ulm university universität ulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Datenbanken und Informationssysteme

Abstraction, Visualization, and Evolution of Process Models

Dissertation zur Erlangung des Doktorgrades Dr. rer. nat.
der Fakultät für Ingenieurwissenschaften, Informatik und Psychologie der Universität Ulm

Vorgelegt von:
Jens Kolb aus Kirchheim/Teck

2015

Amtierende Dekanin: Prof. Dr. Tina Seufert
Gutachter: Prof. Dr. Manfred Reichert
Prof. Dr. Michael Weber
Tag der Promotion: 29. Juli 2015

Abstract

The increasing adoption of process orientation in companies and organizations has resulted in large process model collections. Each process model of such a collection may comprise dozens or hundreds of elements and captures various perspectives of a business process, i.e., organizational, functional, control, resource, or data perspective. Domain experts having only limited process modeling knowledge, however, hardly comprehend such large and complex process models. Therefore, they demand for a customized (i.e., personalized) view on business processes enabling them to optimize and evolve process models effectively.

This thesis contributes the *proView* framework to systematically create and update *process views* (i.e., abstractions) on process models and business processes respectively. More precisely, process views abstract large process models by hiding or combining process information. As a result, they provide an abstracted, but personalized representation of process information to domain experts. In particular, updates of a process view are supported, which are then propagated to the related process model as well as associated process views. Thereby, up-to-dateness and consistency of all process views defined on any process model can be always ensured. Finally, *proView* preserves the behaviour and correctness of a process model.

Process abstractions realized by views are still not sufficient to assist domain experts in comprehending and evolving process models. Thus, additional process visualizations are introduced that provide text-based, form-based, and hierarchical representations of process models. Particularly, these process visualizations allow for view-based process abstractions and updates as well. Finally, process interaction concepts are introduced enabling domain experts to create and evolve process models on touch-enabled devices. This facilitates the documentation of process models in workshops or while interviewing process participants at their workplace.

Altogether, *proView* enables domain experts to interact with large and complex process models as well as to evolve them over time, based on process model abstractions, additional process visualizations, and process interaction concepts. The framework is implemented in a proof-of-concept prototype and validated through experiments and case studies.

Parts of this thesis have been published in the following publications.

- Kolb, J., Rudner, B., Reichert, M.: Towards Gesture-based Process Modeling on Multi-Touch Devices. In: Proc. 1st Int'l Workshop on Human-Centric Process-Aware Information Systems (HC-PAIS'12), Gdansk, Poland (2012) 280–293

- Kolb, J., Huebner, P., Reichert, M.: Model-Driven User Interface Generation and Adaptation in Process-Aware Information Systems. UIB 2012-04, Technical Report, Ulm University (2012)

- Kolb, J., Reichert, M., Weber, B.: Using Concurrent Task Trees for Stakeholder-centered Modeling and Visualization of Business Processes. In: Proc. 4th Int'l Conf. S-BPM ONE 2012, CCIS 284. (2012) 237–251

- Reichert, M., Kolb, J., Bobrik, R., Bauer, T.: Enabling Personalized Visualization of Large Business Processes through Parameterizable Views. In: Proc. 26th Symposium On Applied Computing (SAC'12), Riva del Garda (Trento), Italy, ACM (2012) 1653–1660

- Kolb, J., Kammerer, K., Reichert, M.: Updatable Process Views for User-centered Adaption of Large Process Models. In: Proc. 10th Int'l Conf. on Service Oriented Computing (ICSOC'12), Shanghai, China (2012) 484–498

- Kolb, J., Kammerer, K., Reichert, M.: Updatable Process Views for Adapting Large Process Models: The proView Demonstrator. In: Proc. 10th Int'l Conf. on Business Process Management (BPM'12), Demonstration Track, Tallinn, Estonia (2012) 6–11

- Kolb, J., Hübner, P., Reichert, M.: Automatically Generating and Updating User Interface Components in Process-Aware Information Systems. In: Proc. 10th Int'l Conf. on Cooperative Information Systems (CoopIS'12). (2012) 444–454

- Kolb, J., Reichert, M.: Data Flow Abstractions and Adaptations through Updatable Process Views. In: Proc. 27th Symposium on Applied Computing (SAC'13), Coimbra, Portugal, ACM (2013) 1447–1453

- Kolb, J., Reichert, M.: Supporting Business and IT through Updatable Process Views: The proView Demonstrator. In: Proc. 10th Int'l Conference on Service Oriented Computing (ICSOC'12), Demonstration Track, Shanghai, China (2013) 460–464

- Kolb, J., Reichert, M.: A Flexible Approach for Abstracting and Personalizing Large Business Process Models. ACM Applied Computing Review **13**(1) (2013) 6–17

- Lanz, A., Kolb, J., Reichert, M.: Enabling Personalized Process Schedules with Time-aware Process Views. In: Proc. 25th CAiSE 2013 Workshops, 2nd Int'l Workshop on Human-Centric Information Systems (HCIS 2013), Valencia, Spain (2013) 205–216

- Kolb, J., Leopold, H., Mendling, J., Reichert, M.: Creating and Updating Personalized and Verbalized Business Process Descriptions. In: Proc. 6th IFIP WG 8.1 Working Conference on the Practice of Enterprise Modeling (PoEM'13), Riga, Latvia (2013) 191–205

- Kolb, J., Rudner, B., Reichert, M.: Gesture-based Process Modeling Using Multi-Touch Devices. Int'l Journal of Information System Modeling and Design (IJISMD) **4**(4) (2013) 48–69

- Kolb, J., Zimoch, M., Weber, B., Reichert, M.: How Social Distance of Process Designers Affects the Process of Process Modeling: Insights From a Controlled Experiment. In: Proc. 29th Symposium on Applied Computing (SAC'14), Gyeongju, Korea, ACM (2014) 1364–1370

Contents

Part I

Problem Description and Backgrounds

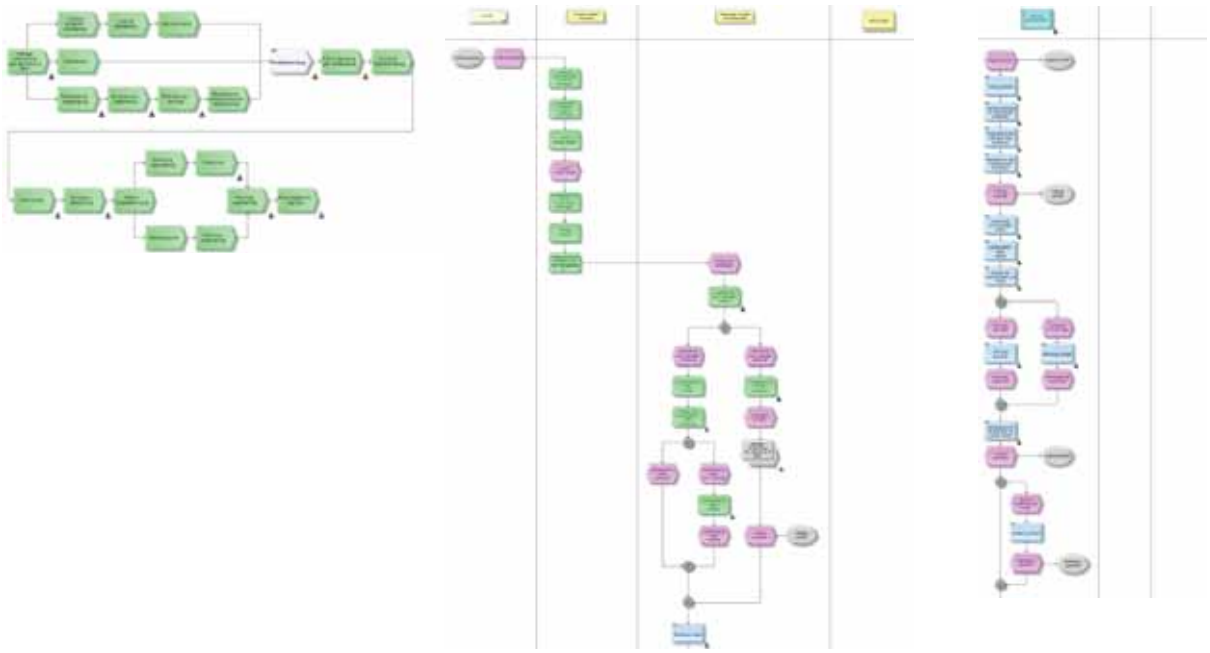
1 Introduction

The adoption of *Process-aware Information Systems (PAISs)* in companies has resulted in a large number of business processes involving various organizational units, domain experts, business partners, and customers. A PAIS provides support for business processes at an operational level [15]. Thereby, it strictly separates process logic from application code, relying on explicit *process models*, e.g., graphical representations of real-world business processes. This enables a separation of concerns, which is a well-established principle in computer science to increase maintainability and to reduce costs of change [16]. Particularly, the practice of documenting business processes, i.e., *process modeling*, is ranked fourth when it comes to what conceptual modeling in companies is used for [17]. As a result, large process model collections have emerged in companies for several years. Each process model describes a business process and captures various perspectives of the latter, i.e., organizational, functional, control, resource, or data perspective. Furthermore, a process model may consist of dozens or hundreds of activities [18].

As has been shown, large process models tend to contain a multitude of errors [19], which, in turn, have a negative impact on *Business Process Management (BPM)* success factors [20]. Furthermore, organizational units or domain experts may have their own perspective on business processes, which need to be reflected through specific process models. To cope with these requirements, in current practice, a large number of process models of varying granularity have emerged, which provide different perspectives on the business process [21, 22]. While managers are more likely to prefer an abstract process overview, process participants need a detailed view on those process parts they are involved in. For example, Figure 1.1 shows three different perspectives of an *order-to-cash* process[1]. Figure 1.1a depicts coarse-grained steps of the process and, hence, provides an abstract view on it. In turn, Figure 1.1b provides a different perspective on the same business process, assigning individual process activities to concrete organizational units. Finally, Figure 1.1c depicts a perspective referring to involved information systems as well.

Another driver for the presence of multiple process models representing specific perspectives on one and the same business process are the different use cases for process modeling, e.g.,

[1] Originates from United Motor Group Scenario of IDS Scheer ARIS Business Architect 7.2



a) Overview

b) Responsibilities of Organizational Units (simplified)

c) Involvement of Information Systems (simplified)

Figure 1.1: Perspectives on Order-to-Cash Process

process documentation or automation. For example, process automation necessitates technical process models capturing implementation details (e.g., fetching data from a database) as well. Furthermore, technical process models are intended to be executed in a PAIS (cf. Figure 1.1c). By contrast, process models created for documentation purpose are used to closely reflect real-world business processes (cf. Figure 1.1a+b) [23, 24, 25]. Although various process models refer to the same business process, in general, the models may comprise disjoint sets of activities. For example, a business process model describes that a truck drives from factory A to factory B. By contrast, a technical process model documents the invocation of a database to fetch order details. Furthermore, variants of process models describing the same business process may exist, e.g., in regards to different products or countries [26, 27]. For example, multiple process models separately document the *order-to-cash* process for different countries. Merging these process variants into one single process model, in turn, would result in a large and complex process model as well [26, 28, 29].

Creating process models of different granularity and maintaining them separately requires large efforts from process designers. Besides this, domain experts (e.g., process participants, process owners, business analysts, software engineers) lack an understanding of large and complex process models [30]. Therefore, personalized information on processes is a much needed feature for them providing individual and customized perspectives on business processes. To be more precise, complex process models need to be abstracted to the specific needs of domain experts, i.e., personalized process abstractions are required [31, 32, 33]. In particular, studies have shown that the involvement of domain experts constitutes a critical factor for successful process modeling initiatives [20, 34].

Personalized process abstractions are not sufficient to fully meet the requirements of domain experts. Since the latter usually have limited process modeling knowledge and are unfamiliar with process model languages, process models shall be visualized in a way that enables domain experts to understand them without high training efforts. In this context, studies have shown that 24% of the domain experts in process modeling initiatives *never* and 52% have been *occasionally* trained to analyze, design, and manage process models [35, 36]. Therefore, companies offer, in addition to graph-based *process representations* like *Business Process Model and Notation (BPMN)*, other kinds of model representations. For example, these include textual descriptions in the context of the ISO 9000 quality management certification or process landscapes for managers. In particular, the communication of process information to and from domain experts is identified as a crucial factor within BPM [17].

Business processes are subject to continuous change due to evolving markets, organizational optimizations, or legal regulations [37, 38]. In turn, this needs to be mirrored by evolving process models over time [39, 40]. According to Gartner, changing process models is of high relevance to keep them synchronized with business processes enabling continuous process improvements and optimizations [41]. Studies have shown that European companies spend between 10% and 55% of their average profit margin on business process changes [42]. Nowadays, all process models related to a particular business process must be manually adapted to reflect business process changes. However, due to time pressure not all process models are adapted or the changes are not accomplished consistently. Consequently, process models get outdated and, hence, do not represent business processes in an appropriate way anymore.

The process modeling component of a PAIS (*process modeling tool* for short) should not only enable process model abstractions, but also guarantee that they are kept up-to-date in the context of process optimization and changes. Usually, the latter should not always be introduced by process designers, but may be introduced by domain experts referring to their process abstractions as well. In particular, the involvement of domain experts in changing business processes is significantly correlated with a successful outcome of BPM projects [43]. To guarantee up-to-dateness of all abstractions defined on a given process model, changes must be automatically propagated to all other process abstractions defined on the same business process. Note that the change of business processes based on process abstractions contributes better process model quality as well [44].

To change process models, domain experts must interact with a process modeling tool. In order to identify changes and optimizations in business processes, interviews and workshops are conducted [45]. However, process modeling tools have, typically, been designed for desktop computers or notebooks [46] and do not adequately support such interviews and workshops. As a result, changes are annotated on process models, which are printed on paper sheets or described textually [47], i.e., additional effort is required to transfer the changes to a process modeling tool. Figure 1.2, for example, shows a whiteboard used by domain experts to discuss and evolve process models by utilizing sticky notes. However, to support domain experts in such situations, a process modeling tool should provide process interaction support on touch-enabled devices (e.g., tablets, touch tables) [46]. Particularly, the latter have become widely used in companies [46, 48].

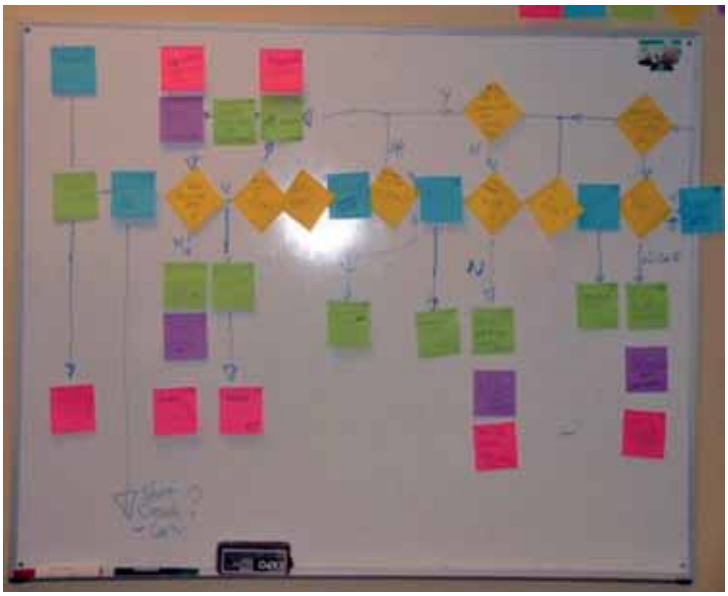


Figure 1.2: Collaborative Process Modeling on a Whiteboard

1.1 Research Contribution

This thesis aims to provide techniques for abstracting and visualizing large and complex business process models, which may involve multiple partners, organizational units, and user groups. The thesis contributes generic techniques, concepts, and algorithms for creating, abstracting, and changing process models as well as for interacting with them. Note that the thesis does not target at a process modeling tool relying on a specific process modeling language.
The main research contributions can be summarized as follows:

1. We provide concepts to abstract complex process models according to the specific needs of domain experts through so-called *process views*. In particular, the presented process view concept reduces process model complexity. More precisely, elements of process models may be *reduced* in a process view to hide process information not relevant in the current context. Alternatively, process elements may be *aggregated* to abstract from detailed process information. In this context, well-defined view creation operations are introduced that may be combined to provide semantically-enriched abstractions. Note that view creation considers various perspectives on a process model, e.g., control and data flow.

2. A complete set of operations that allow for view-based updates of a process model is presented. To be more precise, domain experts may evolve a process model based on the personalized views they have on this process model. Process view updates are automatically propagated to the process model the view is based on. In turn, this process model is then adapted accordingly. Furthermore, other process views associated with the same process model are migrated to the new model version to immediately reflect the process change if applicable. In particular, this allows guaranteeing that all process views are up-to-date and outdated process models are prevented. Note that the provided update operations ensure correctness of the resulting process model.

3. We provide three alternative process representations that focus on domain experts with limited modeling experience. Furthermore, process model updates may be accomplished based on the provided representations as well, i.e., the developed concepts are orthogonal to each other. This enables domain experts to evolve and optimize process models and process views respectively. In the context of this thesis, form-, tree-, and text-based representations are provided. The form-based representation displays a process model in terms of nested rectangles. The tree-based representation, in turn, enables interaction with process models in a top-down manner. Finally, the text-based representation describes a business process in a verbalized way while not using any graphical elements.

4. We provide a multi-touch gesture set to create and evolve process models and process views on touch-enabled devices (e.g., tablets, touch tables). This gesture set results from empirical evaluations.

Altogether, the results presented in this thesis aim to enable domain experts to effortlessly create and update process models as well as to interact with them based on process views, process representations, and process interaction concepts.

1.2 Outline

The remainder of this thesis is structured as follows:

Part I discusses the challenges that emerge when interacting with process models to create, abstract, and update them. Chapter 2 introduces the research methodology applied in the context of the thesis. Chapter 3 presents the requirements to be met. Chapter 4 discusses state-of-the-art process modeling tools. Finally, Chapter 5 gives an overview on the framework developed in this thesis. Furthermore, it discusses general design decisions made.

Part II introduces the concepts of updatable process views. Chapter 6 presents view creation operations to construct process views, including a detailed description on how flexible process hierarchies may be created based on process views. Chapter 7 deals with view update operations showing how their effects may be propagated to the underlying process model and associated process views to keep them up-to-date. Chapter 8 presents advanced operations to create and update process views providing more convenience for domain experts.

Part III introduces process representation and interaction concepts supporting users to understand and evolve process models. Chapter 9 presents process representations that aim to reduce the effort required from domain experts to comprehend and evolve process models and process views respectively. Chapter 10 introduces process interaction concepts for touch-enabled devices.

Part IV evaluates developed concepts and algorithms. Chapter 11 introduces a proof-of-concept prototype, demonstrating the technical feasibility of the developed concepts. Chapter 12 presents a case study based on real-world process models to show the practical relevance of our contribution. Experiments are presented to evaluate process representations and interaction concepts.

Part V concludes the thesis. Chapter 13 summarizes the developed concepts and gives an outlook on future work.

2 Research Method

The research of this thesis makes use of well-defined research methods that allow for generic solutions [49]. In this context, Section 2.1 introduces the research questions addressed by this thesis. Section 2.2 presents the research framework we applied to answer these research questions.

2.1 Research Questions

The research questions addressed by this thesis are based on the initial problem statement that contemporary process modeling tools do not provide proper support to all domain experts [50]. When dealing with large process models in particular, no dedicated support for domain experts who do not have a process modeling background exists. Consequently, these users are also not able to adapt these complex models to changing situations [44, 51, 52]. In particular, large process models are often too detailed for domain experts only interested in specific process parts or abstract overviews. Research question RQ1 addresses this issue.

Research Question RQ1
Which abstractions of large and complex process models are useful to increase model comprehensibility and, thereby, to enable domain experts to evolve the process models over time?

Typically, process models are visualized using a specific process modeling language (e.g., BPMN 2.0). However, as known from related research, this rigid representation of process models is too restrictive and does not meet the visualization requirements of more advanced scenarios (cf. Research Question RQ2) [50, 52, 53]. Domain experts that are unfamiliar with process modeling need to know which steps in a business process have to be performed in an easily understandable way [51, 54, 55].

Research Question RQ2
Which representations of process models are required by domain experts unfamiliar with business process modeling?

Contemporary process modeling tools restrict users to interact with them using a desktop computer or a notebook [56, 57]. They are not customized for usage on touch-enabled devices. Due to the increasing adoption of such devices, however, both process representation and interaction should be feasible on these devices as well (cf. Research Question RQ3), e.g., to record or optimize business processes while interviewing process participants "on-site".

Research Question RQ3
Which process interaction concepts are required to interact with process models using touch-enabled devices?

2.2 Research Framework

Information System (IS) research may be based on two complementary paradigms: *behavioural science* and *design science* [58, 59, 60, 61, 62]. *Behavioural science* aims to develop and evaluate theories explaining or predicting organizational and human phenomena [63]. In turn, *design science* originates from the engineering domain and constitutes a problem-solving paradigm. Furthermore, design science aims at building innovative IT artifacts (i.e., constructs, models, methods, or instantiations) as well as evaluating them [64].

This thesis applies *design science* to build a novel and innovative IT artifact. Furthermore, it utilizes evaluation methods known from behavioural science, i.e., empirical analysis. Generally, in design science a differentiation between *routine* and *scientific design* is made [49]. *Routine design* is performed by applying existing knowledge and best practices (e.g., to develop an ERP[1] system based on a particular software development method). In contrast, *scientific design* solves current issues in an innovative way and contributes new knowledge to the common knowledge base. In this thesis, we contribute concepts to create, visualize, and interact with complex business process models, which foster both the understanding and evolution of complex process models by domain experts, i.e., *scientific design* is applied.

Figure 2.1 shows the research framework applied in this thesis and presents major steps of our research [49, 65]. *Requirements* are identified from the *environment* taking the described research questions into account (cf. Figure 2.1, Step 1a). In particular, the roles, skills, and objectives of domain experts are relevant to extract these requirements. Furthermore, organizations as well as the technologies they apply must be taken into account by our research. Concepts developed in this thesis are evaluated against the environment by proof-of-concept prototypes, case studies, and empirical evaluations (cf. Figure 2.1, Step 4a).

Regarding research questions RQ1-RQ3, the existing *knowledge base* comprises knowledge from fields like business process management, process model abstractions, process change, process representation, and process interaction (cf. Figure 2.1, Step 1b). Moreover, the research results of this thesis are transferred to the knowledge base by publications, business forums, and exhibitions as well as industry projects (cf. Figure 2.1, Step 4b).

Figure 2.1 further indicates, which chapter addresses the various parts of the research framework.

[1] Enterprise Resource Planning

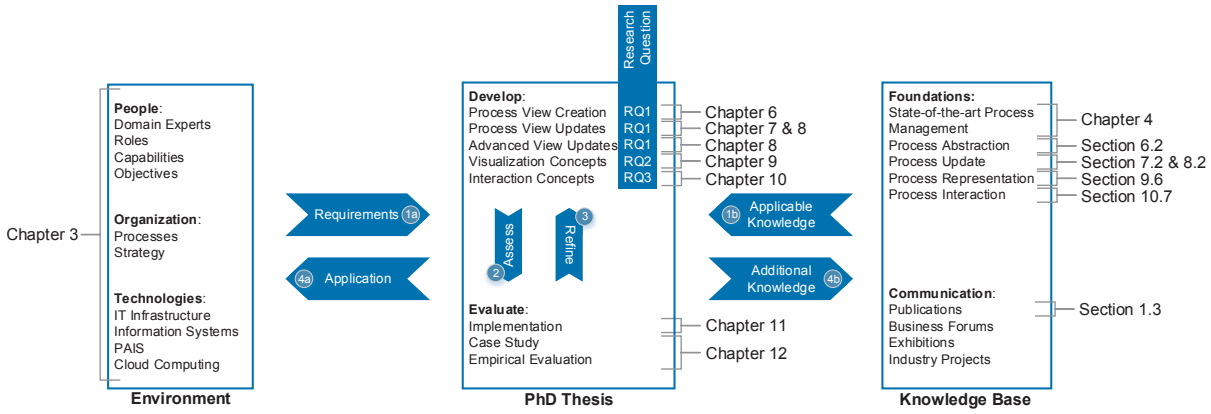


Figure 2.1: Research Framework

We discuss the research framework and the research contribution of this thesis based on the seven design science research guidelines suggested in [49]. In particular, this shall prove that the research framework meets established research standards. Note that the seven guidelines assist researchers from the information system field in better understanding and assessing design science research.

- *Guideline 1 (Design as an Artifact).* Design science aims to create an artifact in terms of a construct, model, method, or instantiation [49]. This thesis deals with process views (i.e., process model abstractions) as *construct* and provides *methods* for creating, maintaining, and updating respective process views as well as for interacting with them. For all methods, an *instantiation* is provided by a proof-of-concept prototype to demonstrate their feasibility in terms of the defined research questions. Furthermore, the evaluation of the developed *constructs* and *methods* is accomplished through case studies and empirical evaluations (cf. Figure 2.1, Step 2).

- *Guideline 2 (Problem Relevance).* The relevance of conducted design science research shall be assessed towards the *community* [49]. Regarding BPM research, in particular, the *community* consists of domain experts who create, evolve, implement, execute, optimize, and monitor process models that represent real-world business processes (cf. Figure 2.1, Step 1a). We address relevant problems of this community that have resulted from the increasing adoption of process-orientation [66], which has led to process model repositories comprising large numbers of process models [67]. Any process modeling tool, therefore, must support domain experts in creating, modeling, and evolving process models.

- *Guideline 3 (Design Evaluation).* Resulting artifacts must be evaluated based on well-established methods [49]. This thesis uses descriptive, observational, and experimental evaluation methods (cf. Figure 2.1, Step 2). In particular, descriptive and observational evaluation methods are applied for evaluating process model abstractions and the handling of process model updates on them. Regarding the representation of and the interaction with (abstracted) process models experimental methods are utilized. Note that evaluation results, in turn, are applied to optimize research contributions (cf. Figure 2.1, Step 3).

- *Guideline 4 (Research Contribution).* Design science research must provide a contribution to the existing knowledge base [49]. This thesis contributes concepts and algorithms to abstract and visualize process models. In particular, the concepts not only foster process model comprehensibility, but allow evolving business processes by changing their abstracted models in order to enable domain experts to understand and update them. Furthermore, process interaction concepts are introduced, which support domain experts interacting with process models on touch-enabled devices. This research contribution is communicated to the environment (cf. Figure 2.1, Step 4a) and transferred to the knowledge base through scientific publications (cf. Figure 2.1, Step 4b).

- *Guideline 5 (Research Rigor).* Generally, design science research needs to be performed in a rigorous way [49]. This thesis, therefore, refers to existing literature. Furthermore, it builds on the existing knowledge base and artifacts (cf. Figure 2.1, Step 1b). In particular, we make use of prior knowledge about the modeling, visualization, abstraction, evolution of, and interaction with process models.

- *Guideline 6 (Design as a Search Process).* In order to create both a *satisfying* as well as *effective* solution in respect to the research questions, design science constitutes an iterative process (cf. Figure 2.1, Steps 2+3). To be more precise, a solution is required, which addresses research questions and resulting requirements in the best possible way [68]. The main issue for information system research is an infinite solution space due to the size and complexity of research problems [49]. Since it is not possible to specify all feasible solutions, a solution adequately addressing our research questions has to be determined. Furthermore, related approaches dealing with process representation, abstraction, and evolution as well as process interaction have to be evaluated and differences to our contribution have to be discussed.

- *Guideline 7 (Communication of Research).* The developed results must be communicated to researchers as well as practitioners. Both groups should benefit from the research results. Furthermore, the contribution of these results to the existing knowledge base is ensured through scientific publications.

All design science research guidelines are discussed in relation to this thesis and, as a result, illustrate that the presented research framework meets the research standards established in the context of design science. Finally, this thesis contributes to the existing knowledge base of information system research or—to be more precise—PAIS research.

3 Requirements Analysis

This chapter elicits the requirements related to the research questions defined in Chapter 2. In this context, case studies are conducted to gain insights into real-world business processes (cf. Section 3.1). Note that this is essential to be able to develop a proper solution in design science (cf. Chapter 2). Referring to these real-world processes, Section 3.2 derives the requirements to be met. Section 3.3 concludes with a summary.

3.1 Case Studies

In the following we introduce two case studies. Their goal is to elicit major requirements related to the research questions described in Section 2.1. To meet confidentiality restrictions, certain details of the case studies are anonymized, e.g., activity labels are changed. Nonetheless, the case studies reflect real-world issues in business process modeling.

3.1.1 Case Study CS1: Order Processing in Mechanical Engineering

Case study CS1 is conducted in a medium-sized company producing customized machines. In particular, we consider the *order-to-cash* process of this company. In the following, we describe the company's situation as well as process models we encounter at project start.

The *order-to-cash* process constitutes of activities starting with the issuing of an order and completing with the delivery of the product and, optionally, its first maintenance. Order processing comprises activities to check back with the customer, the construction of electrical and mechanical components, production, packaging, and delivery of the product. Thereby, every product features a high complexity and needs to be customized according to the customer's demands. Figure 3.1 provides a coarse-grained overview of the *order-to-cash* process as it is perceived by the upper management of the company. For its representation, *Value-Added Chain Diagrams (VCDs)* are used [21]. As can be seen, this process model abstracts from user roles as well as any details regarding the course of action.

Figure 3.1: Coarse-grained Overview on Order-to-Cash Process

In addition to the process model shown in Figure 3.1, six more detailed process models exist, which comprise 70 activities in total. For representing the process models, BPMN is used. Figure 3.2 shows one of the process models, describing the pre-audit of product requirements. Note that this process is part of the *Order Handling* activity in Figure 3.1. Furthermore, the customer is represented by the *collapsed pool* on the top, whereas the company itself is represented by the rectangle on the bottom. The company's organizational structure, in turn, is detailed through five user *lanes* corresponding to roles, i.e., *project manager*, *project planner*, *mechanics*, *core component designer*, and *electronics designer*. To be more precise, each lane contains a process fragment describing the activities each user role has to perform. The project manager (i.e., topmost lane) starts the pre-audit of the order-to-cash process as soon as an order message arrives and the other user role's process fragments are initiated through an event. Although user roles perform almost similar activities (e.g., *Perform Pre-Audit* must be performed by all user roles), the company decided to document such activities in separate process fragments. After completing the pre-audits, the different process fragments are synchronized through a final discussion. The latter is visualized as the sub-process *Order Discussion*, which abstracts from concrete activities to be performed. The sub-process itself is documented by a separate process model in another document.

In addition to the six process models, each activity is described in a verbalized way. The latter is required in the context of an ISO 9000 certification [69]. Furthermore, spreadsheets exists, which describe information required in the *order-to-cash* process. To be more precise, the spreadsheets describe data objects the activities produce or consume. Since information on the *order-to-cash* process is scattered over various artifacts, an update of the latter must be manually propagated to all artifacts, i.e., the existing process models, text documents, and spreadsheets. As a consequence, not all process models, text documentations, and spreadsheets of the *order-to-cash* process were updated consistently updated in the past. Figure 3.3 shows the relation between the various artifacts and the procedure how updates are propagated between these artifacts.

Note that the various types of process-related artifacts refer to the same business process (i.e., order-to-cash) on different levels of detail. Furthermore, they are intended for different groups of domain experts, each requiring a specific perspective on the business process. Unfortunately, process information is scattered over a set of artifacts that must be kept up-to-date causing a high maintenance effort and being an error-prone task.

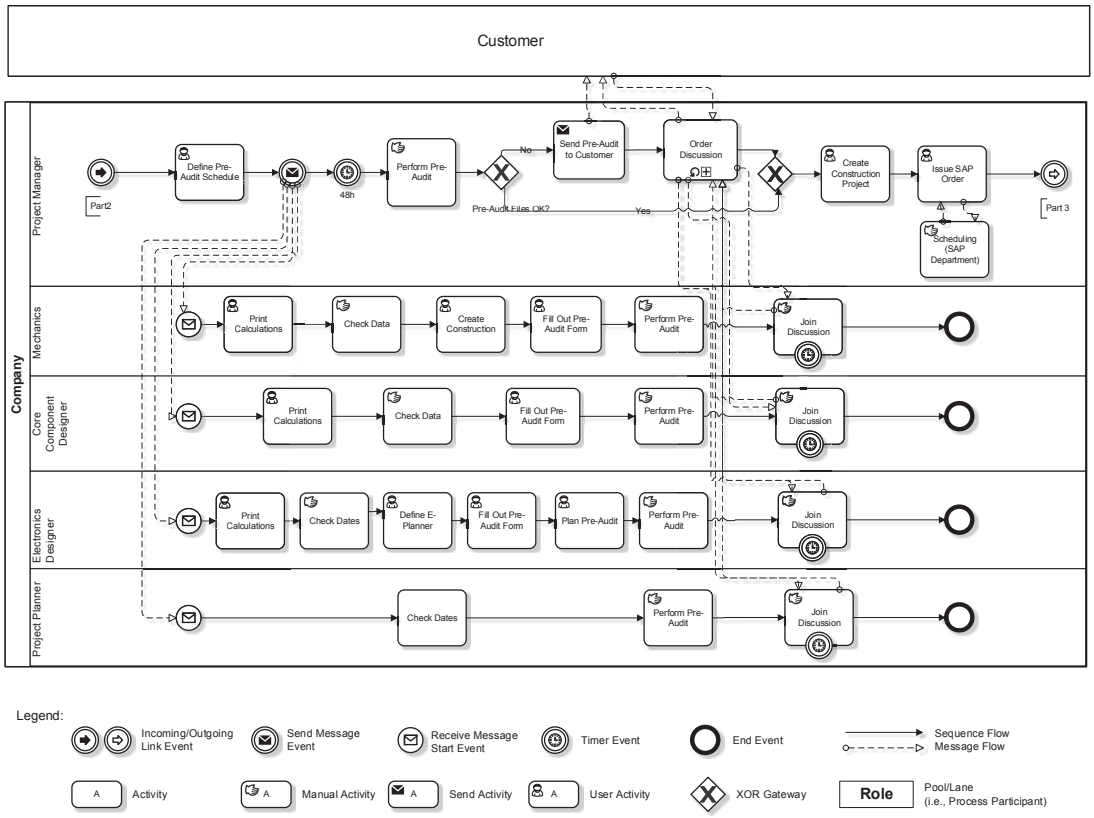


Figure 3.2: Pre-Audit Part of Order Processing Process (in BPMN)

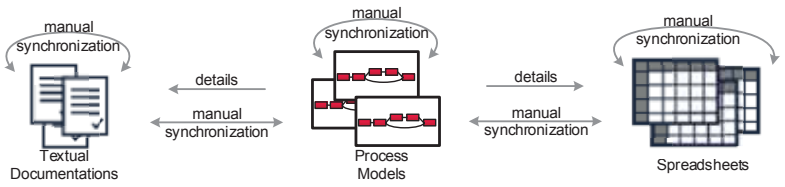


Figure 3.3: Relation Between Process Artifacts

3.1.2 Case Study CS2: Payroll Processing in an Accountant Company

Case study CS2 investigates the *payroll process* of an accountant company. The latter offers the service of processing payrolls to small and medium-sized companies. We describe the current situation of the company as well as how its process models looks like. In detail, the payroll process consists of three major phases: *Preparation*, *Review*, and *Delivery* (cf. Figure 3.4). Each of them is detailed by a specific sub-process model.

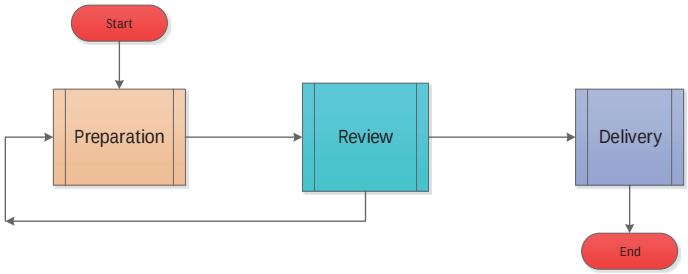


Figure 3.4: Overview of Payroll Process as Flow Chart

Phase *Preparation* comprises activities for fetching and pre-processing information of the employees (e.g., working hours) the payroll is processed for. Furthermore, working hours are registered in an accounting information system and payment schedules are defined. Finally, the available payroll information is merged for reviewing. The *Review* phase comprises activities for cross-checking payroll information with employee records as well as the time sheets provided by the customer. This phase must be performed by another accountant (i.e., separation of duties). In this context, the contract of the employee as well as financial data of the customer are checked. If any of the checks fails, the process loops back to the *Preparation* phase to correct the payroll information. Phase *Delivery* finalizes all documents and payrolls are transferred to the customer either by email or a personal pick up. In the latter case, the customer is called to pick up the documents in the office of the accountant company.

The accountant company captures its payroll process in process models of different levels of granularity. Furthermore, different process representations are used. More precisely, two kinds of graph-based process models exist—one providing a high-level view on the business process using flow charts as process modeling language (cf. Figure 3.4) and the other describing the course of action more detailed. Figure 3.5 depicts these detailed process models for each payroll phase, i.e., Preparation, Review, and Delivery.

In turn, Figure 3.6 shows excerpts of the textual descriptions of the payroll process. Figure 3.6a depicts work instructions for individual activities. More precisely, it describes which information system is used by specific activities and which user form shall be used in this context. In particular, the work instructions shall support less experienced accountants. Figure 3.6b shows checklists to be used in the different phases of the payroll process. These checklists refer to selected activities that were omitted frequently in the past and do not comprise all the process models' activities (cf. Figure 3.5). Particularly, the checklists shall be used on a daily basis and reduce error rates.

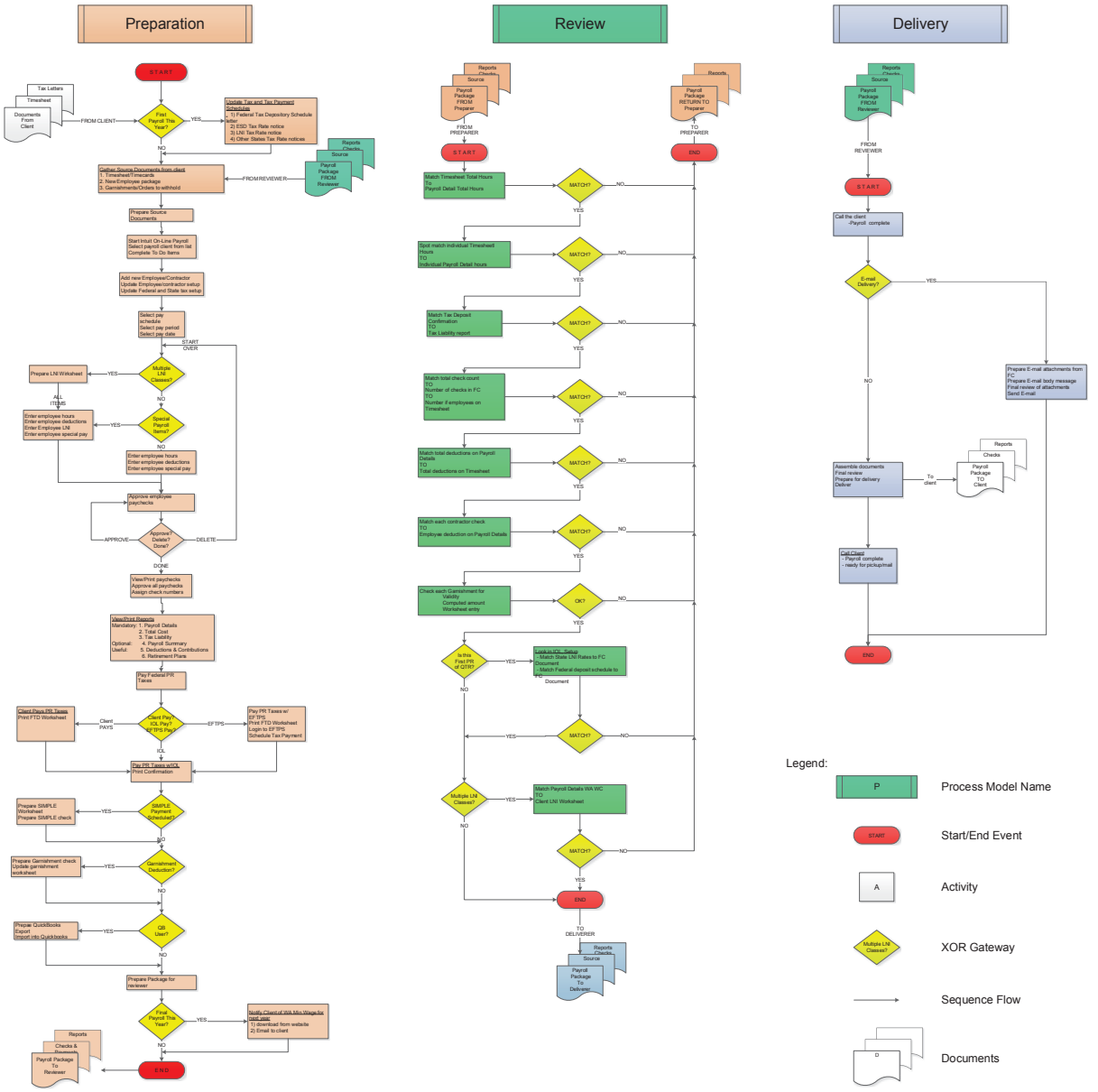
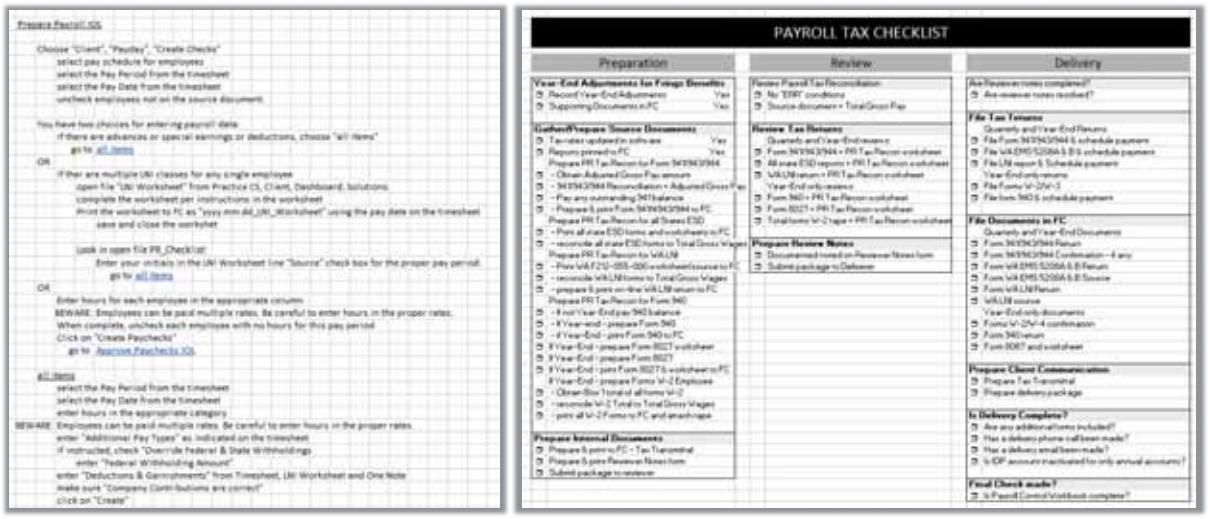


Figure 3.5: Payroll Process Models as Flow Charts



a) Payroll Work Instructions

b) Payroll Checklists

Figure 3.6: Payroll Process Descriptions

Altogether, the accountant company employs four different ways to document its payroll process. In turn, this results in high efforts for capturing as well as maintaining the business processes. In particular, the accountants complained that the update of process models due to organizational changes or changes of tax laws constitutes an error-prone task. Hence, the accountant company has decided to reduce the number of documents despite the fact that all of them are considered as being valuable.

3.1.3 Discussion

Case studies CS1 and CS2 indicate that process models of different levels of granularity exist, describing one and the same business process. In both case studies, process participants with varying knowledge in process modeling can be identified and different process representations are employed to them (i.e., graph-based, verbalized, table-based, and checklist-based). In both case studies, process models are maintained manually, which not only constitutes a time-consuming task, but also results in inconsistencies and errors as well. For example, when changing a particular artifact of the business process, this change is not always correctly propagated to all other artifacts related to the same business process. In turn, this leads to outdated process model versions. Another serious issue concerns versioning, i.e., to identify which version of a process model corresponds to which version of the textual documentation or spreadsheet.

The case studies further reveal that there exists no proper tool support for process designers to create and update process models. When creating a process model, the designer starts with only rudimentary information about the business processes. Then, process participants are interviewed in order to collect information about their activities. Based on these interviews and the notes made, a process model is created stepwise or—to be more precise—multiple process models of different granularity are created. However, no tool support exists when interviewing process participants or discussing process updates *on-site* (i.e., at the users' workplaces).

3.2 Requirements

Based on the described case studies CS1 and CS2 as well as the insights gained from literature study major requirements for a process modeling framework addressing the described research questions are elicitated. To stay focused, we exclude requirements related to (process-aware) information systems in general [70, 71, 72].

The case studies have revealed that domain experts involved in documenting and evolving business processes have limited process modeling knowledge although it is relevant to all of them. As a consequence, different process-related artifacts are (manually) created. In case study CS2, for example, the accountant company provides detailed activity descriptions to new staff members. In general, a process modeling tool to create, visualize, and change process models should involve all domain experts (cf. Requirement REQ-1).

Requirement REQ-1 (Accessible for Domain Experts)
Domain experts should not need to have specific process modeling knowledge to create, comprehend, or update process models.

3.2.1 Process Abstraction and Update

All relevant information of a business process must be documented in process models. For example, in the context of case study CS2, activities, data objects, user roles, and required information systems are described. In general, a process modeling tool should enable domain experts to properly capture all process information required (cf. Requirement REQ-2).

Requirement REQ-2 (Process Model Elements and their Representation)
All relevant process information should be captured and represented in process models.

Case studies CS1 and CS2 further show that one and the same business process may be captured in different process models using different levels of granularity; e.g., process models providing a coarse-grained view on the business process and other ones describing parts of the business process in detail. Furthermore, process models have been created for specific groups of domain experts showing only the activities relevant for them (e.g., case study CS2). Finally, a process model should be as compact as possible, i.e., unnecessary elements should not be displayed (cf. case study CS1 and CS2). For example, [73, 74] have shown that the size of a process model has an impact on its understandability as well as the number of errors caused by process designers (cf. Requirement REQ-3).

Requirement REQ-3 (Process Model Abstractions)
It should be possible to abstract process models by hiding or aggregating information. In particular, personalized (i.e., user- or role-specific) process abstractions are required. Process model abstractions should be compact to ease their understanding.

In general, business processes evolve over time. Hence, the corresponding process models and related abstractions need to be updated continuously. In particular, the latter should be automatically and consistently handled. Note that the case studies have revealed that several process

models have been outdated compared to the actual process version. In turn, outdated process models are not useful for domain experts. Consequently, domain experts should be enabled to change process models—no matter which level of abstraction the latter offers—in order to represent the change of the business process (cf. Requirement REQ-4). In turn, all process models referring to the same business process should be updated as well (cf. Requirement REQ-5).

Requirement REQ-4 (Process Model Updatability)
Domain experts should be enabled to evolve process models or even process abstractions.

Requirement REQ-5 (Up-to-Date Process Models)
When updating a process model all process models and process model abstractions, respectively, referring to same business process should to be updated as well.

3.2.2 Process Representation

Case studies CS1 and CS2 have confirmed that process models not only vary in their level of granularity, but in the way they are represented (i.e., visualized) to the user as well. The intention behind this is to take the vocational background and process modeling knowledge of domain experts, i.e., process models shall be presented in a shape that it is easy to understand (cf. Requirement REQ-6).

Requirement REQ-6 (Intuitive Process Representations)
Process models should be visualized in a way easily understandable for domain experts not having specific process modeling knowledge.

To still allow domain experts to change process models, respected update functionality should be available for process representations (cf. Requirement REQ-7). Both case studies have shown that it is important that process models can be updated no matter which process representation is applied.

Requirement REQ-7 (Updatability of Process Representations)
Process models should be updatable by domain experts independent from the kind of process representation applied .

In practice, different process representations are used to cope with the varying process modeling knowledge (cf. case study CS2). The respective representation is selected depending on the given application scenario. For example, in case study CS2 textual representation is applied for new employees and checklists for experienced employees. Hence, a process representation should not be static, but chosen depending on the current application scenario (cf. Requirement REQ-8).

Requirement REQ-8 (Adjustable Process Representations)
Domain experts should be able to dynamically adjust the process representation of process models.

3.2.3 Process Interaction

Touch-enabled devices, like smartphones, tablets, and touch tables, are becoming more common in companies [46]. In turn, the use of these devices requires advanced user interaction concepts [75]. In both case studies domain experts complained that process modeling tools require desktop computers or notebooks to evolve process models, i.e., they usually print process models discuss with process participants the process activities they are involved. As a consequence, handwritten notes must be manually transferred to a process modeling tool after completing the interviews. In general, touch-enabled devices should be supported when creating and evolving process models (cf. Requirement REQ-9).

Requirement REQ-9 (Intuitive Process Interaction)
Domain experts should be enabled to interact with process models using touch-enabled devices.

3.2.4 Discussion

The elicitated requirements are not specific to two case studies, i.e., they can be identified in other domains and related work as well. For example, [31] deals with a large business process of a health insurance company. The considered process model comprises 150 activities and 300 process elements (i.e., activities, data objects, user roles, and temporal constraints). The authors state that the involvement of all domain experts in process management increases process knowledge in a company. Furthermore, abstractions of process models are requested as well as methods to keep them consistent. These problems are addressed by REQ-1-REQ-5.

The need for sophisticated process model abstractions (i.e., REQ-1-REQ-3) and representations (i.e., REQ-6+REQ-8) has been discussed in [50, 76] as well. Furthermore, [32] demands for an appropriate interaction concept for process models (i.e., Requirement REQ-9).

Requirements to change process models (i.e., REQ-4+REQ-7) and keeping associated process models up-to-date (i.e., Requirement REQ-5) have been discussed in the context of business-IT alignment [24], process variability management [26], and process flexibility [15]. Finally, the requirement to keep process models as compact as possible is referred by process model refactoring [18] or process mining [77].

Obviously, the presented list of requirements is not complete in respect of a fully featured process modeling tool. Instead, we focus on requirements in the context of our research questions.

3.3 Summary

This chapter presents two case studies to illustrate emerging challenges that need to be tackled when facing large process models. The case studies underline that several process models of different levels of granularity, referring to the same business process, exist in companies. Usually, the process models are created for different domain experts or application scenarios (e.g., ISO certification). Furthermore, different process representations (e.g., graph-based, textual, and checklists) are employed to cope with varying process modeling background of domain experts.

Table 3.1 summarizes the requirements elaborated in the case studies and relates them to the research questions presented in Chapter 2. Generally, the requirements not focus on a specific application domain, but aims to be generic.

Requirement	Description
Functional Requirements - General	
Requirement REQ-1	Accessible for Domain Experts
Functional Requirements - Research Question 1	
Requirement REQ-2	Process Model Elements and their Representation
Requirement REQ-3	Process Model Abstractions
Requirement REQ-4	Process Model Updatability
Requirement REQ-5	Up-to-Date Process Models
Functional Requirements - Research Question 2	
Requirement REQ-6	Intuitive Process Representations
Requirement REQ-7	Updatability of Process Representations
Requirement REQ-8	Changeability of Process Representations
Functional Requirements - Research Question 3	
Requirement REQ-9	Intuitive Process Interaction

Table 3.1: Requirements Overview

4
Process Modeling Tools: State-of-the-Art

4.1 Introduction

This chapter gives an overview on contemporary business process modeling tools, which provide a variety of process representation, modeling , and interaction concepts. In general, one can distinguish between *drawing* and *process modeling* tools [32]. *Drawing* tools (e.g., Microsoft Visio, Microsoft PowerPoint) are not explicitly developed with the goal to create process models. In practice, however, respective tools are frequently used for process modeling as well [17]. By contrast, the functions of *process modeling tools* are specifically designed for the purpose of documenting business processes. Hence, the latter target at an integrated support for creating, maintaining, and representing process models. This chapter focuses on selected process modeling tools and their functions related to the requirements set out in Chapter 3. The respective selection is based on process modeling surveys [56, 78].

Table 4.1 depicts the process modeling tools considered. The latter may be categorized as *desktop*, *cloud*, and *mobile tools*. *Desktop* modeling tools need to be installed on the computer of the process designer, but may have a sever-side counterpart as well; e.g., to enable access to a process repository or to allow for collaborative process modeling. As a representative of this type, we consider *IBM Business Process Manager (IBM BPM for short)* [79], which offers a broad range of functions for modeling, executing, and monitoring business processes. As opposed to IBM BPM, *ARIS Business Designer* solely provides process modeling functions [21], but can be integrated with other tools to simulate and execute process models as well. Another desktop modeling tool is *MID Innovator for Business Analysts* [80]. Finally, *Bizagi Process Modeler* allows creating executable business process models [81].

Cloud-based process modeling tools are hosted on a web server. Usually, a browser is required to interact with them, i.e., respective modeling functions are provided as *Software as a Service (SaaS)*. *IBM BlueworksLive* provides a web front-end for creating business process models. In particular, it focuses on users having no or only little process modeling knowledge [82]. The

Vendor	Product	Tool Type	Address
IBM Corporation	Business Process Manager	Desktop	www.ibm.com
Software AG	ARIS Business Designer	Desktop	www.softwareag.com
MID GmbH	MID Innovator for Business Analysts	Desktop	www.mid.de
Bizagi	Bizagi Process Modeler	Desktop	www.bizagi.com
IBM Corporation	BlueworksLive	Cloud-based	www.ibm.com
Signavio GmbH	Signavio Process Editor	Cloud-based	www.signavio.de
semture GmbH	Cubetto	Mobile	cubetto.semture.de
Orchard ebusiness Pty Ltd.	Orchard Mobile Process Designer	Mobile	www.orchardprocess.mobi
Tabtou Ltd.	Process Craft	Mobile	www.showgen.com

Table 4.1: Overview of Process Modeling Tools

Signavio Process Editor, in turn, offers a broad range of process modeling languages (e.g., BPMN, *Event-driven Process Chains (EPC)*) as well as collaboration features [83].

Mobile tools developed for mobile operation systems (e.g., Apple iOS or Google Android). *Cubetto*, for example, constitute an Apple iOS app, which supports BPMN, EPC, and UML [84]. In turn, *Orchard Mobile Process Designer* is a simple process modeling tool supporting BPMN [85]. *Process Craft* allows creating process models with BPMN using special gestures [86].

Section 4.2 analyzes process representation and modeling features of the aforementioned tools. Section 4.3 introduces concepts current tools provide for abstracting process models. Section 4.4 presents interaction concepts provided by the tools. Section 4.5 summarizes the chapter.

4.2 Process Representation and Modeling

All considered process modeling tools support a core set of BPMN 2.0 process modeling elements, whereas only three of them provide full BPMN 2.0 support [87]. Furthermore, five tools provide other process modeling languages like EPC, *Unified Modeling Language (UML)*, and Petri-Nets. Once a user has chosen a particular process modeling language, it can no longer be changed. Furthermore, in tools like ARIS the elements of the process modeling language can be restricted to meet, for example, company standards in terms of process documentation. Finally, certain tools allow modeling different perspective of a business process: *data*, *organizational*, *functional*, or *process perspective*.

IBM BlueworksLive provides a proprietary process modeling language (i.e., *discovery map*) to capture the activities of a business process. The *discovery map* allows grouping the activities in milestones (i.e., logical phases of the business process). In particular, ordering constraints need not to be provided at this modeling stage. Following this phase, a BPMN process model can be generated automatically based on the activities specified in the discovery map. As an extension, defined milestones are visible in the BPMN process model as well. Afterwards, for example, the temporal ordering of activities or data objects can be provided. If a user adds an activity to the process model, it appears in the corresponding discovery map and vice versa. Furthermore, *documentation* provides a textual description of the created process model. For this purpose, activities are represented as a bullet list. For each activity, a subordinated bullet list describes corresponding activity attributes. When adding new bullet items (i.e., activities), these are synchronized to the other process representations. In contrast to all other analyzed

tools, changes in a process representation are propagated to other representations. Thus, outdated process models can be prevented.

Users are assisted in creating process models in various ways. *Modeling aids* shall enable users with limited process modeling knowledge to interact with process models. In particular, recommendations regarding the use of the process model elements in the next steps and applicable in the given context are provided. Furthermore, *automatic layouting* contributes to arrange process model elements. Especially, tools for mobile devices provide an automatically layout (e.g., Cubetto). *Syntax checks* support users to create syntactically correct process models to increase process model quality [88].

Table 4.2 compares process representation and modeling features provided by the process modeling tools. Although the tools focus on the creation of process models, alternative process representations fostering model comprehensibility are not supported. In particular, once a process modeling language is chosen, a transformation to other representations is no longer possible (except for IBM BlueworksLive). Finally, users only get limited support to interact with process models. Rudimentary modeling aids and syntax checks are the only features provided.

Feature	**IBM BPM**	**ARIS Business Designer**	**MID Innovator**	**Bizagi Process Modeler**	**IBM BlueworksLive**	**Signavio Process Editor**	**semture Cubetto**	**Orchard Mobile Process Designer**	**Process Craft**
BPMN 2.0 Core Set	●	●	●	●	●	●	●	●	●
BPMN 2.0 Full Set		●				●	●		
EPC		●				●	●		
UML		●				●	●		
Process Landscapes		●	●			●	●		
Other Languages		●	●		●	●	●		
Modeling Aids	●	●	●	●	●	●	●		●
Syntax Validation		●	●	●		●			●
Automatic Layouting		●	●		●	●	●		

Table 4.2: Process Representation and Modeling Features

4.3 Process Abstraction

Contemporary process modeling tools provide a number of techniques to abstract process models. IBM BPM offers two levels of granularity for process models. The first one allows modeling the process activities from a business perspective, whereas in the second level each of these high-level activities is refined by providing, e.g., processing data from a database or integrating different user forms). In turn, ARIS Business Designer and MID Innovator allow for the hierarchical structuring of process models in process hierarchies [45]. Thereby, a particular hierarchy level comprises process models at the same level of granularity.

Process models may be interlinked across hierarchy levels to enable users to navigate between

them. Generally, an activity on a higher level may be refined by a process model on a lower one. Accordingly, the linkage between different levels is based on *sub-process* activities. Once a process hierarchy has been fixed, it can not be changed anymore or this would require huge efforts. This means that the set of activities assigned to a particular level is fixed. A notable exception is Signavio, which allows creating sub-processes dynamically based on existing process models, i.e., a set of activities may be dynamically selected and then integrated in a sub-process. As another abstraction feature, Signavio allows users to manually specify *process views*. In this context, single process elements (e.g., activities) may be hidden or pools (i.e., the activities of a particular process participant) be collapsed to reduce overall complexity for users.

Table 4.3 summarizes the abstraction features provided by the various tools. As can be seen, most tools support process hierarchies. Furthermore, support for navigating across process models is provided. Furthermore, certain tools allow users to expand sub-processes, i.e., the sub-process model is then displayed within the sub-process activity. In turn, this helps users to better understand the context of a sub-process. However, only Signavio provides features for hiding process elements. No tool enables personalized process abstractions.

Feature	**IBM BPM**	**ARIS Business Designer**	**MID Innovator**	**Bizagi Process Modeler**	**IBM BlueworksLive**	**Signavio Process Editor**	**semture Cubetto**	**Orchard Mobile Process Designer**	**Process Craft**
Process Hierarchies	●	●	●	●	●	●	●		
Links between Process Models	●	●	●	●	●	●	●		
Expanding Sub-Process				●	●		●		
Hiding Process Elements						●			
Personalized Process Model Abstraction									

Table 4.3: Abstraction Features

4.4 Process Interaction

The user interfaces of the presented process modeling tools are clear and intuitive. Obviously, more sophisticated tools (e.g., ARIS Business Designer) show a higher complexity of their user interface compared to tools with only limited functionality (e.g., Bizagi Process Modeler). Desktop and cloud tools provide menu-based interaction, while not considering the specific characteristics of mobile devices (e.g., small display size and multi-touch capabilities). By contrast, mobile modeling tools provide specific user interfaces addressing these characteristics.

