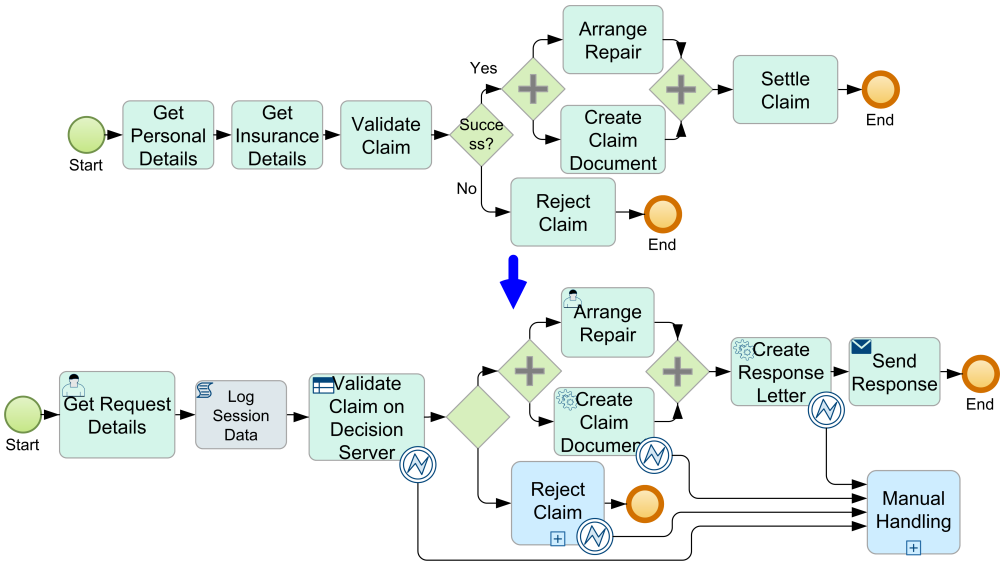


Figure 8.2: Illustration of some refinements often made going from the business to the IT model

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. 8.2 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 thesis.
    On the other hand, multiple tasks on the business level may be collapsed into a single service call or into a single human task to map the required business steps to existing services and sub-engines (human task, business rules). For example, in Fig. 8.2, 'Get Personal Details' and 'Get Insurance Details' are collapsed 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. 8.2).

- *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. 8.2, some fault handling has been added to the IT model to catch failing service calls.
- *Correction and revision of the flow.* Some business-level process models would not pass syntactical and semantical validation checks on the engine. They may contain modeling errors in the control- or data flow that need to be corrected before execution. Sometimes activties also need to be reordered to take previously unconsidered data and service dependencies into account. These changes generally alter the behavior of the process. A special case is the possible parallelization of activities through IT, which may or may not be considered a behavioral change.

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

However, the decision which model elements are business relevant depends on the project and should not be statically fixed, as in the BPMN 2 conformance classes. Therefore, an implementation of progressive disclosure requires extensions that specify which element belongs to which layer. Additional behavior can be handled through progressive disclosure in a similar way as long as there are no dependencies to the business layer. For example, according to the BPMN 2 metamodel, if we add an error boundary event to a task with a 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 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.

8.2.2 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., [17, 120, 128]). 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 [17].

There are various reasons why business and IT models become inconsistent over time. We explained above in Section 8.2.1 (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 already inconsistent in the initial implementation phase. Respondents in our earlier survey [17] have agreed that inconsistency arises already in that phase because the initial business model is incomplete, contains modeling errors, the business model contradicts some IT requirements and the business model does not faithfully represent the actual business process.

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 affect, may change 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.

8.3 Requirements for a Shared Process Model

8.3.1 The Shared Process Model Concept

The *Shared Process Model*, which we now present, has the capability to synchronize process model views that reside on different abstraction levels. The concept is illustrated by Fig. 8.3. The Shared Process Model provides two different views, a business view and an IT view, and maintains the consistency between them. A current view can be obtained at any time by the corresponding stakeholder by the *get* operation. A view may also be changed by the corresponding stakeholder. With a *put* operation, the changed view can be checked into the Shared Proess Model, which synchronizes the changed view with the other view.

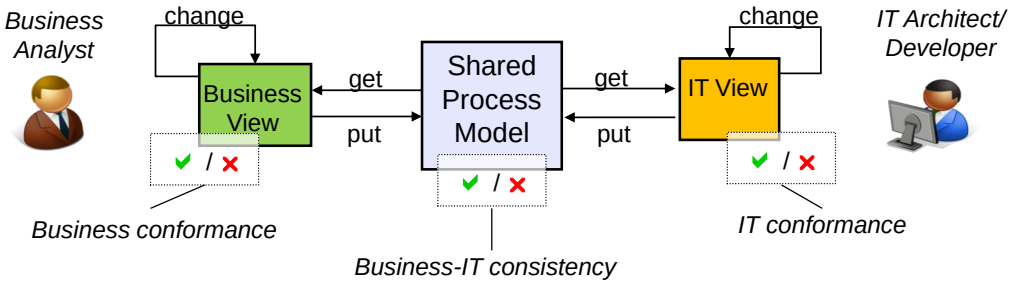


Figure 8.3: Process view synchronization via a Shared Process Model

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

Fig. 8.3 also illustrates the main three status conditions of a Shared Process Model: *business conformance, IT conformance,* and *Business-IT consistency*. The business view is *business conformant* if it is approved by the business analyst, i.e., if it reflects the business requirements. This should include that the business view passes basic validity checks of the business modeling tool. The IT view is *IT conformant* if it is approved by the IT architect, i.e., if it meets the IT requirements. This should include that the IT view passes all validity checks of the IT modeling tool and the execution engine. *Business-IT consistency* means that the business view faithfully reflects the IT view, or equivalently, that the IT model faithfully implements the business view.

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

8.3.2 Usage Scenarios and Requirements

We distinguish the following usage scenarios for the Shared Process Model. In the *presentation* scenario, either the business or IT stakeholder can, at any time, obtain a current state of his view

with the *get* operation. The view must reflect all previous updates, which may have been caused by either stakeholder.

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

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

A private change only takes effect in one view, while the other remains unchanged. Any public change on one view must be propagated to the other view in an automated way. We describe in more detail in Sect. 8.4, 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. 8.4.

The designation of whether a change is private or public is in principle a deliberate choice of the stakeholder that changes his view. However, we imagine that governance rules are implemented that disallow certain changes to be private. For example, a private change 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 [17], 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. 8.2.1 should be compatible with the consistency notion, i.e., should not introduce inconsistencies, wheras changes that cannot be considered refinements should create consistency violations. Checking consistency should be efficient in order to be able to detect violations immediately after a change.

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

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

8.4 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. 8.3. We detail how we have designed and implemented them.

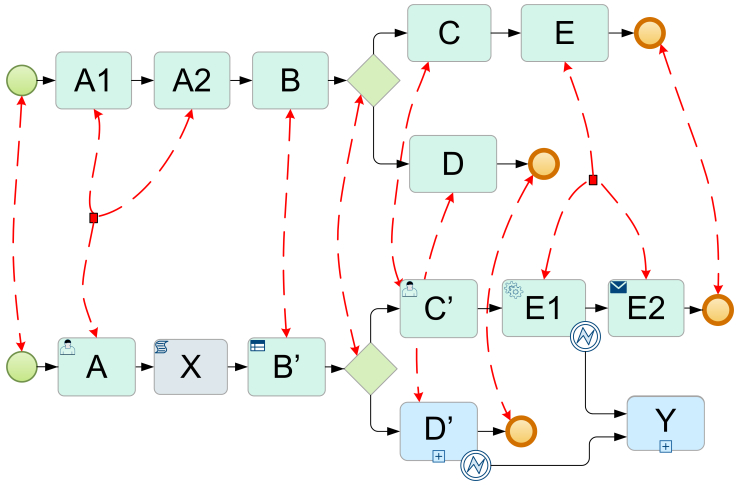


Figure 8.4: The Shared Process Model as a combination of two individual models, coupled by correspondences

8.4.1 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. 8.4. In the upper part, the process model for business is shown, in the lower part the process model for IT. A *correspondence*, shown by red dashed lines, is a bidirectional relation between one or more elements of one model and one or more elements of the other model.

For example, in Fig. 8.4, task B of the business layer corresponds to task B' of the IT layer, which is an example for a one-to-one correspondence. Similarly, task D of the business layer corresponds to subprocess D' of the IT layer and tasks A_1 and A_2 correspond to the (human) task A of the IT layer, which is an example for a many-to-one correspondence. Many-to-many correspondences are theoretically 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.

Alternatively, we could have chosen to represent the Shared Process Model differently by merging the business and IT views into one common model with overlapping parts being represented only once. This ultimately results in an equivalent representation, but we realized that keeping both views separate in the shared models would afford us more flexibility during the development of the prototype.

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

- Import/export to/from the Shared Process Model: From the Shared Process Model, a process model must be created (e.g. business view) that can be shown by an editor. This is straight-forward in our representation. We use BPMN 2 internally in the Shared Process Model, which can be easily consumed outside by existing editors. Likewise, other tools working on BPMN 2 can be leveraged for the Shared Process Model prototype easily.
- Generalization to a Shared Process Model with more than two process models: Such a generalization is easier to realize with correspondences rather than with a merged metamodel. This includes generalization to three or more stakeholder views, but also when one business model is implemented by a composition of multiple models (see Sect. 8.2.1) or when a business model should be traced to multiple alternative implementations.

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

8.4.2 Establishing and Maintaining Correspondences

A possible initialization of the Shared Process Model is with a single process model, which can be thought of the initial business view. This model is then internally duplicated to serve as initially identical business and IT models. This creates one-to-one correspondences between all main elements of the process models for business and IT. The creation of such correspondences is completely automatic because in this case a correspondence is created between elements with the same universal identifier during the duplication process. Another possible initialization is with a pair of initial business and IT views where the two views are not identical. 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 [14].