Cubetto provides specific interaction concepts with respect to process modeling. When inserting a process element, the latter is positioned according to a specific process layout, i.e., the layout algorithm chooses an optimal position and no manual positioning of process elements is required. Instead of a menu bar offering process elements, a context menu solely offers process elements, which may be inserted next. The layout of the resulting process model is always appropriate due to the automatic layouting.

ProcessCraft and Process Designer are mobile tools providing a limited set of multi-touch gestures to interact with process models. More precisely, multi-touch gestures are provided to zoom and pan process models. Gestures for modifying a process model or interacting with it are not supported; instead users must use menus when inserting or deleting process elements.

Table 4.4 summarizes the interaction capabilities of the presented process modeling tools. In a nutshell, the provided user interfaces are intuitive to use, but lack multi-touch support. In particular, no specific multi-touch gestures for creating or changing process models are provided.

Feature	IBM BPM	ARIS Business Designer	MID Innovator	Bizagi Process Modeler	IBM BlueworksLive	Signavio Process Editor	semture Cubetto	Orchard Mobile Process Designer	Process Craft
Intuitive Usage	●	●	●	●	●	●	●	●	●
Mobile User Interface							●	●	●
Multi-touch Support									●
Touch-enabled Processs Modeling									

Table 4.4: User Interaction Features

4.5 Summary

This chapter evaluates state-of-the-art process modeling tools. In this context, selected tools of various categories are considered.

The nine tools support a wide range of process modeling languages. Once a particular language is selected for process modeling, however, the created process model is bound to the notation provided by that language. If the notation of another process modeling language shall be used, the process must be remodeled. All tools provide modeling aids, i.e., process designers are rudimentarily assisted in creating process models. Still considerable process modeling knowledge is required to create and change process models. IBM BlueworksLive provides more sophisticated support for users without profound process modeling knowledge.

Process hierarchies shall structure large business process models. In particular, process models may be linked across several levels through sub-process activities. However, only Signavio provides more advanced abstraction concepts (i.e., process views hiding process information). All process modeling tools provide interaction concepts known from standard desktop applications. No dedicated support of mobile devices exists to assist users in creating process models.

Table 4.5 evaluates the tools based on the requirements presented in Chapter 3. Contemporary process modeling tools target at process designers rather than domain experts (cf. REQ-1). Further, they support various process modeling languages. Although all tools provide BPMN 2.0 support, only three consider all BPMN elements (cf. REQ-2 and REQ-6). Switching between the notations of different process modeling languages is not supported (cf. REQ-8). Process

hierarchies are used to abstract process models according to users' needs (cf. REQ-3). Updates in process model collections must be performed manually on all process models affected (cf. REQ-4, REQ-5, and REQ-7). Although mobile tools exist, process modeling on mobile devices is only rudimentary supported (cf. REQ-9).

Requirement	**Support in Process Modeling Tools**
REQ-1 Accessible for Domain Experts	Tools are made to be used by process designers.
REQ-2 Process Model Elements and their Representation	Not all tools support the BPMN 2.0 full set, i.e., not all process information can be documented.
REQ-3 Process Model Abstractions	Only supported by process hierarchies and by simple process views in Signavio.
REQ-4 Process Model Updatability	Updates in process hierarchies are possible. No updates are allowed in process views (in Signavio).
REQ-5 Up-to-Date Process Models	Not supported
REQ-6 Intuitive Process Representations	Process designers require process modeling language knowledge. Exception: process landscapes and discovery maps in IBM BlueworksLive.
REQ-7 Updatability of Process Representations	Not supported
REQ-8 Changeability of Process Representations	Only basic support in IBM BlueworksLive.
REQ-9 Intuitive Process Interaction	Optimized for desktop PCs; insufficient for mobile devices.

Table 4.5: Tool Support of Requirements

Note that we do not present related research in this chapter. The latter is separately discussed in the context of process views (cf. Chapter 6), process view updates (cf. Chapter 7), process representation (cf. Chapter 9), and process interaction concepts (cf. Chapter 10).

5 An Overview on the proView Framework

5.1 Introduction

Chapter 3 elicitated the requirements relevant in the context of the considered research questions. The goal is to enable domain experts to understand the models of the business processes they are involved in. In particular, they shall be enabled to evolve and, hence, to update the process models at a high level of abstraction. Parts II and III of this thesis introduce the *proView* framework, which addresses introduced requirements. In order to reduce process model complexity as well as to increase model comprehensibility two strategies are applied. First, techniques are provided to abstract from unnecessary details regarding the *structure* of a process model. Second, domain experts are enabled to customize the *representation* of a process model. Abstracting a process model requires techniques for reducing (i.e., hiding) process elements as well as for aggregating (i.e., combining) them. The process model resulting from such an abstraction is denoted as *process view*. For a given process model, specific views may be created reflecting the needs the different domain experts and use cases.

Customizing the *representation* of a process model or process view, in turn, shall allow varying the way the model is displayed to the user. For example, a graph-based process model may be visualized using various layouts, process element visualizations, or process modeling languages. In turn, this enables us to cope with the vocational background and varying process modeling knowledge of domain experts. Process model abstractions and representations can be considered as independent concepts, but may be applied in combination with each other to allow for personalized process views. Moreover, domain experts shall be enabled to perform changes on process model abstractions (i.e., updates on process views) independent from the representation used. As shown in the case studies, respective process modeling tasks may be performed on touch-enabled devices as well. When updating a process view, the change is propagated to the related process model. Furthermore, other process views are updated accordingly.

Section 5.2 gives an overview on the *proView* framework and its components. Section 5.3 discusses ways to store process views in a PAIS. Section 5.4 summarizes the chapter.

5.2 Components of the proView Framework

We first show how process models can be abstracted and visualized. Following this, we deal with changes of the respective process abstractions and representations.

5.2.1 Abstracting and Visualizing Process Models

We rely on fundamental methods of information visualization research to create process views and realize process representations. In particular, the *data state model*—a data visualizing procedure for complex data collections—is applied [89, 90, 91, 92]. Thereby, data visualization is performed in three phases: *data transformation*, *visualization transformation*, and *visual mapping transformation* (cf. Figure 5.1). *Data transformation* loads selected raw data from various data sources and merges them. *Visualization transformation* further abstracts the data (i.e., it hides or combines selected data). Finally, *visual mapping transformation* visualizes the data by a dedicated graphical representation presented to the user.

Figure 5.1: Data State Model

Taking the data state model, Figure 5.2 gives an overview on the *proView* framework. The different phases for creating, visualizing, and updating process views as well as the interacting with them, together with their relation to the *data state model*, will be discussed in the following. Furthermore, the phases are correlated with the elaborated requirements (cf. Chapter 3).

To set a focus this thesis excludes the data transformation phase, i.e., we assume that the process model of the respective business process already exists [93]. To distinguish the original process model from the models representing process views, we denote it as *Central Process Model (CPM)*. Thereby, CPMs may be stored in the *Central Process Repository (CPR)* (cf. Figure 5.2a), saving as basis for creating process views. Note that there exists considerable work to derive a CPM through the mining of process execution and change logs [77, 94, 95, 96] or the application of techniques like model merging [32, 97]. In addition, process views may be used to initially document a business process. Therefore, one may start with an empty CPM as well as an associated process view and apply view updates to it to document a business process.

Visualization transformation is performed in the *view creation* phase, which applies concepts to abstract a CPM (i.e., a process model). In turn, this leads to the creation of the structure of a process view (cf. Figure 5.2b). In particular, a process view abstracts from process information by *reducing* (i.e., hiding) or *aggregating* (i.e., combing) process elements of the CPM. Finally, unnecessary control flow structures (e.g., empty branches) resulting from the transformation are refactored in the view creation phase to obtain compact process view models (cf. Chapter 6).

The *representation mapping* phase generates a *process representation* for a specific user (cf. Figure 5.2c), i.e., a *visual mapping transformation* is applied. Examples of such a process representation are graphical process modeling languages or textual process descriptions (cf. Chapter 9).

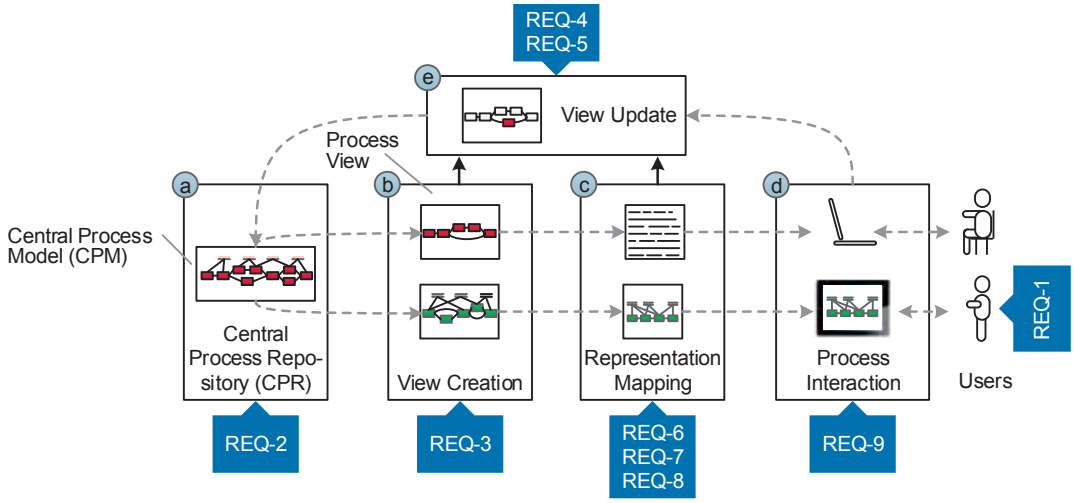


Figure 5.2: Overview on the proView Framework

5.2.2 Updating Process Models and Interacting with them

Enabling users to create and evolve process models requires *process interaction* support (cf. Figure 5.2d), i.e., authorized users should be enabled to evolve process models through updates of related process views, e.g., using multi-touch gestures. Such an interaction support is required to enable users to intuitively update and evolve process models and process views. Chapter 10 introduces corresponding process interaction concept for touch-enabled devices.

If a user triggers a *view update*, the latter must be propagated to the CPM (cf. Figure 5.2e). This propagation is required to guarantee the up-to-dateness of the CPM. In order to ensure that process views refer to the most recent version of the CPM, all associated process views then need to be updated as well. Chapter 7 and Chapter 8 presents an approach enabling updates on process views and ensuring up-to-dateness of all other process views associated with this CPM.

Generally, the phases of view creation, view update, representation mapping, and user interaction are orthogonal. Hence, they may be applied independently from each other. For example, process representations may be directly utilized to process models in the CPR. Alternatively, view creation and user interaction may be combined to enable intuitive interactions with process views. Independence of the phases enables us to provide a powerful and flexible framework for process representation and modeling.

5.3 Materialized and Virtual Process Views

Before introducing concepts for creating and updating process views, we discuss how the latter may be represented in a CPR. Generally, process views and their relationship to a given CPM must be maintained in a way that allows propagating process view updates to the CPM (cf.

REQ-4). Furthermore, all other process views associated with the CPM have to be migrated to the new CPM version (cf. REQ-5). For representing process views, two alternatives exits, i.e., *materialized* or *virtual* views (cf. Figure 5.3) [32].

In case of *materialized process views*, the CPR stores the process models of the CPMs as well as their related process views. Furthermore, references are maintained describing the relationship between CPMs and associated process views. However, following this approach no information is available on how (i.e., by which view creation operations) a process view was created (cf. Figure 5.3a). As opposed to materialized database views, no information is maintained on how the views are created [98]. If a user wants to access a process view, it will be directly fetched from the CPR. Operations to create process views in the view creation phase are only required to create new process views. Applying updates to materialized process views requires to determine corresponding updates in the model of the CPM and all associated process views. Note that, this is far from being trivial since the relations between the nodes in the CPM and the process views are not maintained. If unnecessary control flow structures are refactored, for example, determining a corresponding update of the CPM is not straightforward. Certain updates may concern large regions in a process view (e.g., parallel insertion). In particular, [99, 100] have shown that it is not always possible to determine corresponding CPM updates when applying a view update.

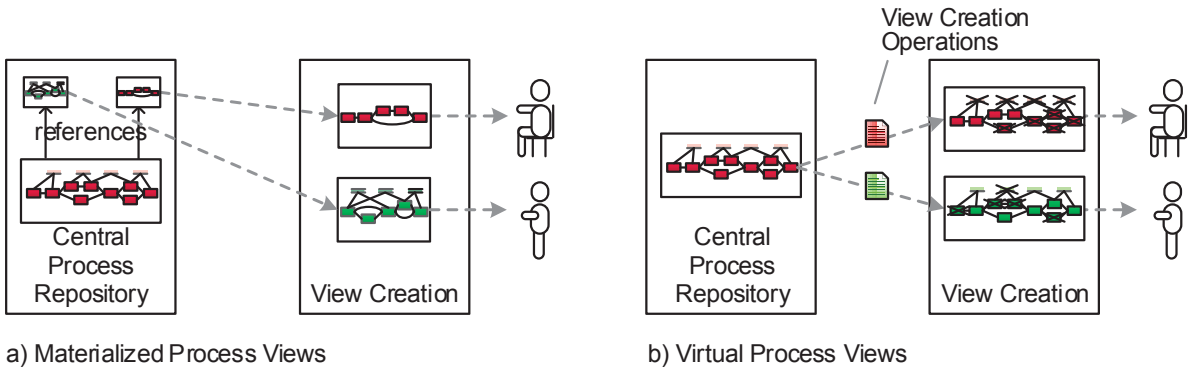


Figure 5.3: Representation of Process Views

Virtual process views are created based on the CPM as well as a set of *view creation operations*. The latter express the process information to be reduced or aggregated. If a user wants to access a particular process view, the latter is derived in the view creation phase by applying a set of view creation operations to the CPM (cf. Figure 5.3b). In particular, there is no need to maintain the relations between the CPM elements and the ones of the associated process view. These relations are specified by the set of view creation operations. Thus, updates on process views can be propagated to an update of the CPM. In turn, associated process views can be re-created based on the set of view creation operations.

In this thesis, we apply virtual process views to allow for updates based on process views. Chapter 6 provides details on the creation of process views and Chapter 7 deals with updates triggered on a process view and their propagation to the CPM and associated process views.

5.4 Summary

This chapter gives an overview on the *proView* framework, which allows creating and updating personalized process abstractions (i.e., process views). Furthermore, it discusses alternatives for representing and maintaining process views in a process repository. Figure 5.4 gives an overview of chapters and shows their relation to the components of the *proView* framework. In Part II, Chapter 6 introduces operations to create process views. Chapter 7 provides operations to update process views as well as the migration of associated process views. Chapter 8 discusses more advanced process view updates. Part III introduces in Chapter 9 process representations. Finally, Chapter 10 presents interaction concepts for touch-enabled devices to interact with process models and process views.

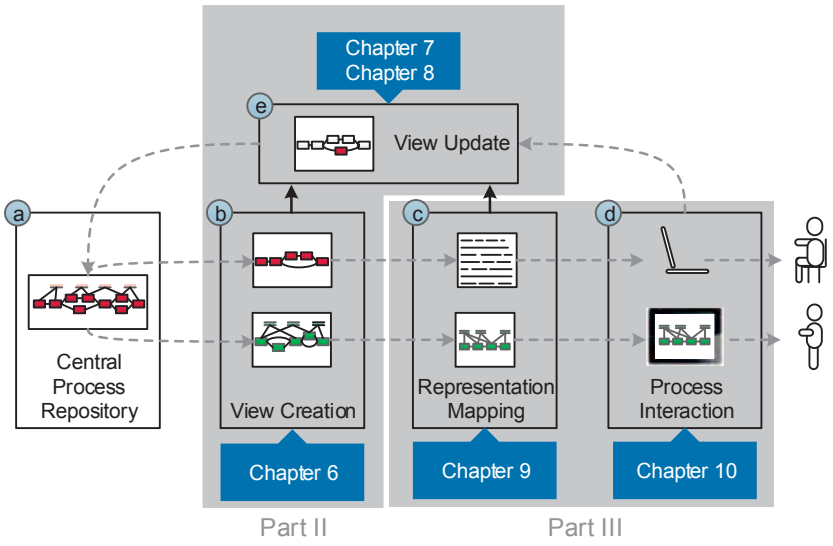


Figure 5.4: Overview proView Framework and Chapters

Part II

Updatable Process Views

6 Creating Process Views

6.1 Introduction

Companies must support a large variety of business processes comprising different organizational units, user roles, business objects, and business functions (i.e., activities). Usually, information about a business process is captured in a process model that is stored in a process repository. In practice, such a repository may comprise hundreds or thousands of process models. In turn, a single process model may include numerous activities, business objects, and different user roles. Typically, each user role has a specific perspective on the respective business process requiring a different level of process information granularity. For example, managers prefer an abstract view on a business process, whereas process participants require a more detailed view on those parts of the business process they are involved in. To reflect this need, it is common to manually create specific process models for the various user roles, which then have to be explicitly maintained. Obviously, creating and maintaining separate process models representing the same business process (or parts of it) causes high efforts for process designers. Consequently, the automated provision of personalized views on process models, which abstract or hide certain process information, is a much needed feature in contemporary *Process-aware Information Systems (PAISs)* to properly support all domain experts involved in a business process (cf. Requirement REQ-3).

This chapter introduces well-defined operations for creating abstractions of process models, which we denote as *process views*. A process view abstracts a process model by reducing (i.e., hiding) or aggregating (i.e., combining) parts of it. Thereby, we restrict aggregations on activities having a direct precedence relation (i.e., "adjacent" activities in the process model). Respective aggregations ensure that no additional precedence relations are added when creating a process view. This restriction guarantees that the order in which the process activities shall be performed is not distorted in the process view. Furthermore, it enables us to apply well-defined update operations on process views.

Example 6.1 illustrates how process views may be used in context of the *credit application process*.

Example 6.1 (Application of Process Views)
Consider the credit application process of a bank, which involves: a customer, a clerk, and a manager. Furthermore, a PAIS is used to coordinate the process activities (including automated activities). The corresponding process model is depicted in Figure 6.1.

The credit application process starts with the customer filling out an inquiry. If the customer has not been registered yet, a corresponding record is created in the database. Otherwise, the existing customer record is retrieved by the PAIS. Next, the address and credit rate are provided by the clerk and the credit risk is determined. Based on the information provided, the manager decides whether the credit shall be granted. Afterwards, the PAIS notifies the customer and updates the customer database. Finally, the clerk calls the customer to discuss next steps.

Personalized process views are provided for the customer as well as the clerk to foster their understanding of the credit application process. In detail, the process view of the customer solely shows the major phases of a credit application, i.e., Customer Inquiry, Inquiry Screening, Decision Making, and Customer Notification. Three of the four phases aggregate major parts of the credit application process (cf. Figure 6.1). In turn, the process view of the clerk hides automated activities, which are executed by the PAIS and aggregates the activities the manager has to perform (cf. Figure 6.1).

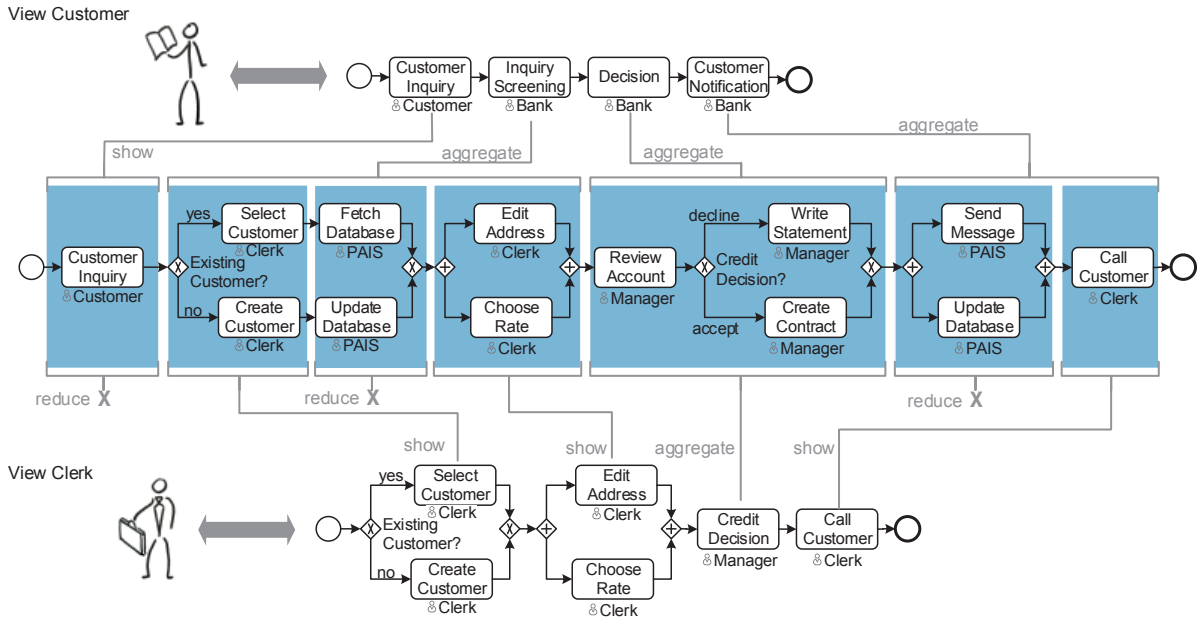


Figure 6.1: Credit Application Process and its Process View

Creating a process view on a process model might result in control flow structures no longer needed (e.g., empty parallel branches) and making the resulting process view unnecessarily com-

plex. In general, a process view should be as compact as possible to reduce complexity for users (cf. Requirement REQ-3). In order to tackle this challenge, in addition to view creation operations, this chapter provides process model refactorings that allow simplifying the structure of the created process view. In particular, refactorings preserve the behaviour of a process view. Thus, they do not influence the meaning of a process view.

Figure 6.2 gives an overview on how process views are created. First, a CPM is retrieved from the CPR (i.e., Central Process Repository) and a set of view creation operations are applied to it. Then, refactoring rules are applied to the resulting process view. In order to further structure process views, these steps are repeated to create flexible *process hierarchies*, i.e., process views on process views.

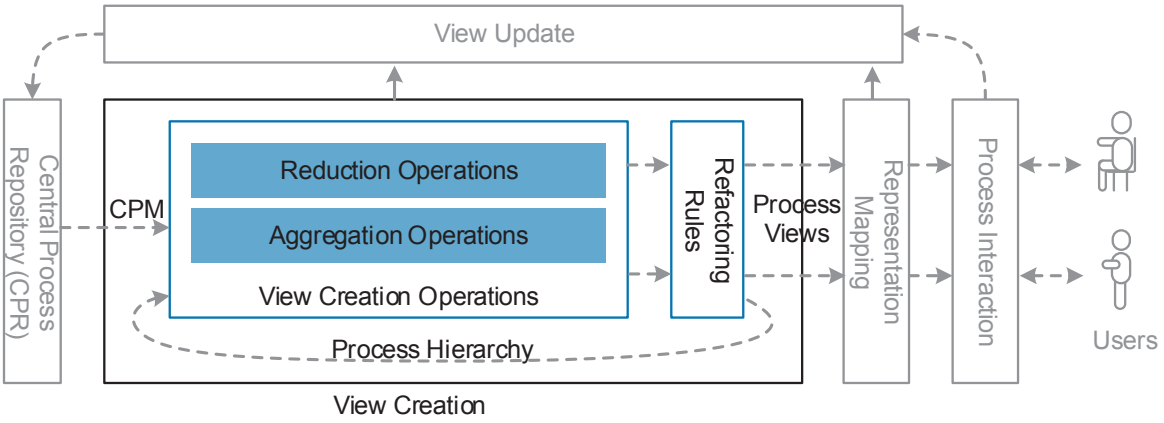


Figure 6.2: Details on View Creation

The chapter is structured as follows: Section 6.2 introduces basic notions and fundamentals of process models and process views. Section 6.3 then introduces elementary operations for creating process views, whereas Section 6.4 provides behaviour-preserving refactorings that simplify the control flow structure of a process view. Section 6.5 introduces flexible process hierarchies. Related work is discussed in Section 6.6. Section 6.7 concludes the chapter.

6.2 View Creation Fundamentals

This section presents basic notions and definitions related to process models and process views.

6.2.1 Process Model

In PAISs, a business process is described in terms of a *process model*. In this thesis, the latter corresponds to a directed graph that comprises a set of *process elements* and a set of *edges* connecting them (cf. Definition 6.1).

Definition 6.1 (Process Model)
A process model is defined as a tuple $P = (N, D, NT, CE, EC, ET, DE, DET)$ *where:*

- N *is a set of process nodes, i.e., activities, gateways, and start/end nodes.*
- D *is a set of data elements.*
- $NT : N \rightarrow NodeType$ *with* $NodeType := \{StartFlow, EndFlow, Activity, ANDsplit, ANDjoin, XORsplit, XORjoin, LOOPsplit, LOOPjoin\}$ *assigns a node type* $NT(n)$ *to each node* $n \in N$. N *can be divided into disjoint sets of activity nodes* A $(NT(n) = Activity)$ *and structural nodes* S $(NT(n) \neq Activity)$, *i.e.,* $N = A \dot{\cup} S$.
- $CE \subset N \times N$ *is a set control edges expressing precedence relations* $(e := (n_{src}, n_{dest}) \in CE$ *with* $n_{src}, n_{dest} \in N \wedge n_{src} \neq n_{dest})$.
- $EC : CE \rightarrow Conds \cup \{\text{TRUE}\}$ *assigns to each control edge either a branching condition or* $TRUE$ *(i.e., the branching condition of the control edge always evaluates to true).*
- $ET : CE \rightarrow EdgeType$ *with* $EdgeType := \{ET\_Control, ET\_Sync, ET\_Loop\}$ *assigns to each control edge* $e \in CE$ *an edge type* $ET(e)$.
- $DE \subset (D \times N) \cup (N \times D)$ *is a set of data edges between activities and data elements. For* $n \in N, d \in D$ *we denote* $(n, d) \in DE$ *as write data edge and* $(d, n) \in DE$ *as read data edge.*
- $DET : DE \rightarrow DEdgeType$ *with* $DEdgeType := \{mandatory, optional\}$ *assigns to each data edge its type of data access.*

The *control flow* of a process model connects *process nodes* through control edges. Hence, it describes the stream of action within a process model. Process nodes may be further divided into node types *Start-/EndFlow*, *activity*, and *gateway*. *Start-/EndFlow* nodes correspond to start/end points of a process model. Without loss of generality, we restrict a process model to have exactly one *StartFlow* and one *EndFlow* node. In turn, *activities* represent elementary actions of the business process (e.g., fetching information from a database or a user form to be filled out). Finally, *gateways* of different semantics are used to define parallel/conditional splits and joins of the control flow. An *XORsplit* gateway allows choosing exactly one of its outgoing edges (i.e., branches) based on the branching conditions assigned to them. In turn, an *ANDsplit* gateway splits the control flow into a set of concurrently executed parallel branches. Accordingly, an *XORjoin* (*ANDjoin*) gateway joins multiple branches and corresponds to an XORsplit (ANDsplit). Finally, a *LOOPsplit* gateway enables backward "jumps" in the control flow based on an explicit branching condition. In particular, a *LOOPsplit* has exactly one outgoing edge of type *ET_Loop* connecting it with the corresponding *LOOPjoin* gateway. Synchronization edges are described below in the context of the structure of a process model.

Definition 6.1 further covers the *data flow* perspective of a process model; i.e., *data elements* and *data edges*. *Data elements* share process information between process nodes. In particular, data elements are connected with process nodes through data edges. In turn, a data edge expresses that a certain data element is either read or written by a particular process node. In Figure 6.3, for example, activity A writes data element $d1$, which is then read by activity C. Furthermore, for each data edge de, function $DET(de)$ indicates whether the corresponding data element is accessed *mandatorily* or *optionally* when executing the respective activity.

Note that Definition 6.1 serves as basis for representing the structure of a CPM as well as its corresponding *process views* (cf. Chapter 5). Moreover, it can be used in combination with any activity-centered process modeling language, i.e., Definition 6.1 is not language-dependent. Generally, this thesis visualizes process models in terms of the *Business Process Model and Notation (BPMN)* 2.0 due to its widespread use (cf. Figure 6.3). To set a focus only selected BPMN elements are considered. These correspond to the ones most frequently used in practice [101]. Furthermore, based on the selected process elements, more complex process structures may be composed if required. For example, an ORsplit gateway may be expressed by the combined use of ANDsplit and XORsplit gateways [102]. Finally, to comply with BPMN, we use the same visualization for XORsplit, LOOPsplit, XORsplit, and LOOPsplit gateways.

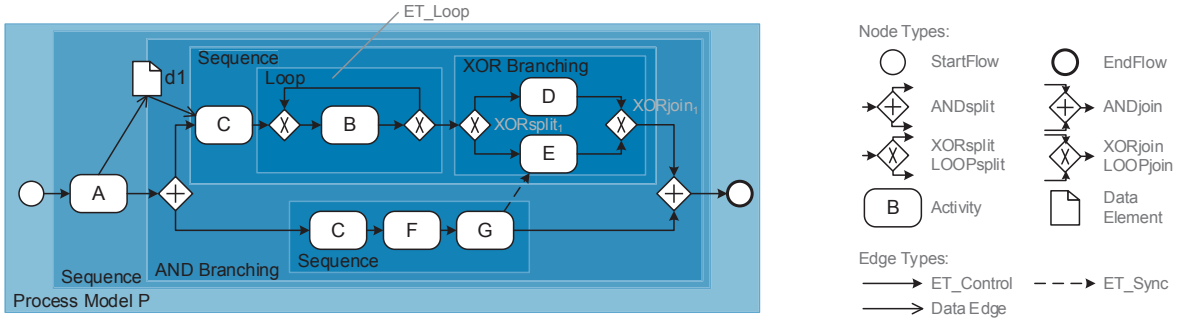


Figure 6.3: Example of a Process Model

To further characterize the structure of a process model, the notion of a *Single Entry Single Exit (SESE)* block is introduced in Definition 6.2.

Definition 6.2 (SESE Block)
Let $P = (N, D, NT, CE, EC, ET, DE, DET)$ *be a process model and* $P' = (N, D, NT, CE', EC, ET, DE, DET)$ *be the subgraph of* P *that excludes synchronization edges, i.e.,* $CE' := CE \setminus \{e \in CE \mid ET(e) = ET\_Sync\}$. *Further, let* $N_S \subseteq N$ *be a subset of process nodes in* N.

Then, a subgraph $P_S = (N_S, D_S, NT_S, CE_S, EC_S, ET_S, DE_S, DET_S)$ *of* P' *is called* SESE *(Single Entry Single Exit) block iff:*

1. P_S *is the subgraph of* P' *induced by node set* N_S,
2. P_S *is connected considering control edge set* CE_S *and node set* N_S, *and*
3. *there is exactly one node in* N_S *without any incoming control edge and one node without any outgoing control edge.*

Finally, $(n_s, n_e) \equiv MinimalSESE(P, N_S)$ *denotes the start and end node of the minimal SESE block comprising all activities from* $N_S \subseteq N$.

As example consider Figure 6.3. Based on Definition 6.2, the subgraph induced by node set $N_S = \{XORsplit_1, D, E, XORjoin_1\}$ constitutes a *SESE block*. Moreover, the *minimal SESE block* comprising activities C and D in Figure 6.3 has start node C and end node $XORjoin_1$, i.e., $MinimalSESE(P, \{C, D\}) = (C, XORjoin_1)$.

The approach developed by this thesis is restricted to *well-structured* (i.e., block-structured) process models; i.e., activity sequences, parallel branchings (i.e., *AND branchings*), conditional branchings (i.e., *XOR branchings*), and loops are expressed in terms of SESE blocks (cf. Definition 6.2) with well defined start and end nodes having the same gateway type (or the same node type for activity sequences). SESE blocks may be nested, but must not overlap [103, 104]. Since we presume well-structured process models, a minimal SESE can always be determined. How SESE blocks can be determined in an efficient way is described in [105].

In practice, well-structured process models do not really limit process designers [106, 107]. In particular, block-structuring is a well-known concept from programming languages [108] as well as process composition languages like WS-BPEL [109]. Furthermore, modern process management systems (e.g., AristaFlow BPM Suite [110] and Cake II [111]) that rely on well-structured process models have been successfully introduced in a variety of application domains [112]. Moreover, well-structured process models are easier to understand and are less error-prone compared to unstructured process models [74, 113, 114, 115, 116]. This is of particular importance when users having limited process modeling knowledge shall change process models. Finally, it has been shown that most unstructured process models can be transformed to well-structured ones [117, 118, 119], i.e., the applied structuring does not actually constitute a limitation of the presented approach. As will be shown, however, it fosters user-driven process changes.

To increase expressiveness of the process modeling language, *synchronization edges* (i.e., $ET(e) = ET\_Sync$), as originally introduced in [120], are used. The latter allow for a *cross-block* synchronization of activities from parallel branches (similar to the *link* concept known from *Web Service Business Process Execution Language (WS-BPEL)* [121]). In Figure 6.3, for example, activity E must not be executed before G is completed.

Definition 6.3 summarizes the aforementioned constraints on structuring process models. Further, it introduces a *correctness* notion for process models.

Definition 6.3 (Correctness of a Process Model)
Let $P = (N, D, NT, CE, EC, ET, DE, DET)$ be a process model. Then:

a) *P describes a structurally correct control flow, iff Constraints 1-4 are met:*

1) *P has exactly one start node n_s (i.e., $\exists! n_s \in N : NT(n_s) = StartFlow$) and exactly one end node n_e (i.e., $\exists! n_e \in N : NT(n_e) = EndFlow$).*

2) *Each process node $n \in N \setminus \{n_s, n_e\}$ has at least one incoming and one outgoing control edge $e \in CE' := \{e \in CE | ET(e) \neq ET\_Sync\}$, i.e., $\forall n \in N \setminus \{n_s, n_e\} : \exists (n, \bullet) \in CE' \wedge \exists (\bullet, n) \in CE'$.*
Moreover, an activity n (i.e., $NT(n) = Activity$) has exactly one incoming and one outgoing control edge $e \in CE'$, i.e.,
$\forall n \in N \setminus \{n_s, n_e\}, NT(n) = Activity : \exists! (n, \bullet) \in CE' \wedge \exists! (\bullet, n) \in CE'$.

3) *Let $P_{S1} = (N_{S1}, D_{S1}, NT_{S1}, CE_{S1}, EC_{S1}, ET_{S1}, DE_{S1}, DET_{S1})$ and $P_{S2} = (N_{S2}, D_{S2}, NT_{S2}, CE_{S2}, EC_{S2}, ET_{S2}, DE_{S2}, DET_{S2})$ be two different SESE blocks induced by node sets N_{S1} and N_{S2} respectively (cf. Definition 6.2).[1]*
Then: Node sets N_{S1} and N_{S2} are either disjoint (i.e., $N_{S1} \cap N_{S2} = \emptyset$) or one is contained within the other (i.e., $N_{S1} \subset N_{S2} \vee N_{S2} \subset N_{S1}$).

4) *Let $P' = (N, D, NT, CE', EC, ET, DE, DET)$ be a subgraph of P with $CE'' = \{e \in CE | ET(e) \neq ET\_Loop\}$. Then: P' must be an acyclic graph. Particularly, the use of control and synchronization edges must not lead to cycles.*

b) *P describes a correct data flow, iff Constraints 5 and 6 are met:*

5) *For any activity linked to a data element $d \in D$ through a mandatory read data edge $de \in DE$ (i.e., $DET(de) = mandatory$), it must to be ensured that d will be always written by preceding activities independent of the execution path chosen.[2]*

6) *For any data element $d \in D$, d must not connect two activities $n_1, n_2 \in N$ through write data edges (i.e., $(n_1, d), (n_2, d) \in DE$), iff n_1 and n_2 are located on parallel branches of the same AND branching, i.e., lost updates will not occur.[2]*

c) *P describes a structurally correct process model, iff Constraints 1-6 are met.*

[1] Sequences of activities must be of maximal length to be considered.
[2] A formal definition and respective validation algorithms may be found in [104].

In the following, functions are introduced that are applied in the context of view creation and view update operations.

Given a process model P and a SESE block induced by node set N', function $first(P, N')$ returns the entry node of the SESE block, i.e., the node whose single incoming control edge connects the SESE block with a preceding fragment of P. Accordingly, function $last(P, N')$ returns the last node of the SESE block that connects it with a directly succeeding fragment P.

Definition 6.4 (First/Last SESE Node)
Let $P = (N, D, NT, CE, EC, ET, DE, DET)$ be a process model and $N' \subset N$ be a set of nodes. Let further CE' be the set of control edges of the minimal SESE block containing all nodes from N'. For $n \in N'$, it then holds:

- $first(P, N') \equiv n_1$ *with* $\nexists(\bullet, n_1) \in CE'$
- $last(P, N') \equiv n_2$ *with* $\nexists(n_2, \bullet) \in CE'$

As example consider Figure 6.4a and the SESE block induced by node set $N' = \{XORsplit_1, B, C, D, XORjoin_1, E\}$. Then, $first(P, N')$ returns $XORsplit_1$ as first node and $last(P, N')$ returns E as last node of the SESE block.

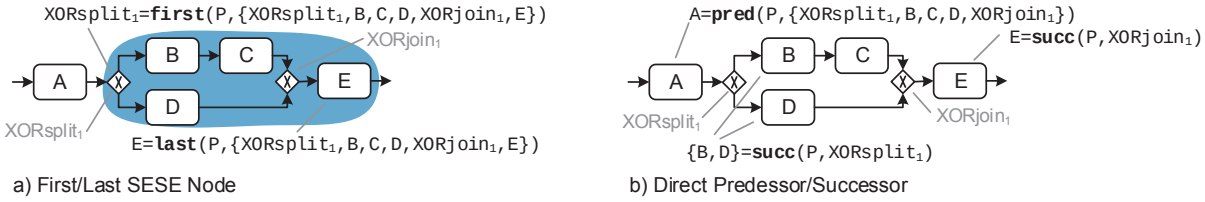


Figure 6.4: Examples of Auxiliary Functions

Regarding a process model P, *pred* (*succ*) returns the direct predecessors (successors) of a process node or node set. As example consider Figure 6.4b: $pred(P, \{XORsplit_1, B, C, D, XORjoin_1\})$ returns A. Moreover, $succ(P, XORsplit_1)$ returns B and D. Finally, $succ(P, XORjoin_1)$ returns E (cf. Figure 6.4b).

Definition 6.5 (Direct Predecessor/Successor)
Let $P = (N, D, NT, CE, EC, ET, DE, DET)$ be a process model.

i) For $n \in N$ we define:
- $pred(P, n) := \{n_p \in N | (n_p, n) \in CE\}$
- $succ(P, n) := \{n_s \in N | (n, n_s) \in CE\}$

If pred (succ) returns a single node set as result, we use $pred(P, n) = n_p$ ($succ(P, N) = n_s$) as simplified notation.

ii) For a SESE block with node set $N' \subset N$ we define:
- $pred(P, N') := pred(P, first(P, N'))$
- $succ(P, N') := succ(P, last(P, N'))$

6.2.2 Process View

Process views abstract from particular details of a process model by reducing or aggregating process elements. To create a process view on a given CPM (i.e., process model), well-defined

are required. For this purpose, *elementary view creation operations* are provided. At the elementary level, two kinds of operations are distinguished: *reduction* and *aggregation*. An elementary *reduction* operation hides a specific process element (e.g., data element or activity), i.e., an element present in the CPM will be hidden in the respective process view. In turn, an elementary *aggregation* operation combines a set of process elements to an abstracted element. For example, a set of activities (data elements) may be aggregated to an abstract activity (business object). An overview of the elementary view creation operations considered as well as their formal semantics are presented in Section 6.3.

A process view on a CPM is created through the consecutive application of elementary view creation operations. In general, for a given CPM, multiple process views may be defined (cf. Definition 6.6).

Definition 6.6 (Process View)
Let CPM be a process model and let $\mathcal{OP}$ be the set of available elementary view creation operations.[1] *Then:*
A process view V on CPM is represented by a creation set $CS_V = (CPM, Op)$ with:

- *CPM being the Central Process Model (i.e., process model) based on which V is created,*
- *$Op = \langle op_1, \ldots, op_k \rangle$, $op_i \in \mathcal{OP}$ is a sequence of elementary view creation operations consecutively applied to CPM.*

The process model of V can be created through the consecutive application of the elementary view creation operations: $CPM = P_0 \xrightarrow{op_1} P_1 \xrightarrow{op_2} \ldots \xrightarrow{op_{k-1}} P_{k-1} \xrightarrow{op_k} P_k = V$ with $P_0, P_1, \ldots, P_{k-1}$ being intermediate process models ($CPM \xrightarrow{Op} V$ for short).

[1] as introduced in Section 6.3

Figure 6.5 shows an example of the consecutive application of view creation operations.

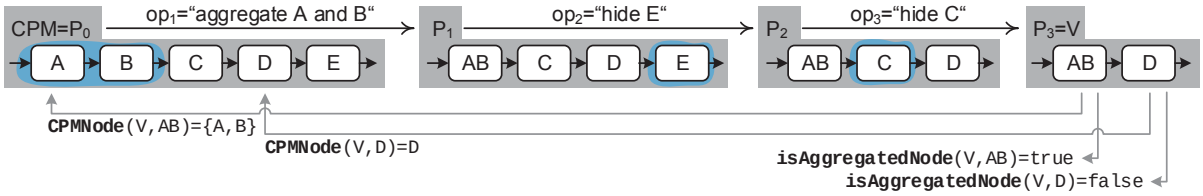


Figure 6.5: Consecutive Application of View Creation Operations

For any process node n of process view V, $CPMNode(V, n)$ determines the corresponding nodes of the underlying CPM (cf. Definition 6.7). Note that n either corresponds to a particular process node n in the CPM or abstracts from a set of CPM nodes. Figure 6.5 shows the application of $CPMNode$.

Definition 6.7 (CPMNode)
Let $CPM = (N, D, NT, CE, EC, ET, DE, DET)$ *be a process model and* $V = (N_V, D_V, NT_V, CE_V, EC_V, ET_V, DE_V, DET_V)$ *be a process view with creation set* $CS_V = (CPM, Op)$ *and* $Op = \langle op_1, \ldots, op_k \rangle$*. Then:*

For any process node $n \in N_V$*,* $CPMNode(V, n)$ *either returns node* $n \in N$ *(i.e., the same node) or node set* $N_n \subset N$ *after applying view creation operation* op_i*:*

$$CPMNode(V, n) \equiv \begin{cases} n & n \in N \\ N_n & \exists op_i \in Op : op_i \text{ aggregates } N_n \text{ to } n \text{ in } V \end{cases}$$

Function $isAggregatedNode(V, n)$ returns *true*, if node n has been the result of applying an aggregation operation op_i to a node set. Otherwise, $isAggregatedNode(V, n)$ returns $false$ (cf. Figure 6.5 for example).

6.3 Process View Creation Operations

This section presents elementary view creation operations for reducing and aggregating process elements, i.e., activities, gateways, and data elements. Then, we show how the respective operations affect process attributes. For sake of readability, we solely show the application of a single view creation operation at a time. In general, however, a process view may be created through the consecutive application of a set of view creation operations (cf. Definition 6.6).

6.3.1 Creating Process Views Through Model Reduction

In order to hide certain elements of the original process model, the view creation framework shall allow reducing process elements. In particular, irrelevant or confidential process information may be hidden from particular users. For example, "technical data elements" (e.g., database connection data) or confidential data elements (e.g., user names or passwords) must not be displayed to certain users in their process views. For this purpose, two elementary operations are provided: *RedActivity* and *RedDataElement*.

RedActivity(CPM,n). View reduction operation *RedActivity(CPM,n)* creates a process view on CPM, which corresponds to CPM except for activity $n \in N$ (i.e., $NT(n) = Activity$), which is hidden from the user together with its incoming and outgoing control edges. Furthermore, the operation re-inserts a control edge linking the direct predecessor of n with its direct successor in the resulting process view[1]. Furthermore, $RedActivity(CPM, n)$ removes all data edges associated with activity n. Note that this might result in a process model with an "incorrect" data flow (cf. Definition 6.3). In Figure 6.6, for example, activity B is reduced (i.e., removed from process model CPM) by applying $RedActivity(CPM, B)$. After removing the data edges associated with B, data element $d2$ is not written by any activity from the perspective of process view $V1$ (i.e., a loss of information occurs when applying *RedActivity*).

[1]Note that activity nodes have exactly one direct predecessor and one direct successor in a process model

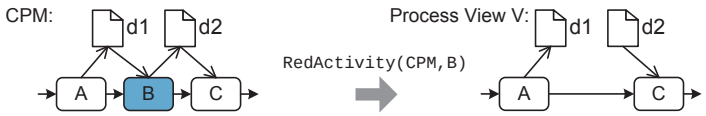


Figure 6.6: View Creation Operation RedActivity

Algorithm 6.1 formally defines *RedActivity*. First of all, *InitializeView(CPM)* (cf. Algorithm A.1) clones CPM, which serves as starting point. Then, activity n is removed from the model of the process view (Line 3). In this context, Lines 4-8 update the set of control and data edges accordingly.

Algorithm 6.1: RedActivity(CPM,n)

```
   Input:  Process model CPM = (N, D, NT, CE, EC, ET, DE, DET)
1          Activity n ∈ N (with NT(n)=Activity) to be reduced
   Output: Process view V = (N', D', NT', CE', EC', ET', DE', DET')
2  begin
3      V := InitializeView(CPM);
4      N' := N' \ {n};
5      e' := (pred(CPM, n), succ(CPM, n)); ET(e') := ET_Control;
6      // Removes control edges connecting node n and adds control edge e'
7      CE' := CE' ∪ {e'} \ {e ∈ CE' | e = (•, n) ∨ e = (n, •)};
8      // Removes data edges connected to node n
9      DE' := DE' \ {e ∈ DE' | e = (•, n) ∨ e = (n, •)};
10     return V;
11 end
```

RedDataElement(CPM,d). View creation operation *RedDataElement(CPM,d)* creates a process view on CPM hiding data element $d \in D$ as well as all data edges associated with d (cf. Algorithm 6.2). As opposed to *RedActivity*, data flow correctness of the process view (cf. Definition 6.3) is preserved since all data edges associated with d (i.e., both read *and* write data edges) are removed together with the data element (cf. Figure 6.7).

Figure 6.7: View Creation Operation RedDataElement

Algorithm 6.2: RedDataElement(CPM,d)

```
   Input:  Process model CPM = (N, D, NT, CE, EC, ET, DE, DET)
1          Data element d ∈ D that shall be reduced
   Output: Process view V = (N', D', NT', CE', EC', ET', DE', DET')
2  begin
3      V := InitializeView(CPM);
4      D' := D' \ {d};
5      DE' := DE' \ {e ∈ DE' | e = (•, d) ∨ e = (d, •)};
6      return V;
7  end
```

6.3.2 Creating Process Views Through Model Aggregation

In order to enable abstractions of certain process model information, a view creation component should allow aggregating the respective process elements to an abstracted one. As opposed to reduction operations, which hide process information, aggregation operations preserve the respective process information in an abstracted way. In particular, this allows presenting certain process information to a specific user group in a more abstract way compared to the original process model. Specifically, it should be possible to aggregate the nodes of a SESE block or several branches of a branching. For example, the activities processed by a particular department may have to be abstracted. Moreover, aggregation may affect the data flow of a process model as well. For example, a set of atomic data elements (e.g., describing attributes of a person) may be aggregated to a business object (e.g., representing a customer).

To support these different use cases, a set of elementary *aggregation* operations are provided: $AggrSESE$, $AggrComplBranches$, and $AggrDataElements$.

$AggrSESE(CPM, N_a)$. Let N_a be the node set of any SESE block of the CPM. View creation operation $AggrSESE(CPM, N_a)$ replaces this SESE block by an abstracted activity node in the resulting process view. For example, Figure 6.8 shows the aggregation of node set $N_a = \{ANDsplit_1, B, C, D, E, ANDjoin_1\}$ and the SESE block induced by N_a respectively. In the resulting process view V, therefore, the SESE block is replaced by activity $BCDE$. In turn, this abstract activity is then embedded in the view model by connecting it with the direct predecessor and the direct successor of N_a. In addition, edge condition $c1$ of the control edge connecting the SESE block with its predecessor in the original model must be set for the edge connecting $BCDE$ with its predecessor as well. Only then a proper control flow is guaranteed. Finally, any synchronization edge connecting a node from N_a with another node of the process model must be reconnected to activity $BCDE$. Note that for the sake of readability, the labels of aggregated activities are concatenated to label the abstracted activity. The labeling of aggregated nodes is described in Section 6.3.3 since activity labels are described by process attributes.

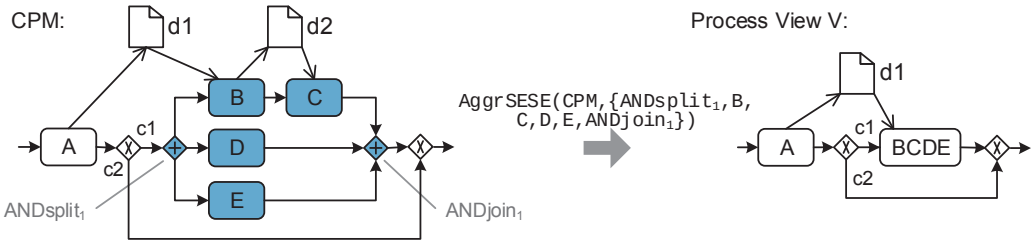


Figure 6.8: View Creation Operation AggrSESE

In general, an aggregation may affect the data flow of a process model as well. For example, the data edges that connect data elements with nodes in N_a must be reconnected with the abstracted activity. Moreover, data elements solely written or read by nodes from N_a are not relevant for the process view. This behavior corresponds to private variables in object-oriented programming, which are only accessed and visible within the respective class. Respective data

elements and their data edges are hidden in the process view as well. In Figure 6.8, for example, data element $d1$ is kept in the process view as it is written by activity A, which is still contained in the process view. By contrast, data element $d2$ is removed in the process view as it is only accessed by the aggregated process nodes.

Algorithm 6.3 formally specifies view creation operation *AggrSESE*. First, the set of process nodes is updated (cf. Line 3), i.e., the nodes to be aggregated are removed from the set of nodes and the abstracted node n_{new} is inserted instead. In Line 7, $updateControlFlowEdges$ is called. It removes control edges connected with nodes in N_a and it adds new control edges that connect node n_{new} with its new predecessor and successor. If applicable, synchronization edges are reconnected with node n_{new}. Lines 8-27 show how the data flow is updated. First, the data elements associated with nodes from N_a are determined. Then, for each of these data elements it is checked whether it is solely written by nodes from N_a. For this case, the data element is removed. Otherwise, it is determined whether all branches of XOR branchings induced by node set N_a access the respective data element or not. In case all branches access the data element, a mandatory data access between the aggregated node and the data element is added. Otherwise, an optional data access is added.

In Figure 6.9, for example, data element $d1$ is read by both branches (i.e., the two branches induced by $\{B, C\}$ and $\{D\}$ respectively). To be more precise, no matter which branch is chosen at run-time, $d1$ will be always read. Hence, $d1$ is connected through a *mandatory* read data edge with the abstracted activity BCD in process view V. By contrast, $d2$ is only read by one XOR branch (i.e., the branch containing D). Therefore, an *optional* data edge is used to connect $d2$ with BCD in V.

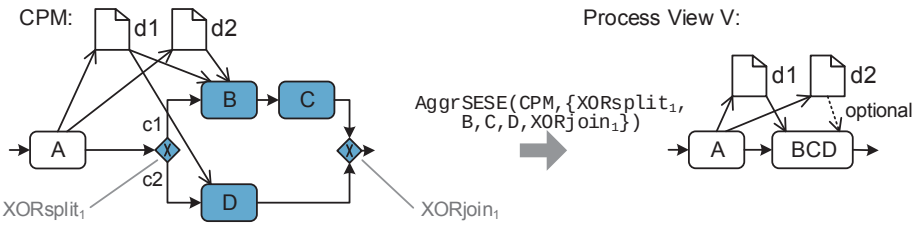


Figure 6.9: View Creation Operation AggrSESE Affecting Data Flow

Finally, Line 29 shows how obsolete data flow edges are removed. Note that a list of supportive functions used in Algorithm 6.3 together with their description, is provided in Appendix A.

Algorithm 6.3: $AggrSESE(CPM, N_a)$

```
   Input:  Process model CPM = (N, D, NT, CE, EC, ET, DE, DET)
1          SESE block described by node set N_a to be aggregated
   Output: Process view V = (N', D', NT', CE', EC', ET', DE', DET')
2  begin
3      V := InitializeView(CPM);
4      N' := N' \ N_a ∪ {n_new}; // n_new represents the abstracted node
5      NT'(n_new) := Activity;
6      // removes control flow and synchronization edges connecting the SESE,
7      // adds control and synchronization edges connecting node n_new
8      CE' := updateControlFlowEdges(CE', N_a, n_new);
9      D_a := {d ∈ D' | ∃de ∈ DE' : de = (d, •) ∨ de = (•, d)};
10     forall (d ∈ D_a) do
11         // checks whether data edge d is accessed (i.e., read or written)
12         // by process nodes not in N_a
13         if isOnlyAccessedInSet(d, DE', N_a) then
14             D' := D' \ {d};
15         else
16             if (isReadAccess(d, DE', N_a)) then
17                 DE' := DE' ∪ {(d, n_new)};
18                 // checks whether data element d is accessed by all branches
19                 // in the SESE
20                 if (accessedByAllTraces(d, DE', N_a)) then
21                     DET'((d, n_new)) := mandatory;
22                 else
23                     DET'((d, n_new)) := optional;
24                 end
25             end
26             if (isWriteAccess(d, DE', N_a)) then
27                 // analogous for write access
28             end
29         end
30         DE' := DE' \ {de ∈ DE' | ∃n ∈ N_a : de = (d, n) ∨ de = (n, d)};
31     end
32     return V;
33 end
```

$AggrComplBranches(CPM, N_a)$. This view creation operation aggregates multiple branches of a given branching to one branch with exactly one abstracted activity (cf. Figure 6.10a). The operation is applied, for example, if a user does not want to aggregate the complete branching, but only selected branches; e.g., an XOR branching of a business process may consist of one standard branch and several exceptional branches. A process view may then aggregate all exceptional branches. Data flow is treated the same way as in the context of $AggrSESE$.

The behaviour of *AggrComplBranches* is similar to the one of *AggrSESE*. In particular, N_a must comprise all activities of the branches to be replaced by a single branch with one abstracted activity. To be more precise, as can be seen in the following preconditions, N_a corresponds to a disjoint union of node sets N_{ai} $(i = 1 \ldots k)$. Each of these node sets comprises the activities of a particular branch that, in turn, which constitutes a SESE block itself. Furthermore, all node sets N_{ai} have the same predecessor (successor), which is the split (join) gateway of corresponding branching block.

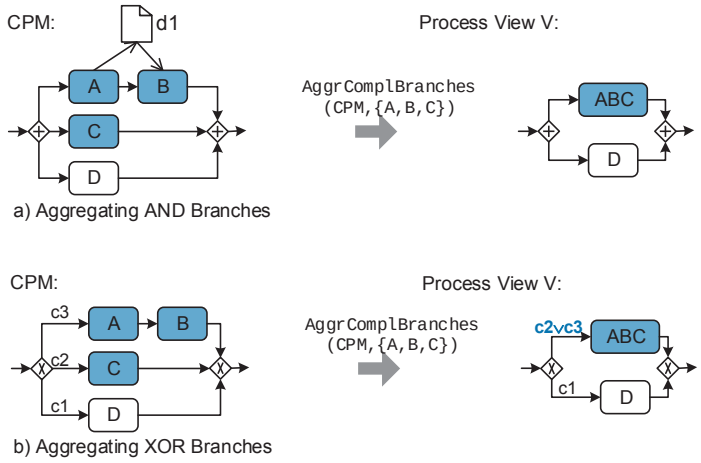


Figure 6.10: Operation AggrComplBranches

Preconditions $AggrComplBranches(CPM, N_a)$.
Let $CPM = (N, D, NT, CE, EC, ET, DE, DET)$ *be a process model. Further, let* $N_a \subset N$ *a set of nodes to be aggregated. Then: The following preconditions must be met in order to apply* $AggrComplBranches(CPM, N_a)$.

- *Node set* $N_a = N_{a1} \dot\cup N_{a2} \dot\cup \ldots \dot\cup N_{ak}$, $k \in \mathbb{N}$
- $\forall i = 1, \ldots, k$: *subgraph induced by node set* N_{ai} *is a SESE block of* CPM
- $\forall N_{ai} : pred(CPM, N_{ai}) =: g_s$ *with* $NT(g_s) \in \{ANDsplit, XORsplit\}$
- $\forall N_{ai} : succ(CPM, N_{ai}) =: g_j$ *with* $NT(g_j) \in \{ANDjoin, XORjoin\}$

Lines 2-6 of Algorithm 6.4 show how the node set of the process model is updated and the new activity is connected with the corresponding split and join gateway. In Line 8, the branching condition of control edge e', which connects the split gateway with activity n_{new}, is set by using logical disjunction (cf. Figure 6.10b). In case of an AND branching, the branching condition of each branch is $TRUE$ and, therefore, the new branching condition is $TRUE$ as well. Function $updateControlFlowEdges$ removes control edges connecting nodes in N_a and reconnects synchronization edges, if applicable. Data flow is handled as set out in Algorithm 6.3 (cf. Lines 8-29).

Algorithm 6.4: $AggrComplBranches(CPM, N_a)$

Input: Process model $CPM = (N, D, NT, CE, EC, ET, DE, DET)$
1 Node set $N_a = N_{a1} \dot\cup N_{a2} \dot\cup \ldots \dot\cup N_{ak}$, $k \in \mathbb{N}$=number of branches to be aggregated.
Output: Process view $V = (N', D', NT', CE', EC', ET', DE', DET')$
2 **begin**
3 $V := InitializeView(CPM)$;
4 $N' := N' \cup \{n_{new}\} \setminus N_a$; $NT'(n_{new}) := Activity$;
5 `// Connecting` n_{new} `with the split/join gateway`
6 $e' := (pred(CPM, N_{a1}), n_{new})$; $e'' := (n_{new}, succ(CPM, N_{a1}))$;
7 $CE' := CE' \cup \{e', e''\}$;
8 `// Updating branching condition of split gateway`
9 $EC'(e') := EC'((pred(CPM, N_{a1}), first(CPM, N_{a1}))) \vee \ldots \vee EC'((pred(CPM, N_{ak}), first(CPM, N_{ak})))$;
10 $CE' := updateControlFlowEdges(CE', N_a, n_{new})$;
11 `// adaptation of data flow like in Algorithm 6.3, Lines 8-29`
12 `// ...`
13 **return** V;
14 **end**

In the following, aggregation operations related to the data perspective of a process model are presented.

AggrDataElements(CPM, D_a). This view creation operation aggregates a set of data elements to an abstract data element (cf. Figure 6.11). As example consider a health care process for which a set of data elements related to a particular patient (e.g., name, birth date, address) shall be combined to one abstracted patient data element. In general, *AggrDataElements*(CPM, D_a) removes all data elements from set $D_a \subseteq D$ and re-inserts an abstract data element d_{new} instead. Furthermore, the corresponding data edges are reconnected with the new data element. Afterwards, the edge type of the new data edge is updated; e.g., aggregating both *optional* and *mandatory* read data elements results in a *mandatory* abstract read data element (e.g., data edge $(d1d2, B)$ in Figure 6.11b). Aggregating two optional read data elements, in turn, results in an optional read data edge (e.g., data edge $(d1d2, C)$). In particular, the resulting data edges do not express which elementary data element is accessed.

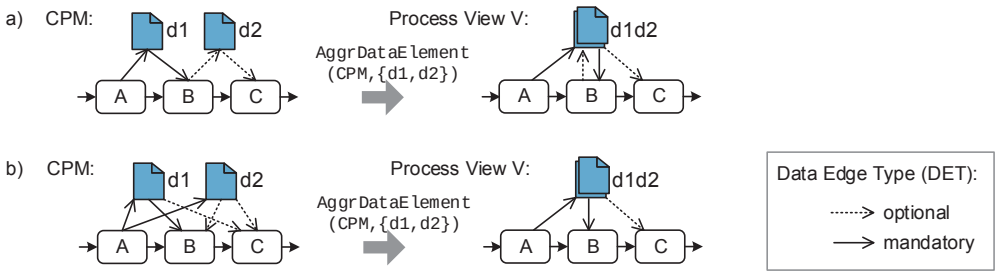


Figure 6.11: View Creation Operation AggrDataElement

Algorithm 6.5: *AggrDataElement*(CPM, D_a)

Input: Process model $CPM = (N, D, NT, CE, EC, ET, DE, DET)$
Set of data elements D_a to be aggregated
Output: Process view $V = (N', D', NT', CE', EC', ET', DE', DET')$
begin
  $V := InitializeView(CPM)$;
  // removes data elements to aggregate; adds data element d_{new}
  $D' := D' \setminus D_a \cup \{d_{new}\}$;
  $DE_r := \{de \in DE' | \exists d \in D_a : de = (d, \bullet)\}$; $DE_w := \{de \in DE' | \exists d \in D_a : de = (\bullet, d)\}$;
  // write access
  forall $(n, d) \in DE_w$ **do**
    if $((DET'((n, d_{new})) = mandatory) \vee (DET'((n, d)) = mandatory))$ **then**
      $DET'((n, d_{new})) := mandatory$;
    else
      $DET'((n, d_{new})) := optional$;
    end
    $DE' := DE' \cup \{(n, d_{new})\} \setminus \{(n, d)\}$;
  end
  // read access
  forall $(d, n) \in DE_r$ **do**
    ... // analogous for read access
  end
  return V;
end

Algorithm 6.5 describes operation *AggrDataElement*. Initially, the data element set is updated (Line 2). Afterwards, data edges are reconnected and their types are updated (Lines 3-17).

6.3.3 View Operations for Handling Process Attributes

In general, process elements constitute complex objects characterized by a number of *process attributes*. For example, a particular attribute of an activity may reflect the costs to execute the activity. When applying view creation operations, process attributes need to be considered as well. As example consider Figure 6.12: activities A, B, and C are aggregated to one abstracted activity ABC. Thereby, the attributes of the respective activities must be aggregated as well. For this purpose, several *attribute functions* are provided; e.g., *CONCAT* concatenates the values of textual attributes as applied for labels of aggregated activities.

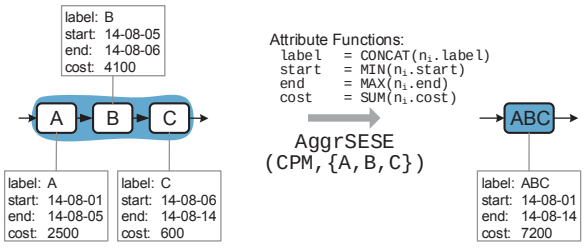


Figure 6.12: Aggregation of Attributes

In order to also cover attributes, we extend our definition of process models (cf. Definition 6.1). Referring to Definition 6.8, we illustrate the application of attribute functions.

Definition 6.8 (Process Model with Attributes)
Let $\mathcal{AS}$ be the set of all possible attributes of process elements. Then: A process model is defined as tuple $P = (N, D, NT, CE, EC, ET, DE, DET, attr, val)$ with:

- *N, D, NT, CE, EC, ET, DE, and DET corresponding to the notions from Definition 6.1.*
- *$attr : N \dot\cup D \dot\cup CE \dot\cup DE \rightarrow \mathcal{AS}$ assigns to each process element an attribute set $AS \subseteq \mathcal{AS}$.*
- *$val : (N \dot\cup D \dot\cup CE \dot\cup DE) \times \mathcal{AS} \rightarrow valueDomain(\mathcal{AS})$ assigns to attribute $x \in \mathcal{AS}$ of process element $elem \in (N \dot\cup D \dot\cup CE \dot\cup DE)$ a respective value or null:*

$$val(elem, x) \equiv \begin{cases} value(x)^1, & x \in attr(elem) \\ null^2, & x \notin attr(elem) \end{cases}$$

[1] $value(x)$ denotes the value of process attribute x
[2] attribute is not available for the respective process element

Furthermore, $elem.x$ refers to process attribute x of process element $elem$. For example, $A.label$ refers to attribute $label$ of activity A.

In general, a view creation operation may abstract from process attributes in an *explicit* or *implicit* way. *Explicit* abstraction aggregates or reduces the process attributes of a particular process element by applying view creation operations *ReduceAttr* or *AggrAttr*. By contrast, an *implicit* abstraction of process attributes occurs when applying aggregation operations to the control or data flow (e.g., *AggrSESE*). In the following, these view creation operations will be discussed in more detail:

ReduceAttr(CPM,elem,attr). This view creation operation reduces attribute *attr* of process element *elem* in the resulting process view, i.e., a specific attribute shall *not* be displayed to a particular user (e.g., for privacy reasons). For example, in Figure 6.13 attribute $A.z$ is hidden in process view V.

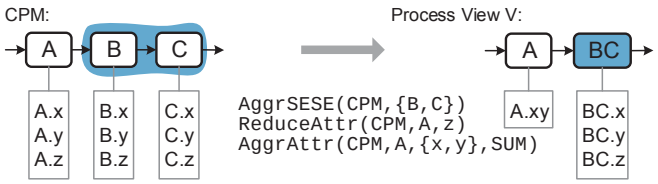


Figure 6.13: View Creation Operations Dealing with Process Attributes

Algorithm 6.6 summarizes *ReduceAttr(CPM,elem,attr).* Particularly, in Line 5 attribute *attr* of process element *elem* is reduced in the resulting process view.

Algorithm 6.6: $ReduceAttr(CPM, elem, attr)$

```
   Input:  Process model CPM = (N, D, NT, CE, EC, ET, DE, DET, attr, val)
1          Attribute attr of process element elem ∈ N ∪̇ D ∪̇ CE ∪̇ DE to be reduced.
   Output: Process view V = (N', D', NT', CE', EC', ET', DE', DET', attr', val')
2  begin
3      V := InitializeView(CPM);
4      AS := attr(elem); // get the set of attributes AS of process element elem
5      //removes attribute attr in set AS
6      attr'(elem) := AS \ {attr};
7      return V;
8  end
```

$AggrAttr(CPM, elem, AS_a, attrFunc)$. This operation aggregates a set of attributes $AS_a = \{attr_1, \ldots, attr_k\}$ of process element $elem$ to an abstracted process attribute based on given attribute function $attrFunc$ (cf. Algorithm 6.7). In the resulting process view, therefore, all attributes from AS_a are removed from $elem$ and an aggregated attribute $attr_{new}$ is added (cf. Algorithm 6.7, Lines 2-4). Based on the values of the attributes from AS_a the new value of $attr_{new}$ is derived by applying $attrFunc$. Depending on the type of the attribute to be aggregated, different attribute functions may be applied. Table 6.1 gives an overview of selected attribute functions, e.g., `SUM` is defined on numerical attributes. It returns the sum of selected attribute values (cf. Figure 6.13).

Algorithm 6.7: $AggrAttr(CPM, elem, AS_a, attrFunc)$

Input: Process model $CPM = (N, D, NT, CE, EC, ET, DE, DET, attr, val)$
1 Set of attributes $AS_a = \{attr_1, \ldots, attr_k\}$ of process element $elem \in N \dot\cup D \dot\cup CE \dot\cup DE$ that shall be be aggregated using function $attrFunc$.
Output: Process view $V = (N', D', NT', CE', EC', ET', DE', DET', attr', val')$
2 **begin**
3 $V := InitializeView(CPM)$;
4 $AS := attr(elem)$; `// get the set of attributes` AS `of process element` $elem$
5 `//remove attributes in set` AS_a`; add abstracted attribute` $attr_{new}$
6 $attr'(elem) := AS \setminus AS_a \cup \{attr_{new}\}$;
7 `//calculating new attribute value`
8 $val'(attr_{new}) := attrFunc(\{val(attr_1), \ldots, val(attr_k)\})$;
9 **return** V;
10 **end**

Note that operation *AggrAttr* neither affects the control nor the data flow of a process model and process view, respectively.

Attribute Functions for Numeric Values		Attribute Functions for Textual Values	
SUM	Sum of attribute values	CONCAT	Concatenation of attribute values
AVG	Average of attribute values	FIRST	First value of attributes applied
MIN	Minimum of attribute values	LAST	Last value of attributes applied
MAX	Maximum of attribute values	MAXFREQ	Most frequent attribute value
COUNT	Number of attributes	MINFREQ	Fewest frequent attribute value

Table 6.1: Attribute Functions for Attribute Values

Implicit Aggregation of Attributes. When aggregating nodes (e.g., when applying *AggrSESE*) the attribute values of the nodes to be aggregated must be aggregated as well. In Figure 6.13, for example, the values of attribute x of activities B and C must be aggregated when applying $AggrSESE(CPM, \{B, C\})$. A trivial solution may be to "copy" attributes $B.x$ and $C.x$ separately to the aggregated activity BC. Since this solution might not be appropriate, attribute functions (cf. Table 6.1), must be applied. However, for such an aggregation, default attribute functions are required in order to automatically aggregate respective attributes. Thereby, each type of attribute has a specific default function. For example, attribute *start (end) time* may be aggregated using the MIN (MAX) attribute function (cf. Table 6.1).

When aggregating branches of an XOR branching (i.e., *AggrComplBranches*), it must be taken into account that only one branch will be executed. In Figure 6.14, for example, the branches containing A and B are aggregated. When aggregating the values of the related attribute *cost* based on attribute function SUM, we obtain attribute value $AB.cost = 3000$. However, since only one branch will be executed, the calculated value is not appropriate.

Moreover, in the CPM from Figure 6.14, the highest possible value of attribute *cost* will be 2000 when selecting the branch containing B. To provide a more realistic representation, *expected* attribute values should be calculated, i.e., each branch will be executed with the same probability at run-time. Consequently, in our example from Figure 6.14, an attribute value of $AB.cost = (2000 + 1000)/2 = 1500$ results. If the probability for selecting a particular branch is known (e.g., based on historical run-time information), however, a more realistic aggregated attribute value can be calculated. However, in certain scenarios it might be required to apply

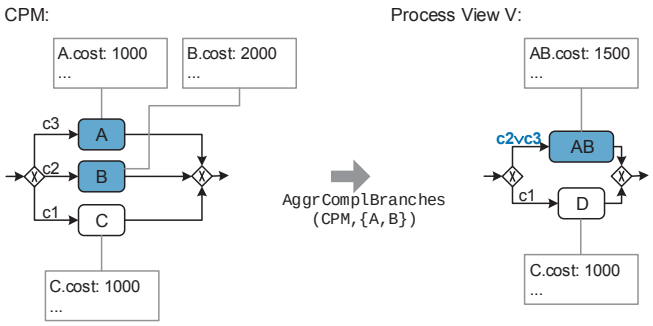


Figure 6.14: Combining Operations AggrComplBranches and Attribute Aggregation

attribute functions in order to provide a worst/best case perspective in the process view. Regarding textual attribute values, the branching probability may be taken into account as well; e.g., to display only attributes values of the branch chosen most frequently.

Attribute functions may not only perform simple calculations (e.g., *SUM*), but more complex ones as well (e.g., to find an appropriate activity label after applying aggregation operation *AggrSESE* or *AggrComplBranches*). For the sake of simplicity, we have applied function *CONCAT* when aggregating activity labels, e.g., when aggregating activities labelled "A", "B", and "C", the abstracted activity is labelled "ABC". Obviously, when aggregating activities with real activity labels (like "Print Letter" or "Send Letter"), such a simple concatenation results in a long activity label that might be hard to comprehend.

Addressing this issue, a (custom) attribute function could match the activity labels to extract similarities, which are then used as new value for attribute *label* [97, 122]. In the example above, a match of "Print Letter" and "Send Letter" will return "Letter". However, such an attribute function might be useless, if there is no match between aggregated activity labels or the match is not meaningful enough (e.g., like the articles "the" and "a").

As an alternative, an attribute function handling activity labels may rely on aggregations made in the context of other process views. To be more precise, when aggregating activity set N_a in process view $V1$, which (or a sub-/superset of it) has already been aggregated in process view $V2$, the label of the aggregated activity in $V2$ is applied to the one in $V1$. If N_a is not aggregated in other process views, the user can be asked to provide a value for attribute *label* of the aggregated node.
Such a behaviour increases the expressiveness of aggregated node labels in process views and introduces an example of a more complex and intelligent attribute function.

6.4 Process Model Refactorings

Applying view creation operations to a CPM might result in unnecessarily complex control flow structures due to the generic nature of the operations applied. For example, single branches of a parallel branching might become empty or a parallel branching might have only one remaining

branch after applying a number of reduction operations. In such scenarios, AND gateways no longer required should be removed in order to obtain a more comprehensible control flow structure of the process view without affecting its behaviour. In particular, simplifying a process model reduces its cognitive complexity [123]. Figure 6.15 shows the creation of a process view V by applying view creation operation *RedActivity* to activities B, C, and G. The resulting view contains an empty AND branching in parallel to D and E as well as an empty AND branch organized in parallel to H. Both can be removed to simplify the structure of the process view. As it can be seen in Figure 6.15, the refactored model of V is more compact compared to the original one.

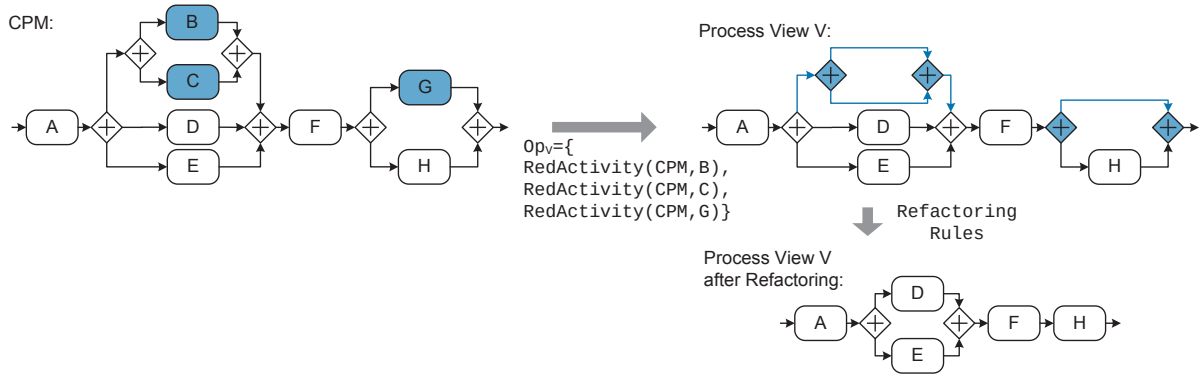


Figure 6.15: Example of Applying Refactoring Rules

This section introduces *refactoring rules* that allow simplifying the structure of a process model. Furthermore, such refactorings must not change the behaviour of the respective process model. In particular, refactoring rules may be applied on both a CPM and on the models of related process views. In order to describe the behaviour of a process model and to show that the applied refactorings are behaviour-preserving, first of all, we introduce the notion of *execution trace* (*trace* for short). Thereby, a trace describes events related to the execution of a process model (e.g., start/end events of activities). Moreover, a trace describes a *valid* and *complete* execution sequence regarding a given process model. Definition 6.9 formally introduces the notion of *execution trace* [124].

Definition 6.9 (Execution Trace)
Let $P = (N, D, NT, CE, EC, ET, DE, DET)$ be a process model and t a sequence of activities with $t = \langle a_1, a_2, \ldots, a_k \rangle, a_i \in N \wedge NT(a_i) = Activity$. Then, t constitutes a trace of P iff:

- *t is valid, i.e., the given execution sequence reflects a possible temporal order in which activities may be completed on P.*
- *t is complete, i.e., a_1 is executed immediately after completing the StartFlow node, and a_k is executed immediately before executing the EndFlow node of P.*

Furthermore, we define $\mathcal{T}_P$ as the set of all traces producible by a process model P.

Consider process model $P1$ from Figure 6.16a: traces $t_{P1.1} = \langle A, B, C, E \rangle$ and $t_{P1.2} = \langle A, D, E \rangle$ can be both produced on $P1$. As known from process mining [125], we assume that the behaviour of a process model P can be described in terms of its set of traces $\mathcal{T}_P$.

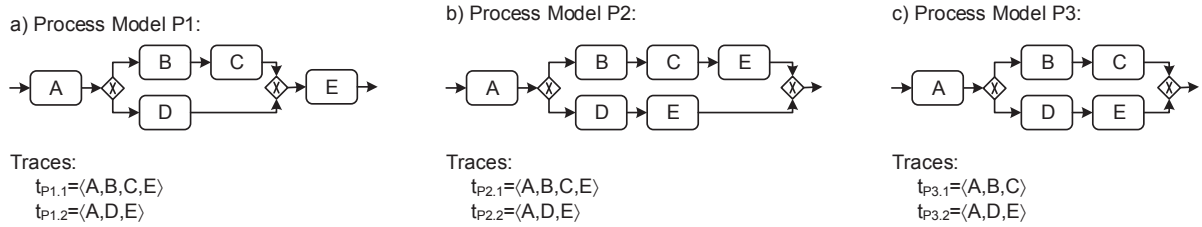


Figure 6.16: Trace Equivalence of Process Models

Two process models are called *trace equivalent*, if and only if both are able to produce the same set of execution traces [126, 67, 127]. Note that trace equivalence can also be shown for process models containing loops. In particular, trace equivalence does not mean that process models are structurally equal (i.e., graph equivalent [128]). However, they show the same (execution) behaviour. Note that graph equivalence is not appropriate in our case as refactoring rules intend to change the structure of a process model.

In Figure 6.16, for example, process models $P1$ and $P2$ are trace equivalent since both are able to produce traces $\langle A, B, C, E \rangle$ and $\langle A, D, E \rangle$. By contrast, $P3$ is neither trace equivalent to $P2$ nor to $P1$. A formal description of this notions is provided by Definition 6.10.

Definition 6.10 (Trace Equivalence)
Two process models $P1$ and $P2$ are trace equivalent iff for the corresponding set of executions traces $\mathcal{T}_{P1}$ and $\mathcal{T}_{P2}$ on process models $P1$ and $P2$, respectively, it holds $\mathcal{T}_{P1} \equiv \mathcal{T}_{P2}$.

In order to show that refactoring rules preserve the behaviour of a process model, trace equivalence has to be checked between the process model before and after applying refactoring rules.

Section 6.4.1 introduces refactoring rules $RR1$-$RR5$ that remove empty branches or simplify unnecessary empty branchings. Section 6.4.2 defines refactorings simplifying consecutive gateways of the same type (i.e., $RR6$-$RR8$). Section 6.4.3 deals with refactoring rules $RR9$-$RR10$ to simplify process structures synchronized through synchronization edges. Section 6.4.4 presents refactoring rule $RR11$ removing unconnected data elements. Section 6.4.5 discusses trace equivalence of process models before and after appyling refactoring rules.

6.4.1 Refactoring Empty Gateways and Branchings

Figure 6.17 shows an example of an AND branching with an empty branch. This branch may be a leftover of a previously applied reduction operation. Since the empty branch is no longer required, the process model may be simplified using refactoring rule RR1 (cf. Figure 6.17).

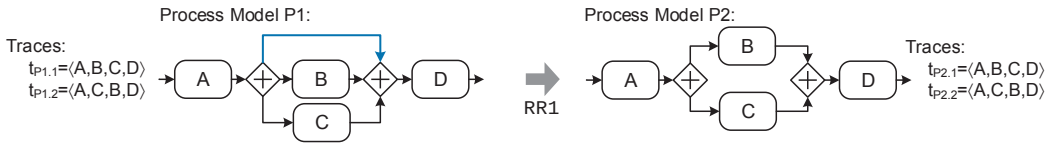


Figure 6.17: Refactoring Rule RR1

Refactoring rule RR1 removes an empty AND branch of an AND branching to obtain a more comprehensive model. Note that RR1 is comparable to refactoring RF10 as presented in [18, 67]. As can be seen in Figure 6.17, both process models $P1$ and $P2$ have the same trace set (i.e., $\mathcal{T} = \{\langle A, B, C, D\rangle, \langle A, C, B, D\rangle\}$). Hence, they are trace equivalent.
While refactoring rule RR1 only removes empty AND branches, refactoring rule RR5 may be applied to empty XOR branches. In particular, in the context of XOR branchings and Loops empty branches must not be simply removed in order to ensure control flow correctness. If the XOR branching is empty, i.e., all branches are empty, RR3 removes the respective block as well. In particular, RR1 is applied, if an ANDsplit gateway g_s and an ANDjoin gateway g_j exist as well as a direct edge (g_s, g_j) (i.e., an empty branch) between them. Furthermore, there must be another non-empty branch. Otherwise, a non-connected process model might result. Then, the empty branch is removed from the process model.

Refactoring Rule RR1 (Empty AND Branch)
Let $P = (N, D, NT, CE, EC, ET, DE, DET)$ be a process model.
***Precondition**:*
$\exists g_s, g_j \in N : \ NT(g_s) = ANDsplit \wedge NT(g_j) = ANDjoin \wedge (g_s, g_j) \in CE;$
$\exists n \in N : (g_s, n) \in CE \wedge n \neq g_j \wedge ET((g_s, n)) = ET\_Control;$

***Postcondition**[1]:*
$CE := CE \setminus \{(g_s, g_j)\};$

Refactoring rule RR2 removes unnecessary AND branchings. For example, the AND branching surrounding activity B in Figure 6.18 is no longer needed since no other parallel branches exist anymore. Note that the depicted process model is valid according to Definition 6.1. It may result when applying RR1 or deleting AND branches.

Figure 6.18: Refactoring Rule RR2

Hence, RR2 removes the ANDsplit and ANDjoin gateways and replaces them by control edges that re-connect the respective preceding and succeeding nodes of the respective process view.

[1]Note that *postcondition* describes the effect on process model P when applying the refactoring rule.

Refactoring Rule RR2 (Unnecessary AND Block)
Let $P = (N, D, NT, CE, EC, ET, DE, DET)$ be a process model. Further, let g_s be an ANDsplit gateway with corresponding ANDjoin gateway g_j.
***Precondition**:*
$\exists! n \in N : (g_s, n) \in CE,\ NT(n) \neq ANDjoin \wedge ET((g_s, n)) = ET\_Control;$

***Postcondition**:*
$n_{s1} := pred(P, g_s);\ n_{s2} := succ(P, g_s);\ n_{j1} := pred(P, g_j);\ n_{j2} := succ(P, g_j);$
$e' := (n_{s1}, n_{s2});\ e'' := (n_{j1}, n_{j2});\ ET(e') := ET(e'') := ET\_Control;$
$EC(e') := EC((n_{s1}, g_s));$
$CE := CE \setminus \{(n_{s1}, g_s), (g_s, n_{s2}), (n_{j1}, g_j), (g_j, n_{j2})\} \cup \{e', e''\};$
$N := N \setminus \{g_s, g_j\};$

Since RR2 removes a pair of ANDsplit and ANDjoin gateways, which are solely connected by a single branch, the set of producible traces does not change. Note that gateways are not represented in an execution trace.

An empty AND or XOR branching (i.e., all branches are empty) may be removed completely (cf. Figure 6.19), i.e., the respective split and join gateways may be removed from the process model. In this scenario, no differentiation needs to be made between AND/XOR branchings.

Figure 6.19: Refactoring Rule RR3

Refactoring rule RR3 removes empty AND or XOR branchings as well as respective control edges. In turn, loops need to be handled separately (cf. refactoring rule RR4). Since an empty branching block has no effect on the traces producible by a process model, it can be removed without changing the behaviour of a process model (i.e., trace equivalence is ensured).

Refactoring Rule RR3 (Empty AND/XOR Branching Block)
Let $P = (N, D, NT, CE, EC, ET, DE, DET)$ be a process model. Further, let g_s be a split gateway (i.e., ANDsplit or XORsplit) with corresponding join gateway g_j (i.e., ANDjoin or XORjoin).
***Precondition**:*
$\exists! (g_s, g_j) \in CE\ \wedge \nexists n \in N : (g_s, n) \in CE \wedge n \neq g_j \wedge ET((g_s, g_j)) = ET\_Control;$

***Postcondition**:*
$n_s := pred(P, g_s); n_j := succ(P, g_j);$
$e' := (n_s, n_j);\ ET(e') := ET\_Control;\ EC(e') := EC((n_s, g_s));$
$CE := CE \setminus \{(g_s, g_j), (n_s, g_s), (g_j, n_j)\} \cup \{e'\};$
$N := N \setminus \{g_s, g_j\};$

Similar to RR3, an empty loop may be removed (cf. Figure 6.20). As opposed to an AND/XOR branching, for loops, the split gateway (i.e., LOOPsplit) succeeds the join gateway (i.e., LOOPjoin). Hence, an empty loop refers to a loop block, which does not contain any activity between its join and split gateways.

Figure 6.20: Refactoring Rule RR4

Refactoring rule RR4 removes empty loops, including corresponding control and loop edges (i.e., $ET(e) = ET\_Loop$). In the context of loops, however, it must be taken into account that there exists another edge pointing from the "second" gateway (i.e., LOOPsplit) back to the "first" one (i.e., LOOPjoin). Similar to RR3, trace equivalence is ensured since no activities are enclosed within the loop anymore.

Refactoring Rule RR4 (Empty Loop Block)
Let $P = (N, D, NT, CE, EC, ET, DE, DET)$ be a process model. Further, let g_s be a LOOPsplit gateway with corresponding LOOPjoin gateway g_j.
***Precondition**:*
$\nexists n \in N : (g_j, n) \in CE \wedge n \neq g_s \wedge ET((g_j, n)) = ET\_Control;$

***Postcondition**:*
$n_s := succ(P, g_s);\ n_j := pred(P, g_j);$
$e' := (n_j, n_s);\ ET(e') := ET\_Control;\ EC(e') := EC((n_j, g_j));$
$CE := CE \setminus \{(g_s, g_j), (g_j, g_s), (g_s, n_s), (n_j, g_j)\} \cup \{e'\};$
$N := N \setminus \{g_s, g_j\};$

As discussed, RR1 cannot be applied to XOR branches as the resulting process model would violate control flow correctness (cf. Definition 6.3). Figure 6.21 shows how to deal with multiple empty XOR branches in order to reduce the complexity of a process view, while guaranteeing control flow correctness. In particular, the branching conditions (e.g., r and s in Figure 6.21) of the empty XOR branches need to be concatenated with logical disjunction operators.

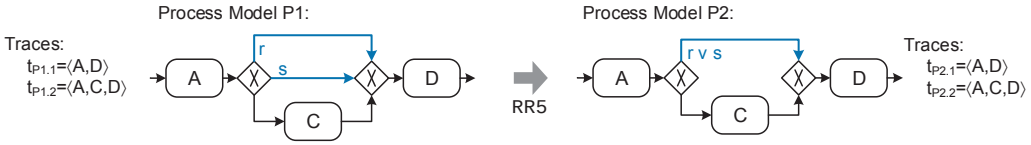


Figure 6.21: Refactoring Rule RR5

Refactoring rule RR5 allows performing such a union of empty XOR branches. Note that this refactoring rule may be only applied if two or more empty XOR branches are present. Compared to RR1, when refactoring multiple empty XOR branches, always an empty branch remains.

Otherwise control flow correctness is violated. Moreover, since the branches the refactoring rule is applied to are empty, RR5 has no effect on the traces producible by a process model, i.e., trace equivalence is ensured.

Refactoring Rule RR5 (Empty XOR Branches)
Let $P = (N, D, NT, CE, EC, ET, DE, DET)$ be a process model and g_s be an XORsplit gateway with corresponding XORjoin gateway g_j.
***Precondition**:*
$CE_{XOR} = \{e \in CE \mid e = (g_s, g_j)\}$ *with* $\mid CE_{XOR} \mid \geq 2$;

***Postcondition**:*
$e_{new} := (g_s, g_j)$;
$EC(e_{new}) := \bigvee_{e_i \in CE_{XOR}} EC(e_i)$;
$CE := CE \setminus CE_{XOR} \cup \{e_{new}\}$;

6.4.2 Refactoring Connected Branchings

Nested branchings of the same type are denoted as *connected branchings* if no activities are enclosed between the respective split (join) gateways. More precisely, the direct predecessor of the inner split gateway must be the outer split gateway. Similarly, the direct successor of the inner join gateway must be the outer join gateway (cf. Figure 6.22). Without loss of generality, we restrict ourselves to two kinds of nested branchings .

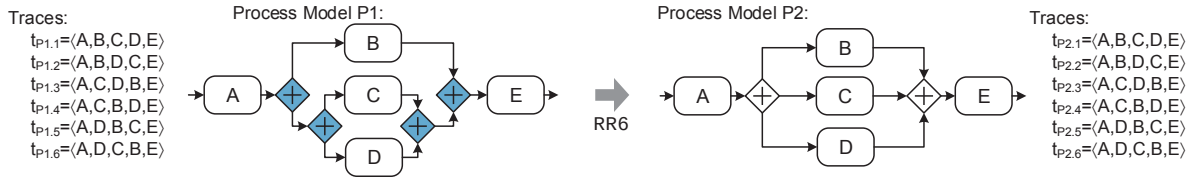


Figure 6.22: Refactoring Rule RR6

Refactoring rule RR6 combines gateways of connected AND branchings in such a way that only one split and one join gateway remains, i.e., it refactors pairs of gateways. If there are more than two connected branchings, RR6 is applied recursively. As RR6 combines a pair of ANDsplit and a pair of ANDjoin gateways, the set of traces is not affected. In Figure 6.22, for example, activity A is executed first; afterwards, one may execute B, C, and D in parallel (i.e., arbitrary order). Finally, E must be executed. When applying RR6, this behaviour is not changed.

Refactoring Rule RR6 (Connected AND Branchings)
Let $P = (N, D, NT, CE, EC, ET, DE, DET)$ be a process model. Further, let g_{s1} and g_{s2} be ANDsplit gateways with corresponding ANDjoin gateways g_{j1} and g_{j2}, respectively. Furthermore, let g_{s2} and g_{j2} represent an AND branching nested with the AND branching represented by g_{s1} and g_{j1}.
***Precondition**:*
$\exists(g_{s1}, g_{s2}) \in CE \ \wedge \ \exists(g_{j1}, g_{j2}) \in CE \wedge ET((g_{s1}, g_{s2})) = ET((g_{j1}, g_{j2})) = ET\_Control;$

***Postcondition**:*
$CE_{old} := \{(g_{s1}, g_{s2}), (g_{j1}, g_{j2})\} \cup \{(g_{s2}, n) \mid n \in succ(P, g_{s2})\} \cup$
$\{(n, g_{j2}) \mid n \in pred(P, g_{j2})\};$
$CE_{new} := \{(g_{s1}, n) \mid n \in succ(P, g_{s2})\} \cup \{(n, g_{j1}) \mid n \in pred(P, g_{j2})\};$
$CE := CE \setminus CE_{old} \cup CE_{new};$
$N := N \setminus \{g_{s2}, g_{j2}\};$

Similar to RR6, refactoring rule RR7 combines connected XOR branchings. Figure 6.23 shows an example of refactoring such connected XOR branchings. Particularly, when combining XORsplit gateways, branching conditions of the outgoing edges must be combined in order to guarantee control flow correctness.

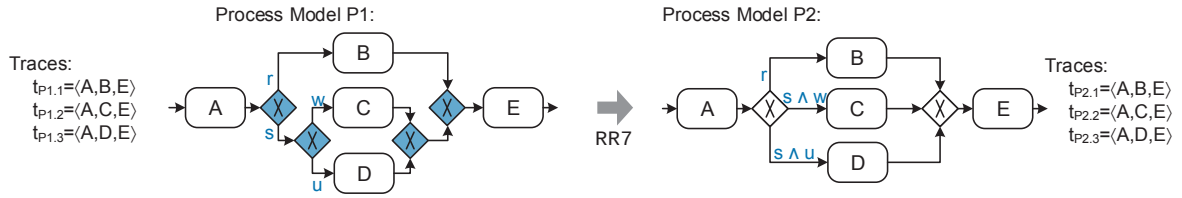


Figure 6.23: Refactoring Rule RR7

Accordingly, refactoring rule RR7 deals with connected XOR branchings. In contrast to refactoring rule RR6, the branching conditions for the added control edges must be set. The new branching conditions concatenate the original branching conditions between the first and second XORsplit gateway (i.e., g_{s1} and g_{s2}) with the one of the respective branch succeeding the second gateway (i.e., g_{s2}) using a logical AND operator (cf. Figure 6.23). This way, control flow correctness is further guaranteed.

Refactoring Rule RR7 (Connected XOR Branchings)
Let $P = (N, D, NT, CE, EC, ET, DE, DET)$ be a process model. Further, let g_{s1} and g_{s2} be XORsplit gateways with corresponding XORjoin gateway g_{j1} and g_{j2}.
Furthermore, let g_{s2} and g_{j2} represent an XOR branching nested with the XOR branching represented by g_{s1} and g_{j1}.
***Precondition**:*
$\exists(g_{s1}, g_{s2}) \in CE \ \wedge \ \exists(g_{j1}, g_{j2}) \in CE \wedge ET((g_{s1}, g_{s2})) = ET((g_{j1}, g_{j2})) = ET\_Control;$

***Postcondition**:*
$CE_{old} := \{(g_{s1}, g_{s2}), (g_{j1}, g_{j2})\} \cup \{(g_{s2}, n) \mid n \in succ(P, g_{s2})\} \cup$
$\{(n, g_{j2}) \mid n \in pred(P, g_{j2})\};$
$CE_{new} := \{(g_{s1}, n) \mid n \in succ(P, g_{s2})\} \cup \{(n, g_{j1}) \mid n \in pred(P, g_{j2})\};$
$CE := CE \setminus CE_{old} \cup CE_{new};$
$N := N \setminus \{g_{s2}, g_{j2}\};$
$\forall e := (g_{s1}, n) \in CE_{new}$: `//i.e., only the edges with` g_{s1} `as source.`
$EC(e) := EC((g_{s1}, g_{s2})) \ \wedge \ EC((g_{s2}, n));$

The set of traces producible by a process model is not changed when applying RR7, since gateways are not represented in a trace and the overall branching conditions are preserved. Consider the process model from Figure 6.23 before applying RR7. The first XORsplit either selects the branch containing B or the one with the nested XOR branching. In the latter case, either the branch containing C or the one containing D is chosen. Hence, only one of the activities B, C, and D will be executed. By contrast, A and E will be always executed when applying RR7, i.e., the same behaviour results (cf. Figure 6.23).

Finally, when dealing with connected branchings, nested loops must be taken into account as well. Figure 6.24 shows an example. Further, it demonstrates how RR8 simplifies respective structures without modifying the behaviour of the process model.

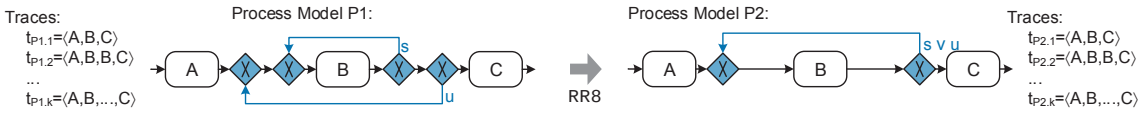


Figure 6.24: Refactoring Rule RR8

As opposed to RR7, the branching condition of the added loop edge must be concatenated using the logical OR operator, since the backward jump may be triggered by the first *or* second loop. This behaviour is defined by refactoring rule RR8. Again the process models before and after applying RR8 are trace equivalent (cf. Figure 6.24).

Refactoring Rule RR8 (Connected Loop Branchings)
Let $P = (N, D, NT, CE, EC, ET, DE, DET)$ be a process model. Further, let g_{s1} and g_{s2} be LOOPsplit gateways with corresponding LOOPsplit gateways g_{j1} and g_{j2}. Furthermore, the loop represented by gateways g_{s2} and g_{j2} is nested within the loop represented by g_{s1} and g_{j1}.
***Precondition**:*
$\exists(g_{s1}, g_{s2}) \in CE \ \wedge \ \exists(g_{j1}, g_{j2}) \in CE \wedge ET((g_{s1}, g_{s2})) = ET((g_{j1}, g_{j2})) = ET\_Control;$

***Postcondition**:*
$CE_{old} := \{(g_{s2}, g_{s1}), (g_{j1}, g_{j2}), (g_{s2}, g_{j2}), (pred(P, g_{s2}), g_{s2}), (g_{j2}, succ(P, g_{j2}))\};$
$CE_{new} := \{(pred(P, g_{s2}), g_{s1}), (g_{j1}, succ(P, g_{j2}))\};$
$CE := CE \setminus CE_{old} \cup CE_{new};$
$N := N \setminus \{g_{s2}, g_{j2}\};$
$EC((g_{s1}, g_{j1})) := EC((g_{s1}, g_{j1})) \vee EC((g_{s2}, g_{j2}));$
$EC((g_{s1}, succ(P, g_{s1}))) := \neg EC((g_{s1}, g_{j1}));$

6.4.3 Refactoring Synchronization Edges

Synchronization edges are used to synchronize the execution of activities belonging to different parallel branches of a process model. Figure 6.25 shows an example for which such a synchronization is no longer needed as A, B, and C are executed in sequential order. The following refactoring rules address process fragments, which result from the application of view update operations.

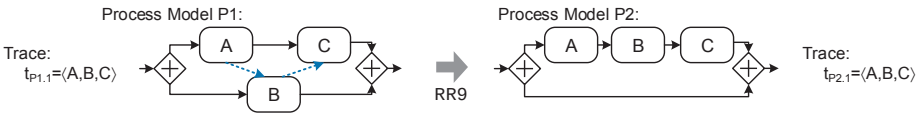


Figure 6.25: Refactoring Rule RR9

Consider Figure 6.25: A and C are connected through a control edge. Furthermore, A and B as well as B and C are connected through a synchronization edge. Consequently, B is executed directly after A and C cannot be executed before finishing B. Obviously, the three activities are ordered sequentially, i.e., exactly one trace $t = \langle A, B, C \rangle$ exists in the process model of Figure 6.25. Such a situation might occur in a process view due to the application of reduction operations. Refactoring rule RR9 inlines respective activities on parallel branches if no other control flow dependencies exist (e.g., an activity between the ANDsplit and B), while preserving the behaviour of the process model (i.e., trace equivalence is ensured). Note that the remaining empty AND branch may be refactored by applying RR1.

Refactoring Rule RR9 (Unnecessary Synchronization Edges)
Let $P = (N, D, NT, CE, EC, ET, DE, DET)$ be a process model and $CE' \subseteq CE$ be subset of control edges excluding synchronization edges, i.e., $\forall e \in CE' : ET(e) \neq ET\_Sync$.
***Precondition**:*
$\exists n_1, n_2, n_3 \in N :$
$(n_1, n_2), (n_2, n_3) \in CE \wedge$
$ET((n_1, n_2)) = ET((n_2, n_3)) = ET\_Sync \wedge$
$\exists (g_s, n_2), (n_2, g_j), (n_1, n_3) \in CE' : NT(g_s) = ANDsplit \ \wedge \ NT(g_j) = ANDjoin;$

***Postcondition**:*
$ET((n_1, n_2)) := ET((n_2, n_3)) := ET\_Control;$
$CE := CE \setminus \{(g_s, n_2), (n_2, g_j), (n_1, n_3)\} \cup \{(g_s, g_j)\};$

Similar to RR9, refactoring rule RR10 simplifies synchronization edges, which synchronize a set of activities in sequential order. In Figure 6.26, D and E are synchronized in sequential order. Although, both activities are located on parallel branches, synchronization edges enforce their sequential execution.

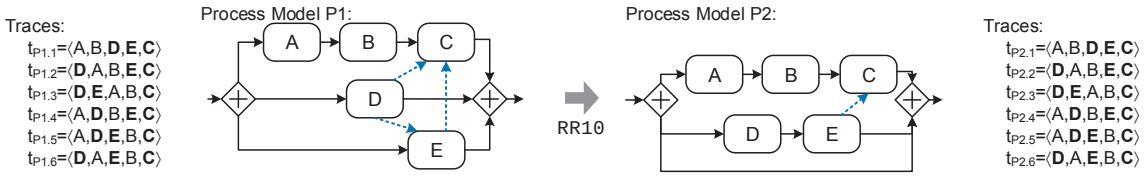


Figure 6.26: Refactoring Rule RR10

Refactoring rule RR10 removes the respective synchronization and inlines E after D. Figure 6.26 shows the set of possible traces for both process models, i.e., $P1$ and $P2$. Note that these are equivalent. Such a situation might occur after reducing activities, e.g., between D and the ANDjoin gateway. The remaining empty AND branch is simplified by refactoring rule RR1.

Refactoring Rule RR10 (Simplifying Synchronization Edges)
Let $P = (N, D, NT, CE, EC, ET, DE, DET)$ be a process model.
***Precondition**:*
$\exists n_1, n_2, n_3 \in N :$
$(n_1, n_2), (n_1, n_3), (n_2, n_3) \in CE \wedge$
$ET((n_1, n_2)) = ET((n_1, n_3)) = ET((n_2, n_3)) = ET\_Sync \wedge$
$\exists (g_s, n_2), (n_2, g_j), (n_1, g_j) \in CE : NT(g_s) = ANDsplit \wedge NT(g_j) = ANDjoin;$

***Postcondition**:*
$ET((n_1, n_2)) := ET\_Control;$
$CE := CE \setminus \{(g_s, n_2), (n_1, n_3), (n_1, g_j)\} \cup \{(n_2, g_j)\};$

6.4.4 Refactoring Data Elements

This section introduces refactoring rules addressing the data flow perspective. Figure 6.27 shows a data element no longer connected to any activity. This scenario results when reducing or deleting activities that have read or write access to the respective data element.

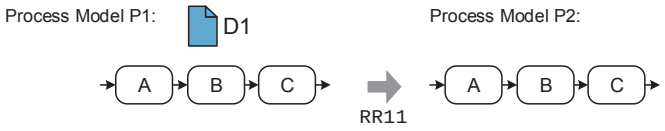


Figure 6.27: Refactoring Rule RR11

RR11 removes unconnected (or unused) data elements from a process model. Obviously, removing data elements not affects the set of traces producible by a process model.

Refactoring Rule RR11 (Unconnected Data Elements)
Let $P = (N, D, NT, CE, EC, ET, DE, DET)$ be a process model. Further, let $d \in D$ be a data element.
***Precondition**:*
$\forall n \in N : \nexists (d, n) \in DE \wedge \nexists (n, d) \in DE;$

***Postcondition**:*
$D := D \setminus \{d\}$

6.4.5 Discussion

Refactoring rules RR1-RR11 contribute to simplify process models without changing their behaviour, i.e., the set of producible execution traces (cf. Definition 6.9) does not change when applying refactoring rules [123]. The presented refactorings do not remove activities from process models, but focus on simplifying the control flow structure by removing or combining gateways, branches, or synchronization edges. Furthermore, RR11 simplifies the data perspective by removing unconnected data elements.

6.5 Constructing Flexible Process Hierarchies

Process hierarchies are commonly used to structure the process landscape of a company [114, 129]. Thereby, a top-level process model, which usually contains only few activities in sequential order, is decomposed through *sub-process activities*. Such a sub-process activity is detailed by another process model, i.e., *sub-process*. Figure 6.28, for example, shows a process model describing a *credit application* (as introduced in Figure 6.1). The process model on the top-most level contains sub-process activity *Inquiry Screening*. The latter is detailed by the respective sub-process that describes which activities must be executed for performing an *Inquiry Screening*. Utilizing sub-processes in this way, *process hierarchies* having multiple levels can be constructed. Thereby, sub-processes at lower levels detail sub-process activities of the respective upper level. A major drawback of such an approach is that once a process hierarchy has been designed,

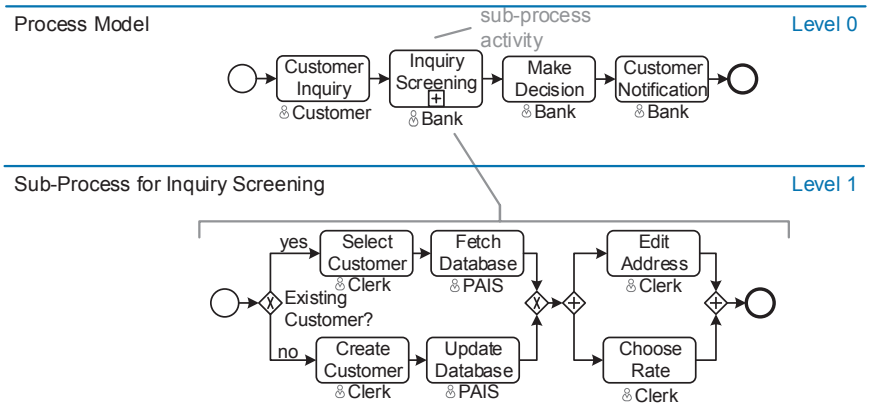


Figure 6.28: Example of a Sub-Process

a modification of its structure requires changes of multiple process models. For example, if a process designer wants to include process elements from process models at level $n+1$ in models at level n, he must modify process models on both levels. Consider Figure 6.28: if the AND branching containing activities *Edit Address* and *Choose Rate* shall be moved from the sub-process (i.e., Level 1) to the superordinated process model both the sub-process and the superordinate process model have to be modified. Moreover, if one intends to adjust the number of levels in a process hierarchy or wants to move different activities to a new sub-process, even more process models must be changed.

Another drawback of such a manually constructed process hierarchy is that only one process hierarchy for a particular business process may exist. In particular, it is not possible (or requires a high effort) to provide multiple process hierarchies based on the same business process to users. Regarding Figure 6.28, for example, one may want to have an alternative process hierarchy, in which all activities between the ANDsplit gateway in sub-process *Inquiry Screening* and activity *Customer Notification* in the top-most level process are shown in a sub-process.

In this context, process views enable us to provide more flexible process hierarchies. In particular, each aggregated activity (i.e., activities that result from the application of the aggregation operations $AggrSESE$ or $AggrComplBranches$) may be interpreted as a sub-process activity, which refers to another process view (i.e., the sub-process). The process view associated to such an activity may then comprise those elements abstracted by the corresponding aggregation operation. As an example consider Figure 6.29: activity $ABCD$ in process view $V2$ results from view creation operation $AggrSESE(CPM, \{A, B, C, D\})$. In turn, process view $V2.1$ represents the sub-process referred to by activity $ABCD$ and consists of activities A, B, C, and D, i.e., all other activities of the CPM are reduced by respective reduction operations. Finally, further view creation operations may be applied to such a sub-process (view) in order to further abstract the latter (cf. Figure 6.29), e.g., $RedActivity(CPM, C)$ *or* $AggrSESE(CPM, \{A, B\})$.

Note that in a process hierarchy based on process views both the superordinate process model and the sub-process constitute process views defined on the same CPM. Furthermore, each

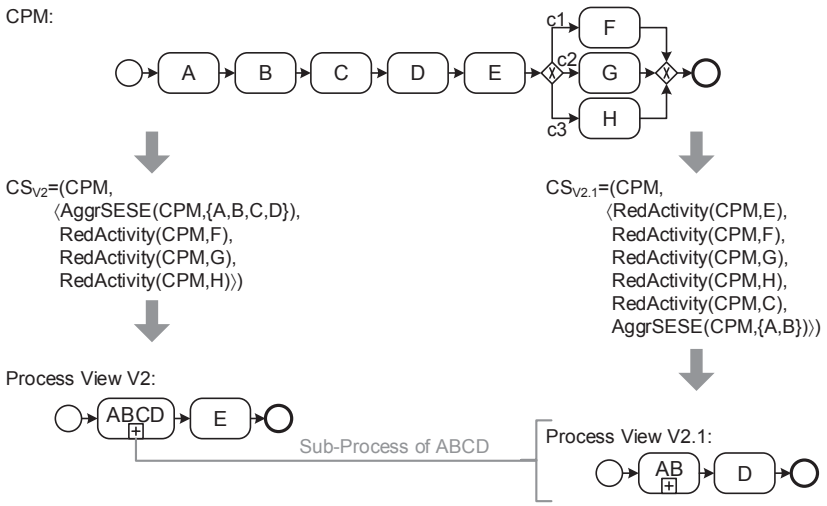


Figure 6.29: Process Hierarchy Utilizing Process Views

sub-process (view) is limited to the set of nodes, which is aggregated by a respective aggregation operation applied to the superordinate process view. Definition 6.11 introduces the notion of sub-process in the context of process views.

Definition 6.11 (Sub-Process)
Let $CPM = (N, D, NT, CE, EC, ET, DE, DET)$ *be a process model. Furthermore, let* $V_p = (N_p, D_p, NT_p, CE_p, EC_p, ET_p, DE_p, DET_p)$ *be a process view on* CPM *with creation set* CS_p*. Then, process view* V_s *with creation set* $CS_s = (CPM, \langle op_1, \ldots, op_i, op_{i+1}, \ldots, op_k \rangle)$ *is a sub-process of node* $n \in N_p$ *in process view* V_p*, iff:*

- $CPMNode(V_p, n) = N' \wedge \ \mid N' \mid > 1$
- $\forall n' \in N \setminus N' : RedActivity(CPM, n') \in \ \langle op_1, \ldots, op_i \rangle$

Consequently, a process hierarchy $PH = (CPM, \mathcal{CS}, CS_t, subpr)$ is defined by a set of creation sets $\mathcal{CS}$ on a CPM. Thereby, creation set CS_t refers to the process view representing the process model on the top-most level of the process hierarchy. Function $subpr(CS_p, n)$ returns the creation set of the sub-process with respect to an aggregated node n in process view V_p where the latter is given by its corresponding creation set CS_p. Definition 6.12 introduces the notion of a *process hierarchy* as used in the context of this thesis.

Definition 6.12 (Process Hierarchy)
Let $\mathcal{N}$ be the set of all possible process nodes. Then: A process hierarchy is defined as a tuple $PH = (CPM, \mathcal{CS}, CS_t, subpr)$ where:

- *CPM is the process model based on which all process views are created,*
- *$\mathcal{CS}$ is a set of creation sets $CS_1, \ldots, CS_k$ (cf. Definition 6.6) representing the sub-processes of the process hierarchy,*
- *$CS_t \in \mathcal{CS}$ is the creation set of the top-most level process view of the process hierarchy,*
- *$subpr : \mathcal{CS} \times \mathcal{N} \mapsto \mathcal{CS}$ with*

$$subpr(CS_p, n) \equiv \begin{cases} CS_s & isAggregatedNode(V_p, n) \\ null & otherwise \end{cases}$$

assigns to each aggregated node $n \in N_p$ of a process view $V_p = (N_p, D_p, NT_p, CE_p, EC_p, ET_p, DE_p, DET_p)$ with creation set CS_p a creation set CS_s representing sub-process V_s; if n is not an aggregated node it returns null, i.e., no sub-process exists.

In particular, a process hierarchy is defined by a set of creation sets required to create the individual (sub-)process models of each hierarchy level. Thereby, it becomes possible to create multiple process hierarchies on a given CPM. Figure 6.30 shows two process hierarchies PH_1 and PH_2 based on a common CPM and their hierarchical dependencies between the individual creation sets.

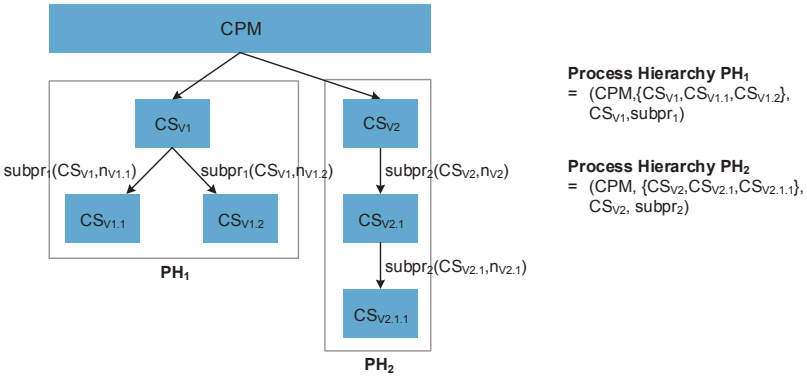


Figure 6.30: Hierarchy of Creation Sets

Figure 6.31 details the process hierarchies PH_1 and PH_2 introduced in Figure 6.30. Process view $V1$ represents the top-most level of PH_1 and aggregates activities A, B, and C to ABC as well as activities F and G to FG. Aggregated activity ABC is refined by process view $V1.1$ and FG by process view $V1.2$. In order to preserve control flow correctness, $V1.2$ not only comprises the aggregated activities, but also the surrounding XOR branching block.

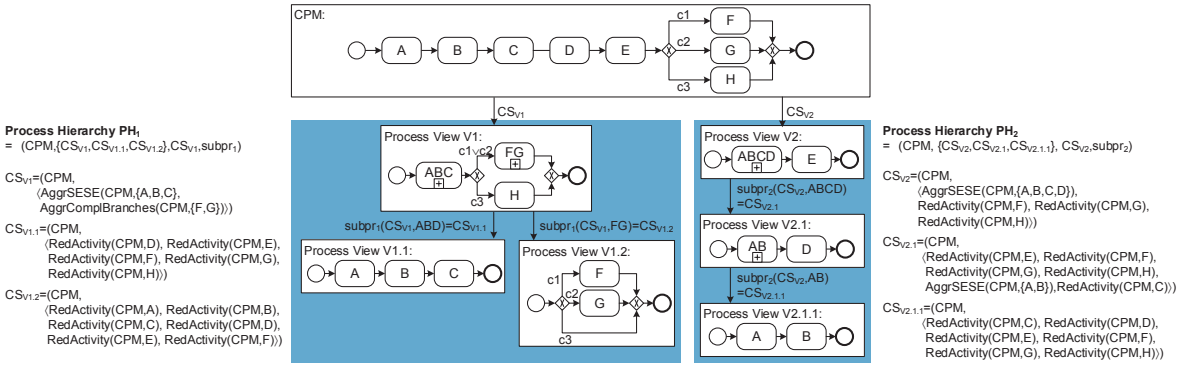


Figure 6.31: Process Hierarchy based on Process Views

Process view $V2$ represents the top-most level of process hierarchy PH_2. Activity $ABCD$ in $V2$ aggregates activities A, B, C, and D of the CPM. Process view $V2.1$ further refines activity $ABCD$. Moreover, in process view $V2.1$ view creation operations reduce activity C and aggregate A and B. Finally, activity AB is further refined by (sub-)process view $V2.1.1$.

Generally, process views allow defining multiple process hierarchies based on the same CPM. Hence, it becomes possible to provide personalized process hierarchies for individual users or user roles. Adapting the structure of a process hierarchy can be accomplished by modifying creation sets of individual process views and does not require modifying multiple process models. Furthermore, we later show that when defining all sub-processes based on a CPM, updates to individual sub-processes can be easily propagated to all other process views (i.e., sub-processes). Hence, all process models in such a process hierarchy are kept up-to-date.

6.6 Related Work

IEEE 1471 recommends user-specific viewpoints for software architectures [130]. These viewpoints constitute templates from which individual views are created for a concrete software architecture. Since this standard does not define any methods, tools, or processes, this thesis provides a powerful framework for this requirements in the context of PAISs. [131] introduces a meta model for process views and shows a general overview of process view patterns. However, no view operations are provided.

Some approaches for creating process views deal with inter-organizational processes and apply views to create abstractions of private processes by hiding implementation details [132, 133, 134, 135, 136, 137]. However, views are specified by the process designer and cannot be easily defined by the user. A notable expection is presented in [138]. Moreover, process views are not intended for internal users, but only for documenting the inter-organizational communication.
Other approaches align business and technical process models [23, 25, 139, 140] to keep them aligned over time (i.e., when evolving them). However, these approaches are limited to establish a mapping between business and technical process models and do not provide personalized process views to users.

In [76], an approach with predefined view types (i.e., human tasks, collaboration view) is presented. Opposed to this thesis, it is limited to pre-specified view types. Furthermore, it is not possible to define user-specific views. In turn, [141] applies graph reduction techniques to verify structural properties of process models. However, aggregation operations as well as other process aspects (i.e., data flow and attributes) are not addressed. Process model abstraction based on SPQR-tree decomposition are presented in [142]. This approach does not cover other process aspects (i.e., data flow and attributes) as well.

Semantic similarity between different process nodes is determined by analyzing the structural information of a process model and its attributes [31, 143]. The discovered similarities are then used to abstract the given process model. However, the approach neither distinguishes between different perspectives of a process model nor does it provide concepts for creating process views.

An approach for creating aggregated views is provided by [144]. It proposes a two-phase procedure for aggregating parts of a process model that must not be exposed to the public. It focuses on block-structured graphs. Neither data flow nor process attributes are considered.
View models for monitoring purposes at run-time are presented in [145, 146]. These approaches focus on the propagation of run-time information to process views in order to visualize the execution progress. In particular, respective process views have to be pre-specified manually by a process designer.

[147] introduces an approach to create process views on a CPM based on reduction operations and merges several process variants into one CPM. Reduction operations are then used to extract process variants again. Aggregation operations are not provided.

Subject-oriented Business Process Management (S-BPM) aims to provide a dedicated view for each user role in the business process [148, 149]. Individual views are connected by a superordinate process model visualizing the communication between the user roles. However, neither a global view (i.e., CPM) is provided nor is it possible to define additional process views. Figure 6.32 documents the credit application process of Example 6.1 utilizing S-BPM.

A set of process view operations addressing control flow, data flow, and process attributes is provided in [32]. However, these operations may destroy the ordering of process nodes in the CPM when creating a process view, e.g., by aggregating activities located on different branches and inserting the aggregated node in front of the respective split gateway. This kind of operations may influence the behaviour (and meaning) of a process model. Furthermore, this approach introduces refactorings for process models, which constitute a subset of the refactoring rules introduced in this thesis.

In the context of process model refactoring various approaches exist. In particular, refactorings for process models stored in process repositories are introduced in [18, 67]. Besides refactorings, which simplify the structure of a process model, the authors introduce operations to maintain (e.g., by renaming) process nodes in such repositories. Refactorings specific to EPCs are defined by [150]. Furthermore, refactorings are also known in the context other graph-based models (e.g., UML) to increase the quality of the resulting model [151, 152]. In software programming,

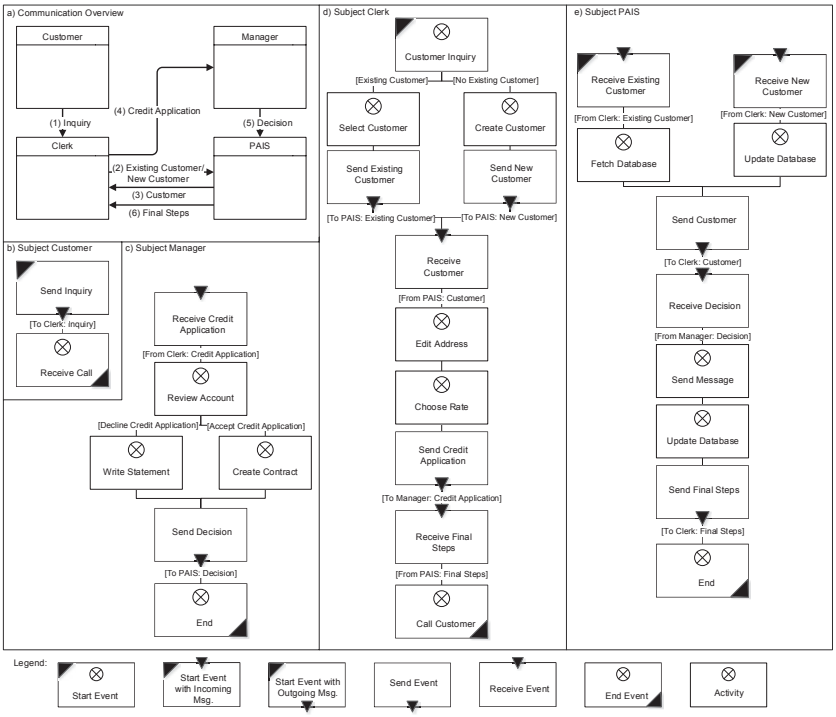


Figure 6.32: User Orientation based on S-BPM

refactorings are used to maintain the source code of information systems [153, 154].