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

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

8.4.3 Business-IT Consistency

As described in Sect. 8.3.2, 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. All private tasks that are not accounted for violate coverage.

For preservation of behavior, we distinguish *strong and weak consistency* according to the IT refinement patterns discussed in Sect. 8.2.1. If business and IT views are strongly consistent, then they generate the same behavior. If they are weakly consistent, then every behavior of the IT view is a behavior of the business view, but the IT view may have additional behavior, for example, to capture additional exceptional behavior. As with coverage, additional behavior in the IT view should be explicitly reviewed to check that it is indeed considered technical exception behavior and not considered business relevant.

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

- We define the Shared Process Model to be *strongly consistent* if the IT view can be derived from the business view by applying only operations from the first three categories in Sect. 8.2.1: complementary implementation detail, formalization and renaming, and behavioral refinement and refactoring. Private tasks in either view are compatible with consistency only if they are connected to shared elements by a path of sequence flow. The operations from the first three categories all preserve the behavior. The Shared Process Model in Fig. 8.4 is not strongly consistent because the IT view contains private boundary

events. Without the boundary events and without activity *Y*, the model would be strongly consistent. Fig. 8.5 shows examples for violating strong consistency.

An initial Shared Process Model with two identical views is strongly consistent. To preserve strong consistency, all flow rearrangements on one view, i.e., moving activities, rearranging sequence flow or gateways must be propagated to the other view as public changes.

- For *weak consistency*, we currently additionally allow only IT-private error boundary events leading to IT private exception handling. Technically we could also allow additional IT-private gateways and additional branches on shared gateways here, but we have not yet seen a strong need for them. The Shared Process Model in Fig. 8.4 is weakly consistent. The examples in Fig. 8.5 also violate weak consistency.

We have used the simplest notion(s) of consistency such that all the refinement patterns we have encountered so far can be dealt with. We have not yet seen, within our usage scenarios, the need for more complex notions based on behavioral equivalences such as trace equivalence [132] or bisimulation [9].

Strong and weak consistency can be efficiently checked but the necessary algorithms and also the formalization of these consistency notions are beyond the scope of this thesis[1]. 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.

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

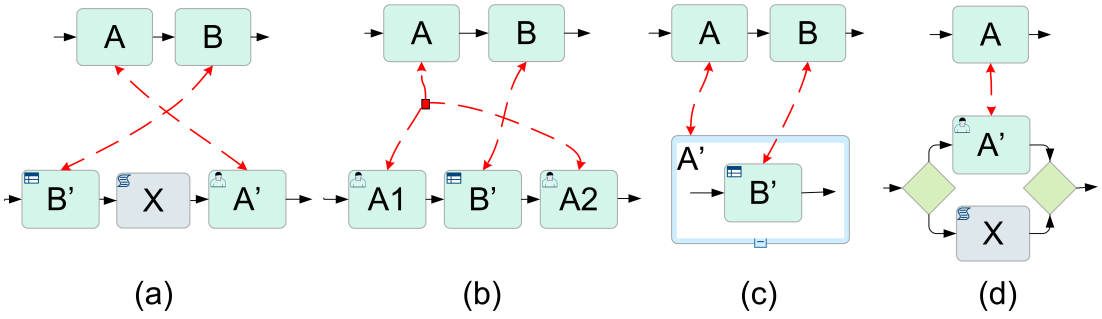


Figure 8.5: Examples of inconsistencies

8.4.4 Computing Changes between Process Model Versions

If the Shared Process Model evolves by changes on the business or IT view, then such changes must be potentially propagated from one view to the other. As a basis for our technical realization of the Shared Process Model, an approach for *compare and merge* of process models is used [83]. We use these compound operations because they minimize the number of changes and represent changes on a higher level of abstraction. This is in contrast to other approaches for comparing and merging models, which focus on computing changes on each model element.

Figure 8.6 shows the change operations that we use for computing 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 Küster et al. [83]

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

Figure 8.6: Change operations according to Küster et al. [83]

As an example for an evolution scenario of the Shared Process Model, consider Figure 8.7. The left hand side shows a part of the initial state of the Shared Process Model in our scenario, which contains a 2-to-1 correspondence and a private IT task. 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. He is wondering why this is not done in

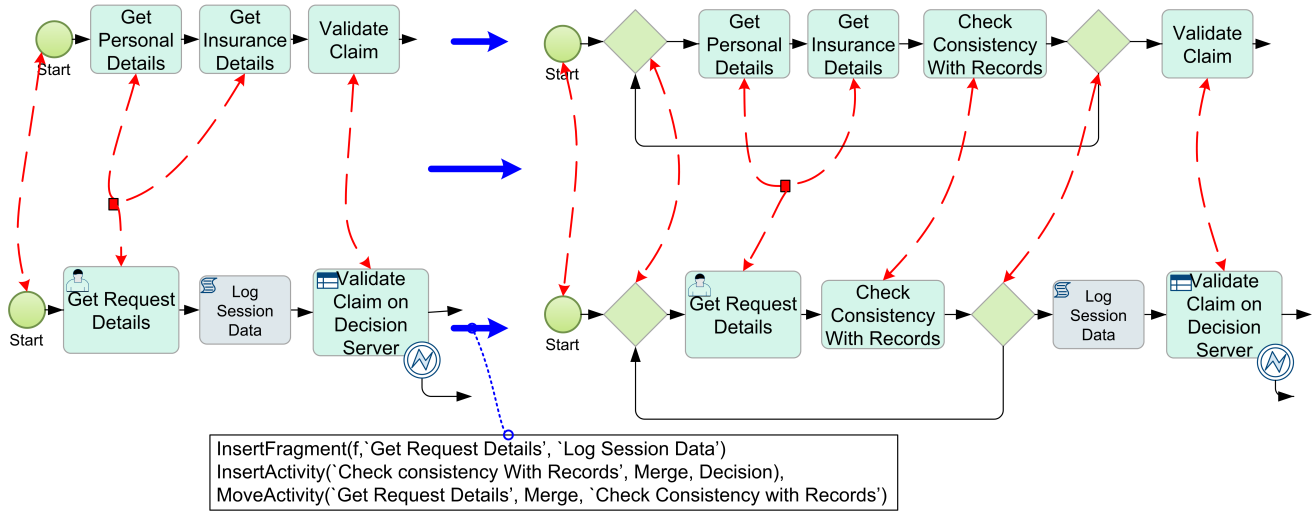


Figure 8.7: Example of a change script on the IT level that is propagated to the business level

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. 8.7. 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. 8.7.

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. 8.7. In the particular example, we compute three compound change operations: the insertion of a new empty fragment containing the two XOR gateways and the loop (*InsertFragment*), the insertion of a new activity (*InsertActivity*), and the move of an activity (*MoveActivity*), illustrated by the change script in Figure 8.7. In the next section, we explain how we use our approach to realize the evolution of the Shared Process Model.

8.4.5 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, 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

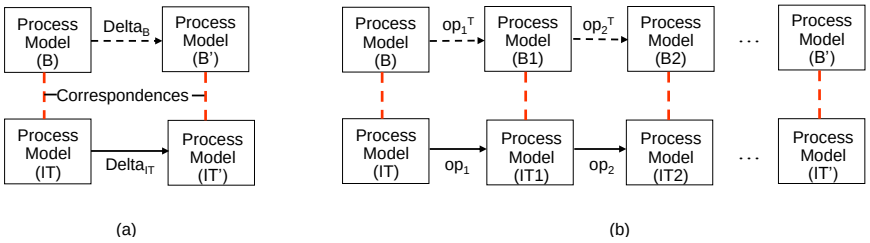


Figure 8.8: Delta computation for propagating changes

between the old model IT and the new model IT', giving rise to a change script $Delta_{IT}$, see Figure 8.8 (a). The change script is expressed in terms of the change operations introduced above, i.e., $Delta_{IT} = \langle op_1, ..., op_n \rangle$ where each op_i is a change operation. In order to propagate the changes to the business level, $Delta_{IT}$ is translated into a change script $Delta_B$ for the business-level. This is done by translating each individual change operation op_i into an operation op_i^T and then applying it to the business-level. Likewise, we also apply each change operation on the IT-level to produce intermediate process models for the IT level. Overall, we thereby achieve a synchronous evolution of the two process models, illustrated in Figure 8.8 (b).

Algorithm 1

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

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

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

IT_i, as illustrated in Figure 8.8 (b). Afterwards, Step 4 updates the correspondences between the business and IT model. For example, If x is the source or target of a one-to-many/many-to-one correspondence, then all elements connected to it must be removed.

For the example in Figure 8.7, the change script $Delta_{IT}$ is translated iteratively and applied as follows:

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

In general, when propagating a change operation, it can occur that the insertion point in the other model cannot be uniquely determined. For example, if a business user inserts a new task between the activity *'Get Insurance Details'* and *'Validate Claim'* in Fig. 8.7, then this activity cannot be propagated to the IT view automatically without user intervention. In this particular case, the user needs to intervene to determine whether the new activity should be inserted before or after the activity *'Log Session Data'*.

In addition to computing changes and propagating them automatically, in many scenarios it is required that before changes are propagated, they are approved from the stakeholders. In order to support this 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.

8.4.6 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. 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 or business views are either all public or all private. With an additional component, this assumption can be removed. Then, the change script computed for the pair of IT views or business views is presented to the user who can then mark the public changes individually. For this scenario, the compare-merge component needs to meet the following two requirements: (i) the change script must be consumable by a human user and (ii) individual change operations presented to the user must be as independent as possible. Note that the change operations in a change script are in general interdependent, which restricts the ability to apply only an arbitrary subset of operations to a model. Therefore, a compare and 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 post-processing. 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. 8.3.2 generate the same requirements for the compare/merge component: human consumability of the change script and separability of approved changes from rejected changes.