A general introduction to create process hierarchies is given in [155]. [22] introduces how to construct process hierarchies in the context of EPCs. In [116], it is suggested to structure process model based on sub-processes to reduce the complexity for users. Furthermore, the benefit of sub-processes is investigated in [129, 156]. An overview on process hierarchies and their realization is given in [114].

This thesis provides a holistic framework for user-centric view creation based on elementary operations addressing the control and data perspective as well as process attributes. Refactoring rules are introduced to simplify the structure of resulting process views. Based on process view creation, we are able to construct process hierarchies, which are easy to define and maintain. Existing approaches neither provide the same expressiveness nor covers all these aspects.

6.7 Summary

Process models describing real-world business processes are complex and may comprise a large number of process nodes (e.g., activities, data elements) [18]. Supporting users with limited background on process modeling requires a framework that provides personalized process views to reduce complexity for respective users. In this chapter a view creation framework is presented, which meets the requirements for creating personalized process views (i.e., REQ-2 and REQ-

3). We have presented aggregation- and reduction-based view creation operations abstracting control flow, data flow, and process attributes. Based on these view creation operations, personalized process views can be created for supporting the individual needs of users.

Finally, this chapter discusses how process views can be used to construct process hierarchies. In this context, aggregated activities may be further refined through a sub-process, i.e., another process view. In particular, process hierarchies based on process views enable us to create multiple process hierarchies based on a given CPM.

7 Changing Process Models Through Process View Updates

7.1 Introduction

As discussed in Chapter 6, process views abstract from complex process models supporting users in better understanding their role in business processes. However, business processes and, hence, the corresponding process models change over time as well; e.g., due to emerging market situations, new legislative regulations, or process reengineering efforts [38, 157]. To reflect the business changes, the respective process models need to be updated accordingly. In this context, business analysts and domain experts having only limited process modeling knowledge should be enabled to update process models on an abstract level to keep them up-to-date when business processes change [15]. Generally, end-users are unable to perform updates of large process models (i.e., *Central Process Models (CPMs)*) containing hundreds or even thousands of elements.

A promising approach is to enable end-users to update and evolve process models based on related abstractions, i.e., process views (cf. Requirement REQ-4). As an advantage, users need not understand all details of a large process model (i.e., CPM), but can focus on a more compact process view when changing the parts of the process relevant for them. In general, a process view update may affect both the CPM and its associated process views (cf. Requirement REQ-5); i.e., a process view update must be applied to the CPM as well as to other related process views (cf. Figure 7.1a). However, manually updating a CPM as well as its associated process views is not appropriate. First, sophisticated modeling skills and high efforts by the user performing the update would be required. Second, users having only limited process modeling knowledge might be unable to update multiple process models. Third, performing the same update on multiple process models is an error-prone task.

In general, an update of any process view must be automatically propagated to the underlying CPM as well as to all other process views associated with the CPM. Accordingly, a view update

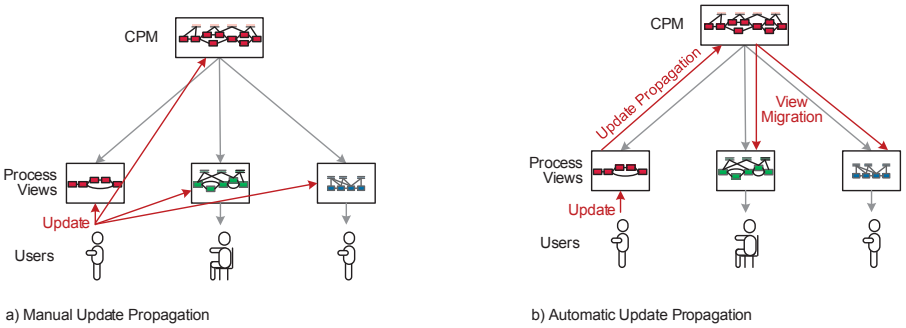


Figure 7.1: Strategies to Propagate View Updates

must be first propagated to the CPM (cf. Figure 7.1b). Then, associated process views must be migrated to the new CPM version. This becomes necessary to guarantee that all process views remain consistent after updates performed by a particular users. Furthermore, a change propagation shall ensure that users work on an up-to-date version of the CPM and of their personalized process views.

First, this chapter formally introduces well-defined view update operations. Thereby, a view update triggered by a user is directly applied to the underlying CPM. Second, the creation sets of other process views associated with the updated CPM need to be migrated to the new CPM version of the CPM, and then be re-created.

Section 7.2 introduces fundamentals of updating process models. Section 7.3 describes view update operations for changing process views and their propagation behaviour to the CPM. Section 7.4 presents rules for migrating process views from an old to a new CPM version after applying an update. Section 7.5 discusses how creation sets of process views can be optimized to improve view creation and view update. Section 7.6 discusses related work. Section 7.7 presents limitation of the introduced concepts. Finally, Section 7.8 summarizes this chapter.

7.2 Fundamentals of Updating Process Models

This section introduces basic update operations that may be applied to process models. We do not introduce a complete set of update operations for process models (see [104] for details), but restrict the approach to those operations required for realizing view updates.

Generally, updates of process models may require adding or deleting process nodes, data elements, control edges, and data edges. Furthermore, process attributes may be changed. *Basic update operations* solely refer to single aspects of a process model. As example consider Figure 7.2a: Activity X shall be inserted into the given process model. To realize this change, a sequence of basic update operations needs to be applied, e.g., to add node X as well as to delete and add related control edges. More precisely, the control edges connecting activities A and B are deleted. Then, activity X is added to the set of process nodes. Finally, control edges connecting X with its predecessor A and successor C respectively are added.

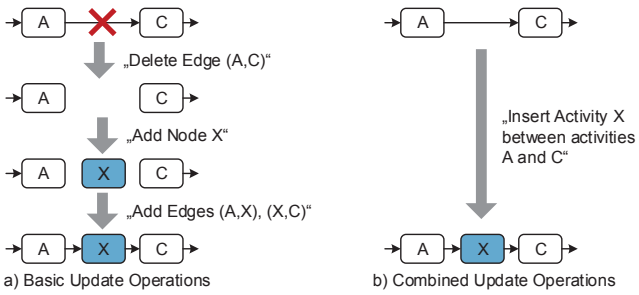


Figure 7.2: Alternatives to Update Process Models

When applying basic update operations process models may result that are not connected anymore (i.e., Constraint 2 of Definition 6.3 is violated). However, basic operations may be *combined* to more complex ones [103]. Figure 7.2b shows an example that inserts activity X between A and C based on such a combined update operation. Still, combined update operations might violate control flow correctness. For example, when solely inserting an ANDsplit gateway without adding a corresponding ANDjoin gateway the control flow is not correct (i.e., Constraint 3 of Definition 6.3 is violated).

Combined update operations may be further assembled to *change patterns* [158]. For example, a change pattern allows inserting or deleting a complete process fragment into a process model. In particular, change patterns allow updating process models without violating control flow correctness. Furthermore, they allow assisting users in updating process models [159]. However, the presented view update operations are transformed to combined update operations used to modify the structure of the CPM. In the following, we introduce in the following update operations *AddNode*, *AddCEdge*, and *AddDEdge*.

Update operation $AddNode(P, n_1, n_2, n_{new}, node\_type)$ adds process node n_{new} with type $node\_type$ between nodes n_1 and n_2 in process model P (cf. Algorithm 7.1). As a precondition, nodes n_1 and n_2 must be direct neighbours in P (i.e., $\exists(n_1, n_2) \in CE$). Figure 7.2b provides an example of applying this update operation.

Algorithm 7.1: $AddNode(P, n_1, n_2, n_{new}, node\_type)$

Input: Process model $P = (N, D, NT, CE, EC, ET, DE, DET)$
Activity n_{new} to be inserted between n_1 and n_2 ($\{n_1, n_2\} \subseteq N$) with node type $node\_type$
begin
  $N := N \cup \{n_{new}\}$;
  $NT(n_{new}) := node\_type$;
  $e_1 := (n_1, n_{new})$; $EC(e_1) := EC((n_1, n_2))$; $ET(e_1) := ET\_Control$;
  $e_2 := (n_{new}, n_2)$; $ET(e_2) := ET\_Control$;
  $CE := CE \setminus \{(n_1, n_2)\} \cup \{e_1, e_2\}$;
end

Update operation $AddCEdge(P, ce, edge\_type, edge\_condition)$ adds a control edge $ce = (n_1, n_2)$ to process model P (cf. Algorithm 7.2). As a precondition, the process nodes to be connected must be already part of the process model (i.e., $n_1, n_2 \in N$). Control edge ce may have edge

type $ET\_Control, ET\_Sync$, or ET_Loop. Finally, an edge condition is assigned in case the source of the edge corresponds to an XORsplit or LOOPsplit gateway (i.e., the edge represents a conditional branch). In all other cases, *edge_condition* is set to $TRUE$.

Algorithm 7.2: $AddCEdge(P, ce, edge\_type, edge\_condition)$

```
   Input: Process model P = (N, D, NT, CE, EC, ET, DE, DET)
1          Control edge ce of type edge_type and edge condition edge_condition to be inserted
2  begin
3  |   ET(ce) := edge_type;
4  |   EC(ce) := edge_condition;
5  |   CE := CE ∪ {ce};
6  end
```

Update operation $AddDEdge(P, de, det)$ adds a data edge $de = (n_1, n_2)$ to process model P (cf. Algorithm 7.3). As a precondition, $n_1 \in N$ and $n_2 \in D$ (or vice versa) must hold, i.e., data edge de connects a process node with a data element or vice versa. Finally, det defines whether the data element may accessed mandatorily or optionally.

Algorithm 7.3: $AddDEdge(P, de, det)$

```
   Input: Process model P = (N, D, NT, CE, EC, ET, DE, DET)
1          Data edge de to be added and data edge type det ∈ DEdgeType = {mandatory, optional}
2  begin
3  |   DE := DE ∪ {de};
4  |   DET(de) := det;
5  end
```

We omit details regarding the operations to remove control or data edges from a process model. Removing such an edge e from a process model is straightforward, i.e., edge e with be removed from the set of control edges (i.e., $CE := CE \setminus \{e\}$) and data edges (i.e., $DE := DE \setminus \{e\}$).

7.3 View Update Operations

When allowing authorized users to update a CPM based on associated process views, it must be ensured that this can be accomplished without violating the correctness of the CPM and the update is propagated to the CPM as desired by the user. In this context, view update operations should allow for the proper propagation of view updates to the underlying CPM.

Propagating view updates to the corresponding CPM is not straightforward. In particular, ambiguities might occur in this context that need to be resolved. Example 7.1 demonstrates that it is not always possible to determine a unique insert position when propagating a view insert operation to the respective CPM. Note that such ambiguities are caused by the information loss that occurs due to the application of reduction operations or refactoring rules (cf. Section 6.4) when creating the process view. In particular, ambiguities not only occur in the context of the control flow, but also when applying updates on aggregated data elements.

Example 7.1 (Updating Process Views)
Consider the credit application process from Example 6.1 and the corresponding process view V displaying solely the activities of user role clerk (cf. Figure 7.3). In this process view, all activities performed by the PAIS as well as the customer (i.e., a1, a2, a5, a11, and a13) are reduced and the activities of user role manager (i.e., a8-a10) are aggregated to abstract activity Credit Decision.

Assume that the clerk inserts activity Create File (i.e., a14) between gateway Existing Customer? (i.e., $XORsplit_1$) and activity Create Customer (i.e., a4). When propagating this update to the underlying CPM, the corresponding insert position in the CPM is unique, i.e., this view update can be automatically propagated to the CPM without any problems.

Consider activity Send Customer ID (i.e., a15) and assume that it shall be inserted between Select Customer and the succeeding $XORjoin_1$ gateway in process view V. When trying to propagate this update to the CPM, multiple insert positions are possible, i.e., there occur ambiguities regarding the transformation of the view update to a corresponding update of the CPM. To be more precise, a15 may be inserted directly after a3 (as shown in CPM') or directly after a2 (as shown in CPM'').

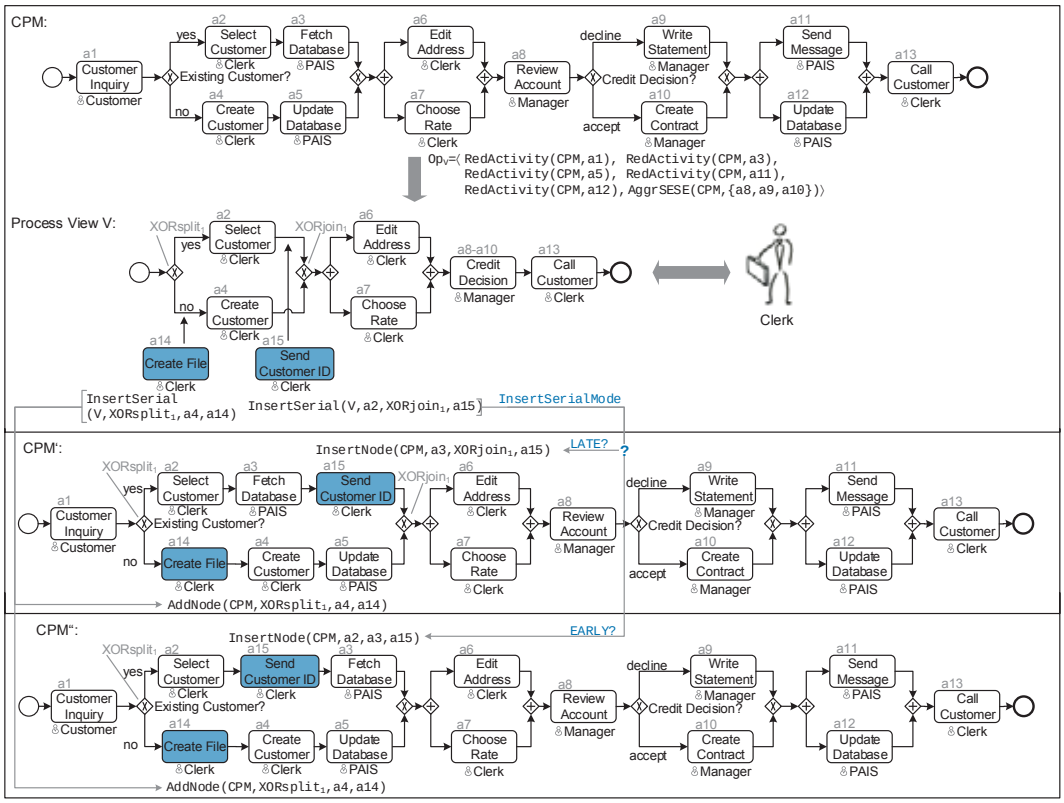


Figure 7.3: Ambiguity when Propagating View Changes to the CPM

In order to enable an automated propagation of process view updates to a CPM, configurable propagation policies are supported; in particular, view update operations may be configured based on a respective set of *configuration parameters* (*parameters* for short). In particular, the latter allow for the automatic resolution of ambiguities, if required. In particular, these parameters describe the *propagation behaviour* of a view update operation, e.g., they control whether an activity shall be inserted at the earliest/latest possible position in the CPM.

Consider view update operation *InsertSerial* in Figure 7.3. Parameter *InsertSerialMode* defines whether activity $a15$ shall be inserted directly after activity $a2$ (i.e., *InsertSerialMode=EARLY*) or directly before activity $a3$ (i.e., *InsertSerialMode=LATE*). Generally, each parameter has a default value, but may be set specifically for a process view. In order to manage this view-specific parameter settings, we replace Definition 6.6 (i.e., process view) by Definition 7.1. Particularly, the creation set is extended by a parameter set PS.

Definition 7.1 (Process View with Parameter)
Let $CPM = (N, D, NT, CE, EC, ET, DE, DET)$ be a process model. Then: A process view V is represented through a creation set $CS_V = (CPM, Op, PS)$, where

- *CPM is the central process model on which V is created,*
- *$Op = \langle op_1, \ldots, op_k \rangle$, $op_i \in \mathcal{OP}$ is a sequence of elementary view creation operations applied to CPM . Thereby, $\mathcal{OP}$ comprises all elementary view creation operations (cf. Section 6.3),*
- *$PS = \{PS_1, \ldots, PS_m\}$ is a set of parameters with corresponding parameter values defined for a specific process view.*

As opposed to basic update operations (cf. Section 7.2), in the following, we deal with elementary view update operations enabling more complex changes, e.g., insert an activity X or an entire branching block. These allow for a more convenient way to update process views; i.e., users do not operate with primitive changes (e.g., insert single edges) when inserting an activity. These view update operations are denoted as *elementary*. In particular, elementary view update operations ensure correctness of both the process view and the CPM (cf. Definition 6.3). This is crucial if users have limited process modeling knowledge, i.e., this way users are prevented from modeling an invalid control flow. However, view update operations may violate data flow correctness (cf. Definition 6.3). In the context of data flow updates, therefore, we discuss which view update operations may violate data flow correctness (cf. Section 7.3.3).

According to Figure 7.4, elementary view update operations can be categorized in operations inserting (cf. Section 7.3.1) and deleting process elements (cf. Section 7.3.2) as well as operations updating the data flow (cf. Section 7.3.3) and process attributes (cf. Section 7.3.5). Furthermore, view update operations are based on change patterns introduced in [158]. Chapter 8 discusses the completeness of our view update operations regarding change patterns for process models [158]. Additionally, we discuss the effects of view update operations on the correctness of process models in Section 7.3.4.

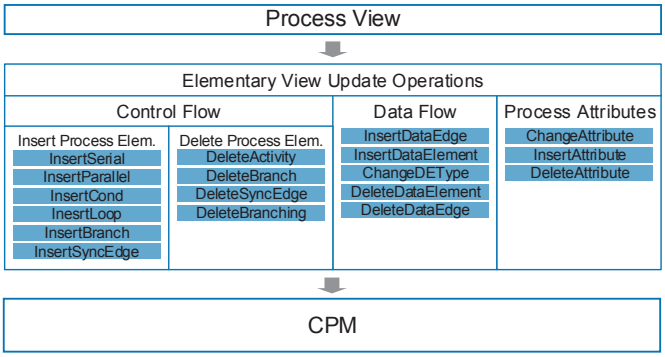


Figure 7.4: Overview on View Update Operations

7.3.1 Inserting Process Nodes and Control Edges

To extend the control flow of a process model, process nodes and control edges may be added to describe new tasks in the respective business process. First, an activity n_{new} may be sequentially inserted between two activities (i.e., $InsertSerial(V, n_1, n_2, n_{new})$). Second, n_{new} may be inserted in parallel to an existing process fragment (i.e., $InsertParallel(V, n_1, n_2, n_{new})$). Third, n_{new} may be inserted as alternative to an existing process fragment, i.e., n_{new} may be executed alternatively to the process fragment (i.e., $InsertCond(V, n_1, n_2, n_{new}, c)$).

Another elementary view update operation allows inserting a loop (i.e., $InsertLoop(V, n_1, n_2, c)$) that encloses an existing process fragment. This way, it can be expressed that the fragment may be executed repeatedly. Furthermore, a new branch may be inserted to an existing branching block (i.e., $InsertBranch(V, g_s, g_j, c)$). Finally, a synchronization edge may be added by applying $InsertSyncEdge(V, n_s, n_e)$. This operation allows synchronizing activities that belong to different branches of an AND branching. Table 7.1 summarizes the elementary view update operations for adding process nodes and control edges to the control flow of a process view and CPM, respectively. For each view update operation, column *configuration parameter & value* contains configuration parameters together with the values that may be assigned to them. Default parameter values are emphasized in bold font. However, for some operations (e.g., *InsertSyncEdge*) no parameter is required since ambiguities can be resolved. In the following, we describe these operations in more detail.

$InsertSerial(V, n_1, n_2, n_{new})$. View update operation $InsertSerial(V, n_1, n_2, n_{new})$ adds an activity n_{new} to process view V and propagated the update to CPM: n_{new} is sequentially inserted between activities n_1 and n_2. For example, Figure 7.5 illustrates the application of $InsertSerial(V, A, C, X)$ on process view V. The latter was derived from the CPM by reducing activity B. Note that propagating this view update to the CPM results in ambiguities regarding the exact position the new activity shall be inserted in the CPM.

In order to resolve such ambiguities, parameter $InsertSerialMode$ needs to be set. It allows selecting the earliest (i.e., $InsertSerialMode = EARLY$) or latest (i.e., $InsertSerialMode = LATE$) insert position in the CPM. Moreover, the activity may be inserted in parallel to the

View Update Operation	Configuration Parameter & Value	Description
$InsertSerial(V, n_1, n_2, n_{new})$	InsertSerialMode ∈ { **EARLY**, LATE, PARALLEL}	Inserts activity n_{new} between n_1 and n_2 in process view V. Parameter $InsertSerialMode$ characterizes the propagation behaviour of the operation.
$InsertParallel(V, n_1, n_2, n_{new})$ $InsertCond(V, n_1, n_2, n_{new}, c)$	InsertBlockMode ∈ { **EARLY_EARLY**, EARLY_LATE, LATE_EARLY, LATE_EARLY}	Inserts activity n_{new} as well as an AND/XOR branching block surrounding the SESE block defined by n_1 and n_2 in view V. Before (after) the underline, the parameter value specifies the propagation behaviour of the split (join) gateway.
$InsertLoop(V, n_1, n_2, c)$	InsertBlockMode ∈ { **EARLY_EARLY**, EARLY_LATE, LATE_EARLY, LATE_EARLY}	Inserts a Loop block surrounding the SESE block defined by n_1 and n_2 in process view V. Before (after) the underline, the parameter value specifies the propagation behaviour of the LOOPsplit (LOOPjoin) gateway.
$InsertBranch(V, g_s, g_j, c)$	InsertBranchMode ∈ { **EARLY**, LATE}	Inserts an empty branch between split gateway g_s and join gateway g_j in process view V. In the context of conditional branchings or loops, in addition, branching condition c is has to be provided.
$InsertSyncEdge(V, n_s, n_e)$	-	Inserts a sync edge from n_s to n_e in V, where n_s and n_e belong to different branches of a parallel branching.

Table 7.1: View Update Operations - Inserting Process Elements

process fragment causing the ambiguity (i.e., $InsertSerialMode = PARALLEL$). As example consider Figure 7.5. Based on the setting of parameter *InsertSerialMode*, process models CPM', CPM'', or CPM''' result. To show the update in process view V, the latter may be re-created by applying view creation operation $RedActivty(CPM, B)$ to the CPM. Independent from the setting of parameter *InsertSerialMode*, the resulting process view model V' is similar to the one that is obtained when inserting activity X directly into process view V. Note that in the context of CPM''', refactoring rule RR8 is applied, which inlines activity X between A and C (cf. Section 6.4).

View update operation $InsertSerial(V, n_1, n_2, n_{new})$ is applicable if n_{new} constitute an activity and nodes n_1 and n_2 are direct neighbours connected by a control edge in process view V:

Preconditions $InsertSerial(V, n_1, n_2, n_{new})$.
Let $V = (N_V, D_V, NT_V, CE_V, EC_V, ET_V, DE_V, DET_V)$ *be the process model of a process view with* $CS_V = (CPM, Op_V, PS_V)$. *Then:*

- $NT(n_{new}) = Activity$
- $\exists (n_1, n_2) \in CE_V : ET((n_1, n_2)) = ET\_Control$

Algorithm 7.4 formally presents view update operation $InsertSerial(V, n_1, n_2, n_{new})$. First, the nodes of the CPM corresponding to n_1 and n_2 are determined (Line 2). If one of them is an aggregated node, *CPMNode* returns the set of aggregated nodes. In this case, *first (last)* returns the first (last) node regarding the control flow within this set (cf. Definition 6.4).

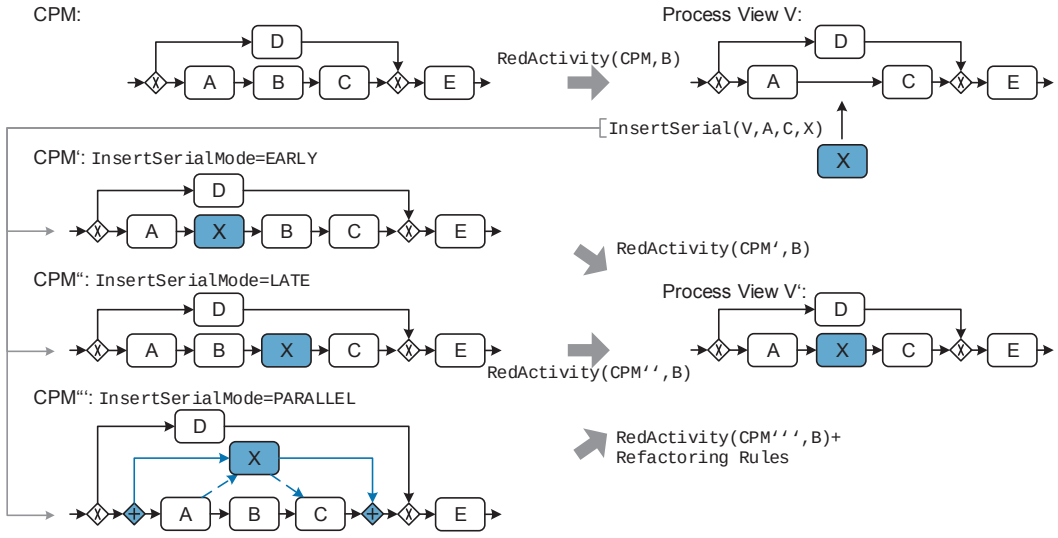


Figure 7.5: View Update Operation: InsertSerial

Algorithm 7.4: *InsertSerial*(V, n_1, n_2, n_{new})

Input: Process View V based on creation set $CS_V = (CPM, Op_V, PS_V)$.
$CPM = (N, D, NT, CE, EC, ET, DE, DET)$ and activity n_{new} to insert between activities $n_1, n_2 \in N$.

1 **begin**
2 $n'_1 := last(CPMNode(V, n_1))$; $n'_2 := first(CPMNode(V, n_2))$;
3 **if** $(succ(CPM, n'_1) = n'_2)$ **then**
4 $AddNode(CPM, n'_1, n'_2, n_{new}, Activity)$;
5 **else**
6 **switch** *InsertSerialMode*
7 **case** *EARLY:*
8 $AddNode(CPM, n'_1, succ(CPM, \{n'_1\}), n_{new}, Activity)$;
9 **case** *LATE:*
10 $AddNode(CPM, pred(CPM, \{n'_2\}), n'_2, n_{new}, Activity)$;
11 **case** *PARALLEL:*
12 $(n_s, n_j) := MinimalSESE(CPM, \{n'_1, n'_2\})$;
13 $AddNode(CPM, pred(CPM, \{n_s\}), n_s, g_s, ANDsplit)$;
14 $AddNode(CPM, n_j, succ(CPM, \{n_j\}), g_j, ANDjoin)$;
15 $AddCEdge(CPM, (g_s, g_j), ET\_Control, TRUE)$;
16 $AddNode(CPM, g_s, g_j, ne new, Activity)$;
17 $AddCEdge(CPM, (n'_1, n_{new}), ET\_Sync, TRUE)$;
18 $AddCEdge(CPM, (n_{new}, n'_2), ET\_Sync, TRUE)$;
19 **end**
20 **end**
21 **end**
22 **end**
23 **end**

Afterwards, it is checked whether nodes n'_1 and n'_2 (i.e., the nodes in CPM corresponding to n_1 and n_2) are direct neighbours (Line 3). In this case, n_{new} may be directly inserted between n_1 and n_2 by applying the basic change operations *AddNode* (cf. Algorithm 7.1) and *AddCEdge* (cf. Algorithm 7.2) to the CPM. In turn, if n'_1 is not directly preceding n'_2 in the CPM, it must be decided at which position in the CPM n_{new} shall be inserted. Note that this is controlled through parameter *InsertSerialMode*. When setting this parameter to *EARLY*, n_{new} is directly

inserted after n'_1 (Line 8). In turn, when setting it to *LATE*, n_{new} is inserted directly before n'_2 (Line 10). Finally, when using parameter value *PARALLEL*, the minimal SESE block containing n'_1 and n'_2 is determined. Then, an AND block surrounding the SESE block is added. The SESE block is created by adding *ANDsplit* and *ANDjoin* gateways as well as an empty branch between them (Lines 12-18). Finally, n_{new} is inserted into this empty branch. To ensure that the same precedence relations as for the process view are obeyed, synchronization edges pointing from n'_1 to n_{new} as well as from n_{new} to n'_2 are inserted as well.

In the following, we show that the propagation of a process view update (based on *InsertSerial*) to the CPM, followed by the re-creation of the process view, results in the same process view model as can be obtained when directly inserting this activity in the process view. We consider this as a fundamental quality property of the view update propagation approach. Note that the user expects the inserted activity at exactly that position in the process view on which he performed the insert operation.

To demonstrate this based on execution traces (cf. Definition 6.9), we introduce the notion of *dependency set* to express direct control flow dependencies (cf. Definition 7.2).

Definition 7.2 (Dependency Set)
Let $P = (N, D, NT, CE, EC, ET, DE, DET)$ be a process model and let $\mathcal{T}_P$ the set of execution traces producible on P. Then:
$\mathbb{D}_P := \{(n_1, n_2) \in N \times N \mid \exists t = \langle \ldots, n_1, n_2, \ldots \rangle \in \mathcal{T}_P\}$ is denoted as dependency set. $\mathbb{D}_P$ reflects all direct control flow dependencies between two activities.

Consider Figure 7.5. The dependency set of CPM' corresponds to $\mathbb{D}_{CPM'} = \{(D, E), (A, X), (X, B), (B, C), (C, E)\}$. Theorem 7.1 expresses that, when propagating view update operation $InsertSerial$ to the CPM, we obtain the same process view in respect to the dependency set compared to the direct insertion of respective process node in the process view.

Theorem 7.1 (Equivalence of Applying InsertSerial to CPM and Process View)
Let CPM be a process model with dependency set $\mathbb{D}_{CPM}$. Further, let V be a process view on CPM with creation set $CS_V = (CPM, Op_V, PS_V)$ and dependency set $\mathbb{D}_V$.
Then: Adding n_{new} to V can be realized based on $InsertSerial(V, n_1, n_3, n_{new})$. Propagating this update to the CPM and re-creating V results in the same dependency set for V as can be obtained when inserting n_{new} directly in V.

View creation operations based on process element reduction (e.g., *RedActivity*) and related refactorings might cause ambiguities. Therefore, we discuss their impact on the dependency set of resulting process views. Applying $RedActivity(CPM, n_2)$ with $(n', n_2), (n_2, n'') \in CE$ to CPM with dependency set $\mathbb{D}$ results in $\mathbb{D}' = \mathbb{D} \setminus \{(n', n_2), (n_2, n'')\} \cup \{(n', n'')\},\ n', n'' \in N$. Applying refactoring rules can then be interpreted as *reducing* dependencies in the process view. Hence, it can be interpreted as part of the view create operations Op in creation set CS.

Proof 7.1 (InsertSerial Equivalence)

Dependency set $\mathbb{D}_{V'} = \mathbb{D}_V \cup \{(n_1, n_{new}), (n_{new}, n_3)\} \setminus \{(n_1, n_3)\}$ *results after inserting* n_{new} *directly in view* V*. When inserting* n_{new} *in the CPM, we must distinguish four cases:*

Case 1. No activity is reduced between n_1 *and* n_3*, i.e., no parameter is required and* $\mathbb{D}_{CPM'} = \mathbb{D}_{CPM} \cup \{(n_1, n_{new}), (n_{new}, n_3)\} \setminus \{(n_1, n_3)\} = \mathbb{D}_{V'}$*.*

Cases 2-4. An activity (activity set) is reduced between n_1 *and* n_3*, i.e., ambiguities occur and parameter InsertSerialMode becomes relevant.*

Case 2. InsertSerialMode=EARLY: We obtain $\mathbb{D}_{CPM'} = \mathbb{D}_{CPM} \cup \{(n_1, n_{new}), (n_{new}, n_2)\} \setminus \{(n_1, n_2)\}$ *and* $RedActivity(n_2) \in Op$ *with* $\{(n_1, n_2), (n_2, n_3)\} \subset \mathbb{D}_{CPM}$*. Without loss of generality, we may assume that exactly one activity is reduced between* n_1 *and* n_3*. Then,* V *is recreated with* $RedActivity(n_2)$ *resulting in* $\mathbb{D}_{V''} = \mathbb{D}_{CPM'} \setminus \{(n_{new}, n_2), (n_2, n_3)\} \cup \{(n_{new}, n_3)\} = \mathbb{D}_{CPM} \cup \{(n_1, n_{new}), (n_{new}, n_2)\} \setminus \{(n_1, n_2)\} \setminus \{(n_{new}, n_2), (n_2, n_3)\} \cup \{(n_{new}, n_3)\} = \mathbb{D}_{V'}$*.*

Case 3. InsertSerialMode=LATE: Similar to Case 2 with n_{new} *inserted directly before* n_3*.*

Case 4. InsertSerialMode=PARALLEL: We obtain $\mathbb{D}_{CPM'} = \mathbb{D}_{CPM} \cup \{(n_1, n_{new}), (n_{new}, n_3)\}$ *and* $RedActivity(n_2) \in Op$ *with* $\{(n_1, n_2), (n_2, n_3)\} \subset \mathbb{D}_{CPM}$*. Then,* V *is re-created with* $RedActivity(n_2)$*. This results in* $\mathbb{D}_{V''} = \mathbb{D}_{CPM'} \setminus \{(n_1, n_2), (n_2, n_3)\} \cup \{(n_1, n_3)\}$*. Note that the parallel branching remains in the process view model, i.e., one branch containing* n_{new} *and another branch containing* n_1 *and* n_2 *synchronized by sync edges. Finally, refactoring rule RR9 (cf. Section 6.4) removes the unnecessary branching:* $\mathbb{D}_{V'''} = \mathbb{D}_{V''} \setminus \{(n_1, n_3)\} = \mathbb{D}_{V'}$*.*

Proof 7.1 shows that inserting a node directly or indirectly in a process view (i.e., by inserting in the CPM) based on *InsertSerial* results in the same process view.

$InsertParallel(V, n_1, n_2, n_{new})$. When inserting an activity, it might become necessary to insert it in parallel to existing activities. This is covered by view update operation $InsertParallel(V, n_1, n_2, n_{new})$. Again, the transformation of the respective view update to an update of the corresponding CPM may raise ambiguities regarding the positions the respective ANDsplit and ANDjoin gateways shall be inserted. To deal with this ambiguity, parameter *InsertBlockMode* is used. It allows configuring the positions at which the ANDsplit (i.e., $EARLY\_*$, $LATE\_*$) and the ANDjoin (i.e., $*\_EARLY$, $*\_LATE$), shall be inserted. For example, *InsertBlockMode=EARLY_LATE* expresses that the ANDsplit gateway shall be inserted at the earliest and the ANDjoin gateway at the latest possible position in the CPM.

Consider Figure 7.6. Activity X is inserted in parallel to the SESE block induced by activities C and D. When propagating this update to the CPM, a proper insert position for the newly added $ANDsplit$ and $ANDjoin$ gateways has to be determined. In Figure 7.6, the $ANDsplit$ gateway of the branching with activity X may be inserted between A and B, B and $ANDsplit_1$, or between $ANDsplit_1$ and C. To resolve this ambiguity, parameter *InsertBlockMode* needs to be set. In Figure 7.6, dotted boxes above CPM' show the possible insert positions for the ANDsplit and ANDjoin gateways depending on the parameter setting. For parameter setting $EARLY\_*$ (i.e., $EARLY\_EARLY$ and $EARLY\_LATE$), for example, the ANDsplit gateway is inserted between A and B in CPM'. When inserting the ANDsplit and ANDjoin gateway, in addition, well-structuredness of the CPM must be preserved. For example, when setting *InsertBlockMode* to $EARLY\_EARLY$, the ANDsplit gateway will be inserted between A and B, and the ANDjoin gateway between D and E. In this case, well-structuredness of CPM' would be violated since the resulting AND branchings are not nested anymore (cf. Constraint 3 in Definition 6.3). Hence, the ANDjoin gateway has to be inserted between $ANDjoin_1$ and G. Possible AND branchings resulting from respective parameter settings are shown below CPM'.

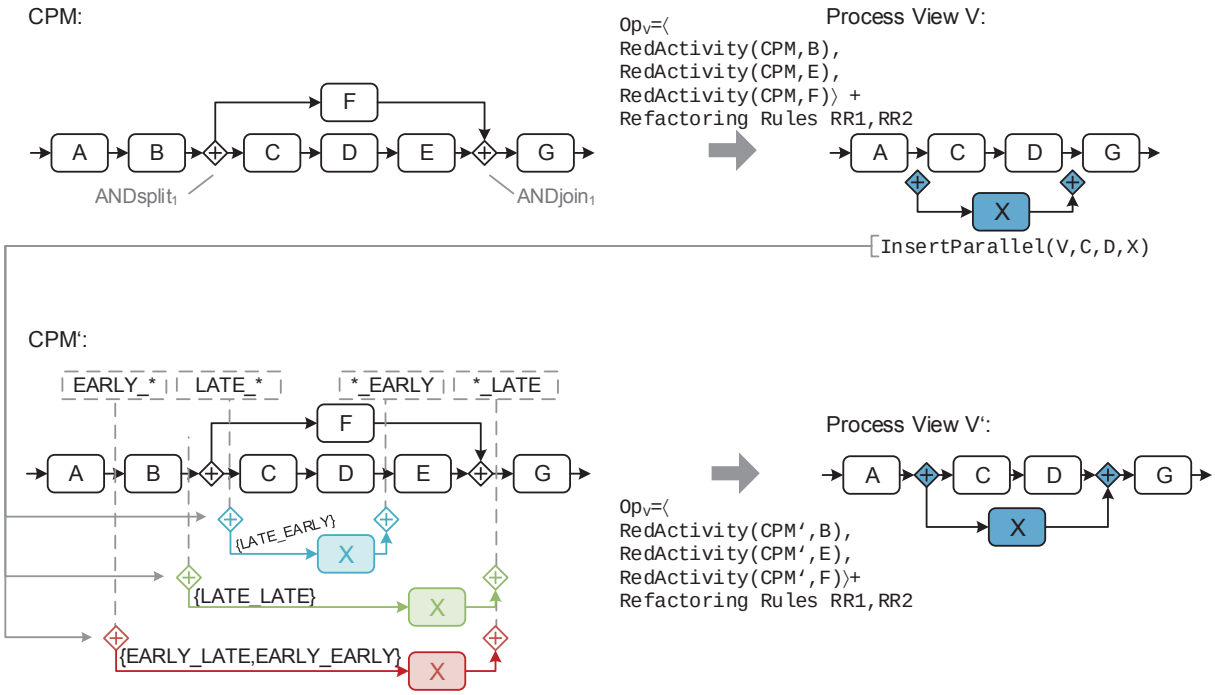


Figure 7.6: View Update Operation: InsertParallel

As can be seen, independent from the concrete parameter value and insert position, re-creating the process view and applying refactoring rules, results in the same process view than applying the update directly to process view V. Before applying view update operation $InsertParallel(V, n_1, n_2, n_{new})$, the following preconditions has to be met.

Preconditions $InsertParallel(V, n_1, n_2, n_{new})$.
Let $V = (N_V, D_V, NT_V, CE_V, EC_V, ET_V, DE_V, DET_V)$ *be the process model of process view* V *with* $CS_V = (CPM, Op_V, PS_V)$. *Then: the following preconditions must be met to apply view update* $InsertParallel(V, n_1, n_2, n_{new})$*:*

- $NT_V(n_{new}) = Activity$
- $NT_V(n_1) \neq StartFlow \ \wedge \ NT_V(n_2) \neq EndFlow$
- $\exists N' \subseteq N_V : MinimalSESE(V, N') = (n_1, n_2)$

Algorithm 7.5 presents the formal behaviour of $InsertParallel(V, n_1, n_2, n_{new})$. Activity n_1 denotes the start and n_2 the end of the SESE block to which n_{new} shall be added in parallel.

Algorithm 7.5: $InsertParallel(V, n_1, n_2, n_{new})$

```
   Input: Process View V based on creation set CS_V = (CPM, Op, PS).
1        CPM = (N, D, NT, CE, EC, ET, DE, DET) and activity n_new to be inserted in parallel to SESE block
         described by start node n_1 and end node n_2.
2  begin
3      n'_1 := last(CPMNode(V, n_1)); n'_2 := first(CPMNode(V, n_2));
4      if (pred(V, last(CPMNode(V, n_1))) ≠ last(CPMNode(V, pred(V, n_1)))) then
5          switch InsertBlockMode
6              case EARLY_EARLY or EARLY_LATE:
7                  n'_1 := succ(CPM, CPMNode(V, pred(n_1)));
8              case LATE_EARLY or LATE_LATE:
9                  n'_1 := last(CPMNode(V, n_1));
10             end
11         end
12     end
13     if (pred(V, last(CPMNode(V, n_2))) ≠ last(CPMNode(V, pred(V, n_2)))) then
14         switch InsertBlockMode
15             case EARLY_EARLY or LATE_EARLY:
16                 n'_2 := first(CPMNode(V, n_2));
17             case EARLY_LATE or LATE_LATE:
18                 n'_2 := succ(CPM, CPMNode(V, pred(V, n_2)));
19             end
20         end
21     end
22     (n_s, n_j) := MinimalSESE(CPM, {n'_1, n'_2});
23     AddNode(CPM, pred(CPM, {n_s}), n_s, g_s, ANDsplit);
24     AddNode(CPM, n_j, succ(CPM, {n_j}), g_j, ANDjoin);
25     AddCEdge(CPM, (g_s, g_j), ET_Control, TRUE);
26     AddNode(CPM, g_s, g_j, n_new, Activity);
27 end
```

When transforming this view update to a corresponding CPM update, it is first checked for the CPM whether the direct predecessor of n_1 is the same node as the one in V (Line 3). If this does not apply, parameter $InsertBlockMode$ is used to decide whether to insert the ANDsplit gateway at the earliest or latest possible location in the CPM (Lines 4-11). The same procedure is applied in respect to the ANDjoin gateway (Lines 12-20). After having determined the insert positions in the CPM, a minimum SESE block is calculated to properly insert the AND branching and to preserve well-structuredness of the CPM (Line 21). Finally, respective process nodes and edges are inserted (Lines 22-25).

Again, it is guaranteed that the application of $InsertParallel$ to the CPM results in the same process view (according to Definition 6.10) as the one we obtain when applying it directly to the process view. The proof is similar to the one provided in the context of operation $InsertSerial$ (cf. Proof 7.1) since $InsertParallel$ can be interpreted as inserting an ANDsplit and ANDjoin gateway serially, i.e., positions of the gateways are in the process view are the same as inserting it directly in the process view.

$InsertCond(V, n_1, n_2, n_{new}, c)$. Similar to *InsertParallel*, the propagation of an update expressed in terms of operation $InsertCond(V, n_1, n_2, n_{new}, c)$ can be accomplished. In addition to the insertion of XORjoin and XORsplit gateways, branching condition c has to be set to guarantee control flow correctness (cf. Table 7.1). To be more precise, branching condition c is assigned to the branch on which n_{new} is located. Accordingly, the branching condition of the other branch

(containing n_1 and n_2) is set to $\neg c$. At run-time, this guarantees that exactly one branch can be selected and no deadlock occurs. As a specific precondition to the ones of $InsertParallel$, the branching condition has to be a valid logical expression in first order logic.

$InsertLoop(V, n_1, n_2, c)$. View update operation $InsertLoop(V, n_1, n_2, c)$ inserts a loop enclosing the SESE block induced by n_1 and n_2. Branching condition c is set for the branch that loops back and $\neg c$ is set for the edge connecting $LOOPsplit$ with the succeeding node of n_2. In order to control the propagation of the view update to the CPM and to avoid ambiguities, parameter $InsertBlockMode$ needs to be set. As opposed to operation $InsertParallel$, however, $InsertLoop$ does not insert an activity. Figure 7.7 shows an example of applying $InsertLoop(V, B, D, c)$ to V. Dotted boxes above CPM' show insert positions of LOOPsplit and LOOPjoin gateways that may be chosen, when propagating the update to the CPM. In particular, determining the position of the LOOPsplit gateway results in a unique insert position between A and B. Due to the reduction of activities E and F as well as the application of refactoring rules, inserting LOOPsplit gateway results in ambiguities regarding its insert position. Depending on *InsertBlockMode*, the LOOPsplit gateway is either inserted between activity D and E (i.e., $*\_EARLY$) or between $ANDjoin_1$ and G. However, to preserve control flow correctness, the latter position has to be selected in both cases. To be more precise, independent of the parameter setting, only one insert position is possible in the example.

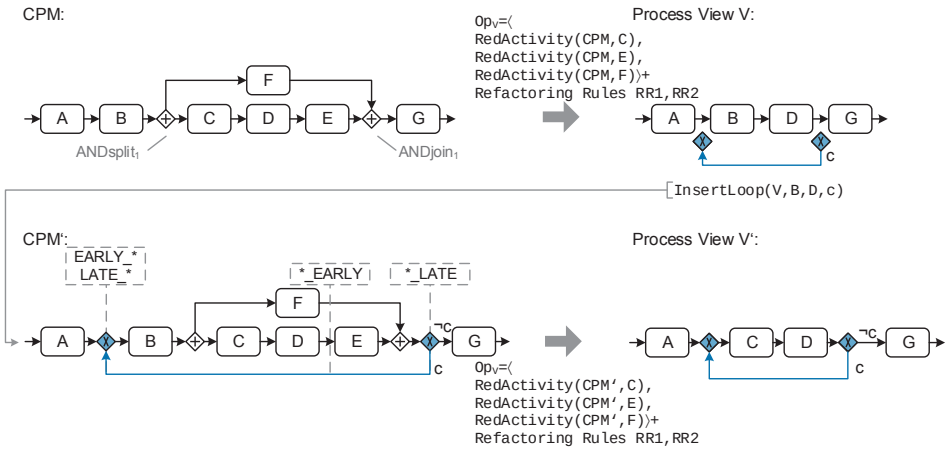


Figure 7.7: View Update Operation: InsertLoop

$InsertBranch(V, g_s, g_j, c)$. View update operation $InsertBranch(V, g_s, g_j, c)$ inserts an empty branch between split gateway g_s and corresponding join gateway g_j. If the gateways are aggregated due to the application of refactoring rules RR5, RR6, or RR7 (i.e., connected branchings), parameter $InsertBranchMode$ determines whether the branch is added to the earliest (i.e., $EARLY$) or the latest (i.e., $LATE$) possible split gateway and, accordingly, to the latest (i.e., $EARLY$) or earliest (i.e., $LATE$) join gateway.

Figure 7.8 shows an example of applying $InsertBranch(V, g_s, g_j, c)$ to the XOR branching described by split gateway g_s and join gateway g_j. Due to the application of refactoring rule RR7,

g_s (g_j) correspond to gateways g_{s1} and g_{s2} (g_{j1} and g_{j2}) in the CPM. Hence, propagating the update to the CPM leads to an ambiguity regarding the insert position of the branch. Parameter $InsertBranchMode$ can be used to deal with this ambiguity. If $InsertBranchMode = EARLY$ holds, a branch is added to the XOR branching built by gateways g_{s1} and g_{j1} (i.e., CPM' in Figure 7.8). In turn, if $InsertBranchMode$ is set to $LATE$, a branch is added to the XOR branching built by g_{s2} and g_{j2} (i.e., CPM'' in Figure 7.8). Moreover, branching condition c has to be added to the new branch. For all other branches, branching conditions have to be extended by a logical AND operator and the negation of branching condition c (i.e., $\neg c$). Thus, we guarantee correctness of the control flow, i.e., no deadlock will occur at run-time.

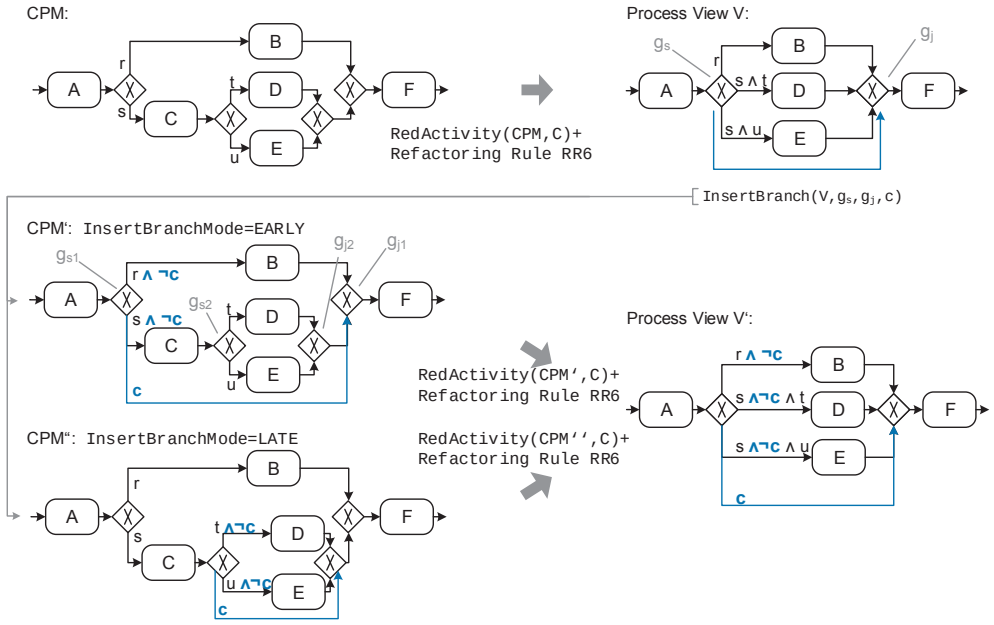


Figure 7.8: View Update Operation: InsertBranch

The listed preconditions must be met to apply view update operation $InsertBranch(V, g_s, g_j, c)$.

Preconditions $InsertBranch(V, g_s, g_j, c)$.
Let $V = (N_V, D_V, NT_V, CE_V, EC_V, ET_V, DE_V, DET_V)$ *be the process model of process view* V *with* $CS_V = (CPM, Op_V, PS_V)$. *Then: The following preconditions must be met to apply* $InsertBranch(V, g_s, g_j, c)$*:*

- $g_s, g_j \in N_V$
- $(NT_V(g_s) = ANDsplit \wedge NT_V(g_j) = ANDjoin) \vee$
  $(NT_V(g_s) = XORsplit \wedge NT_V(g_j) = XORjoin)$
- g_s *and* g_j *are the corresponding gateways of the same XOR/AND branching.*
- *Branching condition* c *is a valid logical expression in first order logic or* $TRUE$ *for AND branchings.*

Algorithm 7.6 describes this operation. Initially, it determines the branching the branch shall be added to (Lines 2-13). In turn, for *XORsplit* gateways, the branching conditions of all branches must be adapted (Line 18). In case of an *ANDsplit*, the edge condition is set to $TRUE$.

Algorithm 7.6: $InsertBranch(V, g_s, g_j, c)$

```
   Input: Process view V = (N_V, D_V, NT_V, CE_V, EC_V, ET_V, DE_V, DET_V) based on creation set
          CS_V = (CPM, Op, PS).
1         CPM = (N, D, NT, CE, EC, ET, DE, DET) and split/join gateway g_s/g_j.
2         Branching condition c in case of an XOR branching; Otherwise: c = TRUE
3  begin
4      if |CPMNode(V, g_s)| = 1 then
5          g'_s := g_s; g'_j := g_j;
6      else
7          G_s := CPMNode(V, g_s); G_j := CPMNode(V, g_j);
8          switch InsertBranchMode
9              case EARLY:
10                 g'_s := first(CPM, G_s); g'_j := last(CPM, G_j);
11             case LATE:
12                 g'_s := last(CPM, G_s); g'_j := first(CPM, G_j);
13             end
14         end
15     end
16     switch NT(g'_s)
17         case ANDsplit:
18             AddCEdge(CPM, (g'_s, g'_j), ET_Control, TRUE);
19         case XORsplit:
20             forall e_b := (g'_s, •) ∈ CE do
21                 EC(e_b) := EC(e_b) ∧ ¬c;
22             end
23             AddCEdge(CPM, (g'_s, g'_j), ET_Control, c);
24         end
25     end
26 end
```

InsertSyncEdge(V, n_s, n_e). View update operation *InsertSyncEdge*(V, n_s, n_e) inserts a synchronization edge from n_s to n_e. Note that propagating this update to the CPM is not straightforward (i.e., ambiguities might occur) since n_s or n_e are aggregated nodes in the process view. Consider the example from Figure 7.9. A synchronization edge shall be inserted between activities DEF and BC in V. Since both activities correspond to abstract nodes that were aggregated using view creation operation *AggrSESE*, the synchronization edge may origin either from D, E, or F and target at either B or C.

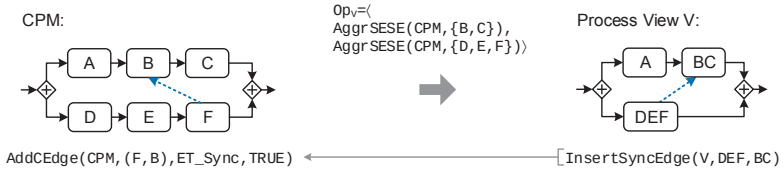


Figure 7.9: View Update Operation: InsertSyncEdge

No parameter is required to resolve this ambiguity. The the synchronization edge expresses that n_e can be activated after n_s has been finished. To be more precise, D, E, and F have to be

finished before B and C may start (cf. Figure 7.9). Accordingly, the synchronization edge must be inserted between activities F and B, i.e., it links the last activity of $CPMNode(V, DEF)$ with the first one of $CPMNode(V, BC)$.

When applying *InsertSyncEdge* to an activity that aggregates several activities of the CPM based on view creation operation *AggrComplBranches*, multiple synchronization edges have to be added to CPM. Figure 7.10 shows an example of inserting a synchronization edge between B and aggregated activity $EFGH$. The latter aggregates the branches containing E, F, G, and H. Propagating the update to the CPM requires to insert a synchronization edge for each branch.

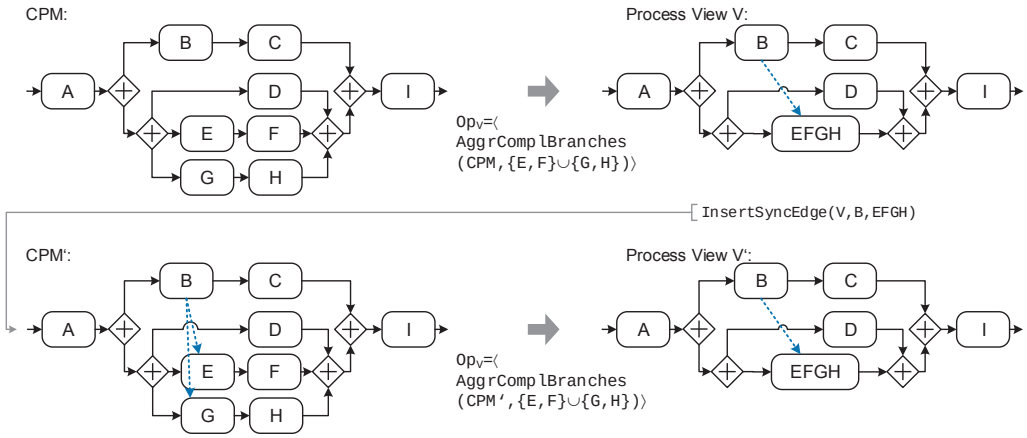


Figure 7.10: View Update Operation: InsertSyncEdge and AggrComplBranches

To apply view update operation $InsertSyncEdge(V, n_s, n_e)$, nodes n_s and n_e have to be activities located on parallel branches.

Preconditions $InsertSyncEdge(V, n_s, n_e)$.
Let $V = (N_V, D_V, NT_V, CE_V, EC_V, ET_V, DE_V, DET_V)$ be the process model of process view V with $CS_V = (CPM, Op_V, PS_V)$. Then: The following preconditions must be met to apply operation $InsertSyncEdge(V, n_s, n_e)$:

- $n_s, n_e \in N_V \wedge NT_V(n_s) = NT_V(n_e) = Activity$
- $\exists p_1, p_2 \in \mathcal{T}_V : \langle \ldots, n_s, \ldots, n_e, \ldots \rangle \wedge \langle \ldots, n_e, \ldots, n_s, \ldots \rangle$, *i.e., activities n_s and n_e are located on parallel branches*

Algorithm 7.7 presents view update operation $InsertSyncEdge(V, n_s, n_e)$ formally. In Line 2, it is checked whether n_s results from the application of view creation operation *AggrComplBranches*. If this applies, for each branch the last node is determined and added to node set N_s (Lines 3-7). Otherwise, either n_s is not an aggregated node or view creation operation *AggrSESE* is applied. In both cases, the last node regarding the control flow of the corresponding CPM node (set) is added to N_s (Line 9). Accordingly, node set N_e is determined (Lines 11-19). In Lines 20-24, synchronization edges are added to the CPM based on nodes from node sets N_s and N_e.

Algorithm 7.7: *InsertSyncEdge*(V, n_s, n_e)

Input: Process View V based on creation set $CS_V = (CPM, Op, PS)$.
$CPM = (N, D, NT, CE, EC, ET, DE, DET)$ and start/end activity n_s/n_e of the synchronization edge.
begin
  if $(\exists AggrComplBranches(V, N_a) \in Op) \ \wedge \ (N_a = CPMNode(V, n_s))$ **then**
    // N_a corresponds to a disjoint union set $N_{ai}, i = 1, \ldots, k$ where each node set comprises
    // all activities of a single branch
    forall $N_{ai} \subseteq N_a$ **do**
      $N_s := N_s \cup \{last(CPM, N_{ai})\}$;
    end
  else
    $N_s := \{last(CPM, CPMNode(V, n_s))\}$;
  end
  if $(\exists AggrComplBranches(V, N_a) \in Op) \ \wedge \ (N_a = CPMNode(V, n_e))$ **then**
    // N_a corresponds to a disjoint union set $N_{ai}, i = 1, \ldots, k$ where each node set comprises
    // all activities of a single branch
    forall $N_{ai} \subseteq N_a$ **do**
      $N_e := N_e \cup \{first(CPM, N_{ai})\}$;
    end
  else
    $N_e := \{first(CPM, CPMNode(V, n_e))\}$;
  end
  forall $n'_s \in N_s$ **do**
    forall $n'_e \in N_e$ **do**
      $AddCEdge(CPM, (n'_s, n'_e), ET\_Sync, TRUE)$;
    end
  end
end

7.3.2 Deleting Process Nodes and Control Edges

From a user perspective two alternatives exist to "delete" a process element in a process view. First, view creation operations reducing process elements may be applied. In Figure 7.11, for example, activity F is deleted by reducing it from V (i.e., by applying $RedActivity(CPM, F)$).

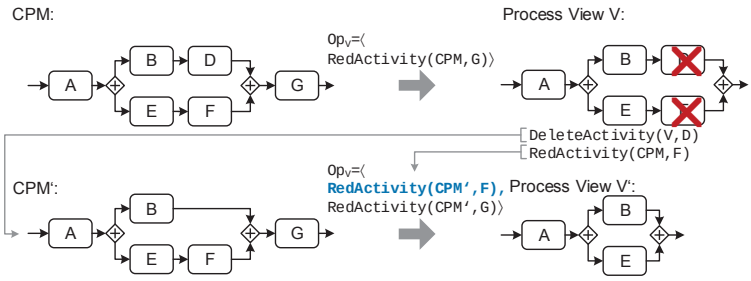


Figure 7.11: Alternatives to Remove an Activity in a Process View

As second alternative, process elements may be removed from a process view by applying view update operations. In Figure 7.11, for example, activity D is deleted by applying view update operation $DeleteActivity(V, D)$. As a result, D is deleted in the CPM as well as in all associated process views. Note that there is no difference between the two alternatives from the perspective of a user interacting with a process view. However, the second alternative modifies

the CPM as well as all other process views. In the following, we focus on operations modifying the underlying CPM as well. View creation operations reducing process elements have been described in Section 6.3. Table 7.2 gives an overview of operations deleting process elements from both a process view and respective CPM.