Chapter 9

Evaluation

9.1 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?* [17]

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 Chapter 4.

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.

9.2 Subjects

To study how the tool works in a concrete setting, we remodeled and replayed change history of a BPM project—Credit Backoffice—from BNB, our industry partner. Table 9.1 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.

Table 9.1: Project Size

		Number of Model Elements				
		Pools	Tasks	Gateways	Events	Flows
Credit Backoffice	Business	6	47	46	18	128
	IT	6	107	52	31	154

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 can change artifacts in ClearCase only 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 and also 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 Chapter 4) 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.

9.3 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 Chapter 4) by splitting a task in the IT model or merging tasks in the business model.

Table 9.2: Correspondence, Actual Edit and Conceptual Edit Patterns

Correspondence Pattern	Actual Edit Pattern	Conceptual Edit Pattern (see Chapter 5)
Add boundary event (Sect. 4.6.2.5)	Add boundary event	Add
Add script task (Sect. 4.6.2.3)	Add business relevant task to Business	Add
Add script task (Sect. 4.6.2.3)	Add business relevant task to IT	Add
Add manual task (Sect. 4.6.2.2)	Add manual task	Add
Add properties (Sect. 4.6.2.1)	Add properties	Attribute Assign
Add protocol task (Sect. 4.6.2.4)	Add protocol task	Add
Add script task (Sect. 4.6.2.3)	Add script task	Add
Add technical exception flow (Sect. 4.6.2.6)	Add technical exception flow	Add
Change activity name (Sect. 4.6.2.7)	Change activity name	Attribute Assign
Change activity type (Sect. 4.6.2.8)	Change activity type	Attribute Assign
Refactor gateway (Sect. 4.6.2.12)	Refactor gateway	Add or Delete
Split task into block (Sect. 4.6.2.10)	Refine task into fragment	Split
Split workflow (Sect. 4.6.2.11)	Refine task into subprocess	Split
Split task into block (Sect. 4.6.2.10)	Simplify selection into task	Collapse
Split task into block (Sect. 4.6.2.10)	Split task into block	Split
Suppress specification activity (Sect. 4.6.2.9)	Suppress specification activity	Delete

It is also important to clarify the relation between the actual edit patterns collected for the evaluation and the conceptual edit operations discussed in the Chapter 5. The first ones are specializations of the second. Table 9.2 shows the relations among correspondence, actual edit and conceptual edit patterns that occurred in the case study.

9.4 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 [102]. 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) 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 they 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. Section 5.2.4 describes this operation conceptually.
- *Turn into Service Task*: A generic task can be changed into an IT service task. Section 5.2.5 describes this operation conceptually.
- *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*. Section 5.2.3 describes this operation conceptually.
- *Refine Task into a Subprocess*: A special case of splitting a task, where the resulting fragment is a subprocess. Section 5.2.3 describes this operation conceptually.

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 employing 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.

9.5 Results: Single Refinement Patterns

Table 9.3 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 9.4 summarizes 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 section 9.7.

9.6 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 7 extra scenarios (as shown in the Table 9.5) 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 seven months of change on 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

Table 9.3: Evaluation Scenarios: Single Refinement Patterns

		Added				Removed				Synchronization Method				
Scenario	Pattern Instance	Tasks	Flows	Events	Gateways	Tasks	Flows	Events	Gateways	Private (Business Only)	Private (IT Only)	Public (B ⇒ IT)	Public (IT ⇒ B)	Tool Refactoring
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	—	—	—	—	—	—			✓		—
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	—	—	—	—	—	—				✓	—

Table 9.4: Evaluation Results, Single Refinements

Scenario	Edit Pattern	Instance of (see Chapter 4)	Result	Comments	Issues
1	Add manual task	Sect. 4.6.2.2	Success	Manual task and corresponding incoming and outgoing flows were correctly updated on the Business view only	—
2	Change activity name	Sect. 4.6.2.7	Success	New name was correctly updated on the Business view only	—
3	Simplify selection into task	Sect. 4.6.2.10	Success	Selection of fine grained changes was correctly simplified into a single task on the Business view	—
4	Add properties	Sect. 4.6.2.1	Success	Specific properties were correctly updated on the IT view only	—
5	Add script task	Sect. 4.6.2.3	Success	Script task and corresponding incoming and outgoing flows were correctly updated on the IT view only	—
6	Add script task	Sect. 4.6.2.3	Success	Script task and corresponding incoming and outgoing flows were correctly updated on the IT view only	—
7	Add protocol task	Sect. 4.6.2.4	Success	Protocol tasks and corresponding incoming and outgoing flows were correctly updated on the IT view only	—
8	Add protocol task	Sect. 4.6.2.4	Success	Protocol task and corresponding incoming and outgoing flows were correctly updated on the IT view only	—
9	Add boundary event	Sect. 4.6.2.5	Success	Event, flow and task were correctly updated on the IT view only	—
10	Add technical exception flow	Sect. 4.6.2.6	Success	Event, flow and task were correctly updated on the IT view only	—
11	Add technical exception flow	Sect. 4.6.2.6	Success	Event, flow and task were correctly updated on the IT view only	—
12	Change activity type	Sect. 4.6.2.8	Success	New type was correctly updated on the Business view only	—
13	Refine task into fragment	Sect. 4.6.2.10	Success	Task was correctly refined into a fragment of other activities, using the refactoring method provided by the tool	—
14	Suppress specification activity	Sect. 4.6.2.9	Success	Task and flows were correctly removed from the IT view only	—
15	Split task into block	Sect. 4.6.2.10	Success	Selection of activities was correctly simplified into a single task on the Business view	Layout
16	Refine task into subprocess	Sect. 4.6.2.11	Success	Task was correctly refined into a subprocess of other activities, using the refactoring method provided by the tool	—
17	Refine task into subprocess	Sect. 4.6.2.11	Success	Task was correctly refined into a fragment of other activities, using the refactoring method provided by the tool	—
18	Refactor gateway	Sect. 4.6.2.12	Success	New activities added and modifed flows were correctly updated on the IT view	—
19	Add business relevant task to Business	Sect. 4.6.2.3	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	Sect. 4.6.2.9	Success	Gateway, tasks and flows were correctly removed on the IT view only	—
21	Add business relevant task to IT	Sect. 4.6.2.3	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	Sect. 4.6.2.10	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	Sect. 4.6.2.12	Success	New activities added and modifed flows were correctly updated on the IT view, and propagated to the Business view as well	Layout

side, whilst 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 9.6 summarizes the results we obtained. We discuss them in the next section 9.7.

9.7 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 to 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, specially when it has dependencies (e.g., flow connections) on private model elements.

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 entangled flows. 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.

Table 9.5: 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	—

For each change period shown in Table 9.5 we performed the following steps (Fig. 9.1 shows an overview):

1. Recovered the initial and final versions of both business and IT models;
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 [15];
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 on the IT model and pushed the updates to the shared model;
5. Replayed all the private changes on 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 possible only by also inspecting the change request log, which contains textual descriptions of each change, and also the project's documentation.

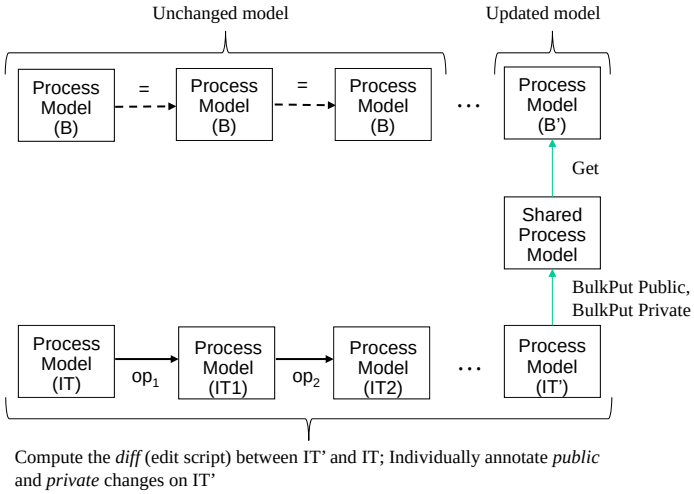


Figure 9.1: Synchronization of Compound Edits

All compound scenarios were correctly synchronized, and a consistent business view was eventually produced, as shown in the Table 9.6. 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 9.2 shows an example of inferred sequence flow between *Task X* and *Task Z* on the business side, after synchronizing all public and private changes on the IT side.

Concurrent Changes

As explained in Sect. 8.3.2, 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. 9.3. 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?*

Table 9.6: 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

The tool was able to deal well with all the evaluated scenarios from BNB, when synchronizing after each edit pattern.

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.

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

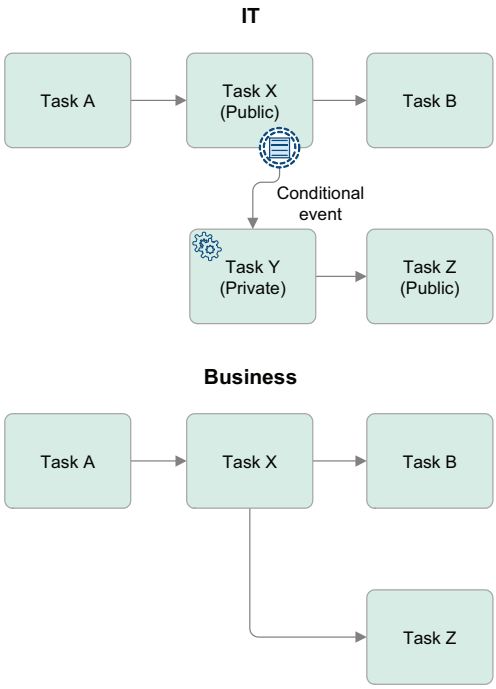


Figure 9.2: Public and Private Synchronization Dependencies

question. However, assessing the feasibility of this approach in industrial practice requires further research.

9.8 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 is 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

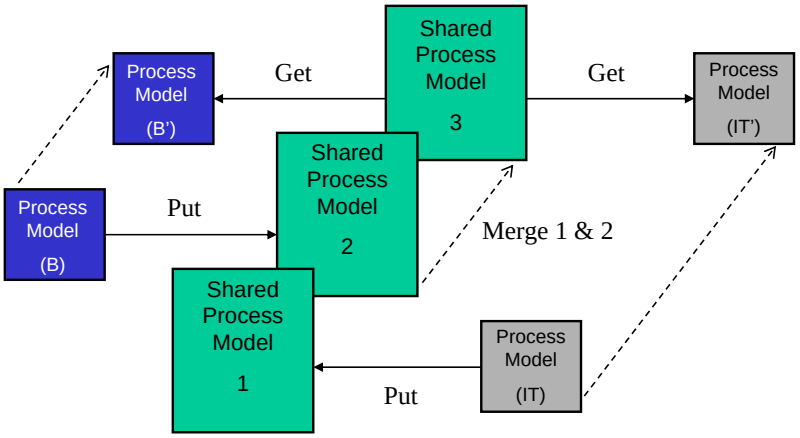


Figure 9.3: Synchronization of Concurrent Changes

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 the 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 *small* changes.

Chapter 10

Conclusions

10.1 Summary

We have developed a practical framework for consistency management in business process modeling. The requirements for the framework come from a comprehensive study of an industrial BPM-driven development process, including the analysis of more than 70 models, 17 hours of interviews with practitioners, and inspection of around 1000 change requests in 5 BPM projects. Our study covers several aspects of consistency management, including types of inconsistencies, causes, impacts, and tool preferences.

The findings detailed in our study highlight some limitations in the way that state-of-the-art BPM solutions work:

- *Development process:* Effective consistency management appears to require a progressive disclosure approach, in which models are created by a smooth progression from high-level specifications to IT-level models, preserving a chain of manageable correspondences. Today, related models are initially created using common refinement patterns, but then maintained separately for satisfying the needs of different stakeholders, possibly in different languages.
- *Stakeholders need a way to define consistency properties:* Consistency is a subjective notion. The same pair of models may or may not be considered inconsistent. The notion of business relevance influences strongly the consistency rules.
- *There is a lack of support for parallel maintenance:* Parallel maintenance requires differencing and merging techniques, something lacking in the major tools.

- *Detection of inconsistencies:* Inconsistencies should be detected and communicated as soon as they occur and then managed according to a clearly defined process.

Our framework aims to mitigate such limitations, by providing an automated way to manage correspondences and to propagate changes between business and IT corresponding models.

We have developed the following techniques, as the main components of the framework:

- *A heuristic process model matching*: Our method discovers which activities in one model correspond to which activities in another model, by taking into consideration frequent correspondence patterns between processes spanning multiple abstraction levels.
- *A process model diff method*: Using the matching as input, we translate the correspondences into human-readable edit operations, such as *insert*, *delete*, *move* and *update*.
- *A process model synchronizer*: Combining the aforementioned techniques together, we present a method for automatically propagate changes between corresponding process models: the Shared Model approach.

We evaluate the framework on 30 concrete modeling scenarios—23 single edit patterns, plus 7 compound edits—and show that the framework successfully deals with all cases, although there may be some issues in the layout of the models and, occasionally, some sequence flows missing.

10.2 Limitations and Future Work

Our framework faces the following main limitations:

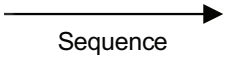
- *Non-hierarchical refinements*: the current matching heuristics are not able to identify *non-hierarchical* refinements (see Sect. 4.6.2). Because of this limitation, such types of refinements can not be managed by the tool. Solving this limitation needs to extend the matching component to deal with user-defined, project-specific patterns (beyond the current lexical and topological correspondences).
- *Concurrent changes*: the current design of the framework is not able to merge simultaneous changes. To solve this issue it would be needed to provide a way to deal with edit conflicts between the views, and merge them back to the Shared Model, along with the updated traceability links.

Regarding future work, we plan to develop improved heuristics to extend the framework, such as matching based on pattern queries and dependencies between consistency rules, as well as investigate different solver algorithms. A promising idea is to add the ability to *learn* user-selected fixes by storing them in a runtime knowledge base and reusing such knowledge for later solution generation. This could be investigated in a model versioning and conflict management case study.

APPENDICES

Appendix A

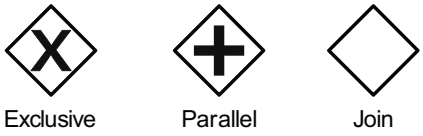
Basic BPMN Notation



(a) Flow

(b) Tasks

(c) Events



(d) Gateways

Appendix B

Matching Algorithm Pseudocode

B.1 Introduction

In this appendix we present the pseudocode of the algorithm described in our MODELS'12 paper [14], entitled *Matching Business Process Workflows Across Abstraction Levels*. In our matching algorithm, we assume that the models to be matched represent the same process, but at different levels of abstraction. We also assume that, although the models are intended to be consistent, inconsistencies can occur during their evolution and the models to be matched may include some inconsistencies. The aim of the algorithm is to automatically identify a correspondence among the models. A correspondence is defined by a set of correspondence links. To achieve that, the algorithm actually deals with the PST representations of the models. Leaves in a PST represent model elements, while inner nodes represent SESE regions. In this document, when referring to the elements in a PST, we will use the term *model element* to refer to leaves, *region* to refer to SESE regions and *node* to refer to both of them indistinctly. Thus, a correspondence link establishes a relation between two nodes in both PSTs.

B.2 Algorithm's Pseudocode

The algorithm receives as input the two business process models ($Model_a$ and $Model_b$) whose correspondence we want to establish and the threshold values, f and l, controlling node's similarity. It produces, as result, the PST representations (PTS_a and PST_b) of the corresponding models as well as the set of correspondence links (*corr*) established among the PSTs' nodes. The objects of type *CorrsLink* are pairs of nodes.

The algorithm is summarized in Listing 1. First, it initializes the output variables: the set of correspondence links is an empty set at the beginning (line 2) and the variables representing the PSTs are initialized with the result of the function *buildPST*, which takes a process model as input and returns its PST's representation (lines 3 and 4). Then, correspondence links established by attribute (first phase of our algorithm) are inserted in *corr* (line 5). It is done by the procedure *matchPSTs_Attribute*, shown in Listing 2. Then, correspondence links determined by structure (second phase of the algorithm) are inserted in *corr* (line 6) by calling the procedure *matchPSTs_Structure*, shown in Listing 3. Finally, the algorithm returns the PST's representations of the input models and the set of correspondence links among their nodes (line 7).

Listing 2 Process models matching procedure

```
1: procedure BPMNMatching(in Model_a, Model_b : Model; in f, l : Real; out PST_a, PST_b :
   PST; out corr : CorrsLink[ ])
2:     corr ← \emptyset;
3:     PST_a ← buildPST(Model_a);
4:     PST_b ← buildPST(Model_b);
5:     corr ← matchPSTs_Attribute(PST_a, PST_b, f, l, corr);
6:     corr ← matchPSTs_Structure(PST_a, PST_b, corr);
7:     return PST_a, PST_b, corr;
8: end procedure
```

The first phase of the algorithm matches the nodes by content similarity, comparing names and types of nodes. When comparing two model elements, the string resulting from the concatenation of their names and types is compared, as explained in the paper. As for regions, the string produced from the concatenation of names and types of all their model elements is compared. This phase is called *attribute matching* and is shown in Listing 2.

This procedure receives as input the PSTs of the models and the threshold values. In adds correspondence links established in this phase to the *corr* variable. The declared variables are *link*, of type *CorrsLink* (pair of nodes), and *maxStringSim*, *leavesComp* and *stringSim*, of type *Real*.

In this phase, the roots of the PSTs are matched by default (line 4). Then, the algorithm performs a depth-first traversal in PST_a (line 5) in order to establish correspondence links with PST_b. If the node in PST_a is a model element, i.e. a leaf (line 6), it traverses PST_b (line 7) in order to find a model element with the same name and type. If it finds it (line 8), a correspondence link is established among them and included in *corr* (line 9).

Listing 3 PSTs matching by attributes

```
1: procedure matchPSTs_Attribute(in PST_a, PST_b : PST; in f, l : Real; in/out corr :
   CorrsLink[ ])
2:     CorrsLink link;
3:     Real maxStringSim, leavesComp, stringSim;
4:     corr ← addLink(getRoot(PST_a), getRoot(PST_b));
5:     for each n_a in getNodes(PST_a) do
6:         if n_a.isLeaf() then
7:             for each n_b in getLeaves(PST_b) do
8:                 if type(n_a) = type(n_b) and name(n_a) = name(n_b) then
9:                     corr ← addLink(n_a, n_b);
10:                end if
11:            end for
12:        else
13:            maxStringSim ← 0;
14:            for each n_b in getRegions(PST_b) do
15:                leavesComp ← common(n_a,n_b) / max(n_a,n_b);
16:                stringSim ← sim_2g(value(n_a), value(n_b));
17:                if leavesComp ≥ f and stringSim ≥ l then
18:                    if stringSim > maxSim then
19:                        maxStringSim ← stringSim;
20:                        link ← (n_a, n_b);
21:                    end if
22:                end if
23:            end for
24:            if maxStringSim > 0 then
25:                corr ← addLink(link);
26:            end if
27:        end if
28:    end for
29:    return corr;
30: end procedure
```

If the node in PST_a is a region (line 12), the algorithm traverses PST_b (line 14) looking for a region to which establish a correspondence link. It keeps in *maxStringSim* the maximum string similarity found (0 at the beginning, line 13), in *leavesComp* the comparison result of both regions in terms of leaves matching (line 15), as explained in the paper, and in *stringSim*

their string similarity (line 16). As long as *leaveComp* and *stringSim* are bigger than their corresponding threshold values (line 17), the algorithm compares if the string similarity of the two nodes (*stringSim*) is bigger than the maximum string similarity (*maxStringSim*) kept until the moment (line 18). If it is, the latter variable is updated with the new string similarity (line 19) and the pair of regions is stored in the variable *link* (line 20). If the current region in PST_a, n_a, was matched with any region n_b in PST_b, i.e., if the *maxStringSim* is bigger than 0 (line 24), then the correspondence link stored in *link* is added to the set of corresponding links (line 25).