View Update Operation	**Configuration Parameter & Value**	**Description**
$DeleteActivity(V, n)$	-	Deletes activity n in process view V.
$DeleteBranch(V, g_s, g_j, C)$	-	Deletes an empty branch between gateways g_s and g_j in process view V.
$DeleteSyncEdge(V, n_s, n_e)$	-	Deletes a synchronization edge between activities n_s and n_e in process view V.
$DeleteBranching(V, g_s, g_j)$	DeleteBranchingMode ∈ {**INLINE**, DELETE}	Deletes an AND/XOR/Loop branching block enclosed by gateways g_s and g_j in process view V. The parameter describes whether elements remaining in the block shall be *inlined* or *deleted*.

Table 7.2: View Update Operations - Deleting Process Elements

DeleteActivity(V, n). View update operation *DeleteActivity*(V, n) deletes activity n in process view V. Figure 7.11 has demonstrated the application of *DeleteActivity*(V, D). To apply *DeleteActivity*, the following precondition has to be meet.

Preconditions *DeleteActivity*(V, n).
Let $V = (N_V, D_V, NT_V, CE_V, EC_V, ET_V, DE_V, DET_V)$ *be the process model of a process view* V *with* $CS_V = (CPM, Op_V, PS_V)$. *Then: The following preconditions must be met in order to apply DeleteActivity*(V, n)*:*

- $n \in N_V \ \wedge \ NT_V(n) = Activity$

Algorithm 7.8 defines *DeleteActivity*(V, n) formally. If activity n has resulted from the application of a view creation operation *op* that aggregates process nodes (Lines 2-4), all aggregated nodes are deleted. Furthermore, view creation operation *op* must be removed from operation sequence Op_V (Line 4). In Lines 8-12, the respective nodes and edges are removed from CPM.

Algorithm 7.8: *DeleteActivity*(V, n)

Input: Process view V based on creation set $CS_V = (CPM, Op_V, PS_V)$.
$CPM = (N, D, NT, CE, EC, ET, DE, DET)$ and activity n to be deleted.

```
1  begin
2      if ∃op ∈ Op_V ∧
       (op = AggrSESE(CPM, CPMNode(V, n)) ∨ op = AggrComplBranches(CPM, CPMNode(V, n))) then
3          N_D := CPMNode(V, n);
4          Op_V := Op_V \ ⟨op⟩;
5      else
6          N_D := {n};
7      end
8      N := N \ N_D;
9      e_new := (pred(CPM, N_D), succ(CPM, N_D));
10     EC(e_new) := EC((pred(CPM, N_D), first(CPM, N_D));
11     ET(e_new) := ET((pred(CPM, N_D), first(CPM, N_D));
12     CE := CE \ {e = (n1, n2) ∈ CE | n1 ∈ N_D ∨ n2 ∈ N_D} ∪ {e_new};
13 end
```

$DeleteBranching(V, g_s, g_j)$. View update operation $DeleteBranching(V, g_s, g_j)$ removes the branching block represented by split gateway g_s and join gateway g_j. When applying operation $DeleteBranching$, however, the respective branching block may enclose further process nodes. In this case, parameter $DeleteBranchingMode$ defines whether remaining activities shall be deleted (i.e., $DELETE$) or inlined sequentially (i.e., $INLINE$). Particularly, the latter discards branching conditions of XOR branchings or loops. Another issue deals with combined gateways resulting from the application of refactoring rules RR6, RR7, and RR8. Figure 7.12 shows an example of deleting a non-empty AND branching in process view V. The latter reduces activity C in CPM. Due to the application of refactoring rule RR6, gateways g_{s1}/g_{s2} and g_{j1}/g_{j2} are combined to gateway g_s and g_j in V.

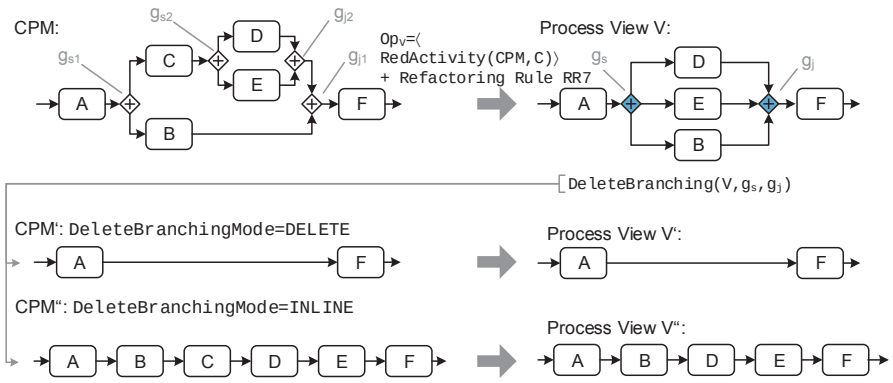


Figure 7.12: View Update Operation: DeleteBranching

Applying view update operation $DeleteBranching(V, g_s, g_j)$ to V in Figure 7.12 leads to the deletion of all nodes between A and F in CPM' (i.e., $DeleteBranchingMode = DELETE$). As an alternative, only the gateways g_{s1}, g_{s2}, g_{j1}, and g_{j2} are deleted and enclosed activities are inlined (i.e., $DeleteBranchingMode = INLINE$), i.e., CPM'' in Figure 7.12.

In the following, preconditions for view update operation $DeleteBranching$ are introduced:

Preconditions $DeleteBranching(V, g_s, g_j)$.
Let $V = (N_V, D_V, NT_V, CE_V, EC_V, ET_V, DE_V, DET_V)$ be the process model of a process view V with $CS_V = (CPM, Op_V, PS_V)$. Then: The following preconditions must be met in order to apply $DeleteBranching(V, g_s, g_j)$:

- $g_s, g_j \in N_V$
- $(NT_V(g_s) = ANDsplit \wedge NT_V(g_j) = ANDjoin) \vee$
  $(NT_V(g_s) = XORsplit \wedge NT_V(g_j) = XORjoin) \vee$
  $(NT_V(g_s) = LOOPsplit \wedge NT_V(g_j) = LOOPjoin)$
- *g_s and g_j gateways of the same branching block.*

Algorithm 7.9 defines this behaviour formally. When parameter $DeleteBranchingMode$ is set to $DELETE$, in Lines 3-9, gateways g_s and g_j in the CPM are determined and, subsequently, all enclosed process nodes (i.e., function $NodesInSESE$) are deleted from the CPM. If $DeleteBranchingMode$ is set to $INLINE$, the respective gateways are deleted. Furthermore, enclosed process nodes are reconnected with each other in a sequential manner.

Algorithm 7.9: $DeleteBranching(V, g_s, g_j)$

Input: Process view V based on creation set $CS_V = (CPM, Op, PS)$.
$CPM = (N, D, NT, CE, EC, ET, DE, DET)$ and SESE block represented by split gateway g_s and join gateway g_j.

```
1  begin
2    switch DeleteBranchingMode
3      case DELETE:
4        g'_s := first(CPM, CPMNode(V, g_s)); g'_j := last(CPM, CPMNode(V, g_j));
5        //returns node set of SESE described by start node g'_s and end node g'_j
6        N_b := NodesInSESE(CPM, g'_s, g'_j);
7        N := N \ N_b;
8        CE := CE ∪ {(pred(CPM, g'_s), succ(CPM, g'_j))};
9        CE := CE \ {e = (n_1, n_2) ∈ CE | n_1 ∨ n_2 ∈ N_b};
10     case INLINE:
11       forall gateways g'_s in CPMNode(V, g_s) do
12         N := N' \ {g'_s, g'_j};
13         n_pre := pred(CPM, g'_s);
14         forall outgoing edges e_s = (g'_s, n_s) of gateway g'_s do
15           e_e := (n_e, g'_j); // last edge on same branch as e_s before join gateway g'_j
16           CE := CE \ {e_s, e_e} ∪ {(n_pre, n_s)};
17           n_pre := n_e;
18         end
19         CE := CE ∪ {(n_pre, succ(CPM, g'_j))} \ {(g'_j, succ(CPM, g'_j))};
20       end
21     end
22   end
23 end
```

$DeleteBranch(V, g_s, g_j, EC')$. View update operation $DeleteBranch(V, g_s, g_j, EC')$ removes an empty branch between split gateway g_s and join gateway g_j (i.e., $\exists(g_s, g_j) \in CE$). Removing such an empty branch is straightforward for AND branchings, but branching conditions of remaining XOR and Loop branches need to be adapted to guarantee control flow correctness. For this, function EC' provides adapted branching conditions for all remaining branches. Particularly, all branching conditions must be valid logical expressions in first order logic. We omit a formal definition of view update operation $DeleteBranch(V, g_s, g_j, EC')$.

$DeleteSyncEdge(V, n_s, n_e)$. Deleting a synchronization edge by applying view update operation $DeleteSyncEdge(V, n_s, n_e)$ is required to remove a control flow dependency between activities on different parallel branches. However, its definition is straightforward, i.e., the corresponding synchronization edge is removed from the set of control edges CE in CPM.

7.3.3 Updating the Data Flow

This section describes operations for updating the data flow of a process view as well as the related CPM. In particular, new data edges may have to be inserted or data elements be inserted or deleted. Table 7.3 summarizes these operations as well as their parameters.

View Update Operation	Configuration Parameter & Value	Description
$InsertDataEdge(V, de, det)$	InsertEdgeMode ∈ {**EARLY**, LATE, ALL}	Inserts a data edge de of type det in process view V. Parameter $InsertEdgeMode$ controls the propagation behaviour in case of ambiguities.
$InsertDataElement(V, d, de, det)$	InsertEdgeMode ∈ {**EARLY**, LATE, ALL}	Inserts data element d in process view V and connects it with data edge $de = (d, n)$ ($de = (n, d)$) to activity n. Parameter $InsertEdgeMode$ controls how to propagate the insertion of data edge de in case of ambiguities.
$ChangeDEType(V, de, det)$	-	Changes the data edge type of data edge de to type det in process view V.
$DeleteDataElement(V, d)$	-	Deletes data element d in process view V as well as all associated data edges.
$DeleteDataEdge(V, de)$	-	Deletes data edge de in process view V.

Table 7.3: View Update Operations - Update Data Flow

$InsertDataEdge(V, de, det)$. View update operation $InsertDataEdge(V, de, det)$ inserts data edge de to process view V and its CPM. Data edge $de = (n_s, n_e)$ indicates whether a read/write data edge is considered. If n_s is a process node, a write data edge is inserted. In turn, if n_s is a data element a read edge is inserted. In particular, it is not allowed that both n_s and n_e constitute either process nodes or data elements. Furthermore, det denotes the data edge type (i.e., mandatory or optional). When inserting a data edge in a process view, ambiguities may occur in respect to the transformation of this view update to the CPM if the activity the data edge is connected with constitutes an aggregated one.

Consider the example from Figure 7.13. Operation $InsertDataEdge(V, (d, BC), always)$ is applied to process view V. Since BC constitutes an aggregated activity (i.e., it represents a set of activities of the CPM), a proper insert position of the data edge has to be determined. In the CPM, data edge de may be connected to B (i.e., $InsertEdgeMode = EARLY$), C (i.e., $InsertEdgeMode = LAST$), or to both activities (i.e., $InsertEdgeMode = ALL$). The ambiguity is caused due to the aggregation of the CPM activities B and C in V. Independent of the value of parameter $InsertEdgeMode$, the same process view V' results (cf. Figure 7.13).

Preconditions $InsertDataEdge(V, de = (n_s, n_e), det)$.
Let $V = (N_V, D_V, NT_V, CE_V, EC_V, ET_V, DE_V, DET_V)$ be the process model of a process view V with $CS_V = (CPM, Op_V, PS_V)$. Then: The follwoing precondition must be met to apply $InsertDataEdge(V, (n_s, n_e), det)$:

- $(n_s \in N_V \wedge NT(n_s) = Activity \wedge n_e \in D_V) \dot{\vee}^1 (n_e \in N_V \wedge NT(n_e) = Activity \wedge n_s \in D_V)$

[1] Logical operator $\dot{\vee}$ stands for exclusive OR

Algorithm 7.10 describes this update operation formally. First, it is checked whether the data edge writes or reads a data element (Line 2). In the latter case, the activity n reading the data element in V is identified. If d corresponds to an aggregated data element (cf. Section 6.3.2), the respective reading edge is set for all aggregated data elements (Line 3).

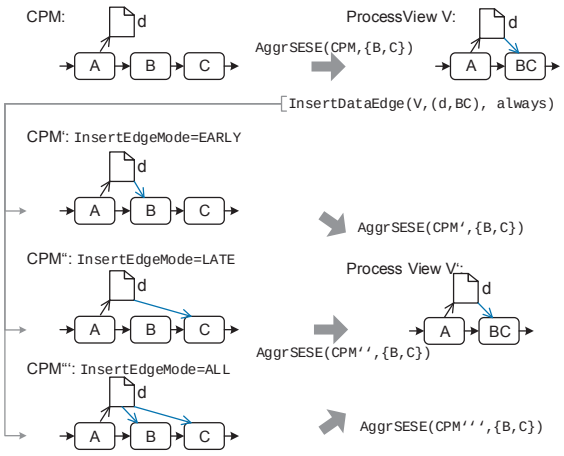


Figure 7.13: View Update Operation: InsertDataEdge

Algorithm 7.10: $InsertDataEdge(V, de, det)$

```
   Input: Process View V based on creation set CS_V = (CPM, Op_V, PS_V).
1          CPM = (N, D, NT, CE, EC, ET, DE, DET) and data edge de to be inserted as well as data edge type det.
2  begin
3      if (de = (d', n) ∧ n ∈ N) //i.e., reading data edge then
4          forall (d ∈ CPMNode(V, d')) do
5              if (isAggregatedNode(V, n)) then
6                  N' := CPMNode(V, n);
7                  switch InsertEdgeMode
8                      case EARLY:
9                          AddDEdge(CPM, (d, first(CPM, N')), det);
10                     case LATE:
11                         AddDEdge(CPM, (d, last(CPM, N')), det);
12                     case ALL:
13                         forall (n' ∈ N') do
14                             AddDEdge(CPM, (d, n'), det);
15                         end
16                     end
17                 end
18             else
19                 AddDEdge(CPM, (d, n), det);
20             end
21         end
22     end
23     else //analogous for a writing edge;
24 end
```

For each data edge, *isAggregatedNode* (cf. Section 7.3) checks whether the associated activity n results from an aggregation (Line 4). If n is not an aggregated node (i.e., the activity is contained in CPM as well), the insertion is straightforward (Line 18). In turn, if n constitutes an aggregated node *CPMNode* (cf. Section 6.2) is applied to obtain the nodes N' from CPM corresponding to activity n. Depending on the value of *InsertEdgeMode*, the data edge is added to the CPM at the *earliest*/*latest* position based on node set N' (Line 5). If $InsertEdgeMode$ is set to ALL, data edges are added to all nodes of N'.

$InsertDataElement(V, d, de, det)$. View update operation $InsertDataElement(V, d, de, det)$ inserts data element d in process view V together with a corresponding data edge de of type det. Inserting a new data element and connecting it directly with an activity contributes to avoid unconnected data elements in the process view or CPM. If a user solely wants to insert a data element, he will most likely connect it to one of the activities in the following modeling steps.

Inserting a new data element is straightforward, i.e., the data element is added to the set of data elements D of the CPM. To insert a corresponding data edge, view update operation $InsertDataEdge$ (cf. Algorithm 7.10) is used. Analogously, parameter $InsertEdgeMode$ resolves ambiguities when inserting a data edge.

$ChangeDEType(V, de, det)$. View update operation $ChangeDEType(V, de, det)$ changes the type of data edge de to det; the latter describes whether the data element is accessed *mandatorily* or *optionally*. However, if either the associated activity or data element is result of an aggregation operation, the types of all data edges are updated. For example, consider Figure 7.14. Data element d is optionally read by B and C in CPM. In turn, B and C are aggregated in process view V. Applying $ChangeDEType(V, (d, BC), mandatory)$ to V and propagating this change to the CPM then requires updating the type of data edges (d, B) and (d, C) in CPM'. We omit a formal definition of $ChangeDEType$ here.

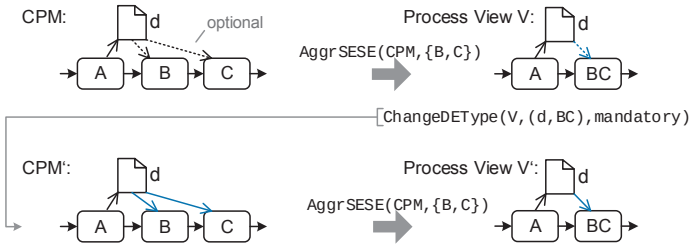


Figure 7.14: View Update Operation: ChangeDEType

$DeleteDataEdge(V, de)$. View update operation $DeleteDataEdge(V, de)$ deletes data edge de from process view V and the related CPM. Due to the application of aggregation operations on data elements or activities associated with de, it might be required to delete multiple elementary data edges in the CPM. In particular, removing write edges might violate data flow correctness of the CPM as well as the process view. Figure 7.15 shows an application of this operation in the context of deleting data elements. In the following, preconditions of view update operation $DeleteDataEdge(V, de)$ are introduced.

Preconditions $DeleteDataEdge(V, de)$.
Let $V = (N_V, D_V, NT_V, CE_V, EC_V, ET_V, DE_V, DET_V)$ *be the process model of a process view with* $CS_V = (CPM, Op_V, PS_V)$. *Then: The following precondition must be met to apply* $DeleteDataEdge(V, de)$*:*

- $de \in DE_V$

If the precondition is met, *DeleteDataEdge* may be applied (cf. Algorithm 7.11).

Algorithm 7.11: $DeleteDataEdge(V, de)$

```
   Input: Process View V based on creation set CS_V = (CPM, Op_V, PS_V).
1          CPM = (N, D, NT, CE, EC, ET, DE, DET) and
2          data edge de = (n_s, n_e) to be deleted.
3  begin
4      forall (n'_s ∈ CPMNode(V, n_s)) do
5          forall (n'_e ∈ CPMNode(V, n_e)) do
6              if (n'_s, n'_e) ∈ DE then
7                  DE := DE \ {(n'_s, n'_e)};
8              end
9          end
10     end
11 end
```

$DeleteDataElement(V, d)$. View update operation $DeleteDataElement(V, d)$ deletes data element d from process view V together with its associated data edges. For deleting the latter, view update operation *DeleteDataEdge* may be used. In case of aggregated data elements, elementary data elements as well as their associated data edges are removed. Figure 7.15 shows an example of deleting an aggregated data element (i.e., $d12$). Propagating the respective update to the CPM requires the deletion of data elements $d1$ and $d2$, as well as data edges $(A, d1), (B, d2)$, and $(d1, C)$. Note that *DeleteDataElement* preserves the correctness of the data flow since all reading and writing edges of the data element are removed. Finally, $AggrDataElement(CPM, d1, d2)$ must be removed from operation sequence Op_V.

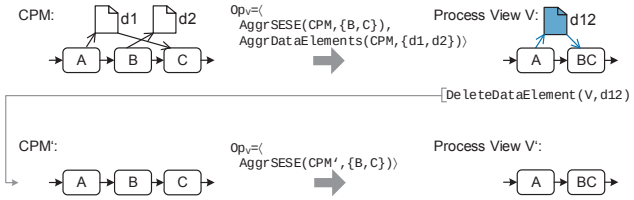


Figure 7.15: View Update Operation: DeleteDataElement

The following precondition for operation $DeleteDataElement(V, d)$ ensures that d constitutes a data element present in process view V.

Preconditions $DeleteDataElement(V, d)$.
Let $V = (N_V, D_V, NT_V, CE_V, EC_V, ET_V, DE_V, DET_V)$ *be the process model of a process view* V *with* $CS_V = (CPM, Op_V, PS_V)$. *Then: The following preconditions must be met in order to apply* $DeleteDataElement(V, d)$*:*

- $d \in D_V$

Algorithm 7.12 presents *DeleteDataElement* formally. In Lines 2-6, both data elements and data edges are deleted in the CPM. If applicable, in Lines 7-10 view creation operation aggregating deleted data elements is removed from operation sequence Op of creation set CS_V.

Algorithm 7.12: *DeleteDataElement*(V, d)

```
   Input: Process View V based on creation set CS_V = (CPM, Op_V, PS_V).
1          CPM = (N, D, NT, CE, EC, ET, DE, DET) and
2          data element d ∈ D to be deleted.
3  begin
4      forall (d' ∈ CPMNode(V, d)) do
5          forall de ∈ {(d', n) ∈ DE ∨ (n, d') ∈ DE | n ∈ N ∧ d' ∈ D} do
6              DeleteDataEdge(V,de);
7          end
8          D := D \ {d'};
9      end
10     if ∃op := AggrDataElements(CPM, CPMNode(V, d)) ∈ Op_V then
11         Op_V := Op_V \ ⟨op⟩;
12     end
13 end
```

7.3.4 Process Model Correctness

This section discusses the impact view update operations have on the correctness of process models. In general, view update operations might violate the correctness of the control flow (cf. Constraints 1-4, Definition 6.3) or the data flow (cf. Constraints 5+6, Definition 6.3) of a process model. For example, one might delete the data edge solely writing a data element. Consequently, succeeding activities might fail when reading the respective data element at run-time.

Table 7.4 gives an overview on the properties of view update operations. Consider a CPM with correct data and control flow. *Data flow correctness preserving* denotes that applying the respective operation on a process view and corresponding CPM the data flow correctness is not violated. Finally, *control flow correctness preserving* expresses whether control flow correctness may be violated by a view update operation.

View Update Operation	**Data Flow Correctness Preserving**	**Control Flow Correctness Preserving**
InsertSerial	✓	✓
InsertParallel	✓	✓
InsertCond	✗	✓
InsertLoop	✓	✓
InsertBranch	✗	✓
InsertSyncEdge	✓	✗
DeleteActivity	✗	✓
DeleteBranch	✓	✓
DeleteSyncEdge	✓	✓
DeleteBranching	✗	✓
InsertDataElement	✗	✓
InsertDataEdge	✗	✓
ChangeDEType	✗	✓
DeleteDataElement	✓	✓
DeleteDataEdge	✗	✓

✓=property fulfilled, ✗=property violated

Table 7.4: Overview of View Operation Properties

Data flow correctness might be violated when applying operations that update the control or data flow. For example, when inserting an XOR branching with *InsertCond* in Figure 7.16a, activity B, solely writing data element d, will only be executed conditionally after inserting the XOR branching. Another example is shown in Figure 7.16b: activity B, solely writing data element d, is deleted. As a result, d is not written anymore.

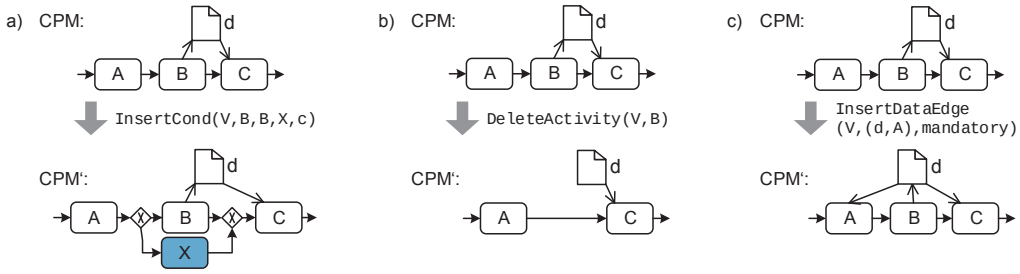


Figure 7.16: View Update Operation Affecting Data Flow Correctness

Most of the view update operations changing the data flow may violate data flow correctness. For example, when inserting a mandatory read data edge (d, A) in Figure 7.16c, data element d will not be written before being read by A. By contrast, *DeleteDataElement* preserves data flow correctness since the data element and its data edges are removed entirely.

Control flow correctness (cf. Definition 6.3) is preserved by most of the view update operations (cf. Table 7.4). To be more precise, *InsertSyncEdge* might violate control flow correctness, since the insertion of a synchronization edge could lead to a deadlock-causing cycle in the CPM. However, the other operations ensure control flow correctness. In particular, users cannot create process models with an incorrect control flow. If data flow correctness is required in addition (e.g., for process execution), it has to be checked when applying operations *not* preserving data flow correctness, if data flow correctness is still given.

7.3.5 Updating Process Attributes

In the following, we introduce operations to insert or delete attributes of process nodes. Furthermore, view update operations are introduced to update the value of existing parameters. The latter, for example, are required to change the label of an activity. Table 7.5 gives an overview of these operations.

View Update Operation	**Configuration Parameter & Value**	**Description**
$UpdateAttribute(V, n.x, val)$	-	Changes the value of attribute x at process node n to val in process view V.
$InsertAttribute(V, n.x, val)$	-	Inserts an attribute x to process node n and assigns value val to x in process view V.
$DeleteAttribute(V, n.x)$	-	Deletes attribute x of process node n in process view V.

Table 7.5: View Update Operation: Updating Process Attributes

UpdateAttribute$(V, n.x, val)$. This operation changes the value of an attribute x of node n to val. Propagating this view update operation to the associated CPM results in three cases:

Case 1 (Direct Update). Attribute value val can be directly propagated to the CPM (i.e., updated in the CPM) if neither n nor x result from the application of an aggregation operation (cf. Section 6.3). Consider the example from Figure 7.17. Applying *UpdateAttribute*$(V, A.label, X)$ to V and propagating this update to CPM results in a changed label "X" of A.

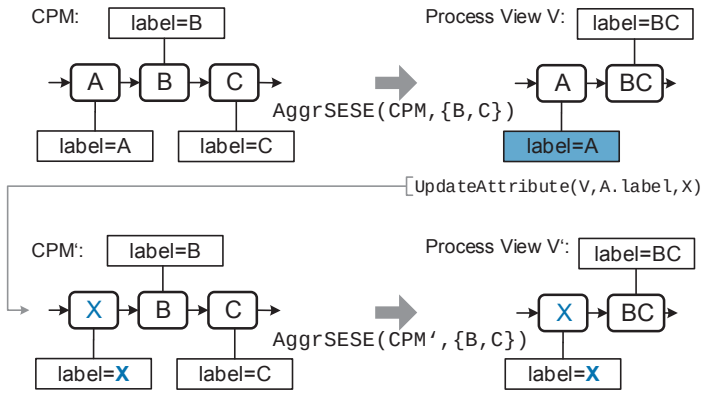


Figure 7.17: View Update Operation: UpdateAttribute

Case 2 (Aggregated Nodes). Process node n in V represents an aggregated set of nodes $N' = \{n_1, \ldots, n_k\}$ from CPM, i.e., the actual value of attribute x has been calculated based on the attribute values of all nodes in N' (cf. Section 6.3.3). In this context, attribute function $attrFunc$ is applied to aggregate respective attribute values: $attrFunc$ (cf. Algorithm 6.7): $val(n_1.x), \ldots, val(n_k.x) \rightarrow val(n.x)$.

Consider the example from Figure 7.17. The value of attribute $BC.label$ is calculated by applying attribute function $CONCAT$. The latter concatenates textual attribute values. When applying *UpdateAttribute*$(V, BC.label, BZ)$ it has to be determined, which part of the new value "BZ" corresponds to attribute $B.label$ and which to attribute $C.label$. Hence, an *inverse function* $CONCAT^{-1}$ calculating attribute values in the CPM is required. When applying $CONCAT^{-1}$, a trivial solution might be to try to match the updated value to respective process nodes (in Figure 7.17, value of $C.label$ is changed to "Z"). Another solution may simply divide the text of the new attribute value into parts of equal length.

A similar issue occurs when applying *UpdateAttribute* to numeric values. Consider transformation function SUM that sums up a set of attribute values, i.e., $n.x = \sum_{i=1}^{k} n_i.x$, $n_i \in N'$. An inverse function SUM^{-1} might assign a new value $val'_i = val/ \mid N' \mid$ to each attribute or the value of attribute $n_i.x$ is calculated proportionally to the old value of attribute $n_i.x$. Furthermore, if node set N' contains an XOR branching, the branching probability has to be taken into account as well. To be more precise, each branch of an XOR branching may be executed with a different probability. In turn, this has an impact on the inverse attribute function. Similar issues arise for loops regarding the number of loop iterations.

Case 3 (Aggregated Attributes). If an attribute $n.x$ has been aggregated in a process view, i.e., view creation function *AggrAttr* has been applied, like in Case 2 the inverse attribute function needs to be defined.

Note that we do not provide a generic solution to determine inverse attribute function for Cases 2 and 3.

$InsertAttribute(V, n.x, val)$. In certain situations, it may be required to insert an attribute to a process node. For example, a user wants to insert an attribute describing the risk of executing the respective process node. View update operation $InsertAttribute(V, n.x, val)$ allows inserting an attribute x to process node n. Furthermore, attribute value val is assigned to $n.x$. Note that such an attribute is not available for all process nodes of the same type (e.g., all activities), but only for that specific node n the operation is applied to.

$DeleteAttribute(V, n.x)$. View update operation $DeleteAttribute(V, n.x)$ deletes an attribute x of process node n in process view V. In case of aggregated attributes (e.g., due to the aggregation of activities), all associated attributes are deleted in the CPM as well.

7.3.6 Summary

This section has introduced view update operations for modifying control and data flow elements of a process model. These operations may be applied to a process view and then be automatically propagated to the CPM. However, regarding process attributes an automated update propagation is not always possible. Furthermore, the impact of view update operations on the correctness of the control and data flow correctness has been discussed. This is required to determine for which view update operations additional checks regarding process model correctness should be applied.

7.4 Migration Rules

After applying a view update operation to the corresponding CPM, all other process views associated with this CPM must be updated accordingly, i.e., it must be guaranteed that all process views created on this CPM are kept up-to-date and that users always interact with the current version of the CPM and its related process views. Formally, after propagating a process view update to the CPM, the creation sets of all other process views must be migrated to the new CPM version (cf. Definition 6.6). Example 7.2 describes such a situation.

Example 7.2 (Migrating Process Views)
Consider the credit application process as introduced in Example 6.1 as well as process views V2 and V3 from Figure 7.18. V2 displays the activities of user role customer and aggregates activities of other users. In turn, V3 is created for user role Manager and only shows the activities of this specific user role, i.e., all other activities are reduced.

Consider the application of $InsertSerial(V, XORsplit_1, a4, a14)$ as described in Example 7.1: Activity Create File (i.e., a14) is inserted between gateway "Existing Customer?" (i.e., $XORsplit_1$) and activity Create Customer (i.e., a4) in the CPM. Subsequently, V2 and V3 have to be migrated to the new CPM version. Regarding V2, the branching in which a14 is inserted, is aggregated. Accordingly, view creation operations in Op_{V2} need to be adopted. Otherwise, operation $AggrSESE(CPM, \{a2, a3, a4, a5, a6, a7\}) \in Op_{V2}$ would not be applied to a SESE block and, hence, its precondition is not met anymore. Therefore, either the set of nodes aggregated by AggrSESE must be extended (i.e., $V2'$) or the aggregation operation must be removed from Op_{V2} (i.e., $V2''$).

Regarding V3, the activities surrounding a14 in CPM' are reduced, i.e., they are not relevant for the Manager. Hence, a14 may be reduced as well (i.e., $V3'$). Alternatively, a14 may be shown to the user when migrating process view V3 (i.e., process view $V3'$).

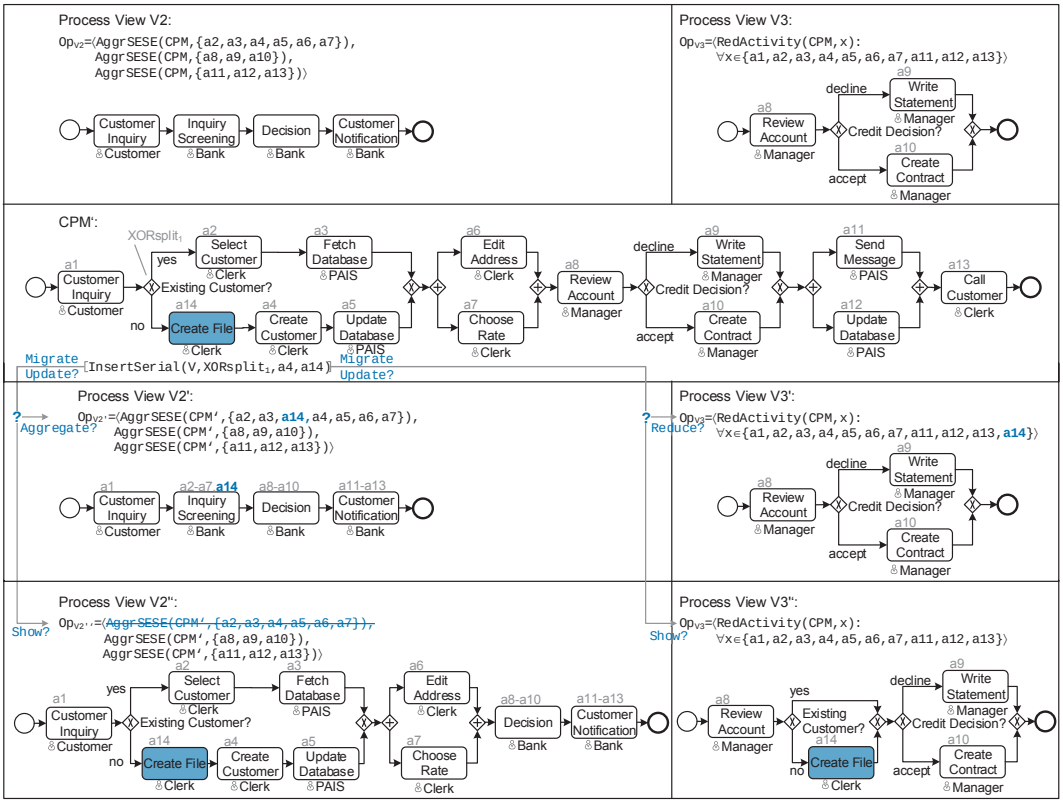


Figure 7.18: Migrating Process Views

Example 7.2 reveals some challenges that arise when migrating process views to a new CPM version. In particular, after applying a CPM it may be required for a process view to adapt their operation sequence or to apply new view creation operations. For this purpose, we introduce *migration rules*, which allow migrating creation sets of process views associated with an updated CPM. In particular, these migration rules are configurable through parameters to control their behaviour. For example, $AggrComplMode = \{AGGR, SHOW\}$ specifies whether activity $a14$ in process view $V2$ (cf. Figure 7.18) shall be aggregated (i.e., $V2'$) or the respective aggregation operation shall be removed from the operation sequence of process view $V2$ (i.e., $V2''$). Analogous to view update operations, for each process view V, the parameters describing the migration behaviour are maintained in parameter set PS_V of creation set $CS_V = (CPM, Op_V, PS_V)$. Consequently, an individual migration behaviour may be configured for each process view.

In order to describe the influence of migration rules on operation sequences of process views, Definition 7.3 describes the concatenation and subtraction of operation sequences.

Definition 7.3 (Concatenation,Subtraction)
Let Op_1 and Op_2 two operation sequences. Then:
The concatenation of $Op_1 = \langle op_1, \ldots, op_i \rangle$ and $Op_2 = \langle op_{i+1}, \ldots, op_k \rangle$ is defined as:
$Op_1 \oplus Op_2 := \langle op_1, \ldots, op_i, op_{i+1}, \ldots, op_k \rangle$

The subtraction of $Op_1 = \langle op_1, \ldots, op_k \rangle$ and $Op_2 = \langle op_i, op_j \rangle, 1 \leq i < j \leq k$ is defined as:
$Op_1 \ominus Op_2 := \langle op_1, \ldots, op_{i-1}, op_{i+1}, \ldots, op_{j-1}, op_{j+1}, \ldots, op_k \rangle$

In the following eight migration rules are introduced: MR1-MR4 may be applied after inserting process nodes to a CPM (cf. Section 7.4.1). Migration rules MR5 and MR6, in turn, may be required when deleting process nodes in the CPM (cf. Section 7.4.2). Migration rule MR7 is required when updating the data flow (cf. Section 7.4.3) and migration rule MR8 is required when updating process attributes in the CPM (cf. Section 7.4.4). Figure 7.19 summarizes these migration rules. Finally, Section 7.4.5 discusses the introduced migrations rules with respect to their completeness.

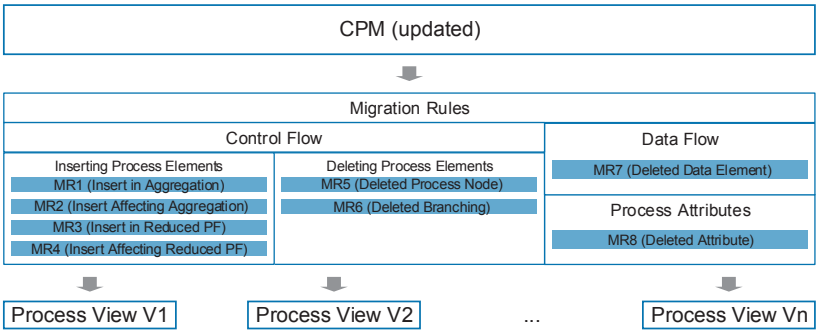


Figure 7.19: Overview on Migration Rules

7.4.1 Migration after Inserting Process Nodes

Inserting a process node in a process fragment, which is aggregated in a process view, requires updating the creation set (cf. Example 7.2). For this purpose, migration rule MR1 is applied.

Let N_c denote the set of process nodes added to the CPM by a view update operation. As example, consider $N_c = \{a14\}$ in Figure 7.18. If the direct predecessors and successors of all process nodes from N_c are aggregated to the same abstracted node[1], migration rule MR1 is applied to migrate the process views to the new CPM version. In this context, two alternatives exist: either N_c is included in the respective aggregation operation or the latter is removed from the process view definition (i.e., Op_V of creation set $CS_V = (CPM, Op_V, PS_V)$). The latter alternative shows all aggregated nodes as well as the inserted process nodes in the respective process view. For each process view, this behaviour is controllable with parameter *AggrComplMode*. If this parameter is set to *SHOW*, the respective aggregation operation is removed from the creation set. In turn, if the parameter is set to *AGGR* (default), the node set of the respective aggregation operation is extended with the process nodes in N_c. In the latter case, the user might be notified about the update within the aggregation (cf. Migration Rule MR1).

Migration Rule MR1 (Insert in Aggregation)
Let $CPM = (N, D, NT, CE, EC, ET, DE, DET)$ be a process model and V be a process view with $CS_V = (CPM, Op_V, PS_V)$. Further, let $N_c \subset N$ be the node set inserted to CPM.
***Precondition**:*
$\exists op_1 \in Op_V$:
$(op_1 = AggrSESE(CPM, N_a) \ \vee \ op_1 = AggrComplBranches(CPM, N_a)) \wedge$
$N_a \supset \{pred(CPM, N_c), succ(CPM, N_c)\} \ \wedge \ (N_a \subset N)$

***Postcondition**:*
AggrComplMode=SHOW: $Op_V := Op_V \ominus \langle op_1 \rangle$
AggrComplMode=AGGR: $op_1 := AggrSESE(CPM, N_a \cup N_c)$
(or $op_1 := AggrComplBranches(CPM, N_a \cup N_c)$ *depending on type of* op_1*)*

Figure 7.20 illustrates the application of migration rule MR1. A user inserts activity X in parallel to C in process view $V1$. Afterwards, process views $V2$ and $V3$ are migrated to the new version of the CPM. Obviously, both process views require the migration of their creation sets since both aggregate B, C, and D. When applying MR1 to operation sequence Op_{V2}, the aggregation operation is updated to also consider activity X. Note that for $V2$, parameter *AggrComplMode* is set to *AGGR*. In turn, aggregation operation $AggrSESE$ is removed from Op_{V3} in $V3$ since *AggrComplMode* is set to $SHOW$. This results in dissolving the aggregation in which the update occurred.

Similar to MR1, migration rule MR2 handles the case in which either the predecessor *or* successor of nodes from N_c is part of an aggregation. Similar to parameter $AggrComplMode$, *AggrPartlyMode* expresses whether the aggregation shall be expanded by process nodes in N_c

[1] To be more precise, both are element of node set N_a that has been aggregated by $AggrSESE(CPM, N_a)$ or $AggrComplBranches(CPM, N_a)$

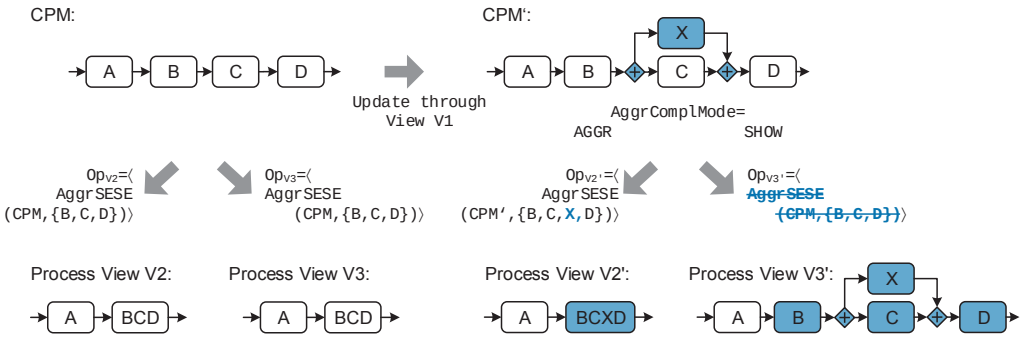


Figure 7.20: Migration Rule MR1

(i.e., $AGGR$) or the aggregation operation shall be removed (i.e., $SHOW$). As example consider Figure 7.21. Activity X is inserted between A and B in the CPM. Subsequently, $V2$ and $V3$ need to be migrated to the new version of CPM. Both process views aggregate activities A, B, and C. Migration rule MR2 with parameter setting $AggrPartlyMode = AGGR$ is applied to $V2$, which leads to the inclusion of X in view creation operation $AggrSESE(CPM', \{B, C, X, D\})$. However, $V3$ applies migration rule MR2 with parameter setting $AggrPartlyMode = SHOW$. As a result, operation sequence Op_{V3} is not modified and X is shown in $V3$.

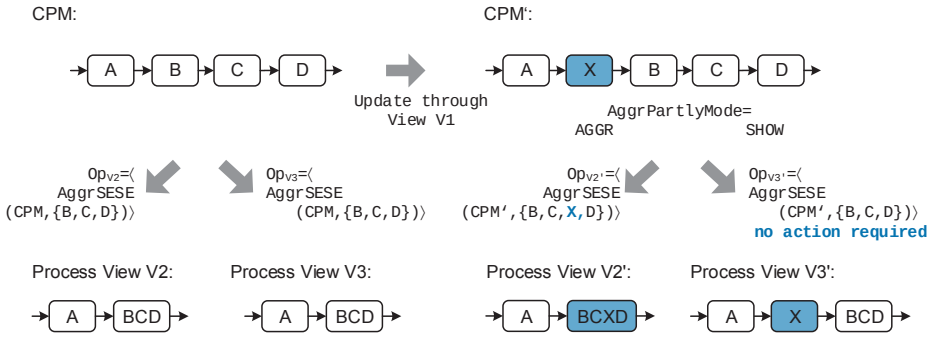


Figure 7.21: Migration Rule MR2

Note that migration rules MR1 and MR2 allow specifying a different behaviour if an update happens completely within an aggregated process fragment (i.e., the predecessor and successor of inserted process fragment are part of the same aggregation) or at the start/end of an aggregated process fragment (i.e., migration rule MR2). In the following, migration rule MR2 is introduced formally.

Migration Rule MR2 (Insert Affecting Aggregation)
Let $CPM = (N, D, NT, CE, EC, ET, DE, DET)$ be a process model and V be a process view with $CS_V = (CPM, Op_V, PS_V)$. Let $N_c \subset N$ be the node set inserted to CPM.
***Precondition**:*
$\exists op_1 \in Op_V$:
$(op_1 = AggrSESE(CPM, N_a) \ \vee \ op_1 = AggrComplBranches(CPM, N_a)) \wedge$
$(pred(CPM, N_c) \in N_a \ \dot{\vee} \ succ(CPM, N_c) \in N_a) \wedge N_a \subset N$

***Postcondition**:*
AggrPartlyMode=SHOW: no action required
AggrPartlyMode=AGGR: $op_1 := AggrSESE(CPM, N_a \cup N_c)$
(or $op_1 := AggrComplBranches(CPM, N_a \cup N_c)$ *depending on* op_1*)*

Migration rules MR3 and MR4 deal with updates of CPM process fragments not present in a process view.

Similar to the handling of aggregation operations, MR3 is applied if the predecessors as well as successors of N_c are removed due to the application of reduction operations. In this case, parameter *RedComplMode* and its values (*SHOW* and *RED* (default)) determine whether N_c shall be shown or hidden in the respective process view. As example consider Figure 7.22: X is inserted between B and C in the CPM. B and C are neither contained in $V2$ nor in $V3$. Since the predecessor and successor of activity X is reduced in process view $V2$, it might be desired to reduce activity X as well. Therefore, migration rule MR3 adds view creation operation $RedActivity(CPM', X)$ to the operation sequence of process view $V2$, in case parameter *RedComplMode* is set to *RED*.

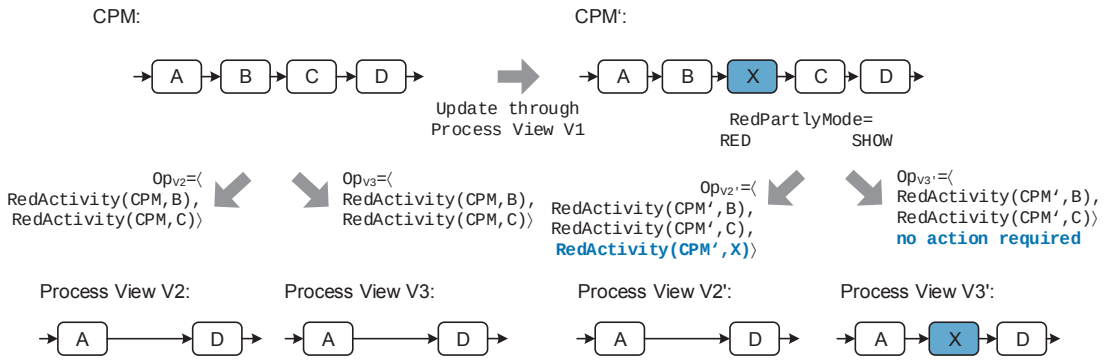


Figure 7.22: Migration Rule MR3

In certain scenarios it is required to show an inserted activity X in a process view. To reflect this, parameter *RedComplMode* must be set to *SHOW* (e.g., process view $V3$ in Figure 7.22).

Migration Rule MR3 (Insert in Reduced Process Fragment)
Let $CPM = (N, D, NT, CE, EC, ET, DE, DET)$ be a process model and V a process view with $CS_V = (CPM, Op_V, PS_V)$. Let $N_c \subset N$ be the node set inserted to CPM.
***Precondition**:*
$\exists op_1, op_2 \in Op_V$:
$op_1 = RedActivity(CPM, pred(CPM, N_c)) \wedge$
$op_2 = RedActivity(CPM, succ(CPM, N_c))$

***Postcondition**:*
RedComplMode=SHOW: no action required
RedComplMode=RED: $Op_V := Op_V \oplus Op_N,\ Op_N := \langle op_{n_1}, \ldots, op_{n_k} \rangle$ *with*
$op_{n_i} := RedActivity(CPM, n_i),\ n_i \in N_c,\ \forall i = 1, \ldots, k$

Migration rule MR4 is applied when reducing either the predecessor *or* successor of node set N_c. Then, parameter *RedPartlyMode* determines whether N_c is visible (i.e., *SHOW*) or reduced (i.e., *RED*) in the respective process view.

Figure 7.23 illustrates the application of MR4. A user inserts activity X between B and C in process view $V1$. In turn, process views $V2$ and $V3$ reduce activity C, i.e., the successor of X (i.e., C) is reduced, but not its predecessor.

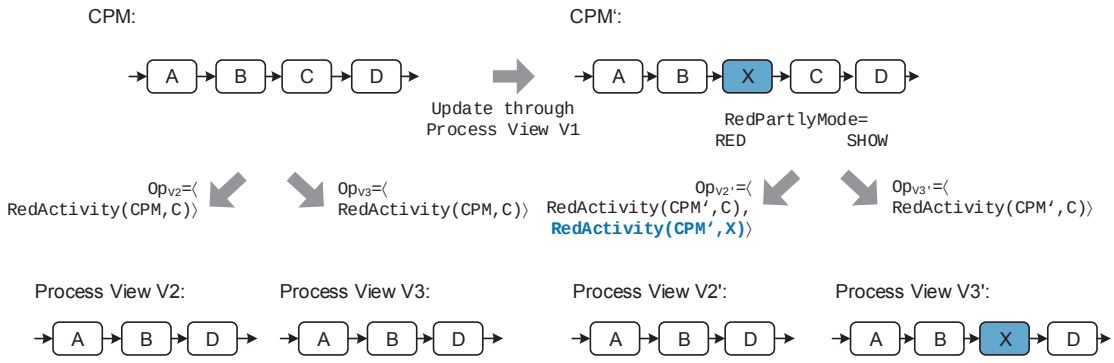


Figure 7.23: Migration Rule MR4

A reduction operation (i.e., to reduce X) is added to operation sequence Op_{V2} (since parameter *RedPartlyMode* is set to *RED*). When parameter *RedPartlyMode* is set to *SHOW*, no adaptation of Op_{V3} is required. Accordingly, X is visible to users interacting with $V3$.

Migration Rule MR4 (Insert Affecting Reduced Process Fragment)
Let $CPM = (N, D, NT, CE, EC, ET, DE, DET)$ be a process model and V be a process view described by $CS_V = (CPM, Op_V, PS_V)$. Further, let $N_c \subset N$ be the node set inserted to CPM.
***Precondition**:*
$\exists op_1, op_2 \in Op_V$:
$op_1 = RedActivity(CPM, pred(CPM, N_c))$ $\dot{\vee}$
$op_2 = RedActivity(CPM, succ(CPM, N_c))$

***Postcondition**:*
RedPartlyMode=SHOW: no action required
RedPartlyMode=RED: $Op_V := Op_V \oplus Op_N,\ Op_N := \langle op_{n_1}, \ldots, op_{n_k} \rangle$ *with*
$op_{n_i} := RedActivity(CPM, n_i),\ n_i \in N_c,\ \forall i = 1, \ldots, k$

7.4.2 Migration after Deleting Process Nodes

CPM updates deleting process nodes have to migrated as well. Otherwise, view creation operations may be applied on process nodes not existing in the CPM anymore. As example consider Figure 7.24. Amongst others, $V2$ reduces C and $V3$ aggregates B, C, and D. However, activity C is deleted in CPM. Accordingly, operation sequences of both process views need to be adapted.

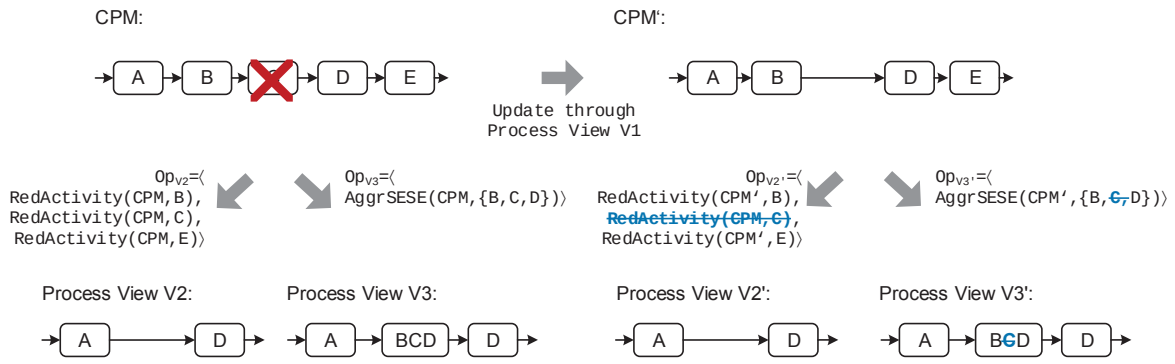


Figure 7.24: Migration Rule MR5

Migration rule MR5 may be applied when a process node n is deleted in CPM. If n has been reduced in a process view V, the respective reduction operation needs to be removed from the corresponding operation sequence Op_V in the creation set (cf. $V2$ in Figure 7.24). If n is affected by an aggregation operation, this operation has to be adapted by removing n from the node set N_a to be aggregated (cf. $V3$ in Figure 7.24).

Migration Rule MR5 (Deleted Process Node)
Let $CPM = (N, D, NT, CE, EC, ET, DE, DET)$ be a process model and V be a process view with $CS_V = (CPM, Op_V, PS_V)$. Further, let $n \in N$ be the node deleted in CPM.
***Precondition**:*
$\exists op_1 \in Op_V$:
$(op_1 = RedActivity(CPM, n) \vee op_1 = AggrSESE(CPM, N_a \cup \{n\}) \vee$
$op_1 = AggrSESE(CPM, N_a \cup \{n\})) \ \wedge \ N_a \subset N$

***Postcondition**:*
op_1 is reduction operation: $Op_V := Op_V \ominus \langle op_1 \rangle$
op_1 is aggregation operation:
$Op_V := Op_V \ominus \langle op_1 \rangle \oplus \langle AggrSESE(CPM, N_a) \rangle$ *(analogous for AggrComplBranches)*

When applying view update operation *DeleteBranching* to a process view V, operations aggregating multiple branches with *AggrComplBranches* have to be migrated. Therefore, MR6 may be applied. Since the *DeleteBranching* operation removes split gateway g_s and join gateway g_j from a branching and either deletes or inlines the remaining activities of its branches (cf. Section 6.3.2), *AggrComplBranches* operations aggregating this branching have to removed in operations sequence Op_V of respective process views. In case of inlining remaining activities, *AggrComplBranches* operation may be replaced by an *AggrSESE* operation aggregating the same node set. However, this causes problems when the node set of *AggrComplBranches* is not a SESE block after the inlining. Since this can not be guaranteed, *AggrComplBranches* is removed.

Migration Rule MR6 (Deleted Branching)
Let $CPM = (N, D, NT, CE, EC, ET, DE, DET)$ be a process model and V be a process view with $CS_V = (CPM, Op_V, PS_V)$. Further, let $g_s, g_j \in N$ be the split and join gateways of a branching deleted in CPM.
***Precondition**:*
$\exists op_1 \in Op_V$:
$op_1 = AggrComplBranches(CPM, N_c) \wedge$
$N_c \subseteq N \ \wedge \ (g_s, g_j) = MinimalSESE(CPM, N_c)$

***Postcondition**:*
$Op_V := Op_V \ominus \langle op_1 \rangle$

7.4.3 Migration after Updating the Data Flow

Migrating process views after data flow updates must be considered as well. Inserting data elements in the CPM, however, does not require any migration. As opposed to the insertion of process nodes, no ordering relation between data elements exists. Hence, already aggregated or reduced data elements are not affected by a newly inserted data element.
When deleting a data element, migration is required to remove view creation operations that reduce or aggregate the data element from the operation sequence. Migration rule MR7 removes respective operations from operation sequences to ensure that view creation operations do not contradict with the corresponding CPM.

Migration Rule MR7 (Deleted Data Element)
Let $CPM = (N, D, NT, CE, EC, ET, DE, DET)$ be a process model and V be a process view with $CS_V = (CPM, Op_V, PS_V)$. Further, let $d \in D$ be the data element deleted in CPM.
***Precondition**:*
$\exists op_1 \in Op_V$:
$(op_1 = RedDataElement(CPM, d) \ \vee \ op_1 = AggrDataElement(CPM, D_a \cup \{d\})) \wedge$
$D_a \subseteq D$

***Postcondition**:*
op_1 is of type $RedDataElement$: $Op_V := Op_V \ominus \{op_1\}$
op_1 is of type $AggrDataElement$:
$Op_V := Op_V \ominus \langle op_1 \rangle \oplus \langle AggrDataElement(CPM, D_a) \rangle$

Finally, when inserting or deleting data edges in the CPM, the operation sequences of associated process views need to be migrated. In particular, no view creation operations exist to aggregate or reduce data edges (cf. Section 6.3). Note that data edges are implicitly aggregated or reduced when aggregating or reducing process nodes or data elements. Hence, there is no need to migrate process views when inserting or deleting data edges.

7.4.4 Migration after Updating Process Attributes

Analogous to the data flow, process attributes have no ordering relation among each other. Hence, when inserting process attributes to the CPM no migration of associated process views is required. Furthermore, updates of process attribute values require to update them in all associated process views as well. This can be accomplished by re-creating the process views not requiring explicit migration rules.

However, deleting process attributes in the CPM requires the migration of associated process views and their operation sequences. Migration rule MR8 removes aggregation and reduction operations for process attributes in operation sequences after deleting attribute *A.del* in the CPM. The behaviour is similar to MR5.

Migration Rule MR8 (Deleted Attribute)
Let $CPM = (N, D, NT, CE, EC, ET, DE, DET)$ be a process model and V be a process view with $CS_V = (CPM, Op_V, PS_V)$. Further, let A.del be a process attribute of process node $A \in N$, which is deleted in CPM.
***Precondition**:*
$\exists op_1 \in Op_V$:
$op_1 = ReduceAttr(CPM, A.del) \vee op_1 = AggrAttr(CPM, AS \cup \{A.del\}, func)$

***Postcondition**:*
op_1 is of type $ReduceAttr$: $Op_V := Op_V \ominus \langle op_1 \rangle$
op_1 is of type $AggrAttr$: $Op_V := Op_V \ominus \langle op_1 \rangle \oplus \langle AggrAttr(CPM, AS, func) \rangle$

7.4.5 Summary

Migration rules MR1-MR8 ensure that the creation set or—to be more precise—the operation sequence of a process view will be migrated to the new CPM version after updating the CPM. In particular, this becomes necessary to guarantee that the CPM does not contradict with operation sequences, e.g., when reducing a process node not existing in the CPM anymore.

Obviously, not all migration rules have to be applied each time a view update operation is performed. For example, when deleting a data element in the CPM, MR8 needs not to be applied. When deleting a data element, in turn, MR7 is only required if *RedDataElement* or *AggrDataElement* is present as creation operations in the operation sequence of a process view.

Table 7.6 describes the relation between specific view update operations and migration rules. Further, it shows which migration rule has to be applied after performing a certain view update operation. For example, when deleting an activity with view update operation *DeleteActivity*, in the corresponding operation sequence, the view creation operations *RedActivity*, *AggrSESE*, and *AggrComplBranches* may be affected. To be more precise, the deleted activity might have been aggregated or reduced in the context of a process view creation. Hence, MR5 must be applied to prevent that an operation sequence of a process view contradicts the CPM.

Applying View Update Operation	**Affects View Creation Operation**						
	RedActivity	AggrSESE	AggrComplBranches	RedDataElement	AggrDataElements	ReduceAttr	AggrAttr
InsertSerial	MR3,MR4	MR1,MR2	MR1,MR2	✓	✓	✓	✓
InsertParallel	MR3,MR4	MR1,MR2	MR1,MR2	✓	✓	✓	✓
InsertCond	MR3,MR4	MR1,MR2	MR1,MR2	✓	✓	✓	✓
InsertLoop	MR3,MR4	MR1,MR2	MR1,MR2	✓	✓	✓	✓
InsertBranch	✓	✓	✓	✓	✓	✓	✓
InsertSyncEdge	✓	✓	✓	✓	✓	✓	✓
DeleteActivity	MR5	MR5	MR5	✓	✓	✓	✓
DeleteBranch	✓	✓	✓	✓	✓	✓	✓
DeleteSyncEdge	✓	✓	✓	✓	✓	✓	✓
DeleteBranching	✓	✓	MR6	✓	✓	✓	✓
InsertDataElement	✓	✓	✓	✓	✓	✓	✓
InsertDataEdge	✓	✓	✓	✓	-	✓	✓
ChangeDEType	✓	✓	✓	✓	✓	✓	✓
DeleteDateElement	✓	✓	✓	MR7	MR7	✓	✓
DeleteDataEdge	✓	✓	✓	✓	✓	✓	✓
InsertAttribute	✓	✓	✓	✓	✓	✓	✓
ChangeAttribute	✓	✓	✓	✓	✓	✓	✓
DeleteAttribute	✓	✓	✓	✓	✓	MR8	MR8

✓=not affected

Table 7.6: Relationship Between Migration Rules and View Creation Operations

After migrating all process views associated with the CPM, the process views are re-created. Applying view update operations to the CPM and re-creating the process views afterwards allows

us to guarantee that all process views are up-to-date. Figure 7.25 shows an example of the interaction of view creation and view update operations, including refactoring and migration rules. First, process views $V1, V2$, and $V3$ are created based on operation sequences Op_{V1}, Op_{V2}, and Op_{V3}. Afterwards, refactoring rules are applied to simplify the resulting process views through purging of unnecessary control flow structures.

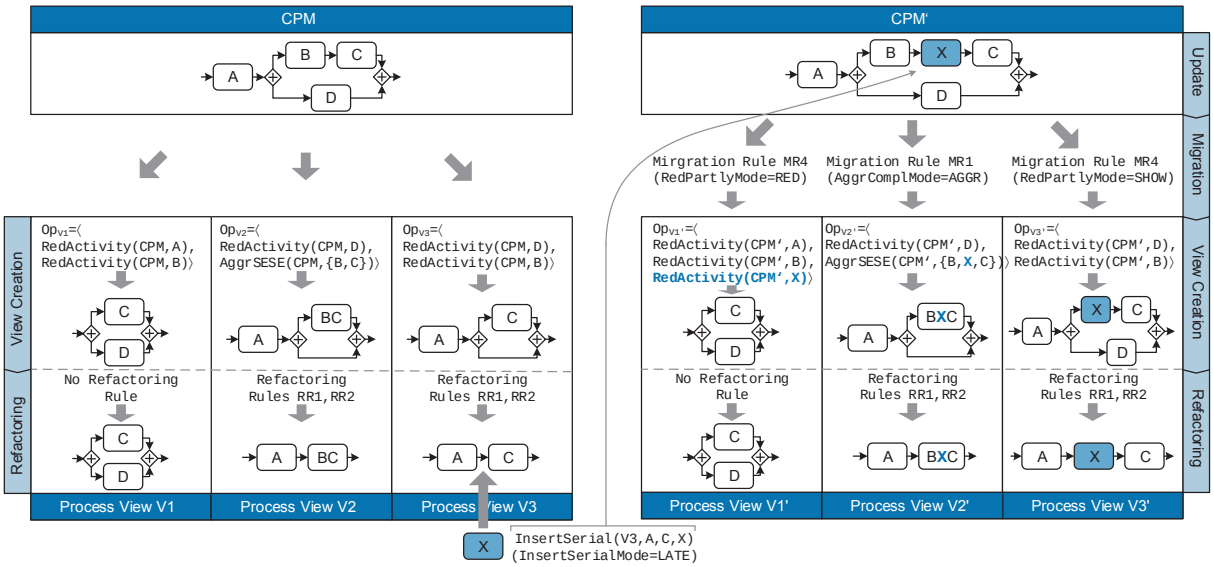


Figure 7.25: Interaction of View Creation and View Update Operations

Then, a user triggers an update by inserting activity X in $V3$, i.e., he applies view update operation $InsertSerial(V3, A, C, X)$. Parameter set PS_{V3} includes parameter $InsertSerialMode$ set to $LATE$, i.e., if ambiguities occur when propagating operation *InsertSerial*, the respective activity will be inserted at the latest possible position in the CPM. Hence, X is inserted between B and C in CPM. Afterwards, operation sets Op_{V1}, Op_{V2}, and Op_{V3} need to be migrated. In particular, MR1 and MR4 has to be applied. Subsequently, process views are re-created based on migrated operation sequences $Op_{V1'}, Op_{V2'}$, and $Op_{V3'}$. Finally, the application of refactoring rules ensures compact process views.

Generally, re-creating of all process views after a CPM update is expensive, especially when considering complex process models with a large number of associated process views. Therefore, a number of optimization techniques exists. First, only those process views are re-created, which are directly affected by an update. For example, $V1$ in Figure 7.25 will not change after inserting activity X in the CPM. Second, when migrating the operation sequence of a process view, the visualization component knows which parts of the process view has changed. Accordingly, only those parts are re-created in the process view.

7.5 Optimizing Process View Definitions

This section discusses maintenance issues. In particular, the creation set or—to be more precise—the operation sequence of a process view needs to be maintained over time. Generally, multiple elementary view creation operations are applied to a CPM when constructing a process view. Respective operations are stored in operation sequence Op_V of creation set $CS_V = (CPM, Op_V, PS_V)$. Figure 7.26 shows the application of $Op_V = \langle op_1, op_2 \rangle$ to a CPM.

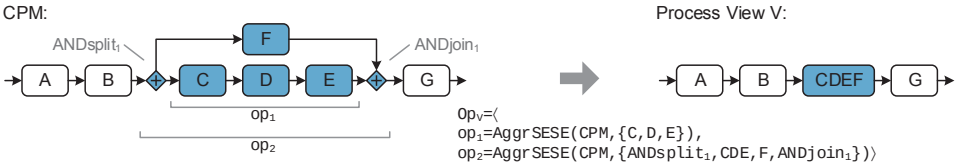


Figure 7.26: Applying an Operations Sequence to a CPM

For example, view creation operation op_2 is defined based on the application of operation op_1 on process view V, i.e., C, D, and E aggregated by op_1 are further aggregated by op_2, including nodes $ANDsplit_1, F$, and $ANDjoin_1$. This may result when a user first applies op_1 and then op_2. Therefore, the specific order in which elementary view creation operations are applied is important to ensure for a proper creation of the respective process view.

Operation sequences containing view creation operations, which are applied on each other, have drawbacks. First, creating the structure of a process view is inefficient. For example, in Figure 7.26, activities C, D, and E are first aggregated and, subsequently, nodes $ANDsplit_1, CDE, F$, and $ANDjoin_1$ are aggregated. Obviously, it would more efficient to aggregate all nodes at once. As another drawback, applying a view update operation, which require to update an operation sequence Op_V (e.g., *DeleteActivity*), may affect multiple operations in Op_V. Therefore, migration rules have to be applied to multiple view creation operations in an operation sequence.

Addressing these drawbacks require to optimize operation sequences, i.e., process view definitions are optimized. Therefore, in the following a set of *optimization rules* are introduced, which eliminate the dependencies between view creation operations in an operations sequence.

Optimization Rule OR1 (Aggregation on Aggregation). Two aggregation operations op_1 and op_2 may be applied on top of each other. In this context, optimization rule OR1 may be used if the node set to which op_2 is applied contains an aggregated node produced by op_1. As example consider Figure 7.27. Activities B and C are aggregated through view creation operation op_1. Subsequently, BC and D is aggregated by operation op_2. When applying rule OR1, operations op_1 and op_2 are removed and a new operation aggregating activities B, C, and D is added. From a user perspective, the structure of the process view does not change.

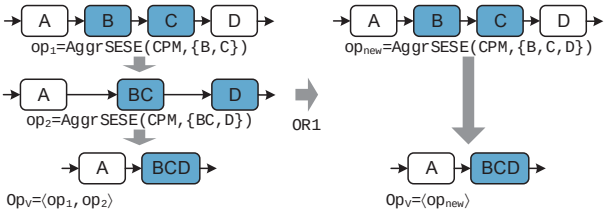


Figure 7.27: Optimization Rule OR1: Aggregation on Aggregation

Optimization Rule OR1 (Aggregation on Aggregation)
Let CPM be a process model. Further, let $Op_V = \langle op_1, op_2 \rangle$ be an operation sequence to create process view V on CPM, i.e., $CPM = P_0 \xrightarrow{op_1} P_1 \xrightarrow{op_2} P_2 = V$ with $P_i = (N_i, D_i, NT_i, CE_i, EC_i, ET_i, DE_i, DET_i)$.
***Precondition**:*
$op_1 = AggrSESE(P_0, N') \wedge op_2 = AggrSESE(P_1, N'') \wedge$
$N' \cap CPMNode(P_1, N'') \neq \emptyset \ \wedge \ N' \subseteq N_0 \ \wedge \ N'' \subseteq N_1$

***Postcondition**:*
$Op_V := Op_V \ \ominus \ \langle op_1, op_2 \rangle \ \oplus \ \langle op_{new} \rangle$
$op_{new} := AggrSESE(P_0, N' \cup CPMNode(P_1, N''))$

In case of view creation operation *AggrComplBranches*, the definition of the respective optimization rule is analogous to OR1.

Optimization Rule OR2 (Aggregation on Reduction). If a process node is reduced and, subsequently, surrounding nodes are aggregated during a process view creation, the corresponding view creation operations may be optimized as well. As example consider Figure 7.28. Activity C is reduced by view creation operation op_1. Afterwards, operation op_2 aggregates B and D. This can be optimized by aggregating all activities at once. Again, the structure of the process views does not change when applying the optimization rule Optimization Rule OR2. As a result, in Figure 7.28 operations op_1 and op_2 are merged into a single aggregation operation (including the reduced activity C).

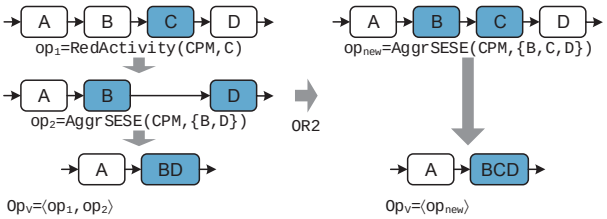


Figure 7.28: Optimization Rule OR2: Aggregation on Reduction

Optimization Rule OR2 (Aggregation on Reduction)
Let CPM be a process model. Further, let $Op_V = \langle op_1, op_2 \rangle$ be an operation sequence to create process view V on CPM, i.e., $CPM = P_0 \xrightarrow{op_1} P_1 \xrightarrow{op_2} P_2 = V$ with $P_i = (N_i, D_i, NT_i, CE_i, EC_i, ET_i, DE_i, DET_i)$.
***Precondition**:*
$op_1 = RedActivity(P_0, n), n \in N_0 \wedge$
$(op_2 = AggrSESE(P_1, N') \vee op_2 = AggrComplBranches(P_1, N')) \wedge$
$pred(P_0, n) \in N' \wedge succ(P_0, n) \in N' \wedge N' \subset N_1$

***Postcondition**:*
$op_{new} := AggrSESE(P_0, N' \cup \{n\})$ *// analogous for AggrComplBranches*
$Op_V := Op_V \ominus \langle op_1, op_2 \rangle \oplus \langle op_{new} \rangle$

Optimization Rule OR3 (Reduction on Aggregation). As opposed to OR2, optimization rule OR3 will be applied, if a node set is first aggregated and the aggregated node is reduced afterwards. In particular, the aggregation operation is not required anymore. Figure 7.29 shows an example aggregating B and C by operation op_1. Subsequently, op_2 reduces abstract activity BC.

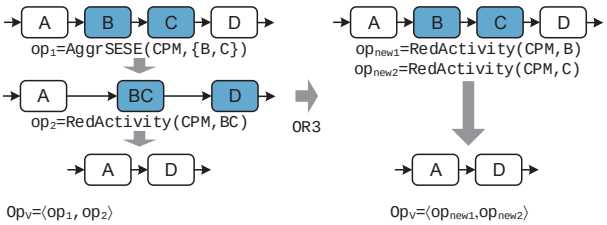


Figure 7.29: Optimization Rule OR3: Reduction on Aggregation

Activities B and C are aggregated by operation op_1, and the aggregated node BC is reduced by op_2. Optimization rule OR3 removes the aggregation operation and adds a respective reduction operation for each aggregated node.

Optimization Rule OR3 (Reduction on Aggregation)
Let CPM be a process model. Further, let $Op_V = \langle op_1, op_2 \rangle$ be an operation sequence to create process view V on CPM, i.e., $CPM = P_0 \xrightarrow{op_1} P_1 \xrightarrow{op_2} P_2 = V$ with $P_i = (N_i, D_i, NT_i, CE_i, EC_i, ET_i, DE_i, DET_i)$.
***Precondition**:*
$(op_1 = AggrSESE(P_0, N') \vee AggrComplBranches(P_0, N')) \wedge$
$op_2 = RedActivity(P_1, n) \wedge N' = CPMNode(P_1, n) \wedge N' \subseteq N_0$

***Postcondition**:*
$Op_N := \langle op_{n_1}, \ldots, op_{n_k} \rangle$ *with* $op_{n_i} := RedActivity(P_0, n_i),\ n_i \in N', \forall i \in 1, \ldots, k$
$Op_V := Op_V \ominus \langle op_1, op_2 \rangle \oplus Op_N$

Optimization rules are required in respect to data flow and process attributes as well. Similar to OR1, optimization rule OR4 is applied in case two aggregation operations aggregate an overlapping set of CPM data elements (cf. Figure 7.30a). In turn, optimization rule OR5 is applied if an aggregated process attribute is reduced by another view creation operation (cf. Figure 7.30b). A formal definition of these optimization rules in provided in Appendix B.

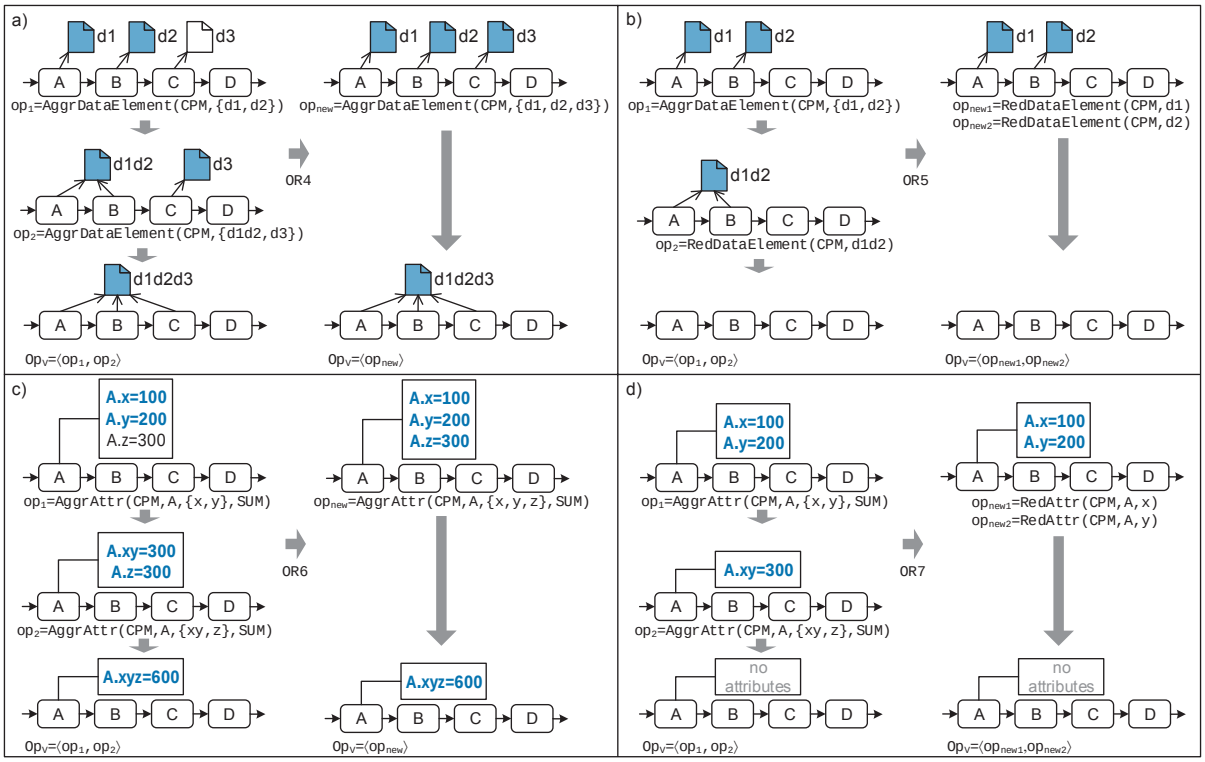


Figure 7.30: Optimization Rules OR4-OR7

Optimization rule OR4 is applied if a set of process attributes is aggregated by view creation operation *AggrAttr* and, subsequently, the result of the latter is further aggregated (cf. Figure 7.30c). Note that process attributes aggregated by these operations must be assigned to the same process node (e.g., A in Figure 7.30c). Finally, OR7 is applied if an aggregated process attribute, i.e., the result of aggregating a set of process attributes by operation *AggrAttr*, is reduced by applying view creation operation *RedAttr*. Figure 7.30c shows an example. A formal definition of these optimization rules are provided in Appendix B.

By applying optimization rules, the dependencies between view creation operations captured in the operation sequence Op_V of a creation set $CS_V = (CPM, Op_V, PS_V)$ are removed. Hence, the order in which view creation operations have to applied on CPM to create process model V is not required anymore, i.e., the operations are commutative. Therefore, view creation operations can be stored in a set $Op_V = \{op_1, \ldots, op_k\}$ as defined in Definition 7.4.

Definition 7.4 (Process View with Operation Set)
Let CPM be a process model. Then: A process view V is represented through a creation set $CS_V = (CPM, Op_V, PS_V)$ where:

- *$CPM = (N, D, NT, CE, EC, ET, DE, DET)$ is the process model based on which process view V is created,*
- *$Op_V = \{op_1, \ldots, op_k\}$ is a **set** of elementary view creation operations applied to CPM: $op_i \in \mathcal{OP}$. $\mathcal{OP}$ comprises all elementary view creation operations (cf. Section 6.3),*
- *$PS_V = (PS_1, \ldots, PS_m)$ is a tuple of parameters and corresponding parameter values defined for a specific view.*

Table 7.7 gives an overview of overlapping node (and attribute) sets that might occur due to the consecutive application of view creation operations defined in an operations sequence. Furthermore, it is shown which optimization rule removes the respective overlap. In some cases, an overlapping node (or attribute) set does not occur. For example, two *RedActivity* operations can not reduce the same process node in the same operations sequence. Furthermore, two view creation operations addressing, for example, the control and data flow do not have overlapping node sets as well.

	View Creation Operation op_2 overlaps...						
...operation op_1.	RedActivity	AggrSESE	AggrComplBranches	RedDataElement	AggrDataElements	ReduceAttr	AggrAttr
RedActivity	✓	OR2	OR2	✓	✓	✓	✓
AggrSESE	OR3	OR1	OR1	✓	✓	✓	✓
AggrComplBranches	OR3	OR1	OR1	✓	✓	✓	✓
RedDataElement	✓	✓	✓	✓	✓	✓	✓
AggrDataElements	✓	✓	✓	OR5	OR4	✓	✓
ReduceAttr	✓	✓	✓	✓	✓	✓	✓
AggrAttr	✓	✓	✓	✓	✓	OR7	OR6

✓=no overlap

Table 7.7: Optimization Rules Removing View Creation Operations Overlaps

7.6 Related Work

Work related to process view updates can be categorized into *process model update*, *process view updates*, and view updates in *databases*.

Process Model Updates. For defining and updating process models, various approaches and process modeling languages exist, e.g., EPC [160], UML activity diagrams [161], WS-BPEL [162], or workflow nets [163, 164]. Currently, BPMN constitutes the most established process

modeling language [165]. These process modeling languages have in common that the complexity of the created process models is high, which results in large efforts when updating the process models. In addition, large process models may result in erroneous process models [19].

As opposed to imperative process modeling languages, declarative process modeling allows defining the behaviour of a business process based on a set of constraints [166, 167, 168, 169, 170, 171]. In general, declarative process models can be updated by inserting or deleting constraints. However, the more constraints a declarative process model has, the harder it is to understand [170] and, hence, to maintain. A comparison between imperative and declarative process modeling is provided in [172]. Furthermore, in [173] a process modeling language is introduced, which focuses on the data flow, i.e., about the processing of data within a business process.

Workflow patterns comprise control and data flow patterns that need to be supported by process modeling languages [174, 175]. [176] introduces *Yet Another Workflow Language (YAWL)*, which realizes all workflow patterns. However, it is not discussed whether non-technical users are able to document business processes based on the patterns. In order to facilitate process modeling for users various patterns are suggested in [177, 178]. Furthermore, in [116, 179] modeling guidelines are introduced to ensure a high quality of resulting process models. In particular, such modeling guidelines assist users in understanding and updating process models. None of the approaches is able to reduce complexity of process models before performing updates on them.

Frequently used patterns for updating process models are presented in [158]. In [18, 180] a subset of these patterns are used for refactoring process models in repositories, while preserving their behaviour. Furthermore, [15] summarizes approaches enabling flexibility in PAISs. In particular, [103] presents an approach for adapting well-structured process models without affecting their correctness properties. Based on this, [39] presents concepts for optimizing process models over time and migrating running processes to new model versions properly. Finally, [181] investigates concurrent changes of running processes and their synchronization.

Recommendation systems and social networks based on process model repositories may be used to support users in updating process models [182, 183]. Furthermore, social networks are used to notify users about process model update. In order to enable non-technical users to change process models, an easy to understand process modeling language targeting at business users may be used [184]. However, these approaches do not support the abstraction of large process models for the various domain experts.

An approach for updating process attributes and, in particular, labels of activities is introduced by [185]. Furthermore, [122] presents an approach for automatically automatic labeling activities.

Process View Updates. Most process view approaches (cf. Section 6.6) do not support updatable process views. Few exceptions are provided by business IT alignment approaches: [100, 186] align technical workflows with business processes. This approach analyzes updates through behavioural profiles and propagates them to change regions of the corresponding technical model. These regions indicate the process model region the change belongs to. An automated propagation of updates is not supported. Similarly, [23] describes a mapping model between a technical workflow and a business process. This approach does not support an automated propagation of updates. [187, 140, 188] are able to synchronize technical workflow descriptions with business process models and to propagate relevant changes between them. However, all these approaches are limited to business IT alignment. In particular, they do not take multiple personalized process views into account.

The view approach suggested by [144] is extended to update process views as well [189]. The approach is limited to the control flow perspective. It does neither supports data flow nor process attributes nor the migration of associated process views.

In the following, we introduce an example, which intends to demonstrate the effort required to update process models contemporary. Figure 7.31 visualizes Example 6.1 in terms of S-BPM [190, 148]. Assume user role *manager* wants to update his process model by inserting activity *Load Contract Template* between activities *Review Account* and *Create Contract* (i.e., between XORsplit *Credit Decision?* and activity $a9$ in Figure 7.3). In Figure 7.31, this update affects the

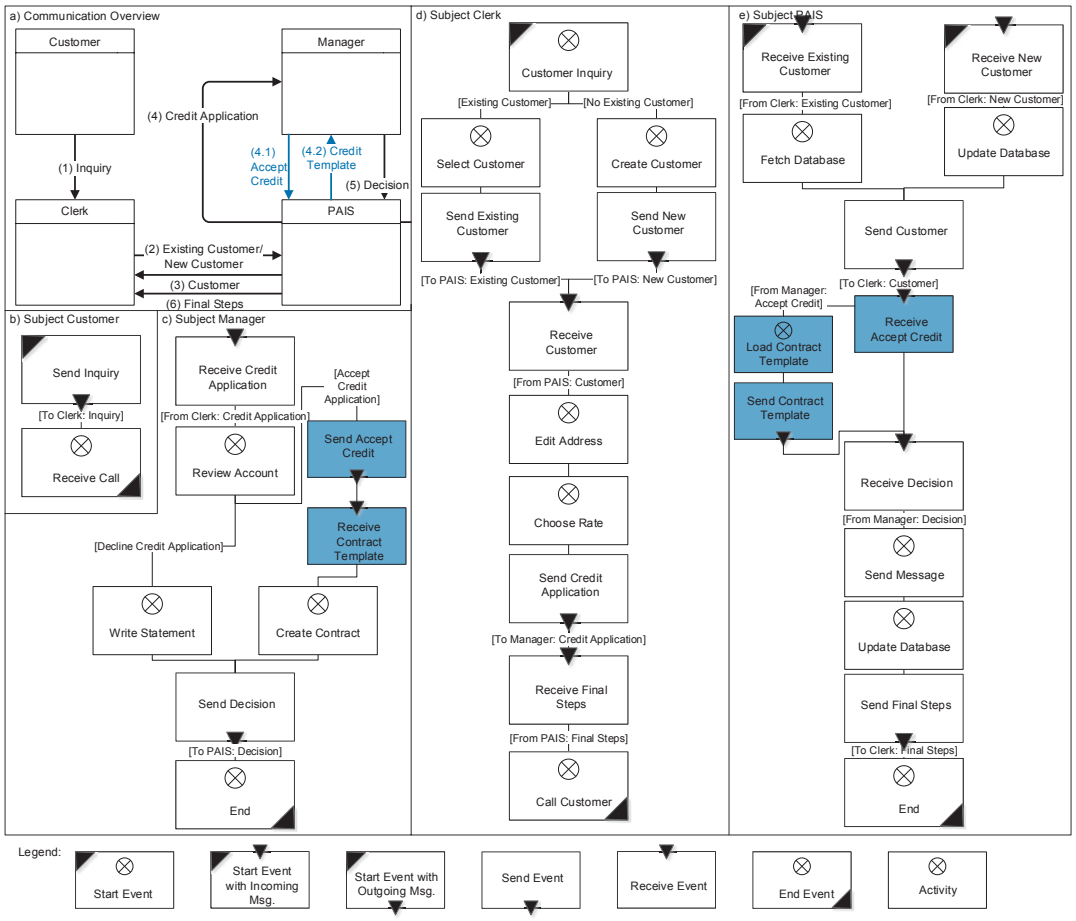


Figure 7.31: Updating S-BPM Process Models

process models of several subjects (i.e., users). In particular, a message has to be sent from the manager (i.e., Figure 7.31c) to the PAIS (i.e., Figure 7.31e) to trigger the latter to load the contract template. Then, the contract template has to be sent back to the manager. Furthermore, the superordinate communication diagram has to be adapted (i.e., Figure 7.31a). Coloured elements show updated process fragments. In particular, no automatic migration or propagation is supported by S-BPM and, hence, such updates require a high effort by non-technical users and might lead to wrong updates.

View Updates in Databases. Database Management Systems (DBMS) provide two kinds of views: *read-only* and *updatable* views [98, 191]. Read-only views must not be changed. In turn, updatable views are solely supported if the DBMS is able to determine an unambiguous mapping between the view on one hand and the corresponding database table on the other, i.e., in case of aggregated database columns an ambiguous mapping exists.

Other domains deal with similar issues. *Concurrent engineering*, for example, provides concepts for maintaining various artifacts describing the same product from a different point of view [192, 193]. In particular, it is stated that concurrent engineering increases the quality of the resulting product, while decreasing development time.

The *proView* framework enables users to change business processes based on their views and guarantees that other views of the CPM are adapted accordingly. None of the existing approaches covers all these aspects or is based on rigid constraints not considering practical requirements.

7.7 Limitations

This chapter introduces operations to update process models (i.e., CPMs) based on process views. Furthermore, rules are provided to migrate process views after updating the CPM. Finally, optimization rules are introduced to optimize process view definitions (i.e., creation sets). In the following, limitations of the approach are discussed.

View update operations update single aspects of a process model, e.g., solely inserting an activity. However, users may require a more convenient and advanced way to update process models and process views, respectively. For example, users may want to insert an entire process fragment to a process view or move a process fragment to another position within the process view. Such more advanced update operations are covered by the change patterns desired in [158].

To resolve ambiguities when propagating view updates to the underlying CPM, parameters for view update operations are introduced. For example, parameter *InsertSerialMode* defines whether an activity shall be inserted at the earliest or latest possible position, or in parallel to existing activities. Default values for parameter settings may be overwritten for each process view and stored in the respective creation sets. However, to provide a more intelligent propagation behaviour parameter settings must be defined each time a view update operation is applied. Thus, it can be decided individually how to resolve an ambiguity based on, for example, the definition of the respective process view.

Parameters can be used to define how process views shall be migrated to a new version of the CPM after an update. Again, these parameters must be specified individually for each process view. The migration must take view creation operations as well as the applied view update operation into account to provide a more intelligent migration behaviour. Consequently, parameter settings need to be adjusted individually when migrating process views.

These limitations are addressed in Chapter 8, which introduces advanced view update operations as well as a more advanced migration behaviour.

Allowing multiple users to update a CPM based on their process views, requires a concept for handling concurrent updates. Thereby, it must be guaranteed that an update is properly propagated to the CPM and, subsequently, all process views are migrated. Otherwise, when dealing with concurrent updates of different process views, these updates may influence each other. In general, updates on process models are rather rare. Hence, an optimistic technique may be applied to synchronize concurrent updates; as known from *optimistic concurrency control* in databases [194, 194, 195]. Furthermore, in case of a conflict it is "cheap" to abort an update instead of locking a process model for other users.

Another aspect is access control. To be more precise, a proper access control is required to prevent non-authorized users to read, create, or update process views. However, existing research on access control of PAISs does not take process views into account and must be extended to address questions that arise when dealing with process views [196, 197, 198, 199, 200]. However, initial research exist addressing access control in the context of process views [201, 202].

7.8 Summary

This chapter introduces operations for view-based updates of process models (i.e., CPMs), i.e., process abstractions not only serve visualization purposes, but also lift process updates up to a higher semantical level. A set of view update operations enables users to update their process view and to propagate the respective update to the related CPM representing a holistic view on the business process. For this purpose, view update operations are introduced. A flexible parameterization of these operations allows for the automated resolution of ambiguities when propagating view changes to the CPM; i.e., the update propagation behaviour may be defined for each process view. Finally, the view update operations not only address the control flow perspective, but data flow and process attributes as well.

Migration rules to update all process views associated with an updated CPM are presented. Similar to update propagation, for each process view, it can be decided how much information about the update shall be displayed to the user. Finally, optimization rules have been introduced to optimize the creation of process views by optimizing process view definitions.

8

Advanced View Operations and their Application

8.1 Introduction

Previous chapters introduced elementary operations for creating, updating, and migrating process views. In particular, each of these operations covers a single perspective like control flow, data flow, and process attributes (e.g., resources). However, creating and updating process views solely based on elementary operations causes high user efforts. Therefore, high-level view creation operations combine elementary ones to provide a more convenient way to create process views for users, e.g., "Show all my activities". However, there exists no abstract view update operations. In particular, respective operations should properly combine elementary operations to provide users a more comfortable way for updating their process models. For example, a view update operation may insert an entire process fragment comprising multiple activities into a process view. Furthermore, such *advanced view update operations* can be semantically enriched. For example, a user may want to insert an activity *directly after* another one. Semantically enriching view update operations may provide information that allows resolving ambiguities when propagating view updates to the CPM. Note that the automatic resolution of ambiguities based on elementary view update operations might result in non-intended updates of the CPM. In turn, knowing *what* the user intends to update enables a more appropriate propagation behaviour when facing ambiguities. Furthermore, when creating a process view based on high-level view creation operations, additional information (e.g., about the criteria applied for filtering out (i.e., reducing) activities) will be available to resolve ambiguities. Example 8.1 gives insights into how advanced view update operations foster process view updates.

Example 8.1 (Advanced View Updates)
Consider the CPM describing a credit approval process (cf. Example 6.1). Furthermore, assume that a related process view $V1$ is created with a high-level view creation operation, which only shows activities of user role Clerk, while aggregating the ones of user role Manager and reducing the ones of any other user role (cf. Figure 8.1).

Assume that activity Send Customer ID (i.e., activity $a15$) shall be added to $V1$ by inserting it directly after activity Select Customer (i.e., activity $a2$). As opposed to Example 7.1, propagating this update to the CPM does not result in any ambiguity. To be more precise, the information that $a15$ shall be inserted directly after $a2$ may be used to map the update to (elementary) view update operation $InsertSerial(V1, a2, XORjoin_1, a15)$ with $InsertSerialMode$ being set to $EARLY$.

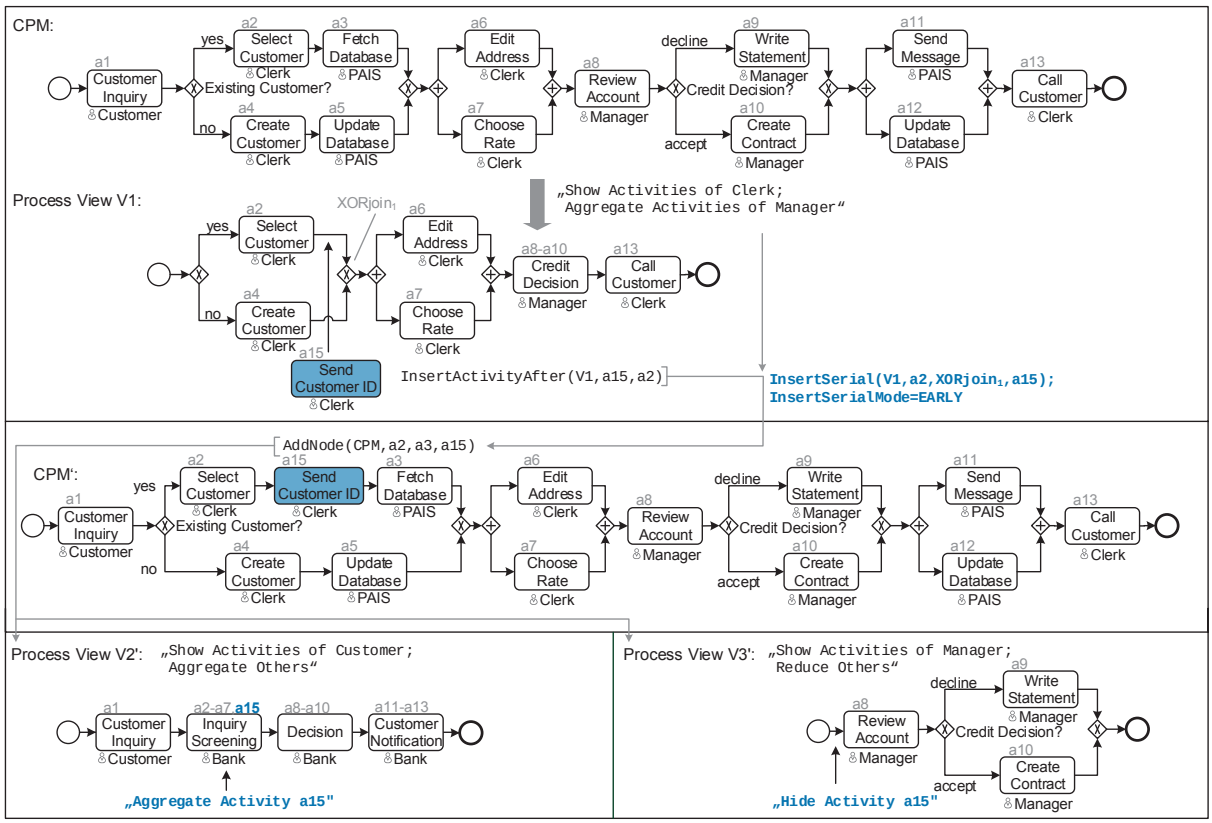


Figure 8.1: Advanced Process View Update

Knowing the intended use of a process view, provides important information to decide whether or not an update on a CPM is relevant for an associated process view. To be more precise, in case high-level view creation operations are applied to create a process view, migrating process views after an update of the CPM may be accomplished more appropriately. Example 8.2 gives insights of advanced process view migration.

Example 8.2 (Advanced View Migration)
Consider the view update described in Example 8.1. When propagating it to the CPM, associated process views $V2$ and $V3$ may have to be migrated as well.

$V2$ has been constructed based on a high-level view creation operation showing solely the activities of user role Customer, while aggregating all other activities. Consequently, the inserted activity $a15$ is not relevant for $V2$, but is aggregated as well (cf. $V2'$ in Figure 8.1).

$V3$ has been constructed based on a high-level view creation operation reducing all activities except the ones of user role Manager. This update is not relevant for $V3$. Accordingly, $a15$ is reduced (cf. $V3'$ in Figure 8.1).

Figure 8.2 illustrates the relation high-level view creation and advanced view update operations have with elementary operations. In particular, high-level view creation operations are applied on a CPM (cf. Figure 8.2a). These operations are then mapped to a sequence of elementary view creation operations for deriving a process view (cf. Figure 8.2b). If a user updates V_1 by applying an advanced view update operation (cf. Figure 8.2c), the latter is mapped to a sequence of elementary view update operations, which are then applied to the CPM (cf. Figure 8.2d). In case of ambiguities, mapping from advanced to elementary view update operations is influenced by high-level view creation operations used to create the respective process view.

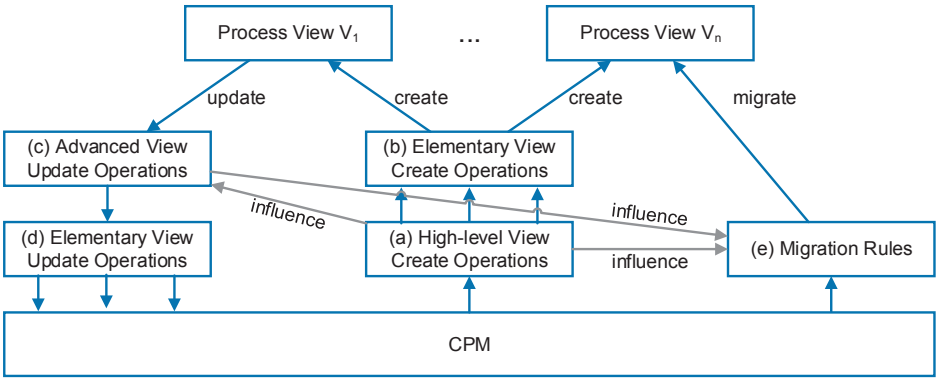


Figure 8.2: High-Level View Creation and Advanced Update and Migration

Following this, all process views are migrated from the old to the new CPM version (cf. Figure 8.2e). In this context, migration rules are affected by high-level view creation operations, i.e., respective parameter settings of migration rules are set accordingly.

Section 8.2 introduces fundamentals on high-level view creation operations. Section 8.3 provides advanced view update operations. Section 8.4 then introduces advanced view migrations. Section 8.5 discusses advanced view updates and Section 8.6 summarizes the chapter.

8.2 Fundamentals of High-Level View Creation Operations

In the following, high-level view creation operations and their mapping to elementary operations are presented. These operations are partially based on concepts presented in [32, 4].

Manually selecting the elementary view creation operations to be applied to a CPM may not be convenient for end-users. In particular, this would require in-depth knowledge of these operations as well as their semantics. Therefore, [32, 4] provides high-level view creation operations, which hide as much complexity from the user as possible. In particular, these operations provide a predicate-based way of selecting the process elements to be aggregated or reduced. For example, operation "Show only activities of user (role) X" eliminates all activities not assigned to user X. To provide such built-in intelligence, view creation operations are organized in four layers. Figure 8.3 gives an overview of the layers and their interactions. In order to define a process view on a CPM, operations from all layers may be applied in a combined way [32].

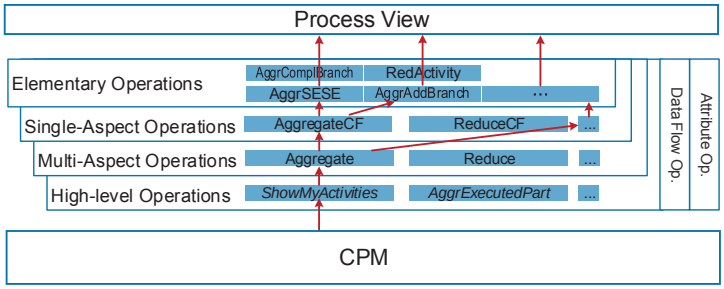


Figure 8.3: Multi-Layer View Creation Operations

Elementary view creation operations are tailored for a specific ordering of the selected node set in the CPM (cf. Chapter 6). *Single-aspect* view creation operations aggregate or reduce a set of unconnected process nodes of the same type (e.g., activities or data elements). These operations analyze the structure of CPM process nodes and select appropriate elementary operations based on a given parameterization. In turn, *multi-aspect* view creation operations consider multiple process perspectives (e.g., activities or data elements) and apply respective single-aspect operations for each of them. Finally, *high-level* operations abstract from aggregations or reductions necessary to build a particular process view. Instead they provide a more intelligent way of creating process views. Figure 8.4 shows an example illustrating the way high-level operation *ShowActivitiesOfUser(CPM, Manager)* is translated into elementary view creation operations that aggregate respective process fragments.

Mappings of single-aspect to elementary view creation operations are not always straightforward. For example, the node set $\{a1, a2, a3, a4, a5, a6, a7, a13\}$ (cf. Figure 8.4) is mapped to elementary view creation operations. The given node set cannot be aggregated at once since it neither constitutes a SESE block (i.e., AggrSESE can not be applied) nor does it form the branches of a branching block (i.e., AggrComplBranches can not be applied). Accordingly, it is required to either split up or extend the given node set in order to obtain SESE blocks. Regarding the example from Figure 8.4, the node set to be aggregated is split up into two groups, i.e., $\{a1, a2, a3, a4, a5, a6, a7\}$ and $\{a11, a12, a13\}$.

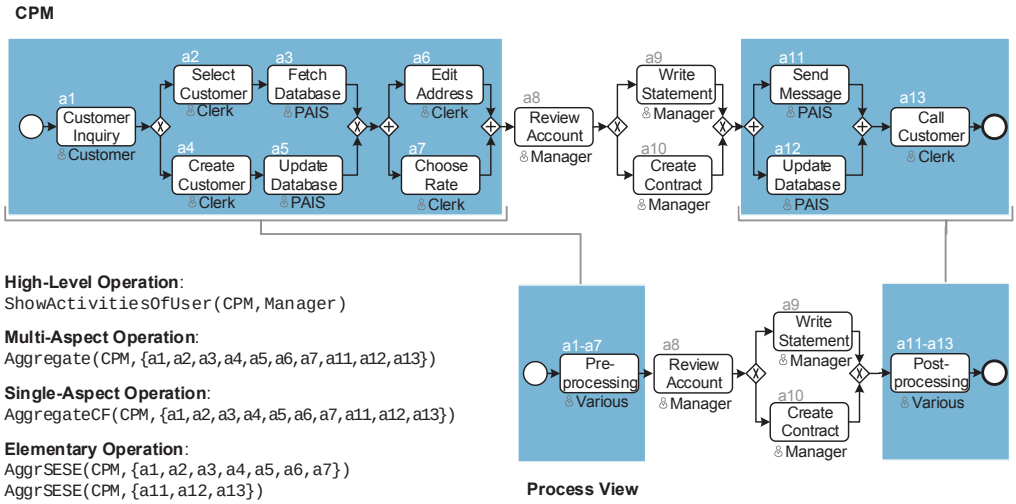


Figure 8.4: Example of Applying a High-Level View Creation Operation

Details on the mapping of high-level view creation operations to elementary ones can be found in [32]. In the following, high-level operations are introduced, which are used in the remaining chapter. Table 8.1 gives an overview of high-level view creation operations.

High-Level Operation	Description
$ViewByPredicate(CPM, p)$	The process nodes of CPM, which satisfy predicate p, are kept in the process view. All other nodes from CPM are either reduced or aggregated depending on parameter $NoShowMode \in \{AGGR, RED\}$.
$GroupByPredicate(CPM, p)$	The process nodes of CPM, which satisfy predicated p, are selected and either reduced or aggregated depending on parameter $NoShowMode \in \{AGGR, RED\}$.
$Subgraph(CPM, N_s)$	The process view correspond to the subgraph of CPM induced by node set N_s; all other activities are reduced.
$SubgraphRange(CPM, n_s, n_e)$	All activities not located between nodes n_s and n_e are reduced.
$ShowActivitiesOfUser(CPM, u)$	Activities of user (role) u are kept in the process view. All other activities are reduced.
$AggrExecutedPart(i)$	Activities of process instance i already executed or skipped are aggregated to an abstract activity.
$ShowExecutedPath(i)$	The activities executed by process instance i are kept in the process view. Skipped and not yet executed activities are reduced.
$ViewByRelevance(CPM, f_r)$	Activities of CPM are selected depending on their relevance described by relevance function f_r. Then those activities are either aggregated or reduced in the resulting process view depending on parameter $NoShowMode \in \{AGGR, RED\}$.

Table 8.1: Overview of High-Level View Creation Operations

A process view created with high-level view creation operation $ViewByPredicate(CPM, p)$ only depicts those activities of CPM that satisfy predicate p. For example, when applying operation $ViewByPredicate(CPM, User = Manager)$ to process model CPM in Figure 8.5, the resulting view only comprises activities whose assigned user role corresponds to $Manager$. Other activities not matching with predicate p are either reduced (i.e., parameter $NoShowMode$ is set to RED) or aggregated (i.e., parameter $NoShowMode$ is set to $AGGR$).

As opposed to operation $ViewByPredicate$, $GroupByPredicate(CPM, p)$ selects those activities of process model CPM, which fulfill predicate p and either aggregates or reduces them in the resulting process view. Figure 8.5 depicts the process view resulting from the application of operation $GroupByPredicate(CPM, p)$ with predicate $p = [User = Manager]$. Similarly, parameter $NoShowMode \in \{AGGR, RED\}$ specifies whether or not selected activities shall be aggregated or reduced in the resulting process view.

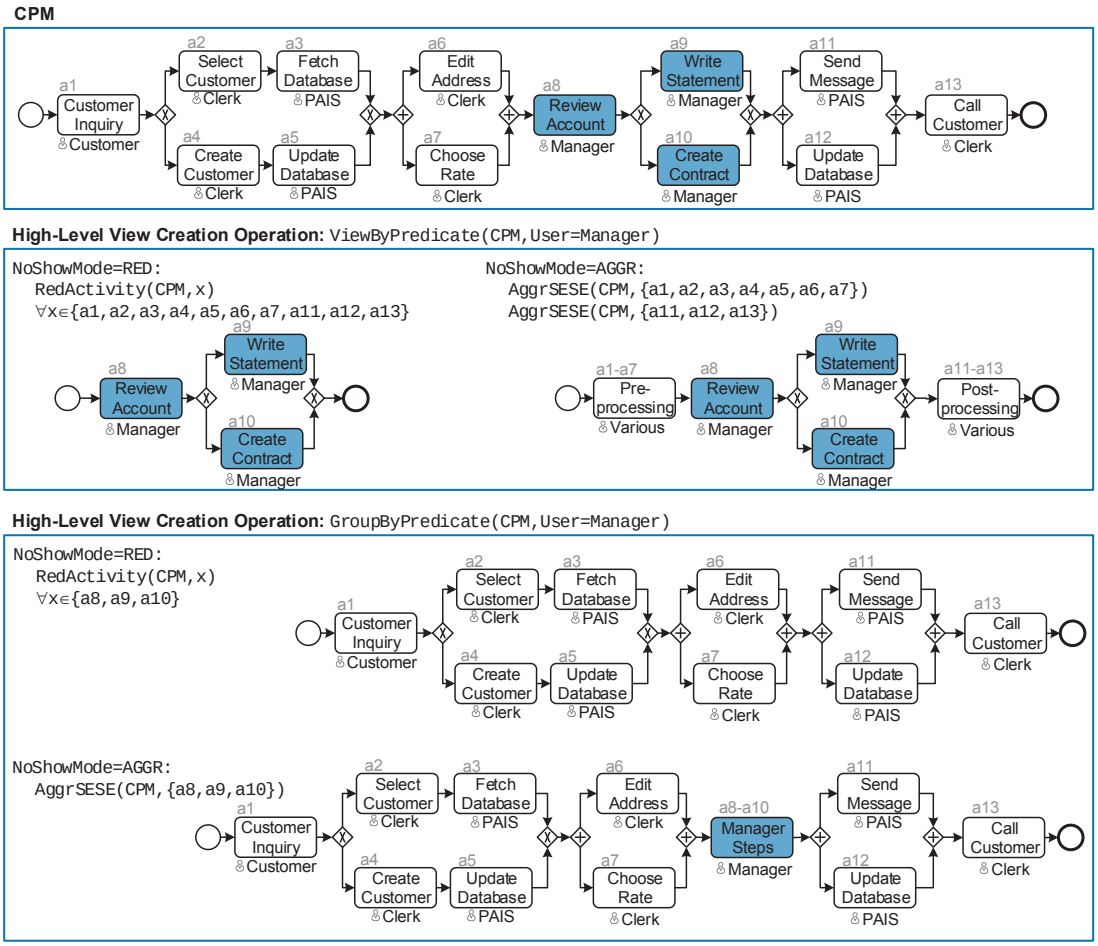


Figure 8.5: View Creation Operations ViewByPredicate and GroupByPredicate

High-level view creation operation $Subgraph(CPM, N_s)$ shows the subgraph of CPM induced by node set N_s (cf. Table 8.1). Applying this operation, the user is able to select those activities relevant to him. Similarly, $SubgraphRange(CPM, n_s, n_e)$ shows the subgraph between first node n_s and last node n_e. If nodes n_s and n_e are not the first and last node of a SESE block, a minimal SESE block is determined (cf. Definition 6.2).
Based on these high-level view creation operations users may create process views in a more convenient way compared to elementary view creation operations.

8.3 Advanced View Updates

Applying elementary view update operations, which only modify single aspects of a process view and its corresponding CPM, may be too fine-grained requiring in-depth knowledge by users about elementary view update operations. *Advanced view update operations* shall enable a more abstract way to update process views. These operations either combine a set of elementary view update operations (e.g., to insert a process fragment) or provide a more intelligent way to update a process view (e.g., insert a node directly after another one).

Advanced view update operations address two limitations of elementary ones. First, user may require a more convenient way to update process views. To achieve this, an advanced view update operation should combine elementary ones. Figure 8.6 provides an example: activity *Call Customer* shall be moved to another position in process view V by applying advanced view update operation $MoveActivity$. The latter deletes the respective activity (i.e., $DeleteActivity$) and re-inserts it at the desired new position (i.e., $InsertActivity$).

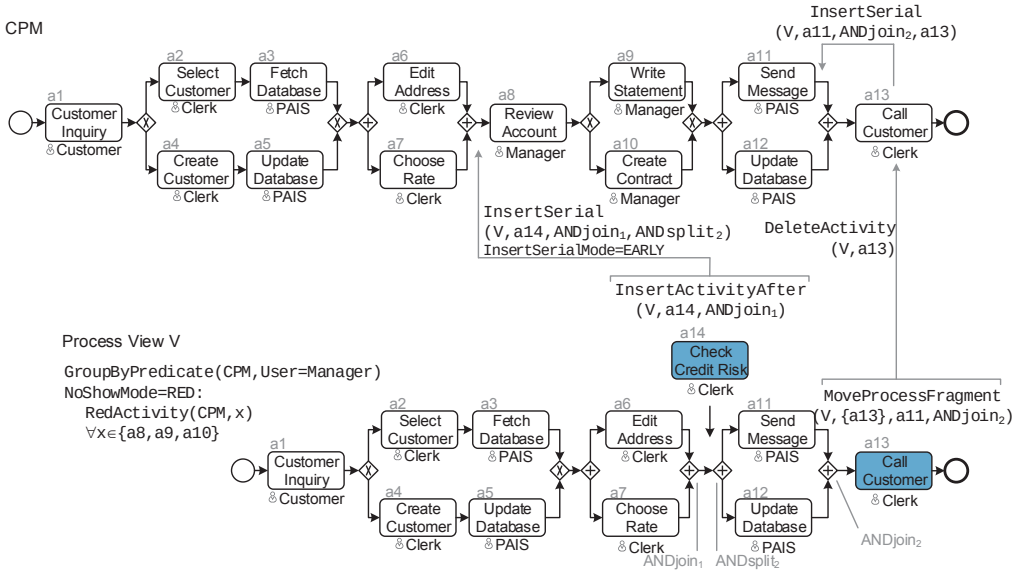


Figure 8.6: Advanced View Update Operations

Another limitation of elementary view update operations concerns ambiguities when propagating view updates to the CPM. Parameters are used to resolve such ambiguities and to enable automatic propagation of updates to the CPM. However, semantically enriched view update operations may be used to resolve such ambiguities.

In certain situations, a user may want to resolve such ambiguities himself. In Figure 8.7, for example, a user wants to insert activity *Check Credit Risk* (i.e., activity $a14$) in process view $V1$. Since the activity shall be inserted in a process fragment, which is reduced, an ambiguity occurs. To resolve the latter, the reduced process fragment is shown to the user who then decides on the desired insert position.

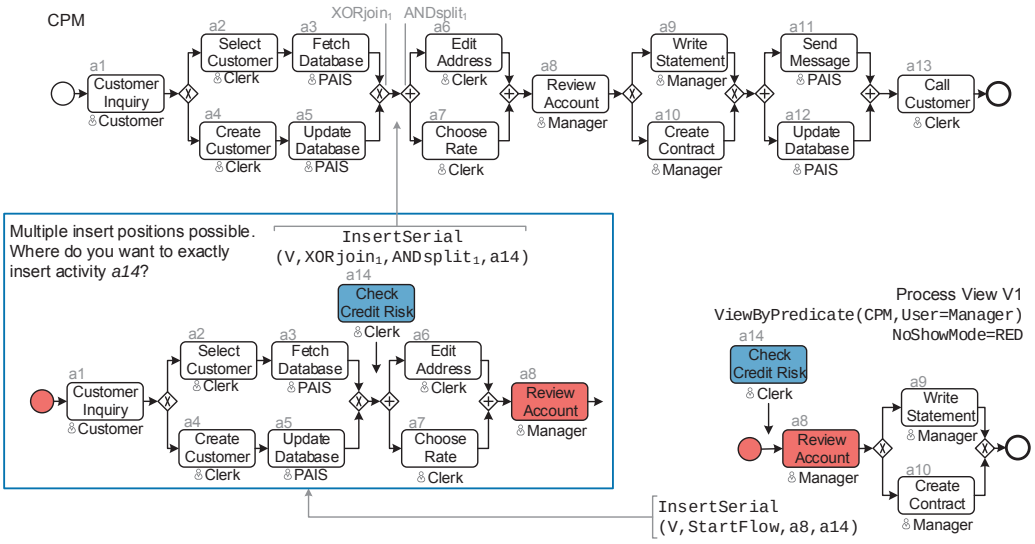


Figure 8.7: Semi-Automatic Propagation of View Changes

Obviously, such a *semi-automatic propagation* requires a minimum of process modeling knowledge to be able to select the desired insert position. Depending of the process modeling knowledge of the user, it might be more adequate to hand the decision over to a process owner or a dedicated process designer. However, to resolve ambiguities automatically, advanced view update operations may provide helpful information to resolve such ambiguities. In Figure 8.6, for example, the ambiguity that occurs when inserting activity *Check Credit Risk* can be resolved by applying operation $InsertActivityAfter$. Note taht high-level view creation operations provide additional information for resolving ambiguities. For example, if a process view only shows the activities of a particular user, inserting an activity in a process fragment assigned to other users will not be adequate.

In the following, we discuss how high-level view creation operations contribute to improve the propagation of updates on a process view to the corresponding CPM (cf. Section 8.3.1). Section 8.3.2 provides a set of advanced view update operations. Finally, Section 8.3.3 evaluates the completeness of view update operations.

8.3.1 Influence of High-Level Operations on Update Propagation

The propagation of a view update to the CPM may result in ambiguities regarding the exact update position (i.e., the position in the CPM to which the update is applied to). To be able to automatically resolve ambiguities for elementary view update operations, parameters (e.g., *InsertSerialMode*) are used (cf. Section 7.4). As a drawback, however, these parameters only consider structural aspects of an update (e.g., to insert an activity at the earliest or latest possible position), but does not take any other information into account to resolve ambiguities. An automated resolution of ambiguities based on the parameter value set for elementary operations may lead to an undesired update position in the CPM.

In order to provide a more *precise* way for propagating view updates to the CPM, the *meaning* of an update and its context need to be taken into account as well. Thereby, *precise* refers to how close the actual update position in the CPM to the intended position by the user is.

A more precise propagation behaviour may be achieved by utilizing high-level view creation operations, i.e., utilizing the information on how a process view is constructed. In addition, in certain situations, high-level view creation operations need to be adapted after performing an update, e.g., by reducing an inserted activity being not relevant for the respective process view. Figure 8.8 shows an example of how high-level view creation operations allow resolving ambiguities.

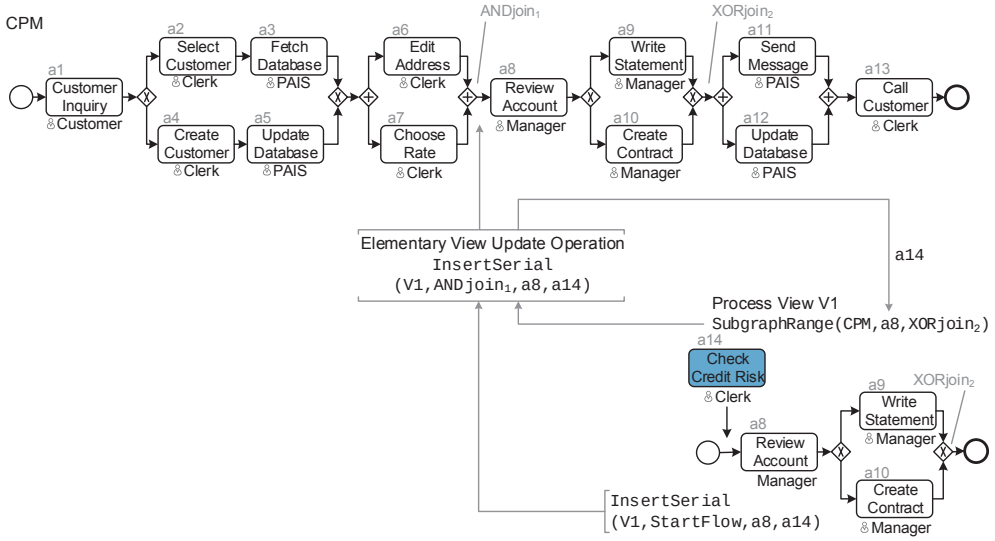


Figure 8.8: Update Propagation based on High-Level View Creation Operations

In process view $V1$, activity *Check Credit Risk* is inserted between $StartFlow$ node and activity *Review Account*. Propagating this update to the CPM results in an ambiguity regarding the insert position (i.e., somewhere between the two nodes). However, since $V1$ is created by applying operation $SubgraphRange$ (cf. Section 8.2), it is more likely that inserted activity $a14$ shall be positioned at the beginning of the selected range (i.e., between nodes $a8$ and $XORjoin_2$). Hence, $a14$ is inserted directly before $a8$. Furthermore, $a8$ will not be shown in the process view, since high-level view creation operation $SubgraphRange(CPM, a8, XORjoin_2)$ reduces it. Hence, the view creation operation $SubgraphRange$ must be adapted in order to show the newly inserted activity (i.e., $SubgraphRange(CPM, a14, XORjoin_2)$).

8.3.2 Advanced View Update Operations

Advanced view update operations combine a set of elementary view update operations to provide a more convenient way for updating a process view and CPM respectively. Furthermore, advanced view update operations may be categorized into two categories: Category *AU1 (CPM*

Update) groups operations only affecting the structure of the CPM and process views respectively (cf. Figure 8.9a). By contrast, operations belonging to category *AU2 (CPM-CS Update)* affect both the CPM and the creation set CS of the process view on which the update is applied to (cf. Figure 8.9b). Note that the update of creation set CS is performed by the update operation itself, but is not result of any view creation operations (cf. Section 8.3.1) or migration rules.