Eventually, this procedure returns the set of correspondence links (line 29).

The second phase of the algorithm matches the remaining unmatched nodes by structural similarity, comparing neighborhood matches among their parents and siblings. This phase is called *structure matching* and is shown in Listing 3.

Listing 4 PSTs matching by structure

```
 1: procedure matchPSTs_Structure(in PST_a, PST_b : PST; in/out corr : CorrsLink[ ])
 2:     NodesPair[ ] np;
 3:     np ← getUnmatchedNodes(PST_a, PST_b, corr);
 4:     for all (n_a, n_b) in np do
 5:         if areMatched(n_a.parent(PST_a), n_b.parent(PST_b)) and
              areMatched(n_a.leftsibling(PST_a), n_b.leftsibling(PST_b)) or
              areMatched(n_a.rightsibling(PST_a), n_b.rightsibling(PST_b)) or
              n_a.isFirstChild(PST_a) and n_b.isFirstChild(PST_b) or
              n_a.isLastChild(PST_a) and n_b.isLastChild(PST_b)
            then
 6:             corr ← addLink(n_a, n_b);
 7:         end if
 8:     end for
 9:     return corr;
10: end procedure
```

This procedure declares a set of node pairs, *np*, of type *NodesPair* (line 2), where it keeps all the pairs of nodes in PST_a and PST_b that have not been matched with any node. These pairs are retrieve by the method *getUnmatchedNodes* and stored in the variable *np* (line 3). For every pair of nodes, n_a and n_b, belonging to *np*, if their parents are matched, and if at least one sibling (the left or right one) are matched or if none of the siblings match but both n_a and n_b are the last or first node in the child list (line 5), then a correspondence link among the nodes is added to *corr* (line 6). Finally, this phase returns the updated set of correspondence links (line 9).

Appendix C

Academic and Research Activities

Table C.1 displays a list of relevant academic and research activities accomplished in the course of the Ph.D. program towards its completion. In the following sections we discuss the activities in more detail.

C.1 Accomplished Activities

We have conducted this work in close collaboration with Alex Lau and Phil Coulthard from the IBM Toronto Lab, and Hagen Völzer and Jochen Küster from the IBM Research Zurich. We had weekly calls with the IBM researchers from Zurich from February 2011 to July 2013. We also had roughly monthly consultations with the IBM personnel from the IBM Toronto Lab in the course of 2011. Following we comment on the progress we had towards the accomplishment of the thesis:

Advance our understanding of the relationship between models that target different levels of abstraction and how to characterize consistency among them. Towards achieving this objective (see contribution 1), we conducted an in-depth empirical study of a business-driven engineering process deployed at a large company in the banking sector. We analyzed more than 70 business process models developed by the company, including their change history, with over 1000 change requests. We also interviewed 9 business and IT practitioners and also surveyed—via questionnaires—a total of 23 business and IT practitioners to understand concrete difficulties in consistency management of business process models, the rationales for the specification-to-implementation refinements found in the models, strategies that the practitioners use to detect and fix inconsistencies, and how tools could help with these tasks. Our main accomplishments

Table C.1: Timetable of Academic and Research Activities

Year	Term	Activities
2010	Winter	☑ Course on Model-based Software Engineering (CS846) ☑ CAS UDays Poster: Maintaining Consistency Between BPMN and BPEL Models ☑ Background study on BPMN-BPEL transformation and round-trip engineering
	Spring	☑ Course on Requirements Engineering (CS846) ☑ Initial collaboration with IBM on the multi-perspective BPM project
	Fall	☑ Course on Software Bug Detection and Tolerance (ECE750) ☑ CASCON 2010 Workshop: Effective Consistency Management in BPM [18] ☑ Initial work on matching business process workflows
2011	Winter	☑ Started research collaboration with IBM Research Zurich ☑ CAS UDays Poster: Friendly Change Extraction for BPMN Workflows ☑ IBM TechConnect 2011 Panel: Friendly Change Extraction for BPMN Workflows ☑ Empirical study on process model consistency management at BNB ☑ Collaboration with István Ráth (Budapest Univ. of Technology and Economics)
	Spring	☑ Acceptance of an IBM CAS PhD Fellowship Project ☑ Compilation of main findings regarding the empirical study [16] ☑ Attendance at the 1st SAT/SMT Summer School at MIT ☑ Publication at the IEEE VL/HCC 2011 Conference [58]
	Fall	☑ Submission of a journal paper (SoSyM), result of the empirical study [17] ☑ Presentation at the 3rd ORF-RE Workshop ☑ CASCON 2011 Technology Showcase: Quick Consistency Management in BPM
2012	Winter	☑ Publication of an IBM Technical Report [81] ☑ Evaluation of a heuristic method for process model matching
	Spring	☑ Publication at the MODELS 2012 Conference [14] ☑ Presentation at the 4th ORF-RE Workshop ☑ Writing PhD proposal
	Fall	☑ Work on generating edit lenses for business process modeling ☑ PhD comprehensive examination
2013	Winter	☑ Publication of the SoSyM journal paper [17] ☑ Publication at the ECMFA 2013 Conference [85] ☑ Publication at the CAiSE 2013 Conference [15]
	Spring	☑ Working on the comprehensive evaluation of the shared model approach ☑ Writing PhD Thesis
	Fall	☑ PhD Seminar
2014	Winter	☑ Submission of a journal version of the shared model featuring its evaluation [86] ☑ PhD Defense

were 1) an account of how business process specification and executable models co-evolve and how their consistency is maintained in a concrete industrial setting; 2) a set of recurrent patterns used to refine business-level process specifications into IT-level executable models, and 3) a set of findings that confirm or contradict conventional wisdom on process model consistency management found in the literature. As a result of this study, we have published a paper in the Journal of Software and Systems Modeling (SoSyM)—Special Issue on Enterprise Modeling [17].

Develop techniques to discover correspondence via matching and support traceability among the models. Towards achieving this objective (see contribution 2), we developed an algorithm for matching and differencing arbitrary BPMN models. The algorithm translates BPMN models into corresponding process structure trees and uses techniques for matching ordered trees. We have developed a prototype implementing the algorithm, which we presented at the IBM TechConnect'11. We extended the algorithm to match non-hierarchical refinements (see Sect. 6.9), common in models residing in different levels of abstraction. The extension exploits the patterns that we discovered in the empirical study described above. The paper describing the algorithm has been accepted and presented at the MODELS'12 Conference [14].

Develop a technique to generate edit operations based on the matching and support process model synchronization. Towards achieving this objective (see contribution 3), we developed an approach that leverages the matching and produces an edit script based on edit lenses. We presented the approach at the CAiSE'13 Conference [15].

Develop a technique to automatically generate fixing actions for post-synchronized malformed process models. Towards this objective (see contribution 4), we developed a solution to generate quick fixes for BPMN models, based on predefined repair templates. The solution uses graph matching, graph transformation, and constraint satisfaction problem (CSP) solving over models. This last work was presented at the VL/HCC'11 Conference [58]. We also presented a technology showcase at the CASCON'11, showing the approaches and prototypes developed. We also collaborated with IBM to produce a technical report that proposes the use of custom views on a shared model [81].

References

[1] Steve Adolph, Wendy Hall, and Philippe Kruchten. A methodological leg to stand on: lessons learned using grounded theory to study software development. In *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*, CASCON '08, pages 13:166–13:178, New York, NY, USA, 2008. ACM.

[2] Marcos Almeida da Silva, Alix Mougenot, Xavier Blanc, and Reda Bendraou. Towards automated inconsistency handling in design models. In Barbara Pernici, editor, *Advanced Information Systems Engineering*, pages 348–362. Springer, 2010. LNCS 6051.

[3] Carsten Amelunxen, Elodie Legros, Andy Schürr, and Ingo Stürmer. Checking and enforcement of modeling guidelines with graph transformations. In *Applications of Graph Transformations with Industrial Relevance*, pages 313–328. Springer, 2008. LNCS 5088.

[4] Michal Antkiewicz. Round-trip engineering using framework-specific modeling languages. In *Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion*, OOPSLA '07, pages 927–928, New York, NY, USA, 2007. ACM.

[5] Michal Antkiewicz and Krzysztof Czarnecki. Design space of heterogeneous synchronization. In Ralf Lämmel, Joost Visser, and João Saraiva, editors, *Generative and Transformational Techniques in Software Engineering II*, pages 3–46. Springer-Verlag, Berlin, Heidelberg, 2008.

[6] András Balogh and Dániel Varró. Advanced model transformation language constructs in the VIATRA2 framework. In *ACM Symp. on Applied Computing (SAC 2006)*, page 1280–1287, Dijon, France, 2006. ACM Press.

[7] Gábor Bergmann, Ákos Horváth, István Ráth, and Dániel Varró. Experimental assessment of combining pattern matching strategies with VIATRA2. *Journal of Software Tools in Technology Transfer*, 2009.

[8] Gábor Bergmann, András Ökrös, István Ráth, Dániel Varró, and Gergely Varró. Incremental pattern matching in the viatra model transformation system. In *Proceedings of the Third International Workshop on Graph and model transformations*, pages 25–32. ACM, 2008.

[9] J. A. Bergstra. *Handbook of Process Algebra*. Elsevier Science Inc., New York, NY, USA, 2001.

[10] Daniel Berry, Ricardo Gacitua, Pete Sawyer, and SriFatimah Tjong. The case for dumb requirements engineering tools. In Björn Regnell and Daniela Damian, editors, *Requirements Engineering: Foundation for Software Quality*, volume 7195 of *Lecture Notes in Computer Science*, pages 211–217. Springer Berlin Heidelberg, 2012.

[11] Norbert Bieberstein, Sanjay Bose, Marc Fiammante, Keith Jones, and Rawn Shah. *Service-Oriented Architecture Compass: Business Value, Planning, and Enterprise Roadmap*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[12] Ralph Bobrik, Manfred Reichert, and Thomas Bauer. View-based process visualization. In *Business Process Management*, volume 4714 of *Lecture Notes in Computer Science*, pages 88–95. Springer Berlin Heidelberg, 2007.

[13] E. Börger and R. Stärk. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer-Verlag, 2003.

[14] Moisés Castelo Branco, Javier Troya, Krzysztof Czarnecki, Jochen Küster, and Hagen Völzer. Matching Business Process Workflows Across Abstraction Levels. In *Proceedings of 15th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, MODELS 2012. ACM/IEEE, 2012.

[15] Moisés Castelo Branco and Arif Wider. Generating preliminary edit lenses from automatic pattern discovery in business process modeling. In *Proceedings of the CAiSE'13 Forum at the 25th International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 65–72, 2013.

[16] Moisés Castelo Branco, Yingfei Xiong, Krzysztof Czarnecki, Jochen Küster, and Hagen Völzer. An Empirical Study on Consistency Management of Business and IT Process Models. Technical Report GSDLAB-TR 2012-03-22, Generative Software Development Laboratory, University of Waterloo, Waterloo, 2012.

[17] Moisés Castelo Branco, Yingfei Xiong, Krzysztof Czarnecki, Jochen Küster, and Hagen Völzer. A Case Study on Consistency Management of Business and IT Process Models in Banking. *Software and Systems Modeling (SoSyM) – Special Issue on Enterprise Modeling*, 2013.

[18] Moisés Castelo Branco, Yingfei Xiong, Krzysztof Czarnecki, Janette Wong, and Alex Lau. Effective collaboration and consistency management in business process modeling. In *Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '10, pages 349–350, Riverton, NJ, USA, 2010. IBM Corp.

[19] Stephan Buchwald, Thomas Bauer, and Manfred Reichert. Bridging the gap between business process models and service composition specifications. In *Service Life Cycle Tools and Technologies: Methods, Trends and Advances*, pages 124–153. Idea Group Reference, November 2011.

[20] Hong-Mei Chen. Towards service engineering: Service orientation and Business-IT alignment. In *Proceedings of the Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, HICSS '08, pages 114–, Washington, DC, USA, 2008. IEEE Computer Society.

[21] Krzysztof Czarnecki, J. Nathan Foster, Zhenjiang Hu, Ralf Lämmel, Andy Schürr, and James F. Terwilliger. Bidirectional transformations: A cross-discipline perspective. In *Proceedings of the 2nd International Conference on Theory and Practice of Model Transformations*, ICMT '09, pages 260–283, Berlin, Heidelberg, 2009. Springer-Verlag.

[22] Thomas H. Davenport. *Process innovation: Reengineering work through information technology*. Harvard Business School Press, Boston, MA, USA, 1993.

[23] Valeria De Castro, Esperanza Marcos, and Roel Wieringa. 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. *International Journal of Cooperative Information Systems*, 18(2):225–260, 2009.

[24] Gero Decker. Bridging the gap between business processes and existing IT functionality. In *Proceedings of the 1st International Workshop on Design of Service-Oriented Applications (WDSOA)*, pages 17–24, Amsterdan, The Netherlands, 2005. ICSOC.

[25] Juliane Dehnert and Wil M. P. van der Aalst. Bridging the gap between business models and workflow specifications. *Int. J. Cooperative Inf. Syst.*, 13(3):289–332, 2004.

[26] Andrea Delgado, Francisco Ruiz, Ignacio Garcia-Rodriguez de Guzman, and Mario Piattini. A model-driven and service-oriented framework for the business process improvement. *Journal of Systems Integration*, 1(3):45–55, 2010.

[27] Remco Dijkman. A classification of differences between similar business processes. In *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference*, pages 37–, Washington, DC, USA, 2007. IEEE Computer Society.

[28] Remco Dijkman. Diagnosing differences between business process models. In *Proceedings of the 6th International Conference on Business Process Management*, BPM '08, pages 261–277, Berlin, Heidelberg, 2008. Springer-Verlag.

[29] Remco Dijkman, Marlon Dumas, Luciano Garcia-Banuelos, and Reina Kaarik. Aligning Business Process Models. In *2009 IEEE International Enterprise Distributed Object Computing Conference*, pages 45–53. IEEE, September 2009.

[30] Remco M. Dijkman, Dick A. C. Quartel, Luis Ferreira Pires, and Marten J. van Sinderen. A rigorous approach to relate enterprise and computational viewpoints. In *Proceedings of the Enterprise Distributed Object Computing Conference, Eighth IEEE International*, pages 187–200, Washington, DC, USA, 2004. IEEE Computer Society.

[31] Zinovy Diskin. Model synchronization: mappings, tiles, and categories. In *Proceedings of the 3rd international summer school conference on Generative and transformational techniques in software engineering III*, GTTSE'09, pages 92–165, Berlin, Heidelberg, 2011. Springer-Verlag.

[32] Zinovy Diskin, Krzysztof Czarnecki, and Michal Antkiewicz. Model-versioning-in-the-large: Algebraic foundations and the tile notation. In *Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models*, CVSM '09, pages 7–12, Washington, DC, USA, 2009. IEEE Computer Society.

[33] Zinovy Diskin, Yingfei Xiong, and Krzysztof Czarnecki. From state- to delta-based bidirectional model transformations. In *Proceedings of the Third international conference on Theory and practice of model transformations*, ICMT'10, pages 61–76, Berlin, Heidelberg, 2010. Springer-Verlag.

[34] Zinovy Diskin, Yingfei Xiong, and Krzysztof Czarnecki. Specifying overlaps of heterogeneous models for global consistency checking. In *Proceedings of the First International Workshop on Model-Driven Interoperability*, MDI '10, pages 42–51, New York, NY, USA, 2010. ACM.

[35] Zinovy Diskin, Yingfei Xiong, and Krzysztof Czarnecki. From State- to Delta-Based Bidirectional Model Transformations: the Asymmetric Case. *Journal of Object Technology*, 10, 2011.

[36] Zinovy Diskin, Yingfei Xiong, Krzysztof Czarnecki, Hartmut Ehrig, Frank Hermann, and Fernando Orejas. From State- to Delta-Based Bidirectional Model Transformations: the Symmetric Case. In *Proceedings of the 14th international conference on Model driven engineering languages and systems*, MODELS'11, pages 304–318, Berlin, Heidelberg, 2011. Springer-Verlag.

[37] Steve M. Easterbrook, J. Singer, M. Storey, and D. Damian. Selecting Empirical Methods for Software Engineering Research. In *Guide to Advanced Empirical Software Engineering*, pages 285–311. Springer, 2007.

[38] A. Egyed, E. Letier, and A. Finkelstein. Generating and evaluating choices for fixing inconsistencies in uml design models. In *Automated Software Engineering, 2008. ASE 2008. 23rd IEEE/ACM International Conference on*, pages 99 –108, 2008.

[39] H. Ehrig, G. Engels, and H. J. Kreowski. *Handbook of Graph Grammars and Computing by Graph Transformation: Applications, Languages and Tools*. World Scientific Publishing Company, 1997.

[40] Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors. *Handbook on Graph Grammars and Computing by Graph Transformation*, volume 2: Applications, Languages and Tools. World Scientific, 1999.

[41] Marc Ehrig, Agnes Koschmider, and Andreas Oberweis. Measuring similarity between semantic business process models. In *APCCM 2007*, pages 71–80, Darlinghurst, Australia, 2007. Australian Computer Society, Inc.

[42] W. Emmerich, A. Finkelstein, C. Montangero, S. Antonelli, S. Armitage, and R. Stevens. Managing standards compliance. *Software Engineering, IEEE Transactions on*, 25(6):836–851, 1999.

[43] Jérôme Euzenat and Pavel Shvaiko. *Ontology matching*. Springer-Verlag, Heidelberg (DE), 2007.

[44] Fault Tolerant System Research Group, BME. VIATRA2 Model Transformation Framework. http://www.eclipse.org/gmt/VIATRA2/.

[45] A. Finkelstein and I. Sommerville. The Viewpoints FAQ. *Software Engineering Journal*, 11(1):2–4, January 1996.

[46] B. Fluri, M. Wursch, M. Pinzger, and H.C. Gall. Change Distilling: Tree Differencing for Fine-Grained Source Code Change Extraction. *Software Engineering, IEEE Transactions on*, 33(11):725 –743, nov. 2007.

[47] J.N. Foster, M.B. Greenwald, J.T. Moore, B.C. Pierce, and A. Schmitt. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 29(3):17, 2007.

[48] Christian Gerth, Jochen M. Küster, Markus Luckey, and Gregor Engels. Precise detection of conflicting change operations using process model terms. In *Proceedings of the 13th international conference on Model driven engineering languages and systems: Part II*, MODELS'10, pages 93–107, Berlin, Heidelberg, 2010. Springer-Verlag.

[49] Christian Gerth, Markus Luckey, Jochen M. Küster, and Gregor Engels. Detection of Semantically Equivalent Fragments for Business Process Model Change Management. In *SCC 2010*, pages 57–64, Washington, DC, USA, 2010. IEEE Computer Society.