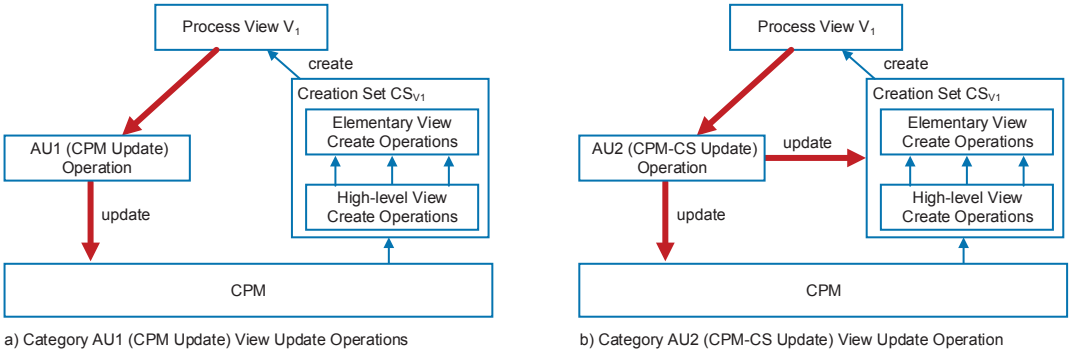


Figure 8.9: Categories of Advanced View Update Operations

Category *AU1 (CPM Update)*. Operation $InsertActivityAfter(V, n_{new}, n)$ constitutes an advanced view update operation belonging to category *AU1 (CPM Update)*. It inserts activity n_{new} directly after process node n in process view V. Therefore, operation $InsertActivityAfter$ is mapped to elementary view update operation $InsertActivity(V, n, succ(V, n))$ with parameter setting $InsertSerialMode = EARLY$, i.e., it is inserted between n and its direct successor in the CPM (cf. Figure 8.6). Similar to $InsertActivityAfter$, $InsertActivityBefore(V, n_{new}, n)$ inserts an activity n_{new} directly before n in process view V.

Operation $InsertProcessFragment(V, P, n_1, n_2)$ inserts a SESE fragment P into process view V between nodes n_1 and n_2 of V. The operation is split up into a sequence of elementary view update operations $\langle op_1, \ldots, op_k \rangle$ inserting the nodes of P stepwise.
Note that $InsertProcessFragment$ corresponds to process change pattern *AP1 (Insert Process Fragment)* as described in [158] and formally specified in [180]. As opposed to AP1, however, operation $InsertProcessFragment$ not only affects the control flow perspective, but data flow and process attributes as well. Figure 8.10 shows an example of inserting a process fragment P into process view V. Three elementary view update operations are required. Note that these operations are able to deal with ambiguities when propagating the change to the CPM due to its mapping to elementary update operations.
View update operation $DeleteProcessFragment(V, N')$ deletes the SESE block induced by node set N' in process view V, i.e., nodes contained in set N' and respective edges are deleted. Note that this operation corresponds to change pattern *AP2 (Delete Process Fragment)* [158, 180].

View update operation $ReplaceProcessFragment(V, P, N')$, in turn, replaces the SESE block induced by node set N' with process fragment P (i.e., SESE block) in V. To be more precise, N' is removed and the nodes from P, together with their precedence relations, are inserted in V and corresponding CPM. Figure 8.11 exemplarily shows the application of this operation. Note

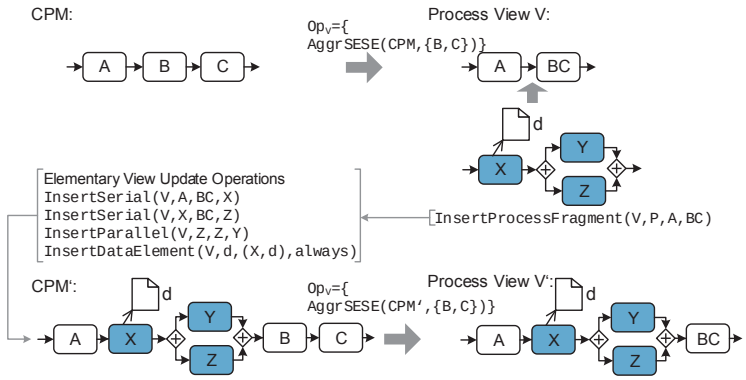


Figure 8.10: Advanced View Update Operation: InsertProcessFragment

that $ReplaceProcessFragment(V, P, N')$ corresponds to change pattern *AP4 (Replace Process Fragment)* [158, 180].

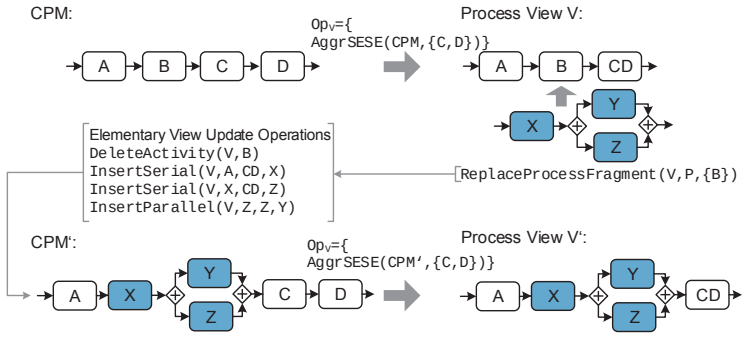


Figure 8.11: Advanced View Update Operation: ReplaceProcessFragment

As illustrated in Figure 8.6, view update operation $MoveProcessFragment(V, N', n_1, n_2)$ moves the SESE block induced by node set N' to the position between process nodes n_1 and n_2. Note that this operation can be also realized by the combined application of view update operations $DeleteProcessFragment$ and $InsertProcessFragment$. Furthermore, $MoveProcessFragment$ corresponds to change pattern *AP3 (Move Process Fragment)* as described in [158].

Table 8.2 summarizes the advanced view update operations from category *AU1 (CPM Update)*.

Category *AU2 (CPM-CS Update)*. Compared to operations provided by category AU1 (CPM Update), operations of category *AU2 (CPM-CS Update)* modify the CPM as well as the creation set of the process view on which the update is applied to. Figure 8.12 shows an example of inserting a business object $d1d2$ (i.e., an aggregation of elementary data elements $d1$ and $d2$) in process view V. Data elements $d1$ and $d2$ are inserted to the CPM together with related data edges by applying the respective elementary view update operations. Afterwards, operation set Op_V of creation set CS_V is extended by view creation operation $AggrDataElement$ aggregating elementary data elements $d1$ and $d2$ to business object $d1d2$.

Advanced View Update Operation	Description
$InsertActivityAfter(V, n_{new}, n)$	Inserts activity n_{new} directly after activity n in process view V.
$InsertActivityBefore(V, n_{new}, n)$	Inserts activity n_{new} directly before activity n in process view V.
$InsertProcessFragment(V, P, n_1, n_2)$	Inserts process fragment P into process view V between nodes n_1 and n_2.
$DeleteProcessFragment(V, N')$	Deletes the SESE block induced by node set N' in process view V.
$ReplaceProcessFragment(V, P, N')$	Replaces the SESE block induced by node set N' with process fragment P.
$MoveProcessFragment(V, N', n_1, n_2)$	Moves the SESE block induced by node set N' to its new position between nodes n_1 and n_2 in process view V.

Table 8.2: Overview of Advanced View Update Operations of Category AU1 (CPM Update)

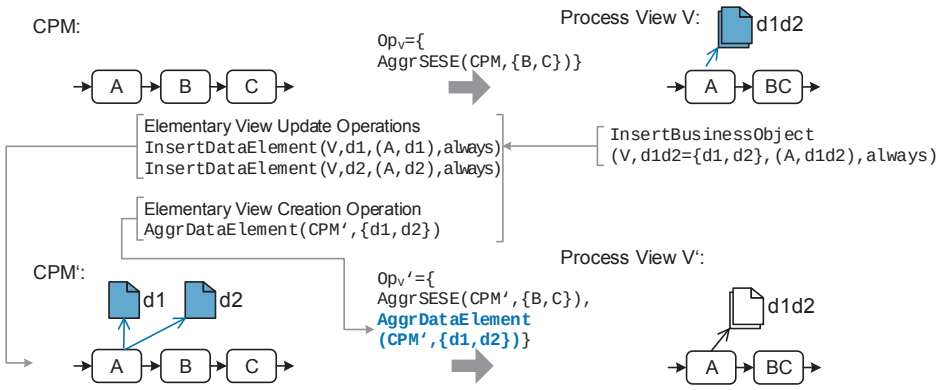


Figure 8.12: Advanced View Update Operation: InsertBusinessObject

View update operation $InsertSubProcess(V, P, n)$ inserts a sub-process model P in process view V by refining activity n. Accordingly, activity n is deleted in the CPM and P is inserted instead. Afterwards, the creation set of process view V is extended with a view creation operation that aggregates all nodes of P. Hence, there is no visible difference for the user of process view V (cf. Section 6.5). Moreover, other process views may display the inserted nodes.

View update operation $LocalInsert(V, op)$ enables a user to apply an insert operation $op \in \{InsertSerial, InsertParallel, InsertConditional, \ldots\}$ solely to his process view V, i.e., all other process views based on the same CPM reduce the inserted nodes. Accordingly, for process view V_i with $CS_i = (CPM, Op_i, PS_i)$ the set of view creation operations Op_i is extended with a reduction operation hiding the update performed by op.

Table 8.3 summarizes advanced view update operations from category *AU2 (CPM-CS Update)*.

The application of advanced view update operations of both categories to the CPM must not be interrupted. To be more precise, elementary view update operations representing the advanced update must be executed as a transaction. This is especially required, if multiple users interact and update a CPM based on their views (cf. Section 7.8).

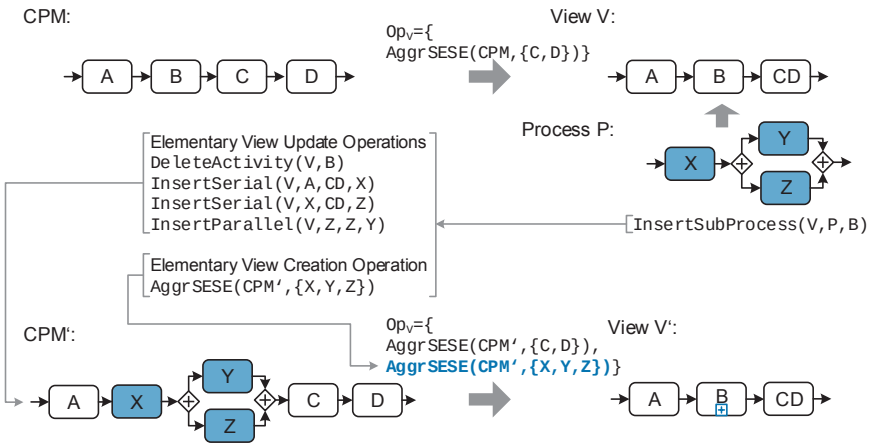


Figure 8.13: Advanced View Update Operation: InsertSubProcess

Advanced View Update Operation	Description
$InsertBusinessObject(V, D', de, det)$	Inserts a business object represented by set $D' = \{d_1, \ldots, d_n\}$ to process view V. Furthermore, data edge de connecting the business object to the nodes of V is inserted with type det.
$InsertSubProcess(V, P, n)$	Inserts a sub-process model P refining activity n of process view V.
$LocalInsert(V, op)$	Applies insert operation op to process view V. Inserted nodes are hidden in all process views associated to the same CPM except process view V by inserting respective reduction operations.

Table 8.3: Overview of Advanced View Update Operations of Category AU2 (CPM-CS Update)

8.3.3 Expressiveness of View Update Operations

This section evaluates the expressiveness of view update operations. We compare the set of view update operations with process change patterns as introduced in [158, 180]. Change patterns describe frequently used updates on process models. In particular, change patterns were empirically evaluated [158]. Table 8.4 summarizes change patterns as well as related view update operations. In particular, a change pattern may be expressed by one advanced view update operation or by multiple elementary operations.

Change patterns *AP1 (Insert Process Fragment)*, *AP2 (Delete Process Fragment)*, *AP3 (Move Process Fragment)*, and *AP4 (Replace Process Fragment)* can be directly realized based on advanced view update operations (cf. Table 8.4). These patterns may also be carried out by a set of elementary view update operations inserting or deleting process nodes. Change pattern *AP5 (Swap Process Fragment)* swaps two process fragments. It is not directly supported by an advanced view update operation, but can be realized by applying operation *MoveProcessFragment* once or twice (depending on whether the process fragments succeed each other).

Change pattern *AP6 (Extract Sub Process)* extracts a process fragment and transforms it to a sub-process. By contrast, change pattern *AP7 (Inline Sub Process)* inlines a sub-process. These patterns can be simulated by adding or deleting operations, which aggregate the respective process fragment to the view definition, i.e., the underlying CPM needs not be changed.

Change Pattern	View Update Operation	
	Advanced	**Elementary**
AP1 (Insert Process Fragment)	InsertProcessFragment	Insert{Serial, Parallel, Conditional, Loop, DataElement, DataEdge}
AP2 (Delete Process Fragment)	DeleteProcessFragment	DeleteActivity, DeleteDataElement
AP3 (Move Process Fragment)	MoveProcessFragment	Insert and Delete Operations
AP4 (Replace Process Fragment)	ReplaceProcessFragment	Insert and Delete Operations
AP5 (Swap Process Fragment)	1-2x MoveProcessFragment	Insert and Delete Operations
AP6 (Extract Sub Process)	View Creation Op: AggrSESE, AggrComplBranches	-
AP7 (Inline Sub Process)	Delete View Creation Op: AggrSESE, AggrComplBranches	-
AP8 (Embed PF in Loop)	-	InsertLoop
AP9 (Parallelize Activity)	-	InsertParallel
AP10 (Embed PF in Cond. Br.)	-	InsertConditional
AP11 (Add Ctrl Dependency)	-	InsertSyncEdge
AP12 (Remove Ctrl Dependency)	-	DeleteSyncEdge
AP13 (Update Condition)	-	ChangeAttribute
AP14 (Copy Process Fragment)	InsertProcessFragment	Insert{Serial, Parallel, Conditional, Loop, DataElement, DataEdge}

Table 8.4: Relation Between Change Patterns and View Update Operations

Change pattern *AP8 (Embed Process Fragment in Loop)* embeds a process fragment in a loop. This pattern is not supported by an advanced view update pattern, but by elementary view update operation *InsertLoop*. The latter inserts a loop surrounding a SESE block. Change patterns *AP9 (Parallelize Activity)* and *AP10 (Embed Process Fragment in Conditional Branch)*, which embed activities in an AND or XOR branching, can be realized like pattern *AP8*. Optionally, a move operation is required to move an existing process fragment to the newly inserted branching.

Change patterns *AP11 (Add Control Dependency)*/*AP12 (Remove Control Dependency)* insert and remove control dependencies in a process model. In particular, *AP11* is used to synchronize the execution of activities from parallel branches. In our framework, this pattern can be realized by elementary view update operations *InsertSyncEdge* and *DeleteSyncEdge*.

Change pattern *AP13 (Update Condition)* updates the branching condition of an XOR branch. For this purpose, the elementary view update operation *ChangeAttribute* may be applied to update the respective branching condition. Change pattern *AP14 (Copy Process Fragment)* copies an existing process fragment to a new position in the process model. To simulate this change, the advanced view update operation *InsertProcessFragment* may be applied referring to an existing process fragment as parameter. Alternatively, elementary view update operations inserting process nodes may be applied.

The comparison of view update operations and change patterns shows that most of the change patterns are directly supported. Furthermore, some of them (e.g., *AP5 (Swap Process Fragment)*) can be realized by combining multiple advanced or elementary view update operations.

8.4 Advanced Process View Migration

Information related to applied high-level view creation operations and advanced view update operations can be used to automatically decide whether a CPM update is relevant for a given process view, e.g., inserted nodes are shown, reduced, or aggregated. Subsequently, improvements based on the application of advanced view update operations are presented.

Presence of high-level view creation operations. High-level view creation operations may provide information whether an inserted process node shall be shown, reduced, or aggregated in a process view to be migrated.

Consider the example from Figure 8.14. Process view $V1$ is created based on elementary view creation operation *RedActivity*, whereas $V2$ is based on high-level view creation operation *ShowActivitiesOfUser*. When inserting activity $a14$ between $ANDjoin_1$ gateway and activity $a8$ in the CPM, both process views (i.e., $V1, V2$) need to be migrated to the new CPM version by applying migration rule MR4. The latter decides whether $a14$ shall be shown in respective process views. Note that the process fragment preceding $a14$ has been reduced in both process views. In this example, parameter *RedPartlyMode* of migration rule MR4 is set *RED*, i.e., inserted activities are reduced. Hence, view creation operation $RedActivity(CPM, a14)$ is added to the respective operation set of $V1$. By contrast, for $V2$ we know that it shall only show activities of user role *Manager*. Hence, parameter *RedPartlyMode* is automatically set to *SHOW* for this migration, i.e., $a14$ is shown in $V2$ since $a14$ is assigned to user role *Manager*. If the information of the high-level view creation operation had not been used, migration rule MR4 would have added operation $RedActivity(CPM, a14)$ to the operation set of $V2$.

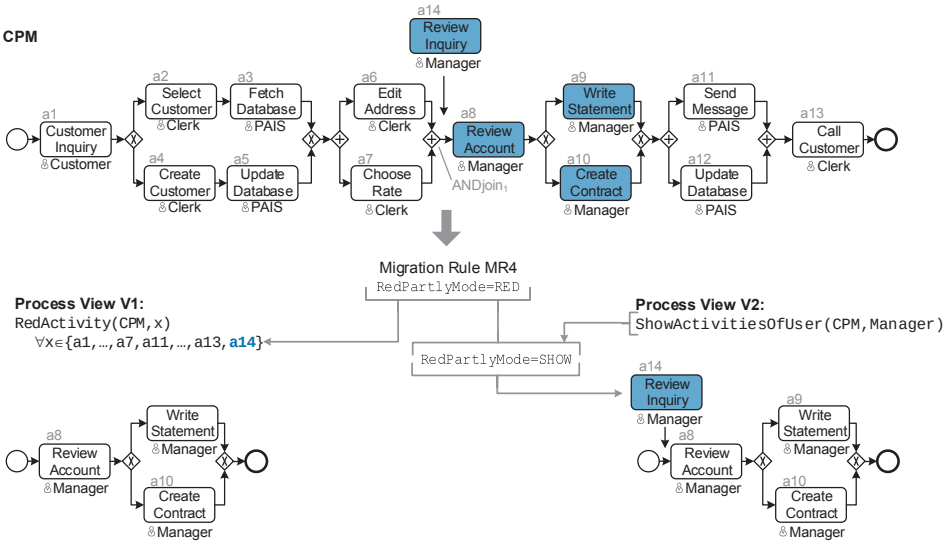


Figure 8.14: Advanced View Migration: ShowActivitiesOfUser

Another example is illustrated in Figure 8.15. Information about high-level view creation operations of the process view, which shall be migrated, is used to influence migration behaviour as

well as about view creation operations of the process view, which triggers the update. $V1$ is created by applying operation $SubgraphRange(CPM, a8, a13)$, whereas $V2$ applies high-level view creation operation $SubgraphRange(CPM, a8, XORjoin)$. Both process views select a range in the CPM with the same first node and differ in their last node. Then, activity $a14$ is inserted between nodes $StartFlow$ and $a8$. This update is propagated to CPM (with parameter setting $InsertSerialMode = LATE$). When migrating $V2$ to the new CPM version the information about how $V1$ has been constructed may be used to decide whether this update is relevant for $V2$. In Figure 8.15, both process views apply the same high-level view creation operation having overlapping ranges. Hence, it is more likely that the update is relevant for $V2$ as well. Accordingly, view creation $SubgraphRange$ is modified in order to show activity $a14$ (i.e., $SubgraphRange(CPM, a14, XORjoin)$).

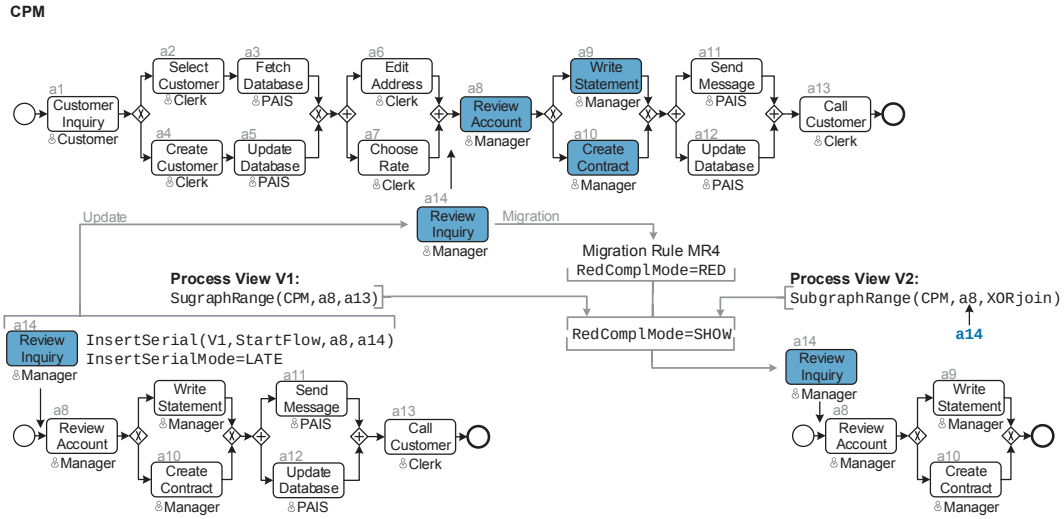


Figure 8.15: Advanced View Migration: SubgraphRange

Presence of advanced view update operations. In the following, the migration behaviour is discussed after a CPM update performed by advanced view update operations.

Consider Figure 8.16. Regarding $V1$, activity $a14$ is inserted through advanced update operation $LocalInsert$. In turn, this leads to the reduction of the inserted node in all associated process view except $V1$. Accordingly, the operation set of $V2$ is extended with view creation operation $RedActivity(CPM, a14)$.

A more sophisticated example is shown in Figure 8.17. Activity $a14$ is inserted directly before $a8$ in process view $V1$ (cf. Figure 8.17). When migrating process view $V2$ to the new CPM version, this information can be used to determine whether $a14$ shall be shown in $V2$. Since the user inserts $a14$ directly before $a8$, both activities may belong together. Hence, it is more likely that $a14$ is relevant for $V2$, i.e., migration parameter $RedPartlyMode$ shall be set to parameter value $SHOW$.

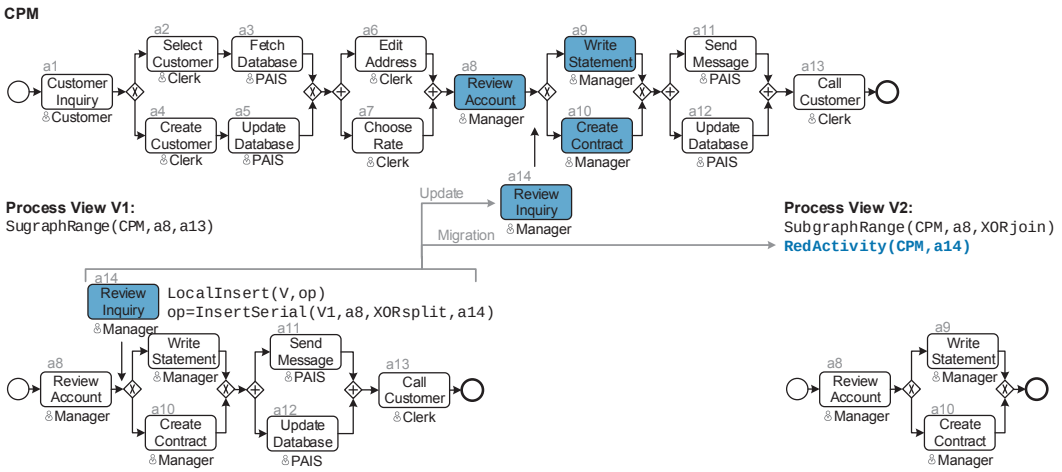


Figure 8.16: Advanced View Migration: LocalInsert

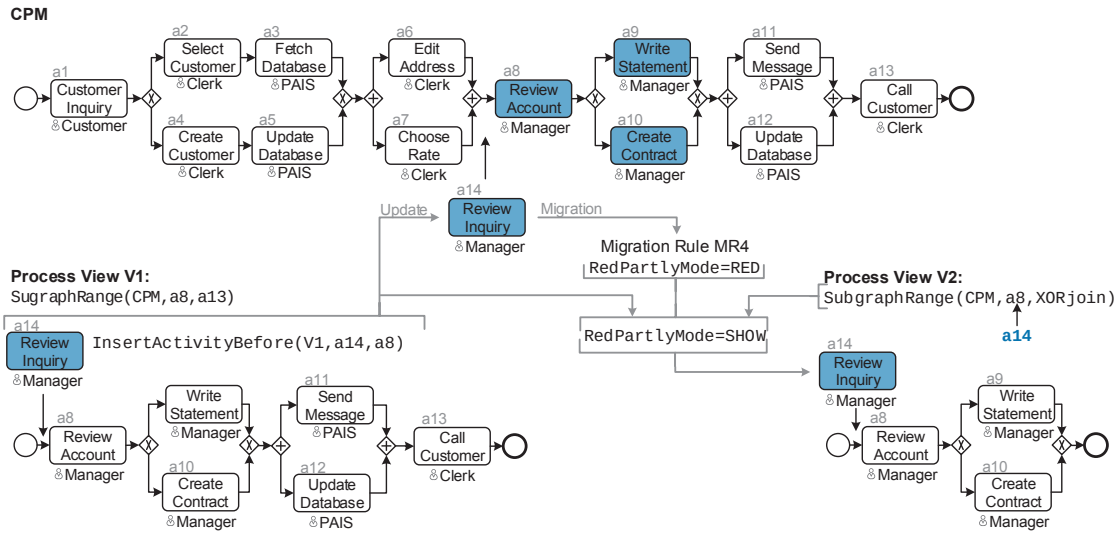


Figure 8.17: Advanced View Migration: InsertBefore

This advanced view migration behaviour needs to be specified in further *advanced migration rules*. However, these rules are highly dependent on advanced view update and high-level view creation operations, i.e., these rules rely on the presence (or absence) of individual operations.

8.5 Discussion

The introduced high-level view creation and advanced view update operations improve the propagation and migration behaviour of updates. In particular, exploiting the information on how the process view triggering an update or the process views to be migrated are constructed, (i.e., by which high-level view creation operations) allows for a more precise propagation and migration behaviour. Furthermore, if updates are triggered through advanced view update operations, even more information becomes available for providing a precise propagation behaviour.

Figure 8.18 shows the dimensions of view updates and migrations. The axes represent the precision regarding update propagation and view migration. If elementary view update operations and migration rules are applied, a *rudimentary* user support can be provided with a low update and migration precision.

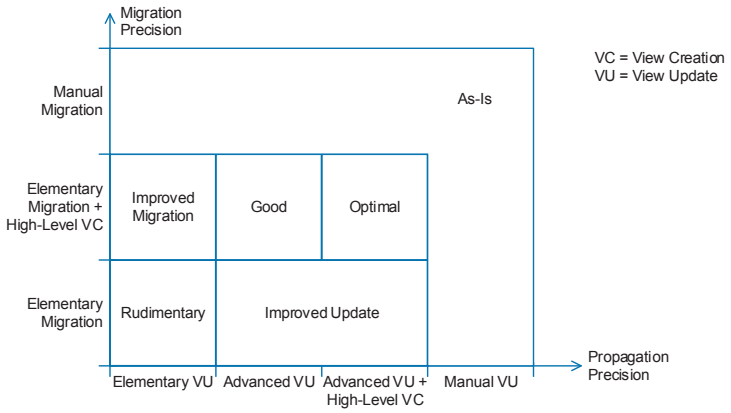


Figure 8.18: Propagation and Migration Dimensions

An *improved migration* behaviour can be provided, if process views are created with high-level view creation operations. In turn, an *improved view update* and migration behaviour can be provided, if advanced view update and/or high-level view creation operations are used. In case the process views to be migrated are built with high-level view creations, a *good* user support can be provided. Finally, an *optimal* precision regarding view update and migration can be provided if all process views are created by high-level view creation operations and a change is triggered through advanced update operations.

The highest precision regarding view update and migration can be achieved, if the user triggers updates and migrates process views manually (i.e., *as-is* situation). In this case, no ambiguities occur and the user decides on his own whether or not he wants to migrate a process view after an update in the CPM. This behaviour may be non-suitable for end-users, but for a process designer.

Although advanced view update operations and advanced migration rules are based on corresponding elementary operations and rules, it is not straightforward to implement them in a PAIS. Particularly, they have to take the circumstances (i.e., the structure of the CPM as well as associated process views) into account.

8.6 Summary

This chapter addresses the limitations of the view update operations and migration rules introduced in Chapter 7. First, high-level view creation operations are presented, which allow defining process views in a more convenient way compared to elementary view creation operations. Then, improvements of view update propagation are discussed by utilizing the semantics of advanced view update operations as well as high-level view creation operations. Finally, insights into improvements regarding view migrations are provided. Such advanced update propagation and migration may improve applicability of a framework providing process abstractions based on process views.

Part III

Process Representations and Process Interaction Concepts

9 Process Representations

9.1 Introduction

The visualization of process models constitutes an integral part of any process modeling tool. Usually, process models are visually represented as process diagrams, e.g., using graphical languages like BPMN [87], EPC [53], or *Workflow Nets* [203]. In turn, respective process modeling languages were designed semantics and expressiveness in mind [204], whereas graphic design conventions and guidelines have not been considered yet [204, 205, 206]. However, the latter are required to reduce model complexity as well as to increase model comprehensibility [54].

Using process views to abstract process models constitutes one building block to foster process model comprehensibility by end-users. Another one concerns the visualization of the process models, denoted as *process representation* in the following, which should address the specific needs of end-users (cf. Requirement REQ-6). In particular, the large number of modeling elements in languages like BPMN 2.0 causes high efforts for users when comprehending or modeling process models. Exactly for this reason, in practice, often only a subset of the available modeling elements is used [101, 207]. Another issue affecting process model comprehensibility concern the degree of connectivity (i.e., number of control edges). Several studies have shown that process models with a low degree of connectivity are easier to comprehend than models with a high degree of connectivity [208, 209, 210].

Before any process modeling initiative, a process modeling language has to be chosen. Replacing this language at a later stage, is hardly supported by contemporary process modeling tools (cf. Chapter 4). Particularly, the same kind of process model visualization is provided to all users, independent from their needs and vocational background. Since each process modeling language may be used in different application domains [211], process modeling tools should allow dynamically switching between different process representations (cf. Requirement REQ-8).

The chosen representation of process models is crucial for enabling end-user process model up-

dates as well (cf. Requirement REQ-7). Thus, users can work with the same process visualization to view and update process models without the need for switching their context (i.e., process modeling language). It may be not suitable for users to use a different process modeling language to perform process updates. Furthermore, guidance shall be provided to support non-technical users to create correct process models and achieve a high process model quality [184].

This chapter introduces alternative *process representations* for CPMs and process views respectively. Since process views themselves are represented as process models (cf. Definition 6.1), any process representation may be used for visualizing process views as well. Hence, this chapter needs not distinguish between CPM and related process views. A process representation aims to visualize a process model in an easy-to-understand representation. In particular, it should be possible to dynamically switch between the various process representations without the need of manually transforming process models. In the *proView* framework, four different process representations are supported (cf. Figure 9.1): BPMN 2.0, hierarchical representation, form-based representation, and verbalized process description. Note that we do not intend to develop new process representations, but adapt existing ones from other domains to address aforementioned issues.

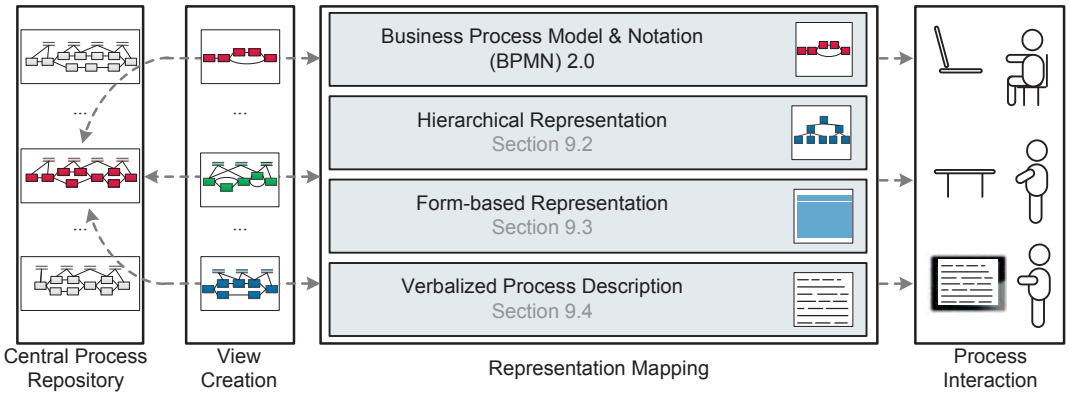


Figure 9.1: Overview of Visualization Component

Visualizing a process model in terms of BPMN 2.0 has been already described in Section 6.2. This chapter focuses on how process models can be transformed to these process representations on the one hand and on the use of these representations for enabling process updates on the other.

Example 9.1 (Process Representations)
Consider the credit application process and the related process views $V1, V2$, and $V3$ from Example 8.1. In addition to Example 8.1, the CPM comprises a loop structure enclosing activities Select Customer (i.e., a2) and Fetch Database (i.e., a3), i.e., the two activities are repeated until the desired customer is found in the database.

Assume that activity Create File (i.e., activity a14) is inserted in $V1$ (see Examples 7.1 and 7.2 for a description of the respective update propagation and migration). Figure 9.2 shows the corresponding process models before and after this update.

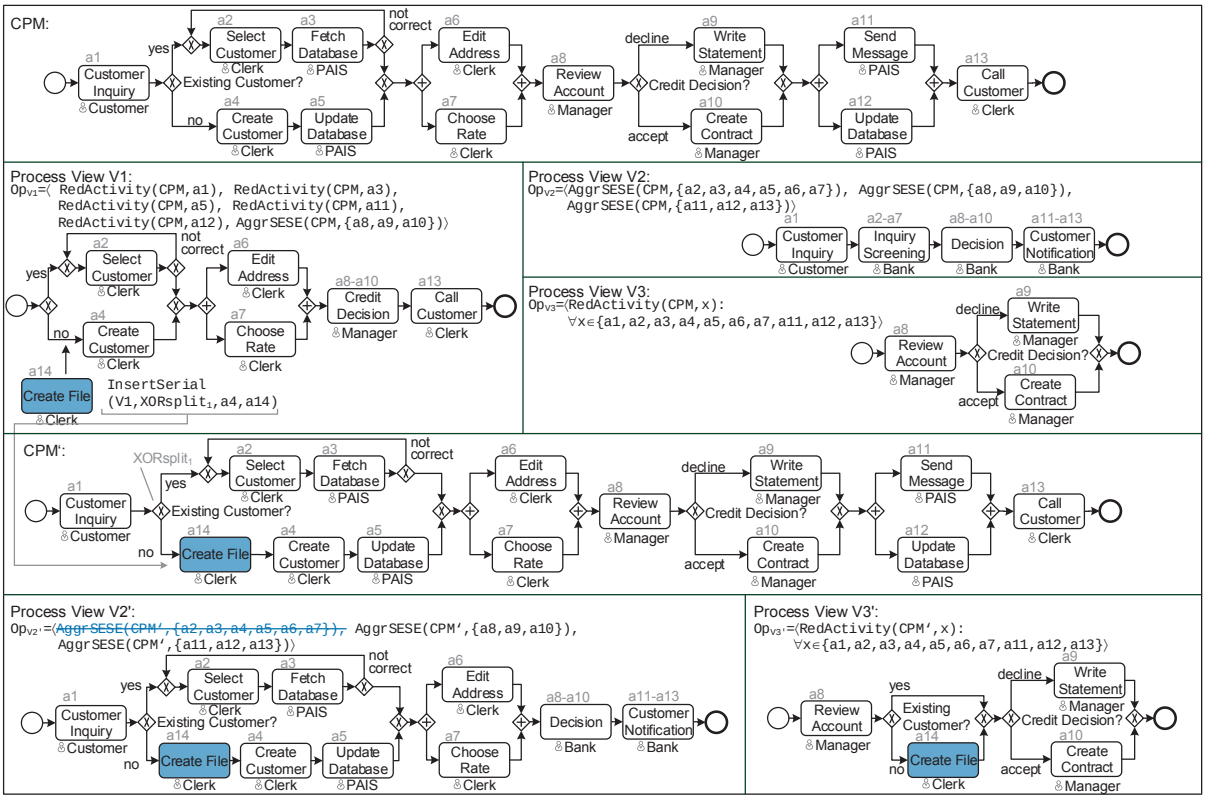


Figure 9.2: Credit Application Process

An effective way of reducing the complexity of diagrams is to decompose them into smaller parts [54]. This is also known as *modularization* [212] or *divide and conquer* [213]. To adopt this principle, Section 9.2 presents a *hierarchical representation*, i.e., a tree-based representation of a process model. The latter is based on *Concurrent Task Trees (CTT)* [214], which originate from end-user development in software engineering [215]. In particular, the chosen hierarchical representation allows for a top-down structuring of process models as suggested in [116].

The *graphic complexity* of diagrams may be reduced as well. The latter is described by the number of graphical conventions, i.e., syntactical rules of process modeling languages [216]. Hence, graphical complexity may be reduced by hiding certain graphical process model elements (e.g., control edges). In this context, Section 9.3 presents a *form-based representation*. It eliminates control edges and reduces the number of different shape types for visualizing process elements. Note that this form-based representation is related to *Nassi-Shneiderman-Diagrams*, which use nested rectangles for visualizing the behaviour of program code [217].

Section 9.4 suggests a *verbalized process description*, which presents a process model in terms of text expressed as natural language. In particular, utilizing natural language minimizes process modeling knowledge required. Finally, Section 9.5 summarizes further process representations. Section 9.6 discusses related work and the chapter is concluded in Section 9.7.

9.2 Hierarchical Representation

The *hierarchical representation* presented in the following is based on *Concurrent Task Trees* (CTT), which provide a widely used task modeling language for describing the behaviour of end-user interfaces [218, 219, 214, 220, 221]. In particular, CTT has been used in the area of *End-User Development (EUD)* [222]. When creating a CTT, the user specifies a *hierarchical task model* (i.e., a tree). Thereby, lower level tasks refine upper level ones. This hierarchical structuring addresses the demand to structure complex conceptual models in a top-down manner [54].

Temporal relations between tasks on the same level specify the order in which these tasks shall be executed. Thus, a CTT allows specifying the behaviour of a user interface in a top-down and left-right direction. Accordingly, a CTT complies with the "usual" reading and writing direction of text in Western countries (i.e., from left to right and from top to bottom) [223].

9.2.1 Creating Hierarchical Representations

First of all, we describe how a process model can be mapped to a hierarchical representation.

Activity. A CTT uses *tasks* to describe changes of the state of a user interface and associated information system, respectively. Thereby, a task represents "work" accomplished by a user of the information system. Since the activities of a process models describe the behaviour of a business process, we map them to CTT tasks (cf. Definition 6.1).

In general, a CTT may comprise five types of tasks (cf. Figure 9.3): A *user task* represents a task to be performed by a user without need for interacting with an information system (e.g., a clerk interviewing a customer). If the user shall interact with an information system (e.g., filling data into a user form), in turn, an *interaction task* is used. Note that interaction tasks must be triggered by a user. Furthermore, *cooperation tasks* may involve more than one user. An *application task* can be executed by an information system without need for any user interaction (e.g., storing data in a database). Finally, an *abstraction task* is used, if none of the aforementioned task types applies. Its semantics may be specified by refining it through subordinated tasks. When generating a hierarchical representation for process models, the user assignment may determine the task type. For example, if a human user is assigned to an activity, an interaction task is used. By contrast, an application task is used when assigning a PAIS to the task. Each task type is visualized through a specific icon (cf. Figure 9.3) [224].

Figure 9.3: Task Types of Hierarchical Representation

Creating a hierarchical representation based on CTT constitutes a top-down approach. The root task describes the name of the process model and may be refined through lower-level tasks. The different levels are connected by *hierarchical relations*. After having specified the hierarchical relations between the tasks of the different tree levels, the *temporal relations* among tasks of the

same tree level need to be specified. Temporal relations describe the order in which the tasks of a particular level shall be executed. CTT allows for eight different kinds of temporal relations of which only a subset is used in this thesis [214, 218]. The *enabling* relation, which is represented by a temproal relation edge with symbol "≫", corresponds to a sequence flow in a process model. Figure 9.4 shows an example of transforming process model P to the hierarchical representation. Particularly, the sequential control flow between activities A, B, and C is mapped to *enabling temporal relations*. Note that the edges between abstraction task P and tasks A, B, and C symbolize hierarchical relations.

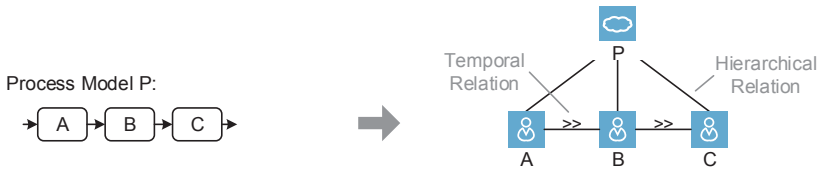


Figure 9.4: Visualizing a Sequence of Activities in Hierarchical Representations

Enabling with information passing (i.e., symbol "[] ≫") constitutes another temporal relation. It has the same semantics as the enabling relation, but additionally expresses a data flow between the two tasks. More precisely, $A[] \gg B$ expresses that when completing A, task B will be enabled and respective data be passed. However, it is not explicitly documented, which particular data elements are passed over.

AND Branching. Parallelism of the control flow is expressed through the temporal relation *parallel* visualized by three vertical lines "|||" (cf. Figure 9.5). For each branch of the AND branching another level is introduced. This allows users to read a process model top-down with increasing level of detail.

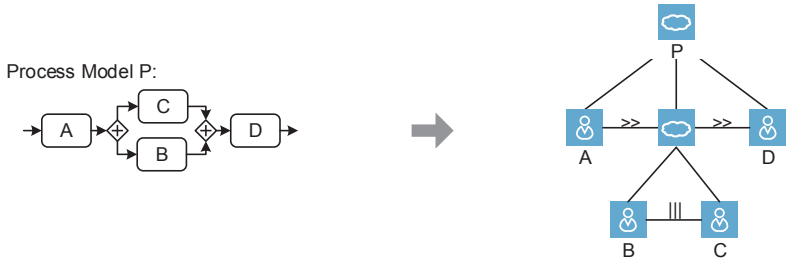


Figure 9.5: Visualizing an AND Branching in Hierarchical Representations

XOR Branching. Another relation type of the CTT is the *choice* relation (expressed by symbol "[]"). For example, $A[]B$ expresses that the user may decide whether A or B shall be executed. As soon as one of the two tasks is selected, the other one is no longer enabled. Note that this behaviour corresponds to the one expressed by the *deferred choice* workflow pattern [174]. In particular, it constitutes the only way in CTTs to express an alternative relation. Usually, in

a process model a choice is based on transition conditions on data values. However, since this workflow pattern (i.e., *exclusive choice*) is not supported by CTT, we extend the set of temporal relations. In Figure 9.6, the XORsplit gateway is represented by task *Choice?*, whereas its child tasks *c1* and *c2* represent the respective associated branches.

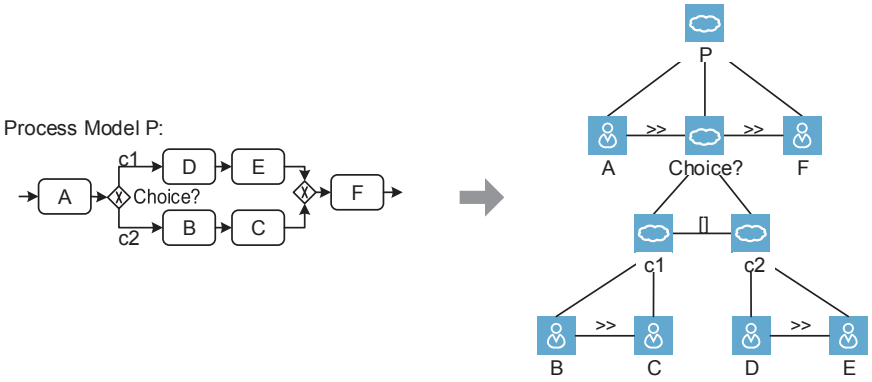


Figure 9.6: Visualizing a XOR Branching in Hierarchical Representations

Synchronization Edge. Control edges synchronizing tasks from parallel branches are not directly supported in CTTs. However, they may be realized by connecting two tasks through an *enabling temporal relation*. Note that such a connection between different levels of a hierarchical representation might affect its understandability.

Loops. Temporal relation *iteration* allows repeating a task until it is terminated by a user (i.e., A^*) or a pre-specified number n of iterations (i.e., $A[n]$) is reached. Note that the *iteration* relation is not represented by an edge in the hierarchical representation, but directly assigned to a task executed repeatedly. However, to express a loop in a hierarchical representation a branching condition must be defined. For this purpose, relation *conditional iteration* is introduced. Figure 9.7 shows conditional iteration $Choice?[c1]$. Corresponding child tasks (i.e., A and B) are executed as long as branching condition $c1$ evaluates to *true*. Otherwise, its succeeding task D is activated.

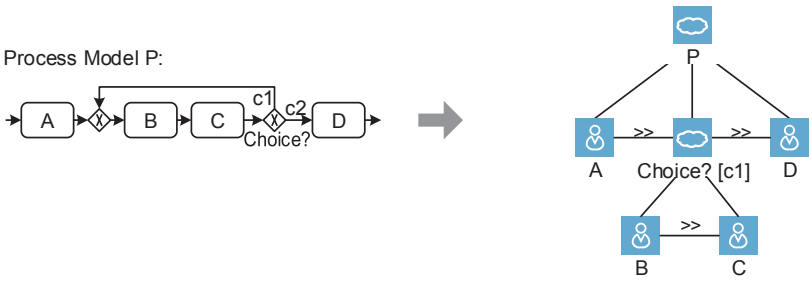


Figure 9.7: Visualizing a Loop Branching in Hierarchical Representations

Data Flow and Process Attributes. Data elements and process attributes can not be visualized in CTT, but need to be described in separate documents [214]. Hiding data flow in a process model reduces complexity for users. Therefore, we do not introduce new elements visualizing data flow and process attributes in the hierarchical representation.

Figure 9.8 visualizes the process model from Example 9.1 in terms of a hierarchical representation (i.e., CTT) comprising five levels. The root task represents the entire process model. Level 1 comprises seven sequential tasks: abstraction task *Preparation* corresponds to the first XOR branching of the credit application process. The respective XOR branches as well as their tasks are mapped to subordinated levels. In analogy to this, abstraction tasks *Credit Conditions*, *Decision* and *Notification* correspond to respective XOR branchings (cf. Figure 9.2).

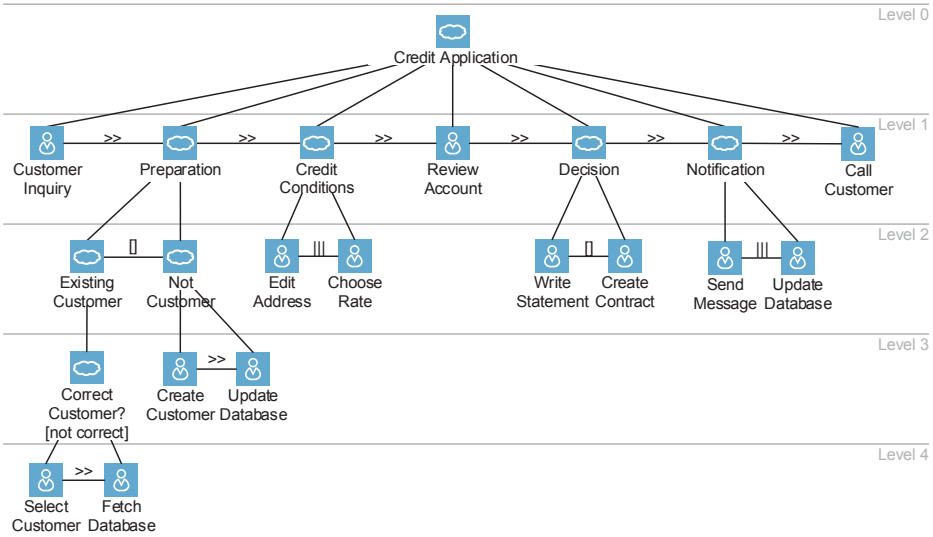


Figure 9.8: Credit Application Process Visualized Using Hierarchical Representation

Figure 9.9 visualizes process views $V1$-$V3$ of the credit application process (cf. Example 9.1) as hierarchical representation. As can be seen, unbalanced trees may result.

The tree-based structure of the hierarchical representation allows for a top-down decomposition. In particular, this structure may be abstracted (etailed) by folding (unfolding) sub-trees (cf. Figure 9.10). As example consider Figure 9.10a: the tree may be unfolded step-by-step by the user, e.g., using a zoom slider to refine the level of detail. Numbers in (red) circles right next to a task symbolize how many tree levels are folded. When reaching the leaf level of a hierarchical representation, the most detailed level is shown to the user. We denote this decomposition as *leveled exploration method*. In particular, it assists users in understanding or exploring processes step-by-step. When refining an activity-oriented process model (like BPMN) by opening a sub-process, in turn, the superordinate process model is no longer displayed. Thus, the latter might loose the context of the sub-process. By contrast, the context is kept when using the leveled exploration method in the hierarchical representation.

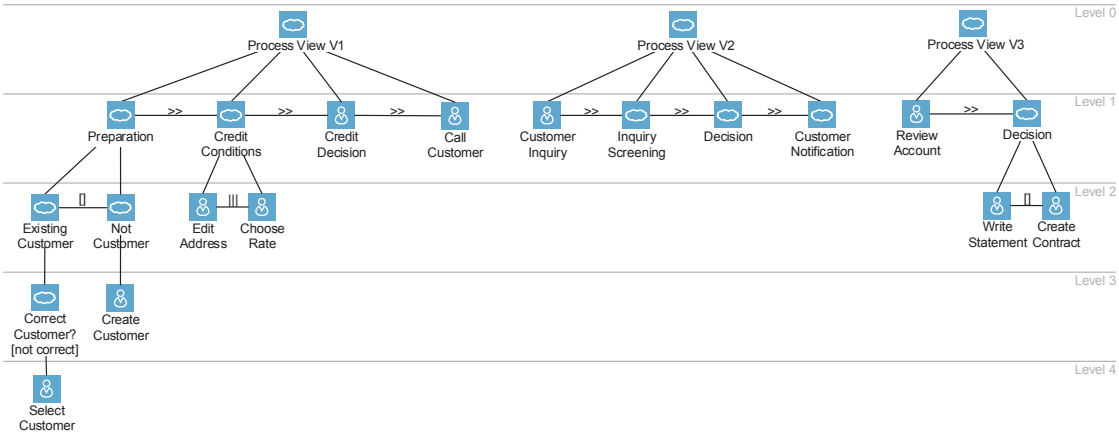


Figure 9.9: Credit Application Process Views Visualized Using Hierarchical Representation

Figure 9.10b shows another method for exploring hierarchical representations. In the *depth exploration method*, the user may click on a folded task (e.g., task *Preparation*). Then, the corresponding sub-tree is unfolded. Using this exploration method, users are able to interactively explore process models. Starting with an abstract tree the user may decide which parts shall be refined. Clicking on the same task again will unfold this sub-tree. This way the user may decide which region of the process model shall be explored at which level of granularity.

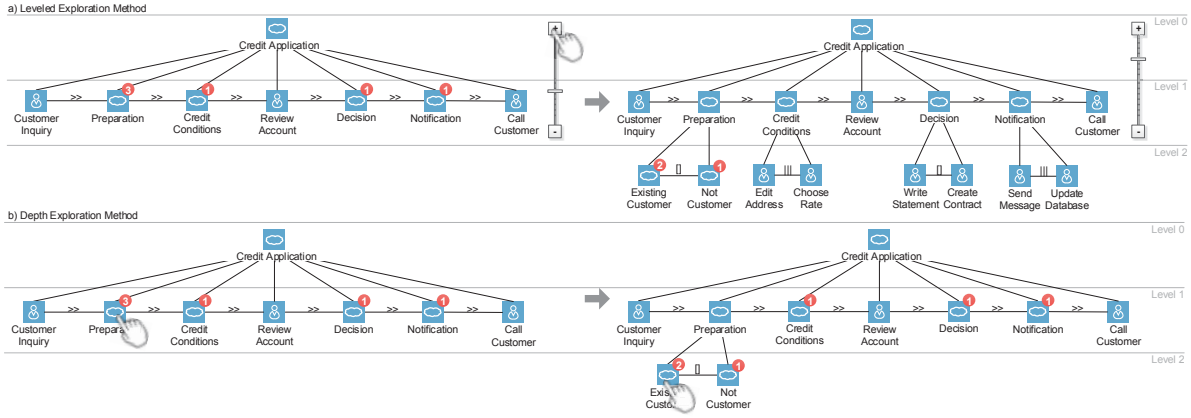


Figure 9.10: Exploration of Credit Application Process

9.2.2 Updating Hierarchical Representations

In general, it should be possible to update process models based on a hierarchical representation, i.e., without need to switch the representation to a BPMN model. In the following, we describe how a process model can be modified through adaptations of the hierarchical representation.

Inserting Process Elements. Inserting a task in a hierarchical representation corresponds to the

addition of an activity to the respective process model. To determine whether the activity shall be inserted sequentially, in parallel, conditionally, or iteratively, respective temporal relations need to be defined for the inserted task. As example consider Figure 9.11a. B is first added and then connected with A using an *enabling temporal relation*. In turn, this update sequentially inserts B after A. By contrast, connecting B with A using a *parallel temporal relation* will be accompanied by inserting B in parallel to A in the process model (cf. Figure 9.11b). If there already exist other tasks connected by parallel relations on the same tree level, another branch will be inserted in the process model. Similarly, loops and XOR branchings may be handled.

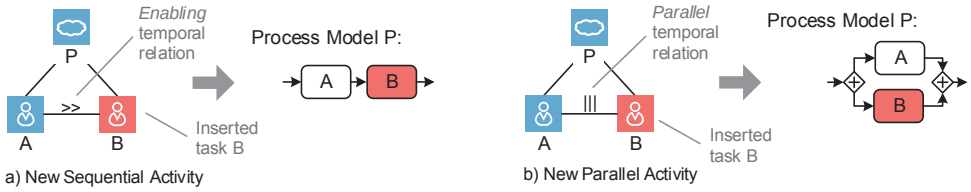


Figure 9.11: Inserting a Task as well as a Temporal Relation

When inserting tasks and respective temporal relations, ambiguities might occur. In particular, this applies when introducing different kinds of temporal relations at the same tree level. As example consider Figure 9.12: A, B, and C are located on the same tree level with temporal relations $A >> B|||C$; i.e., A and B are connected by an *enabling* relation, whereas B and C are connected with a *parallel* relation. Hence, there are two ways of interpreting these relations: $((A >> B)|||C)$ or $(A >> (B|||C))$ (cf. Figure 9.12). In any case, such ambiguity should be resolved in a unique way, i.e., either process model P' or P'' in Figure 9.12 results. CTTs offer two options in this context [214]: either the priority order defined by LOTOS [225] may be used (i.e., *choice* > *parallel* > *enabling*) or another tree level may be added to resolve the ambiguity. In this thesis, the second option is applied as it requires no additional modeling knowledge.

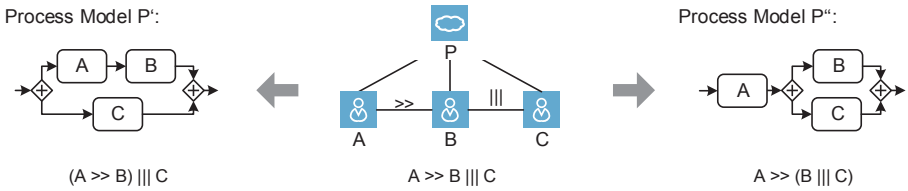


Figure 9.12: Ambiguity after Inserting a Temporal Relation

Inserting Branches. When using the hierarchical representation, the insertion of a new branch to an existing AND branching is not supported since there is no distinct representation of an AND branch. In order to insert an XOR branch, in turn, a task representing the branching condition has to be inserted. For example in Figure 9.13, a task representing branching condition $c3$ is added to the hierarchical representation on Level 1. In turn, this leads to the insertion of a conditional branch to the respective process model. Note that branching conditions $c1$ and $c2$ must be adapted to guarantee control flow correctness (cf. Section 7.3.1).

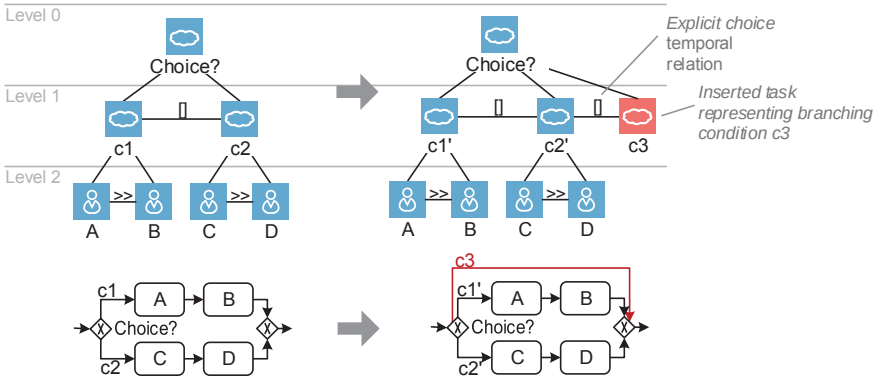


Figure 9.13: Inserting a Conditional Branch

Deleting Process Elements. Removing a task in a hierarchical representation triggers the deletion of the respective activity in the corresponding process model (cf. Section 7.3.2). Depending on the temporal relations connecting the deleted task with tasks on the same tree level, branchings have to be removed as well. In Figure 9.14, for example, task C is deleted. Afterwards, the temporal relation expressing that B and C shall be executed in parallel must be removed to obtain a valid tree again. In turn, this triggers an operation for deleting the AND branching. In the hierarchical representation, the respective abstraction task is removed and the tree level reduced. Note that the removal of the AND branching must not be applied if the latter contains other branches. In this case, however, empty branches should be removed.

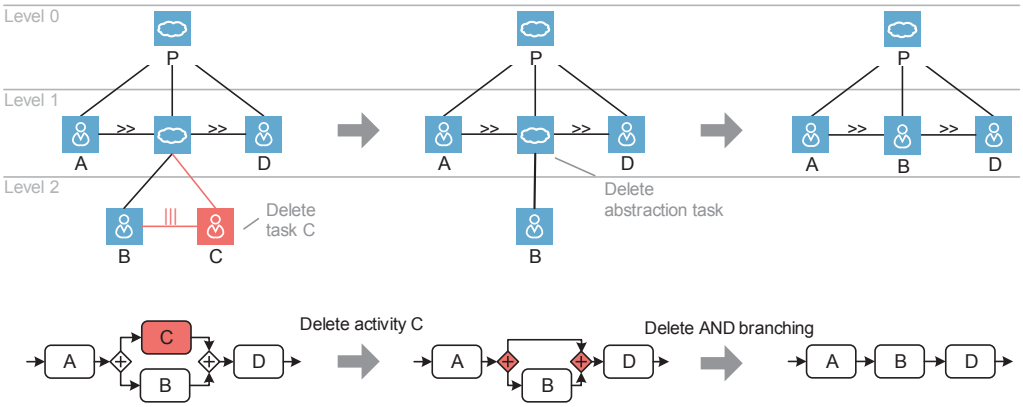


Figure 9.14: Delete a Task in Hierarchical Representation

Changing Temporal Relations. When changing the type of temporal relations between tasks in the hierarchical representation, in the corresponding process model complex structural changes might be required. In Figure 9.15, for example, the type of the temporal relation between A and B are changed from *enabling* to *parallel* in order to express that associated tasks shall be executed in parallel. In turn, an AND branching has to be inserted in the process model and activity B be moved in parallel to activity A.