[50] Christian Gerth, Markus Luckey, Jochen M. Küster, and Gregor Engels. Detection of Semantically Equivalent Fragments for Business Process Model Change Management. Technical Report IBM Research Report RZ 3767, IBM Research, Zurich, Switzerland, 2010. http://www.cs.uni-paderborn.de/uploads/tx_sibibtex/rz3767.pdf.

[51] Phil Gilbert. The next decade of BPM, 2010. Keynote at the 8th International Conference on Business Process Management.

[52] László Gönczy, Ábel Hegedüs, and Dániel Varró. Methodologies for Model-Driven Development and Deployment: an Overview. In M. Wirsing, editor, *Rigorous Software Engineering for Service-Oriented Systems: Results of the SENSORIA project on Software Engineering for Service-Oriented Computing*. Springer-Verlag, 2010.

[53] Esther Guerra and Juan de Lara. Event-driven grammars: Towards the integration of meta-modelling and graph transformation. In *Graph Transformations*, pages 215–218. Springer, 2004. LNCS 3256.

[54] Ábel Hegedüs. A framework for the Dependability analysis of UML-based system designs with maintenance. Master's thesis, Budapest University of Technology and Economics, 2009.

[55] Ábel Hegedüs. A model transformation-based approach for the Dependability analysis of UML-based system designs with maintenance. Technical Report rcl090601, University of Florence, Dip. Sistemi Informatica, RCL group, June 2009.

[56] Ábel Hegedüs. BPEL Verification: The Back-annotation Problem. In *Proceedings of the 17th PhD Minisymposium*, pages 30–31. Budapest University of Technology and Economics, Department of Measurement and Information Systems, 2010.

[57] Ábel Hegedüs. Criteria evaluation-driven state space exploration of graph transformation systems. In *Proceedings of the 18th PhD Minisymposium*, pages 42–45, Budapest, 02/2011 2011. Budapest University of Technology and Economics, Department of Measurement and Information Systems.

[58] Ábel Hegedüs, Ákos Horváth, István Ráth, Moisés Castelo Branco, and Dániel Varró. Quick fix generation for DSMLs. In *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing VLHCC 2011*. IEEE, 2011.

[59] Ábel Hegedüs, Ráth István, and Dániel Varró. From BPEL to SAL and Back: a Tool Demo on Back-Annotation with VIATRA2. Technical report, Consiglio Nazionale delle Ricerche (CNR), 2010. Accepted for the SEFM'2010 "Posters and Tool Demo Session" Track.

[60] Ábel Hegedüs, Ráth István, and Dániel Varró. Back-annotation framework for Simulation Traces of Discrete Event-based Languages. Technical report, BME, April 2010. http://home.mit.bme.hu/~hegedusa/publist.html.

[61] Ábel Hegedüs, István Ráth, and Dániel Varró. Back-annotation of Simulation Traces with Change-Driven Model Transformations. In *Proceedings of the Eigth International Conference on Software Engineering and Formal Methods*, 2010.

[62] Ábel Hegedüs, Zoltán Ujhelyi, Gábor Bergmann, and Ákos Horváth. Ecore to Genmodel case study solution using the Viatra2 framework. In *Transformation Tool Contest 2010*, 2010.

[63] Ábel Hegedüs, Zoltán Ujhelyi, István Ráth, and Ákos Horváth. Visualization of Traceability Models with Domain-specific Layouting. In *Proceedings of the Fourth International Workshop on Graph-Based Tools*, 2010. Accepted.

[64] Ábel Hegedüs and Dániel Varró. Guided state space exploration using back-annotation of occurrence vectors. In *Proceedings of the Fourth International Workshop on Petri Nets and Graph Transformation*, 2010.

[65] Martin Henkel, Jelena Zdravkovic, and Paul Johannesson. Service-based processes: design for business and technology. In *Proceedings of the 2nd international conference on Service oriented computing*, ICSOC '04, pages 21–29, New York, NY, USA, 2004. ACM.

[66] Frank Hermann, Hartmut Ehrig, Fernando Orejas, Krzysztof Czarnecki, Zinovy Diskin, and Yingfei Xiong. Correctness of model synchronization based on triple graph grammars. In *Proceedings of the 14th International Conference on Model Driven Engineering Languages and Systems*, MODELS'11, pages 668–682, Berlin, Heidelberg, 2011. Springer-Verlag.

[67] Anders Hessellund, Krzysztof Czarnecki, and Andrzej Wasowski. Guided development with multiple domain-specific languages. In *In ACM/IEEE 10th International Conference On Model Driven Engineering Languages and Systems (MODELS 2007*, 2007.

[68] Rashina Hoda, Philippe Kruchten, James Noble, and Stuart Marshall. Agility in context. In William R. Cook, Siobhán Clarke, and Martin C. Rinard, editors, *Proceedings of the 25th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2010*, pages 74–88, Reno/Tahoe, Nevada, 2010. ACM.

[69] Martin Hofmann, Benjamin Pierce, and Daniel Wagner. Symmetric lenses. In *Proceedings of the 38th annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '11, pages 371–384, New York, NY, USA, 2011. ACM.

[70] Martin Hofmann, Benjamin C. Pierce, and Daniel Wagner. Edit lenses. In *Proceedings of the 39th annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '12, pages 495–508. ACM, 2012.

[71] Ákos Horváth and Dániel Varró. Dynamic constraint satisfaction problems over models. *Software and Systems Modeling*, 11/2010 2010.

[72] John Hutchinson, Mark Rouncefield, and Jon Whittle. Model-driven engineering practices in industry. In *Proceeding of the 33rd international conference on Software engineering*, ICSE '11, pages 633–642, New York, NY, USA, 2011. ACM.

[73] International Organization for Standardization. *Financial transaction card originated messages – Interchange message specifications – Part 1: Messages, data elements and code values*.

[74] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, 2006.

[75] Mikoláš Janota, Victoria Kuzina, and Andrzej Wasowski. Model construction with external constraints: An interactive journey from semantics to syntax. In *Proceedings of the 11th Int. Conf. on Model Driven Engineering Languages and Systems*, pages 431–445, 2008. LNCS 5301, Springer.

[76] Richard Johnson, David Pearson, and Keshav Pingali. The Program Structure Tree: Computing Control Regions in Linear Time. In *SIGPLAN Conference on Programming Language Design and Implementation*, 1994.

[77] Cem Kaner. Software testing as social science problem, 2006. Canadian Undergraduate Software Engineering Conference, Montreal, Canada.

[78] Jung-Min Kim, Adam Porter, and Gregg Rothermel. An empirical study of regression test application frequency. In *Proceedings of the 22nd international conference on Software engineering*, ICSE '00, pages 126–135, New York, NY, USA, 2000. ACM.

[79] Jana Koehler, Rainer Hauser, Jochen Küster, Ksenia Ryndina, Jussi Vanhatalo, and Michael Wahler. The role of visual modeling and model transformations in business-driven development. *Electron. Notes Theor. Comput. Sci.*, 211:5–15, April 2008.

[80] Jochen Küster. *Consistency Management of Object-oriented Behavioral Models*. PhD thesis, Universität Paderborn, 2004.

[81] Jochen Küster, Hagen Völzer, Cédric Favre, Moisés Castelo Branco, and Krzysztof Czarnecki. Supporting different process views through a shared process model. Technical report, IBM Research Zurich, 2012.

[82] Jochen M. Küster. Towards inconsistency handling of object-oriented behavioral models. *Electronic Notes in Theoretical Computer Science*, 109:57–69, 2004. Proceedings of the Workshop on Graph Transformation and Visual Modelling Techniques (GT-VMT 2004).

[83] Jochen M. Küster, Christian Gerth, Alexander Förster, and Gregor Engels. Detecting and resolving process model differences in the absence of a change log. In *Proceedings of the 6th International Conference on Business Process Management*, BPM '08, pages 244–260, Berlin, Heidelberg, 2008. Springer-Verlag.

[84] Jochen Malte Küster and Ksenia Ryndina. Improving inconsistency resolution with side-effect evaluation and costs. In *MoDELS*, pages 136–150, 2007.

[85] Jochen Malte Küster, Hagen Völzer, Cédric Favre, Moisés Castelo Branco, and Krzysztof Czarnecki. Supporting different process views through a shared process model. In *9th*

European Conference on Modelling Foundations and Applications, ECMFA, pages 20–36, 2013.

[86] Jochen Malte Küster, Hagen Völzer, Cédric Favre, Moisés Castelo Branco, and Krzysztof Czarnecki. Supporting different process views through a shared process model. In *Journal of Software and Systems Modeling, SoSyM*, To appear.

[87] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady.*, 10(8):707–710, February 1966.

[88] Chen Li, Manfred Reichert, and Andreas Wombacher. On measuring process model similarity based on high-level change operations. In *Proceedings of the 27th International Conference on Conceptual Modeling*, ER '08, pages 248–264, Berlin, Heidelberg, 2008. Springer-Verlag.

[89] Jerry Luftman, Raymond Papp, and Tom Brier. Enablers and inhibitors of business-IT alignment. *Commun. AIS*, 1, March 1999.

[90] Andrian Marcus and Jonathan I. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pages 125–135, Washington, DC, USA, 2003. IEEE Computer Society.

[91] S. Mazanek, S. Maier, and M. Minas. Auto-completion for diagram editors based on graph grammars. In *Visual Languages and Human-Centric Computing, 2008. VL/HCC 2008.* IEEE, 2008.

[92] Steffen Mazanek and Mark Minas. Business process models as a showcase for syntax-based assistance in diagram editors. In *Model Driven Engineering Languages and Systems*. Springer, 2009. LNCS 5795.

[93] Steffen Mazanek and Mark Minas. Generating correctness-preserving editing operations for diagram editors. In *Proc. of the 8th Int. Workshop on Graph Transformation and Visual Modeling Techniques. ECEASST*, volume 18, 2009.

[94] Tom Mens, Ragnhild Van Der Straeten, and Maja DHondt. Detecting and resolving model inconsistencies using transformation dependency analysis. In *Proc. of Model Driven Engineering Languages and Systems*, pages 200–214. Springer, 2006. LNCS 4199.

[95] Derek Miers and Stephen A. White. *BPMN Modeling and Reference Guide Understanding and Using BPMN*. Future Strategies Inc., Lighthouse, Pt, FL, USA, 2008.

[96] Tilak Mitra. Business-driven development, 2005. IBM developerWorks article.

[97] Pierre-Alain Muller, Franck Fleurey, and Jean-Marc Jézéquel. Weaving executability into object-oriented meta-languages. In *Model Driven Engineering Languages and Systems*. Springer, 2005. LNCS 3713.

[98] C. Nentwich, W. Emmerich, and A. Finkelstein. Consistency management with repair actions. In *Software Engineering, 2003. Proceedings. 25th International Conference on*, pages 455 – 464, 2003.

[99] A. Nöhrer and A. Egyed. Utilizing the Relationships Between Inconsistencies for more Effective Inconsistency Resolution. In *Proceedings of the 3rd Workshop on Living with Inconsistencies in Software Development*, pages 39–43. CEUR Workshop Proceedings, 2010.

[100] Bashar Nuseibeh, Steve Easterbrook, and Alessandra Russo. Leveraging inconsistency in software development. In *33(4):24-29, IEEE Computer*, pages 1–33. Society Press, 2000.

[101] OASIS. Web Services Business Process Execution Language (WSBPEL) TC. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel.

[102] Object Management Group. Business Process Model and Notation (BPMN) Version 2.0. http://www.omg.org/spec/BPMN/2.0/.

[103] Object Management Group. Object Constraint Language (OCL). http://www.omg.org/spec/OCL/.

[104] Fernando Orejas, Hartmut Ehrig, and Ulrike Prange. A logic of graph constraints. In *Fundamental Approaches to Software Engineering (FASE)*, pages 179–198. Springer, 2008. LNCS 4961.

[105] J. Pinna Puissant, T. Mens, and R. Van Der Straeten. Resolving Model Inconsistencies with Automated Planning. In *Proceedings of the 3rd Workshop on Living with Inconsistencies in Software Development*, pages 8–14. CEUR Workshop Proceedings, 2010.

[106] Artem Polyvyanyy, Jussi Vanhatalo, and Hagen Völzer. Simplified computation and generalization of the refined process structure tree. In *Proceedings of the 7th international conference on Web services and formal methods*, WS-FM'10, pages 25–41, Berlin, Heidelberg, 2011. Springer-Verlag.

[107] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10:334–350, December 2001.

[108] Colette Rolland and Naveen Prakash. Bridging the gap between organisational needs and ERP functionality. *Requirements Engineering*, 5:180–193, 2000. 10.1007/PL00010350.

[109] Michael Rosemann. Gartner fellows interview: Dr. Michael Rosemann on BPM in Australia, 2011. http://www.gartner.com/DisplayDocument?id=1840717.

[110] Sagar Sen, Benoit Baudry, and Hans Vangheluwe. Towards domain-specific model editors with automatic model completion. *Simulation*, pages 109–126, 2010. 86(2).

[111] SOA Tools Platform. Eclipse BPMN Modeler. http://eclipse.org/projects/project.php?id=soa.bpmnmodeler.

[112] Pnina Soffer. Refinement equivalence in model-based reuse: Overcoming differences in abstraction level. *J. Database Manag.*, 16(3):21–39, 2005.

[113] George Spanoudakis and Andrea Zisman. Inconsistency management in software engineering: Survey and open research issues. In *in Handbook of Software Engineering and Knowledge Engineering*, pages 329–380. World Scientific, 2001.

[114] Perdita Stevens. Bidirectional model transformations in qvt: Semantic issues and open questions. In *In International Conference on Model Driven Engineering Languages and Systems (MoDELS*, pages 1–15. Springer-Verlag, 2007.

[115] Anselm L Strauss and J Corbin. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*, volume 2nd. Sage Publications, Inc., 1998.

[116] The Eclipse Project. Eclipse Modeling Framework Project. http://www.eclipse.org/emf.

[117] The Eclipse Project. EMF Validation Framework. http://www.eclipse.org/modeling/emf/?project=validation.

[118] Oliver Thomas, Katrina Leyking, and Florian Dreifus. Using process models for the design of service-oriented architectures: Methodology and E-Commerce case study. In *Proceedings of the Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, HICSS '08, page 109, Washington, DC, USA, 2008. IEEE Computer Society.

[119] Juha-Pekka Tolvanen and Matti Rossi. MetaEdit+: defining and using domain-specific modeling languages and code generators. In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, OOPSLA '03, pages 92–93, New York, NY, USA, 2003. ACM.

[120] Huy Tran, Uwe Zdun, and Schahram Dustdar. View-based integration of process-driven SOA models at various abstraction levels. In *Proceedings of First International Workshop on Model-Based Software and Data Integration MBSDI 2008*, pages 55–66. Springer, 2008.

[121] Christoph Treude and Margaret-Anne D. Storey. Effective communication of software development knowledge through community portals. In *SIGSOFT FSE*, pages 91–101, 2011.

[122] Boudewijn van Dongen, Remco Dijkman, and Jan Mendling. Measuring Similarity between Business Process Models. In Zohra Bellahsène and Michel Léonard, editors, *Proceedings of the 20th Int'l Conference on Advanced Information Systems Engineering (CAiSE 2008)*, volume 5074 of *Lecture Notes in Computer Science*, pages 450–464, Montpellier, France, 2008. Springer Verlag.

[123] Jussi Vanhatalo, Hagen Völzer, and Jana Koehler. The Refined Process Structure Tree. In *Proceedings of the 6th International Conference on Business Process Management*, BPM '08, pages 100–115, Berlin, Heidelberg, 2008. Springer-Verlag.

[124] Jussi Vanhatalo, Hagen Völzer, and Frank Leymann. Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition. In *ICSOC 2007*, LNCS, pages 43–55, Berlin, Heidelberg, 2007. Springer-Verlag.

[125] Dániel Varró. Model transformation by example. In *Model Driven Engineering Languages and Systems (MODELS'06)*, pages 410–424, Genova, Italy, 2006. LNCS 4199, Springer.

[126] Dániel Varró and András Pataricza. VPM: A visual, precise and multilevel metamodeling framework for describing mathematical domains and UML. *Software and Systems Modeling*, 2(3), 2003.

[127] Alain Wegmann, Anders Kotsalainen, Lionel Matthey, Gil Regev, and Alain Giannattasio. Augmenting the Zachman Enterprise Architecture Framework with a systemic conceptualization. In *Proceedings of the 2008 12th International IEEE Enterprise Distributed Object Computing Conference*, pages 3–13, Washington, DC, USA, 2008. IEEE Computer Society.

[128] Matthias Weidlich, Alistair P. Barros, Jan Mendling, and Mathias Weske. Vertical alignment of process models - how can we get there? In *CAiSE 2009 Workshop Proceedings: BPMDS*, pages 71–84, 2009.

[129] Matthias Weidlich, Gero Decker, Mathias Weske, and Alistair Barros. Towards vertical alignment of process models - a collection of mismatches. Technical report, Hasso Plattner Institute, 2008.

[130] Matthias Weidlich, Remco Dijkman, and Jan Mendling. The ICoP framework: identification of correspondences between process models. In *Proceedings of the 22nd international conference on Advanced information systems engineering*, CAiSE'10, pages 483–498, Berlin, Heidelberg, 2010. Springer-Verlag.

[131] Matthias Weidlich, Remco Dijkman, and Mathias Weske. Deciding behaviour compatibility of complex correspondences between process models. In *Proceedings of the 8th international conference on Business process management*, BPM'10, pages 78–94, Berlin, Heidelberg, 2010. Springer-Verlag.

[132] Matthias Weidlich, Jan Mendling, and Mathias Weske. Efficient consistency measurement based on behavioural profiles of process models. *IEEE Transactions on Software Engineering*, 99(PrePrints), 2010.

[133] Matthias Weidlich, Eitam Sheetrit, Moisés Castelo Branco, and Avigdor Gal. Matching business process models using positional passage-based language models. In *Conceptual Modeling*, volume 8217 of *Lecture Notes in Computer Science*, pages 130–137. Springer Berlin Heidelberg, 2013.

[134] Monika Weidmann, Modood Alvi, Falko Koetter, Frank Leymann, Thomas Renner, and David Schumm. Business Process Change Management based on Process Model Synchronization of Multiple Abstraction Levels. In *Proceedings of SOCA 2011*. IEEE Computer Society, 2011.

[135] Dirk Werth, Katrina Leyking, Florian Dreifus, Jörg Ziemann, and Andreas Martin. Managing SOA through business services: a business-oriented approach to service-oriented architectures. In *Proceedings of the 4th international conference on Service-oriented computing*, ICSOC'06, pages 3–13, Berlin, Heidelberg, 2007. Springer-Verlag.

[136] Zhenchang Xing. Model comparison with GenericDiff. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, ASE '10, pages 135–138, New York, NY, USA, 2010. ACM.

[137] Zhenchang Xing and Eleni Stroulia. Umldiff: an algorithm for object-oriented design differencing. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, ASE '05, pages 54–65, New York, NY, USA, 2005. ACM.

[138] Michael Zapf and Armin Heinzl. Evaluation of generic process design patterns: An experimental study. In *Business Process Management, Models, Techniques, and Empirical Studies*, pages 83–98, London, UK, UK, 2000. Springer-Verlag.

[139] Loucif Zerguini. A novel hierarchical method for decomposition and design of workflow models. *J. Integr. Des. Process Sci.*, 8:65–74, April 2004.