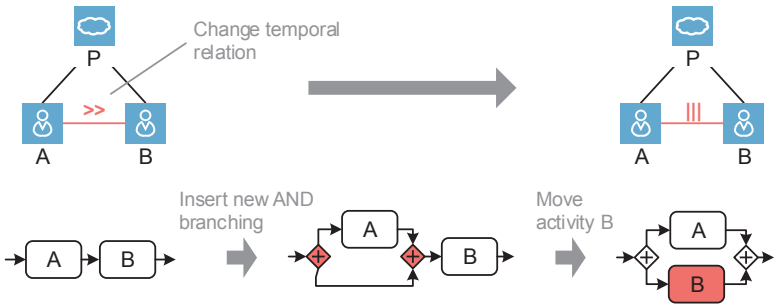


Figure 9.15: Changing Temporal Relations in Hierarchical Representation

Updating Data Flow and Process Attributes. As discussed in Section 9.2.1, data flow and process attributes are not considered in the hierarchical representation. If these shall be changed, therefore, additional dialogs need to be provided. As an alternative, data flow and process attributes may be changed directly in the process model.

Note that the tree corresponding to a hierarchical representation will *increase the number of levels* when inserting new branchings. By contrast, inserting sequential tasks or branches to existing branchings, the tree *width* will be increased. Especially, trees with many levels will become too complex for users.

In Figure 9.16, view updates are applied to the credit application process (cf. Example 9.1). Furthermore, the effects of propagating this update to the respective CPM (cf. Figure 9.8) and view migration (cf. Figure 9.9) are shown. As can be seen, for process views $V2'$ and $V3'$ a new sub-tree subordinating task *Preparation* must be added when propagating the update of $V1$ to them. In particular, the user may comprehend which parts of the tree structure has not changed. Note that this increases understandability.

9.2.3 Discussion

The hierarchical representation allows visualizing process models as a tree-based structure. Thereby, it addresses the requirement of modularization as discussed in [54]. In particular, this allows exploring process models either level-by-level (i.e., leveled exploration) or selectively by following specific sub-trees (i.e., depth exploration).

A hierarchical representation increases the number of elements required to visualize a process model. In turn, this might increase its complexity as well. Concerning process updates, however, it is easier to modify (i.e., delete, move, replace) large process fragments since the respective sub-tree may be deleted, moved, or replaced in the hierarchical representation. Furthermore, updates requiring a set of update operations in process models can be easily performed in the hierarchical representation by switching the type of temporal relation (e.g., moving a sub-tree in parallel to another one).

Table 9.1 shows how change patterns match to updates in such a hierarchical representation.

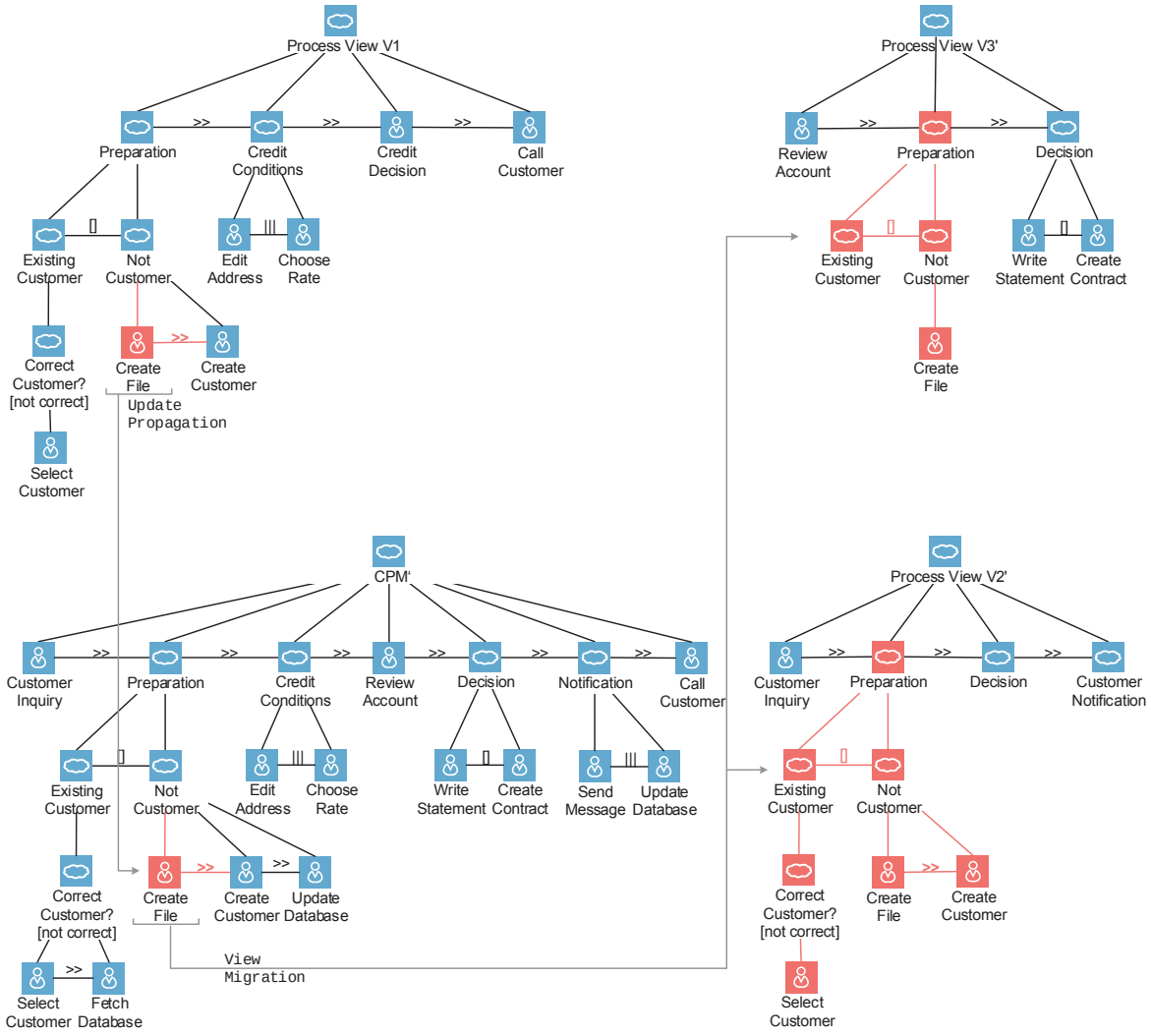


Figure 9.16: Credit Application Process Update in Hierarchical Representation

In particular, change patterns related to sub-processes are not applicable to the hierarchical representation, since the latter does not consider the sub-processes. All other change patterns are applicable.

Change Patterns	Counterpart in Hierarchical Representation
AP1 (Insert Process Fragment)	Inserting a sub-tree and connecting it with a temporal relation. Depending on the type of temporal relation, the respective tasks are inserted conditionally, sequentially, or in parallel.
AP2 (Delete Process Fragment)	Deleting the respective sub-tree in the hierarchical representation.
AP3 (Move Process Fragment)	Moving a sub-tree to another position in the hierarchical representation.
AP4 (Replace Process Fragment)	Replacing a sub-tree of the hierarchical representation by another one.
AP5 (Swap Process Fragment)	Swapping two sub-trees in the hierarchical representation.
AP6 (Extract Sub-Process)	Not applicable.
AP7 (Inline Sub-Process)	Not applicable.
AP8 (Embed PF in Loop)	Inserting a superordinate task to a sub-tree representing a loop.
AP9 (Parallelize Activity)	Changing the temporal relation between respective tasks to a parallel relation.
AP10 (Embed PF in Conditional Branching)	Inserting a superordinate task to the sub-tree representing the conditional branching.
AP11 (Add Ctrl Dependency)	Inserting a respective synchronization edge.
AP12 (Remove Ctrl Dependency)	Deleting the respective synchronization edge.
AP13 (Update Condition)	Updating the abstraction task representing the respective branch.
AP14 (Copy Process Fragment)	Duplicating a sub-tree of the hierarchical representation.

Table 9.1: Change Patterns in Hierarchical Representations

9.3 Form-based Representation

The *form-based representation* for visualizing process models is based on *Nassi-Shneiderman-Diagrams* [217] presuming well-structuredness of the process models (cf. Section 6.2). More precisely, the form-based representation visualizes process models through nested rectangles [226], requiring a minimal set of visualization symbols compared to BPMN. Note that reducing the amount of symbols may increase model comprehensibility [216].

9.3.1 Creating Form-based Representations

We first show how to transform a process model to a form-based representation. Then, the form-based representation is applied to Example 9.1.

Activity. In a form-based representation, an activity is visualized as a rectangle. Its label is added to the center of the rectangle (cf. Figure 9.17). Subsequent activities, in turn, are placed below this rectangle, i.e., activities of a sequence are not connected through edges. Instead, their order is represented through the vertical (i.e., top-down) placement of the rectangles. Finally, a form-based representation has no explicit start and end nodes.

AND Branching. Figure 9.18 shows the form-based representation of an AND branching. As can be seen, each branch is visualized as a separate column, which contains all activities of the respective branch. Note that neither ANDsplit nor ANDjoin gateways are explicitly visualized, which might increase process model comprehensibility.

Figure 9.17: Sequence of Activities in Form-based Representations

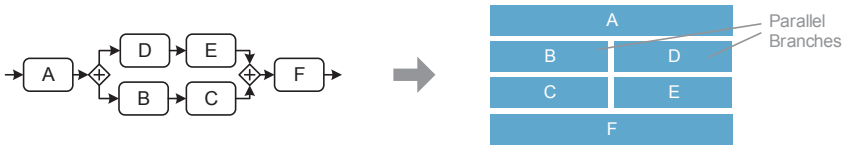


Figure 9.18: AND Branching in Form-based Representations

If the drawing area of the form-based representation is not wide enough to visualize all branches, only a subset of them are shown. The presence of additional branches is then indicated through *narrow columns*. As example consider Figure 9.19. Activity A is succeeded by an AND branching with four branches. The branches with activities B and C (and D and E respectively) are visible, whereas the other two branches are only visualized as narrow columns. If the user clicks on one of the narrow columns, it (i.e., the respective branch) is unfolded; e.g., G and H may then be visible (cf. Figure 9.19).

Figure 9.19: Unfolding a Nested Branching

XOR Branching. As opposed to ANDsplit gateways, XORsplit gateways and their branching conditions need to be explicitly visualized (cf. Figure 9.20). To be more precise, an XORsplit gateway is visualized in terms of a rectangle showing the branching expression (i.e., *Choice?* in Figure 9.20). For each conditional branch, another rectangle, displaying the condition of the respective branch, is added (i.e., $c1$ and $c2$ in Figure 9.20). In turn, the columns below these rectangles comprise the activities (or nested branchings) of the respective branch. Again, in case of limited width *narrow columns* may be used. As opposed to AND branchings, the corresponding XORjoin gateways of XOR branchings are visualized. Note that this becomes necessary to visualize the join (i.e., XORjoin gateway) of an XOR branching. Otherwise, no visual indicator to a succeeding AND branching exists.

Loops. In the form-based representation, a loop is visualized as thick arrow around the loop activities (cf. Figure 9.21). Particularly, that arrow provides the branching condition of the backward jump. The branching condition to continue in the control flow (i.e., not to jump back), is not explicitly visualized in form-based representation.

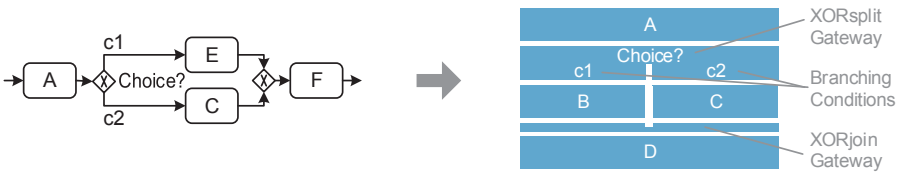


Figure 9.20: XOR Branching in Form-based Representations

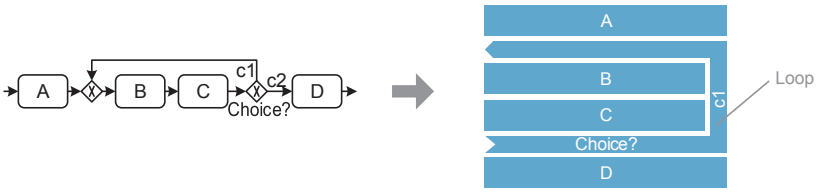


Figure 9.21: Loop in Form-based Representations

Synchronization Edge. As opposed to control edges, synchronization edges of a process model must not be omitted when transforming the model to a form-based representation. Therefore, a synchronization edge is mapped to a directed edge connecting two rectangles.

Data Flow and Process Attributes. As opposed to BPMN, for example, data flow is not explicitly visualized. Therefore, data elements read or written by an activity are visualized underneath the respective rectangle when clicking on the latter. This unfolded area provides additional information to users (cf. Figure 9.22). Its left-hand side shows data elements read or written by the activity, whereas its right-hand side displays process attributes.

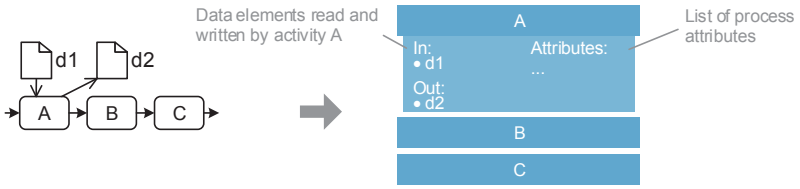


Figure 9.22: Data Elements in a Form-based Representation

The form-based representation could be extended with an interaction concept that allows visualizing data flow, e.g., if the user clicks on a data element in the unfolded area, all other activities accessing this data element may be colored accordingly.

At run-time, the data elements of an unfolded area are replaced by form fields to enter or display data values. In this context, well-known techniques for automatically creating user forms can be used [2, 7, 227, 228].

Figure 9.23 shows the form-based representation of the credit application process (cf. Example 9.1). As can be seen not all rectangles representing activities have same height due to the varying number of process elements on different branches. Note that neither the height nor the width of a rectangle is correlated with the duration or importance of the respective activity. In turn, Figure 9.24 shows the form-based representation of the process views related to the credit application process (cf. Example 9.1). Note that $V2$ does not comprise branchings anymore and, thus, allows for a compact process representation.

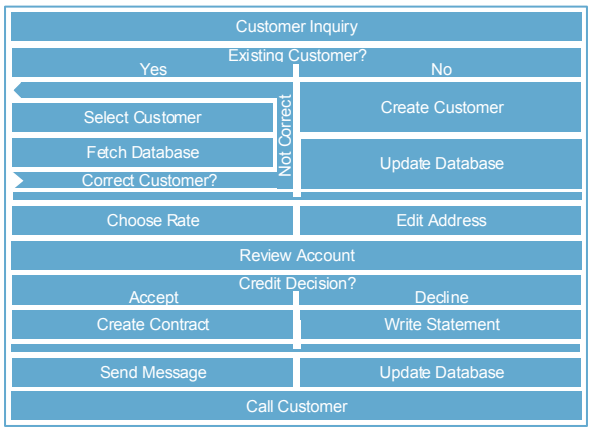


Figure 9.23: Form-based Representation of the Credit Application Process

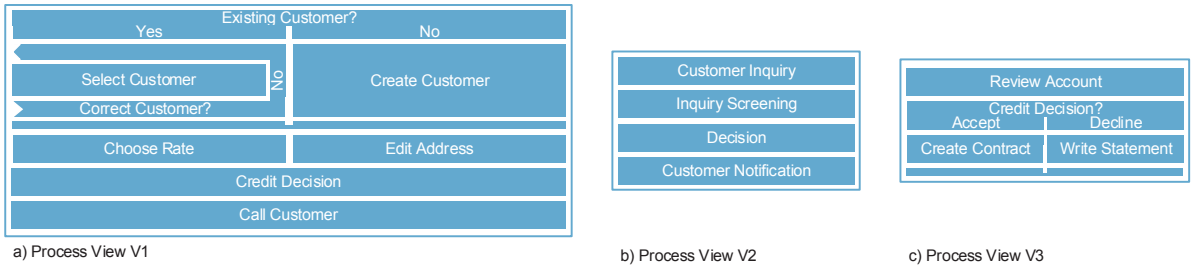


Figure 9.24: Form-based Representation for Views of the Credit Application Process

9.3.2 Updating Form-based Representations

Since a form-based representation visualizes a process model based on well-nested rectangles, users can easily perform updates by inserting or deleting rectangles. Remember that the rectangles correspond to SESE blocks in the corresponding process model.

Figure 9.25 shows a user interface that supports the modification of a process model based on its form-based representation. The column on the left depicts the list of available modeling elements. The column on the right, in turn, displays the form-based representation to be modified. If the

user drags a modeling element from the left to a position close to a rectangle in the form-based representation, possible insert positions are indicated by empty rectangles with dotted lines. When inserting an activity above activity *Create Customer* in Figure 9.25, this corresponds to a serial insertion. By contrast, inserting the new activity on the left of activity *Create Customer* is accompanied by the insertion of an AND branching, i.e., activity *Create Customer* and the newly inserted activity are then located on parallel branches. If there is already an AND branching, another parallel branch will be added.

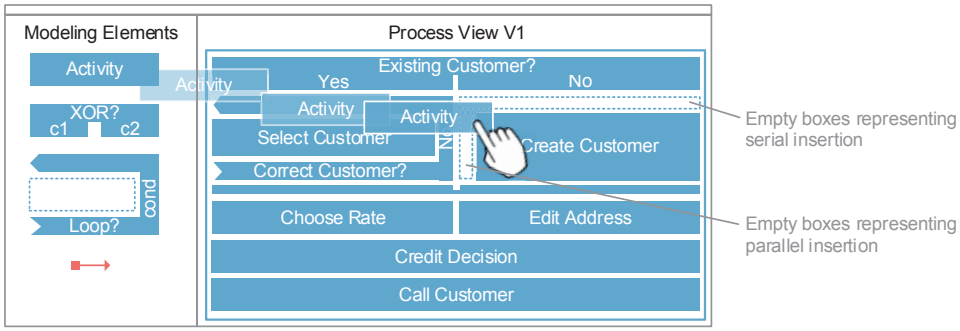


Figure 9.25: Performing Updates in the Form-based Representation

In the same way, XOR branchings and loops may be added (cf. Figure 9.25). Finally, synchronization edges may be inserted, which requires specifying their start and target rectangle.

Process elements may be also removed through drag&drop in a form-based representation. Note that this results in a well-structured process model again.

In order to define the data flow or process attributes values the respective activity is unfolded. Then three types of buttons are displayed right next to the data elements and process attributes: *edit*, *add*, and *delete* (cf. Figure 9.26). The *add* button (displayed using a plus symbol) allows inserting data elements and process attributes. When inserting a data element, in turn, a drop down menu appears in which existing data elements may be selected. Alternatively, the user may enter a new data element name, which triggers respective operations to insert data elements (e.g., *InsertDataElement* for process views). The *delete* button removes the selected data element or process attribute. Finally, the *edit* button (i.e., displayed as a pencil) allows updating values of process attributes, e.g., the label of the activity.

Again, for updating a form-based representation change patterns are supported (cf. Table 9.2).

9.3.3 Discussion

The form-based representation provides an easy to understand top-down visualization of process models. Particularly, the number of different shapes representing process elements is reduced and not all elements of a process model are visualized (e.g., ANDsplit gateway) [216]. Note that this is not contradicting the theory of symbols that requires a 1:1 matching of symbol and its meaning [229]. In turn, data flow and process attributes are only visualized if required. Another strength of the form-based representation is the ease of performing updates by end-users, which

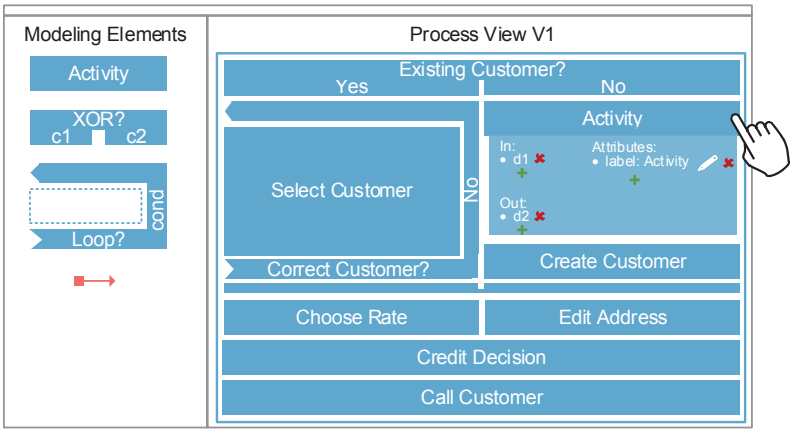


Figure 9.26: Updating Data Flow and Process Attributes of Form-based Representation

Change Patterns	**Counterpart in Form-based Representation**
AP1 (Insert Process Fragment)	Dragging the respective process elements from the sidebar to the form-based representation.
AP2 (Delete Process Fragment)	Deleting the respective rectangle from the form-based representation.
AP3 (Move Process Fragment)	Dragging respective rectangles to a new position in the form-based representation.
AP4 (Replace Process Fragment)	Not applicable. Needs to apply AP3 (Move Process Fragment) twice.
AP5 (Swap Process Fragment)	Not applicable. Needs to apply AP3 (Move Process Fragment) twice.
AP6 (Extract Sub-Process)	Not applicable.
AP7 (Inline Sub-Process)	Not applicable.
AP8 (Embed PF in Loop)	Inserting a loop element surrounding the respective process fragment (i.e., rectangles).
AP9 (Parallelize Activity)	Moving the respective rectangle in parallel (i.e., left or right) to another rectangle.
AP10 (Embed PF in Conditional Branching)	Inserting XOR branching surrounding the respective process fragment (i.e., rectangles).
AP11 (Add Ctrl Dependency)	Inserting a synchronization edge from the sidebar.
AP12 (Remove Ctrl Dependency)	Deleting the respective synchronization edge in the form-based process representation.
AP13 (Update Condition)	Updating the rectangles representing branching conditions.
AP14 (Copy Process Fragment)	Duplicating respective rectangles.

Table 9.2: View Update Operations and their Support in Form-based Representations

utilizes the well-structuredness of process models. Finally, the user obtains direct feedback where to insert new process elements.

In case a branching has numerous branches, which exceeds the width of the drawing area, the user is unable to view the complete process model. Then, further action (i.e., clicking on the narrow columns visualizing hidden branches) is required to explore the process model.

9.4 Verbalized Process Description

In general, domain experts are unfamiliar with process modeling languages and, hence, are unable to understand process models in detail. By contrast, a verbalization (i.e., textual representation) of the process model would enable them to properly understand the process details relevant for them [230]. This section presents such a verbalized representation, which relies on well-established methods to create verbalized process descriptions (see [231]). Further, it extends these methods to support verbalized process updates as well.

9.4.1 Creating Verbalized Process Descriptions

This section describes how a process model may be transformed to a verbalized process description. Figure 9.27 gives an overview of the text generation technique applied in this context. Note that we apply techniques presented in [232] and integrate them with the visualization component.

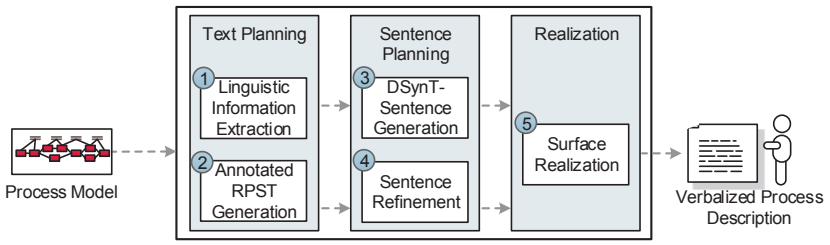


Figure 9.27: Modules for Deriving Verbalized Process Descriptions

The visualization component is extended by five modules for verbalizing a process model, which can be categorized into *text* and *sentence planning* as well as *realization* (cf. Figure 9.27). *Text planning* modules analyze the structure and contents of the process models (e.g., activity labels, user assignments). Its results are structured preparing the text generation. Subsequently, the *sentence planning* modules are applied to prepare the content and structure of the resulting verbalized process description (i.e., the structure of the text). Finally, the *realization* module creates the verbalized process description.
In the following each module is described in detail:

1. *Linguistic Information Extraction.* The first module (cf. Figure 9.27, Step 1) applies a well-known linguistic label analysis technique [232] that utilizes the lexical database

WordNet [233] and the *Stanford Tagger* [234] to recognize the various name patterns that exist for activity labels in a process model. For instance, we are able to decompose an activity label such as *Choose Contact Type* into action *Choose* and object *Contact Type*.

2. *Annotated RPST Generation.* The *Refined Process Structure Tree (RPST)* [235] generation module derives a tree representation from the given process model to provide a basis for a step-by-step process description (cf. Figure 9.27, Step 2). In particular, the computed RPST is a parse tree containing a hierarchy of sub-graphs derived from the process model.[1] RPST is chosen as well-established transformations to the data structure of the following module exist [232]. To be more precise, the RPST represents the hierarchical order of the well-nested SESE blocks of a process model. The result can be visualized as a tree whose root captures the entire process model and whose leaves contain connections between two process elements. Then, we annotate each process element with the linguistic information obtained in the previous phase. For example, the leave node pointing to activity *Choose Contact Type* is annotated with action *Choose* and business object *Contact Type*.

3. *DSynT Sentence Generation.* The sentence generation module maps the annotated RPST to a list of intermediate sentences (cf. Figure 9.27, Step 3). More specifically, each sentence is stored as a *Deep-Syntactic Tree (DSynT)*, which corresponds to a dependency representation introduced by the *Meaning Text Theory* [237]. Such a deep-syntactic tree facilitates the manageable, yet comprehensive storage of the constituents of a sentence. In addition, it can be automatically mapped to a syntactically correct sentence with existing technologies [238]. Taking the example of activity *Choose Contact Type*, the corresponding DSynT consists of a root node pointing to verb *choose* and two subordinate nodes: the first specifies *contact type* as object and the second specifies *clerk* as subject of *choose*.

4. *Sentence Refinement.* Within the sentence refinement module, we take care of sentence aggregation, referring expression generation, and discourse marker insertion (cf. Figure 9.27, Step 4). The need for these measures arises if the considered process model contains long sequences of activities. In such cases, for instance, we aggregate sentences sharing the same object. For example, assume that a sequence of activities comprising *Choose Contact Type* and *Select Contact Type* shall be performed by a user with role *Clerk*. Instead of generating one sentence per activity, these activities are aggregated and described by one sentence such as "The clerk chooses and selects the contact type." An alternative aggregation strategy is to insert referring expressions such as *he* or *it* to ensure lexical variety. Regarding the discourse marker insertion, a set of connectors is used to insert markers such as *then* and *afterwards*, i.e., a well readable text with sufficient variety is obtained.

5. *Surface Realization.* The surface realization takes the DSynT of the previous modules and outputs the final verbalized process description (i.e., text). The complexity of the surface realization task has led to the development of publically available realizers (cf. Figure 9.27, Step 5). We decided to use the DSynT-based realizer RealPro from CoGenTex [238], which requires an XML-based DSynT message as input, transforming it to a grammatically correct sentence. As a result, the DSynT for activity *Choose Contact Type* is automatically transformed into sentence "The clerk chooses the contact type."

[1] A detailed introduction on RPST can be found in [236]

After processing a process model in these modules, the resulting personalized and verbalized process description may be displayed to the respective user. Figure 9.28 shows the verbalized process description of the credit application process (cf. Example 9.1). Branchings (i.e., AND, XOR) and Loops are visualized as an enumeration block with bullet points. Depending on the type of branching an introducing sentence is provided before; e.g., "... the process is split into X parallel streams of action:" in the context of AND branchings. Futhermore, in case of nested branchings, the bullet points are also nested and indentation is increased. Finally, user assignments are integrated with the created sentences.

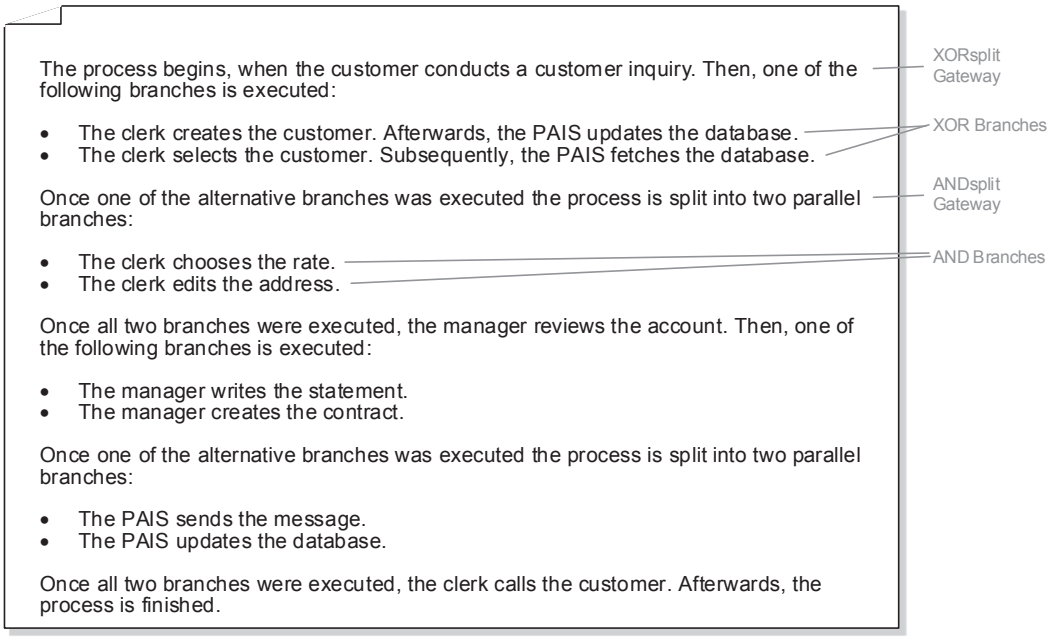


Figure 9.28: Verbalized Process Description of the Credit Application Process

Figure 9.29 shows the verbalized process descriptions of the process views from Example 9.1. Compared to Figure 9.28, imperative formulations are applied since the process views shall assist specific users (e.g., $V1$ is created for the user (role) *Clerk*). In this way, a personalization of the verbalized process description can be achieved.

9.4.2 Updating Verbalized Process Descriptions

This section describes how changes of a personalized and verbalized process description (i.e., text changes) can be mapped to updates of the corresponding process model (cf. Section 7.3). More precisely, the changes of a verbalized process description must be interpreted and mapped to respective updates of the corresponding process model [12, 239].

Inserting a Sentence. When adding a sentence to a verbalized process description, the user expresses that an action (i.e., activity) shall be added to the process model. Figure 9.30 shows

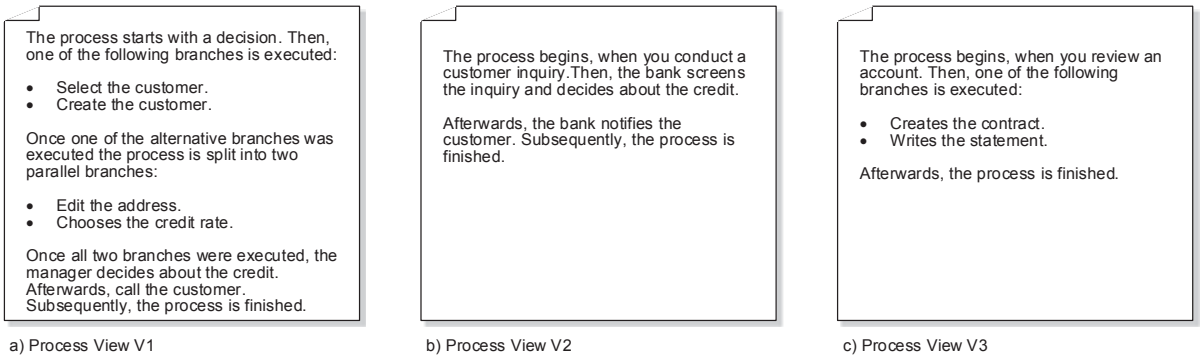


Figure 9.29: Verbalized Process Description of the Credit Application Process Views

an example of such an insertion. Sentence "The clerk prints the details." is added by the user to the verbalized process description and analyzed using the *Stanford Parser* [240]. From the parsing result, the grammatical relation of the words in the sentence can be automatically derived. Relations *nsubj(prints, clerk)* and *dobj(prints, details)* reveal that "clerk" represents the subject and "details" represents the object for predicate "print". Consequently, we extract "clerk" as subject, "details" as object, and "print" as predicate. This information is then used to insert activity *Print Details*, which shall be performed by user role *Clerk*, into the corresponding process model.

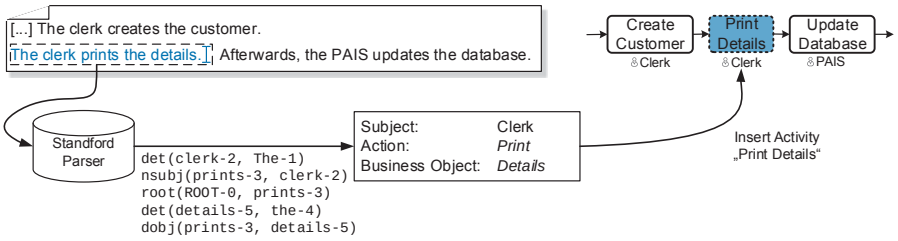


Figure 9.30: Inserting a Sentence and Adapting the Corresponding Process Model

Inserting an Enumeration. Inserting an enumeration block into the verbalized process description implies that the user wants to insert multiple actions that shall be performed in parallel or alternatively. Regarding the corresponding process model, the user intends to insert an AND/XOR/Loop branching. However, he must manually add the information whether the bullet points of the enumeration express parallelism (i.e., AND branching is inserted) or alternative choices (i.e., XOR branching is inserted). Inserting individual sentences to the bullet points triggers again updates to insert the respective activities. Figure 9.31 gives an example of inserting a new enumeration block that performs the bullet points in parallel.

Inserting a Bullet Point. The user adds a bullet point to an existing enumeration in the process description in order to insert a stream of actions that shall be performed in parallel or alternatively to the already existing bullet points. Inserting a bullet point corresponds to the insertion of a branch to an existing AND/XOR branching in the corresponding process model. Initially,

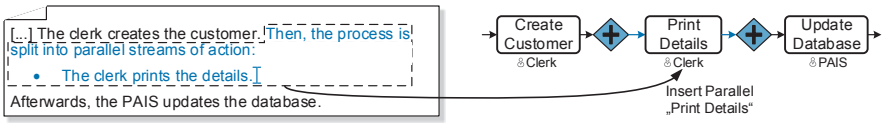


Figure 9.31: Inserting an Enumeration Block and Corresponding Process Model Update

this branch is empty. Activities may then be added by inserting a corresponding sentence describing the desired action.

Change the Object or Verb of a Sentence. When the user changes the verb or object of a sentence, he wants to adapt the activity described by the sentence. Mapping this to the corresponding process model requires the update of activity attributes. The updated sentence is then re-analyzed using the *Stanford Parser* and a verb or object is extracted (cf. Figure 9.30). For example, when changing sentence "The clerk prints the details" to "The clerk prints customer details.", this leads to the renaming of activity *Print Details* to *Print Customer Details* in the corresponding process model.

Inserting a Part to a Sentence. Inserting a new part, like "[...] provides information customer record" to an existing sentence, details the action described through this sentence. In terms of a process model, such information is captured in the data flow. Therefore, data elements or data edges may be inserted in the corresponding process model depending on whether or not the addressed data element already exists. Figure 9.32 shows an example in which the part "[...] requires the information customer record" is added to a sentence. In turn, this requires the addition of data element *Customer Record* as well as a corresponding reading data edge. Generally, the decision whether a reading or writing data edge is required, respective phrases like "provides information" and "requires information" are analyzed and the corresponding data edge is inserted. Note that the phrases are not predefined in a strict sense. Instead, we parse the inserted part of the sentence with the *Stanford Parser* and employ a set of signal verbs and nouns to detect the intention of the user. In order to express whether writing a data element is *optional* or *mandatory*, adjectives like "always" or "sometimes" may be used. If the user changes the adjectives later on, respective operations updating the data edge type will be triggered.

Figure 9.32: Inserting a Data Element and Corresponding Data Edge

Deleting a Sentence. Deleting a sentence removes the dedicated action from the verbalized process description. Accordingly, the respective activity will be deleted in the process model as well.

Deleting an Enumeration. Deleting an enumeration completely in a verbalized process description triggers update operations that delete the respective branching in the process model together

with all its activities. When deleting only one bullet point of an enumeration, the related branch is deleted in the process model.

Deleting a Part of a Sentence. When deleting the part of a sentence describing the data elements provided or required by the activity, the respective data edges have to be deleted in the process model. If no other activity accesses the associated data elements, these have to be deleted as well.

Changing the Subject of a Sentence. If the subject of a sentence is changed by the user, he wants to assign the action described by the sentence to another user. For this purpose, the process attribute describing the user assignment in the corresponding process model is changed. To detect this text modification, the *Stanford Parser* analyzes whether the subject is changed. Since further process attributes are not present, they cannot be updated.

Table 9.3 shows how the various change patterns can be mapped to adaptations of the verbalized process description. Note that synchronization edges are not explicitly supported since links between sentences would be hard to comprehend and maintain by users. Since the verbalized process description shall be intuitive to users, we therefore decided to exclude synchronization edges in this representation. Furthermore, it is not possible to insert and delete attributes to keep the textual description easy to understand. However, it is possible to change user assignments and activity labels by changing the subject or object/verb of a sentence.

Change Patterns	**Counterpart in Verbalized Process Description**
AP1 (Insert Process Fragment)	Inserting new sentences to the verbalized process description.
AP2 (Delete Process Fragment)	Deleting sentences in the generated verbalized process description.
AP3 (Move Process Fragment)	Moving respective sentences to another position.
AP4 (Replace Process Fragment)	Modifying or overwriting existing sentences.
AP5 (Swap Process Fragment)	Not applicable. Needs to apply AP3 (Move Process Fragment) twice.
AP6 (Extract Sub-Process)	Not applicable.
AP7 (Inline Sub-Process)	Not applicable.
AP8 (Embed PF in Loop)	Inserting an enumeration block comprising the sentences representing the process fragment to repeat.
AP9 (Parallelize Activity)	Inserting an enumeration block comprising the sentences (i.e., activities), which shall be parallelized.
AP10 (Embed PF in Conditional Branching)	Inserting an enumeration block comprising the sentences (i.e., activities), which shall be alternatively executed.
AP11 (Add Ctrl Dependency)	Not applicable.
AP12 (Remove Ctrl Dependency)	Not applicable.
AP13 (Update Condition)	Changing the introductory sentence of an enumeration block.
AP14 (Copy Process Fragment)	Duplicating respective sentences.

Table 9.3: View Update Operations and their Support in Verbalized Process Description

9.4.3 Discussion

A verbalized process description enables users with limited or no process modeling knowledge to understand process models. Furthermore, it enables them to realize process updates through corresponding text changes. In particular, the textual representation of a process models allows expressing both control and data flow.

However, the presented approach neither supports synchronization edges nor process attributes (except user assignment and activity labels). Hence, these aspects must be updated using an-

other process representation. Furthermore, note that the application of diagrams to visualize process models may be more precise than natural language [241, 242]. Due to the complexity and expressive power of natural language, the updates performed in the verbalized process description may be propagated in an undesired way, e.g., when writing subordinate clauses. In order to address this issue, predefined sentence templates may be used as known from user stories in agile software development (i.e., "As a [role] I want [something] so that [benefit]") [243].

9.5 Further Process Representations

Other process representations focusing on specific process aspects may be introduced to meet user needs. For example, in [11] we introduced a *Gantt-based representation* for visualizing time-aspects of process models. Figure 9.33 shows an example of this representation. Thereby, the Gantt-based representation visualizes min/max durations of activities as well as temporal relations between them [244]. In this context, we extended Gantt diagrams to visualize branchings as well (cf. Figure 9.33).

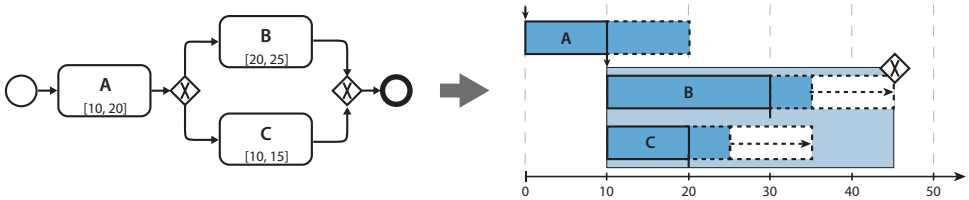


Figure 9.33: Gantt-based Process Representation

The modular design of the *proView* framework allows for the simple integration of additional process representations. Therefore, process representations should base on the definition of a process model to be applicable (cf. Definition 6.1).

9.6 Related Work

The work presented in this chapter is related to several streams of research. Accordingly, we split related work into approaches addressing process *visualization in general*, approaches related to *hierarchical representations*, *form-based representations*, and *verbalized process descriptions*.

Process Visualization in General. Several approaches for visualizing process models exist. In particular, there exist many established process modeling languages, e.g., BPMN 2.0 [87], EPC [203], *UML 2.0 Activity Diagrams* [161], or *Workflow Nets* [203]. However, the variety of modeling elements are overwhelming for users. Moreover, only a small subset of these elements are actually applied when modeling business processes [101, 207].

Few approaches focus on reducing the variety of process modeling elements. For example, S-BPM limits its process modeling language to five elements [148, 149]. Similarly, the *PICTURE* project suggests a domain-specific language for the public domain comprising a predefined set of process

building blocks [245], similar to the activity patterns presented in [246]. All approaches have a limited expressiveness and focus only on human-centric processes.

In [247], an approach based on *storyboards* and *storytelling* is suggested. Thereby, a comic strip-like visualization is chosen to visualize and create process models. However, this approach does not fit for complex business processes since the storyboards would get too big and complex.

By contrast, several approaches aim at supporting users to better understand their process models. The *niPRO* project provides an intelligent process portal using a multi-dimensional navigation through different process aspects [248, 249, 250, 206, 251, 252]. Particularly, a user is able to navigate through process models on different levels of abstraction. Next, [253, 254] introduce three dimensional process models utilizing the third dimension to visualize additional information. In turn, [255, 256] simulate business processes in virtual environments (e.g., a virtual hospital) to support users in understanding the tasks they need to perform in the context of a business process. More precisely, avatars play the role of individual process participants and show which activities shall be performed. None of these approaches supports process updates.

Regarding process execution, [257, 258] apply *sonification* techniques to make process models and their execution performance audible to users. In particular, users need not to learn any modeling language or to have an in-depth knowledge on process dashboards, but can *hear* whether or not a process performs well.

Hierarchical Representation. Similar to the hierarchical approach presented in this chapter, [259] introduces a mapping of CTT process models to UML 2.0 Activity Diagrams. Although UML provides a rich set of diagram types, none of them supports a model-based design of the behavior of user interfaces. To bridge this gap the authors introduce a mapping of CTT modeling elements to UML 2.0 Activity Diagrams. Although UML Activity Diagrams can be used for modeling business processes, the suggested mapping is more complex than ours. Furthermore, the resulting process model would be too complex for users since the number of elements are increased. A similar approach showing the same drawbacks is introduced in [260].

The approach presented in [261] uses BPMN 2.0 for process modeling. However, CTT is used to refine interactive, human-centric activities in BPMN process models. More precisely, the users first models the business process using BPMN 2.0. Then, activities requiring user interactions are refined similarly to the definition of a sub-process by defining a corresponding CTT.

Another CTT use case is presented in [262]. A method for modeling business processes is used based on the *Concurrent Task Tree Environment (CTTE)*—a modeling tool for CTT [221]. The resulting CTT is then transformed into an imperative process model, which may be edited using a standard process model editor (*Task Tree Workflow Management System Editor, TTMS Editor*). This way, the process model can be enriched with explicit choices and execution-relevant aspects. Finally, the process model is exported as XML file to a process engine (*TTMS WfMS*). This transformation cannot be reversed to keep models up-to-date. However, [262] neither provides an explicit mapping to an existing process modeling language nor does it support appropriate visualizing concepts for users.

In [235, 263, 264] another tree structure, denoted as (Refined) *Process Structure Tree (PST)*, is introduced. On one hand, this tree type is used to analyze a process model in respect to control flow errors; on the other, the PST structure is applied to detect SESE regions. The latter allows transforming certain classes of unstructured process models to well-structured ones

[265]. However, user needs are not addressed by the PST notion.

Form-based Representation. The presented form-based process representation relies on the *Nassi-Shneiderman-Diagram* [217]. Similarly, [266] suggests *structured* process modeling languages related to this type of diagram.

In turn, [267] presents an approach for visualizing process models as nested rectangles to get a dense representation. Each nested rectangle represents an individual SESE block in the corresponding process model. Activity labels are not displayed on a rectangle, but in a different area that appears when clicking on the rectangle. Colors symbolize the different types of SESE blocks (e.g., AND branching). However, the approach neither supports process updates nor a printable process documentation.

The *discovery map* provided by *IBMs Blueworks Live* [82] enables a minimalistic way to initially capture milestones and activities within a process in a table-based way. This visualization may be transformed to BPMN 2.0 and allows further specification of the control flow. However, it is limited to sequential activities which may grouped.

In the context of process execution, [2, 7, 227] present an approach to create user friendly user forms for complex process models. Thereby, directly preceding activities may be combined in one user form if they are performed by the same user (role) at the same time. Process updates are supported by dragging individual form fields or sub-forms.

Verbalized Process Description. [268] presents a study investigating the process of process modeling of users not familiar with process modeling languages. As a result, natural language is identified as an important alternative to describe process models. Particularly, structured text may reduce the cognitive load to achieve the transition from informal (i.e., verbal) to formal process descriptions (i.e., process models). The generation of texts in natural language has a long tradition and has been utilized in different application fields like weather forecasting [269] or task documentation [270]. In [271], natural language descriptions are generated based on object models. Furthermore, UML 2.0 Class Diagrams are then utilized to derive textual specifications from it [272]. None of these approaches tackles problems associated with process modeling.

As discussed, the verbalized process description is based on [231], which allows creating natural language descriptions from a given process model. However, it does not support change and evolution. Transforming graphical models to natural language has been investigated in other domains as well; e.g., [273] generates textual requirements descriptions, whereas [274] suggests natural language to define business rules.

Process models are generated based on textual descriptions of corresponding business processes in [275]. Particularly, the authors describe an unidirectional transformation and apply their approach for initially creating process models; process updates are not addressed.

A combination of graphical notations and textual descriptions provides a better comprehension [276]. The *proView* framework enables this by supporting various representations.

9.7 Summary

This chapter introduces three process representations that aim to visualize and update process models for end-users in an easy and intuitive. In particular, the representations support users

in understanding complex process models without requiring specific knowledge about process modeling elements. The presented process representations can be applied to CPM as well as to process views. As opposed to existing work, all representations support process updates to evolve process models.

The *hierarchical process representation* visualizes process models in terms of a tree-based structure. For this purpose we utilize CTT known from end-user development. Reading a CTT from the root to the leafs provides an increasing level of detail on the process model from level to level. As opposed to hierarchical modeling through sub-processes, this hierarchical representation keeps the context between super- and sub-processes. Furthermore, inserting new tasks to the tree of the hierarchical representation or changing relations between tasks, triggers updates in the corresponding process model.

The *form-based representation* visualizes a process model in terms of nested rectangles. Thereby, each rectangle represents an element of a process model (e.g., activity). This reduces the number of elements needed to visualize a process model. Instead, the control flow is given through the top-down ordering of the rectangles. Furthermore, users may add process elements by inserting new rectangles.

Verbalized process descriptions present process models as natural text to users. Although it increases the amount of text a user must read, it does not require any process modeling knowledge. The target group of this representation are users having no process modeling knowledge. Changing a verbalized process description, in turn, can be accomplished through text changes.

Table 9.4 emphasizes the strengths and weaknesses of the three process representations. Opposed to current process modeling approaches, *proView* allow users to dynamically switch between process representations. This enables an appropriate visualization of process models. Finally, updates performed on any of the representations are propagated to the corresponding process model as well.

Hierarchical Representation	
Strengths:	Weaknesses:
• Top-down visualization of process models (i.e., modularization)	• Increased number of model elements due to tree structure
• Enables exploration of process models	• No visualization of data flow and process attributes
• Complex updates can be accomplished through simple modifications of sub-trees	
Form-based Representation	
Strengths:	Weaknesses:
• Decreased number of process elements	• Implicit visualization of data flow as well as process attributes
• Control flow correctness supported due to form-based structure	
Verbalized Process Description	
Strengths:	Weaknesses:
• No specific process modeling knowledge required	• High expressiveness of natural language when applying updates
• Enables imperative formulation of work for users	• Might result in long process descriptions

Table 9.4: Strengths & Weaknesses of Introduced Process Representations

10 Process Interaction

10.1 Introduction

Process models and their optimization are often discussed in the scope of interviews and workshops [45]. Identified optimizations are, typically, annotated on printed process models or documented on paper [47] to lastly transfer them to process modeling tools manually. Due to the media disruption, the latter is a time-consuming and error-prone task.

Mobile devices may support capturing business processes while interviewing process participants [277]. Particularly, mobile devices may increase user productivity [278, 279]. In turn, touch-enabled devices with larger screens (e.g., touch tables) allow for workshops among domain experts to collaboratively create and evolve process models [75, 280, 281]. Consequently, touch-enabled devices offer promising perspectives in respect to process modeling. However, traditional process modeling tools have not been designed with touch-enabled devices in mind. Hence, they do not take their specific properties (e.g., small screen size) and interaction capabilities (e.g., gesture-based interaction) into account [46, 282, 283].

Process views and representations might be leveraged to address limited screen sizes of touch-enabled devices. Regarding interaction support for process modeling, this chapter introduces a well-designed set of process interaction gestures that allow creating and evolving process models on touch-enabled devices. The gesture set was derived in a user study we conducted to identify gestures perceived as intuitive by users. The gesture set is applicable to all screen sizes of touch-enabled devices. Overall, as will be shown in this chapter, gesture-based modeling complements the thesis with sophisticated concepts for intuitively interacting with process models and views.

Section 10.2 introduces fundamentals on gesture-based user interaction. Section 10.3 presents an experiment identifying appropriate and intuitive multi-touch gestures for process modeling tasks. Section 10.4 then introduces a multi-touch gesture set for modeling, visualizing, and evolving process models. Section 10.5 discusses how the gesture set can be applied in the context of

collaborative process modeling and touchless interaction. Section 10.6 discusses results. Finally, Section 10.7 presents related work and Section 10.8 concludes the chapter.

10.2 User Interaction Fundamentals

User interaction refers to the interaction between humans on the one side and machines or information systems on the other. Thereby, a user may interact with an information system using *touch-based* or *touchless* interaction. *Touch-based interaction* applies sensor technologies that need to be touched to transmit information. Therefore, touch-sensitive screens are required as present in smartphones or tablets [284]. *Touchless interaction*, in turn, utilizes sensor technologies, which do not require a touch-based interaction, e.g., an automatic door that opens when someone approaches it. Examples of touchless interactions include gestures based on eye movement [285, 286], hand, body movements [287], or muscle contraction [288]. Another example of touchless interaction is speech recognition [289]. In general, sensors for touchless interactions might be attached to the human body (e.g., gloves for hand interaction) or observe body movements (e.g., through cameras).

Independent of the applied sensor technology, users may interact in different ways when using respective multi-touch applications. First, concepts known from traditional desktop applications can be applied in the context of multi-touch applications as well; e.g., *menu-based interactions* are based on menus and toolbars to provide available functions to users. As an advantage, this concept allows users to easily *explore* the application when searching for a specific function. However, displaying menus and toolbars requires space on the screen, which is limited on small devices (e.g., smartphones) [290]. Hence, menu-based interaction should primarily be used for multi-touch applications running on larger screens. Besides this, selecting menu items might be challenging on a small screen when covering them with fingers or hand [291].

Gesture-based interaction, in turn, relies on gestures for selecting functions of the multi-touch application as well as for interacting with them. Thereby, a (multi-touch) *gesture* constitutes a movement of the human body involving arms, hands, head, or body [292]. This movement is then recognized and interpreted by the respective application. For example, in a slideshow of pictures, the *wipe gesture* (i.e., wiping with one finger from right to left) can be used to switch to the next picture. In general, gestures can be categorized into *physical* and *symbolic gestures* [293]. *Physical gestures* directly manipulate virtual objects on the screen, e.g., by dragging a virtual element on the screen. In turn, *symbolic gestures* are related to the function that shall be executed (e.g., drawing a plus to add an object). As opposed to menu-based interactions, gestures do not require any space on the screen in order to display menus or other graphical elements. However, as a barrier, users do not always know which functions are supported by a multi-touch application and which gestures shall be applied to select them. Especially, the latter is crucial for novices and unexperienced users of the multi-touch application. Usually, this problem is addressed through a tutorial provided the first time the user starts the application.

Finally, a *hybrid interaction* based on both menu- and gesture-based interactions can be realized. For example, a menu bar may be displayed at the top or bottom of the screen, offering the most important functions. Additionally, well-known gestures may be supported, e.g., the aforementioned wipe gesture.

10.2.1 Characteristics of Multi-Touch Applications

In addition to their interaction concepts, multi-touch applications and their multi-touch capabilities should consider at least the following characteristics in the context of process modeling [1].

C1 (Screen Occlusion). The user interface design of a multi-touch application must consider that users directly interacting with the application may cover screen areas with their hand [294, 295]. For example, dragging an object from the top-left corner of a tablet to its bottom-right corner will be a difficult task, if large parts of the screen are covered by the user's hand.

C2 (Handling Precision). Conventional desktop applications require a mouse pointer to interact with them. This pointer scales with the screen resolution in order to ensure a high precision of the interactions. In multi-touch applications, however, the "interaction device" is the user's finger [296]. As opposed to the mouse pointer, a finger neither scales up nor down when changing screen sizes. Accordingly, the handling precision of a multi-touch application changes with the size of an object the user wants to touch. Studies have shown that an object should have a size of 11.52 mm or more to have a hit probability of at least 95% [297]. Regarding multi-touch process modeling a high handling precision is indispensable.

C3 (Fatigue of Extremities). When using a computer mouse, usually, the user's hand does not move a lot. Usually, the arm lays on the table and the mouse is moved with the fingers to reach the corners of the screen with the mouse pointer. Interacting with multi-touch applications, in turn, frequently requires the movement of the entire arm. Furthermore, the latter does not lay on a table, but has to be hovered in the air during the interaction with the multi-touch application [298, 299]. Obviously, the degree of movement strongly depends on the screen size.

C4 (Number of Supported Fingers). Compared to conventional desktop applications with one single mouse pointer, a multi-touch application typically supports more than one touch point. Generally, the maximum number of touch points an application supports is limited through the underlying sensor technology, operation system, and screen size. For example, Apples iPad distinguishes between 11 touch points (i.e., fingers) [300]. While this number is sufficient for single user applications, multi-user applications should be able to support even more touch points, e.g., to enable concurrent process model updates on a touch table.

C5 (Number of Concurrent Users). Multi-touch applications may be concurrently used by multiple users. Obviously, the degree of concurrency is limited by screen size. While concurrent interactions with a tablet would effect each other, the concurrent use of an application on a touch table (e.g., joint editing or modeling) might improve the way of working collaboratively [301]. Especially, in the context of concurrent process modeling this should be exploited [302].

C6 (Usage of Common Interaction Concepts). As known from conventional desktop applications, there are interaction concepts representing de-facto standards. Examples include the menu bar on the top of the application window or *File* as first entry of this menu bar. Such recurrent patterns help users to intuitively interact with various applications. The same applies to multi-touch applications and especially gesture-based interactions; e.g., the *pinch gesture* is typically used for zooming. However, there are only few gestures that have been used in a consistent

manner so far. Therefore, [303] suggests a gesture dictionary with the purpose that different applications show similar behavior in connection with a specific gesture.

C7 (Intuitive Gestures). Gestures used in the context of multi-touch applications should be as intuitive as possible (cf. Requirement REQ-9). This supports the user in memorizing and applying them. Regarding a simple gesture, studies have shown that eight seconds are required to plan and perform it. In turn, complex gestures require per average 15 seconds [293]. Generally, the simpler a gesture is, the more intuitive its use will be [293].

The introduced characteristics of multi-touch applications show their wide range of possible applications. Obviously, these characteristics should be taken into account when designing gestures for process modeling tools.

10.3 Experiment on Multi-Touch Gestures

In order to identify multi-touch gestures, which are appropriate to interact with process models, experimental research is conducted. Experimental research on various BPM issues has already shown promising results regarding user-centric process support [304, 305, 306, 307].

In this thesis, we conducted two user experiments related to touch-based process modeling. The goal of these experiments is to derive a proper multi-touch gesture set [1, 13, 308, 309]. *Experiment 1* focuses on the creation of process models, whereas *Experiment 2* additionally targets at gestures for view creation and tool interactions (e.g., undo actions, selecting elements). Using the *Goal Definition Template* (cf. [310]), the experiments are defined as follows (cf. Table 10.1):

Analyze	*multi-touch process interaction*
for the purpose of	*evaluating*
with respect to their	*intuitive usage*
from the point of view of	*the researchers and developers*
in the context of	*students and research assistants.*

Table 10.1: Goal Definition Template

Taking this goal definition, the experiments evaluate multi-touch interactions applied by users (i.e., by students and research assistants) when interacting with process models on touch-enabled devices. The results of these experiments are then used to develop a consistent set of interaction and modeling gestures covering view creation and process updates as introduced (cf. Chapter 6 and Chapter 7). Based on this goal definition, the following hypothesis is derived:

Users intuitively use gesture-based interaction when interacting with process models on touch-enabled devices.

In the following, Section 10.3.1 describes the setting of the experiments. Then, Section 10.3.2 presents and discusses experiment results.

10.3.1 Experiment Setting

Both experiments have the same setting, but differ in the objects (i.e., process models) and number of tasks [308, 309]. The experiments are designed as a *single object study* [310], i.e., having one group of subjects and one object to evaluate. Furthermore, each experiment is divided into two parts. In the first part, demographic and background information of the subjects are collected (i.e., gender, experience with touch-enabled devices, profession, and handedness). In the second part, each subject has to modify an existing process model (cf. Figures D.1 and D.2 in Appendix D) in 8 (i.e., Experiment 1) or 14 (i.e., Experiment 2) steps. Each step includes a description explaining the subjects what they have to do to perform the respective process modeling task on a given process model. For example, task "Think of a way to create a new activity between *Make Up Package* and *XOR Branch*" corresponds to inserting an activity serially.

	Task	**Experiment 1**	**Experiment 2**
T1	(Insert Activity)	Think of a way to create a new activity between *Make Up Package* and the *XOR Branch*.	Insert an activity between activities *1st Review* and *2nd Review*.
T2	(Rename Activity)	Label the new activity with *Print Invoice*.	Rename activity *2nd Review*.
T3	(Insert Branching)	Insert an AND branching surrounding the XOR block and activity *Send E-Mail*.	Insert an AND branching, which includes activity *Fill Out Credit Request*.
T4	(Insert Branch)	Insert an empty branch between the two existing XOR gateways.	Insert an XOR branch to the first XOR branching.
T5	(Insert Data Element)	Add a new data element *Shipping Number* which is written by activities UPS Shipping and read by *Send E-Mail*.	Insert a data element.
T6	(Insert Data Edge)	Find a way to connect data element *OrderID* to activity *Print Invoice* through a reading data edge.	Insert a reading data edge between data element *CustomerPhone* and activity *Load Customer Data*.
T7	(Delete Activity)	Delete activity *Print Invoice*.	Delete activity *2nd Review*.
T8	(Insert Sync Edge)	-	Insert a synchronization edge between *Calculate Risk* and *Check Credit Protection Agency*.
T9	(Aggregate Activities)	Aggregate the XOR block and activity *Send E-Mail* to a sub-process.	Combine activity *Fill Out Credit Request* and the first XOR branching to a sub-process.
T10	(Reduce Activity)	-	Hide activity *Calculate and Check*.
T11	(Create Process View)	-	Create a process view consisting of activity *Fill Out Credit Request* as well as the first XOR branching.
T12	(Select Activity)	-	Select activity *Load Customer Data*.
T13	(Undo Action)	-	Undo the last action performed.
T14	(Open Help)	-	Open the *help* dialog.

Table 10.2: Task Descriptions of Experiments 1 & 2

The *response variable* of the experiments is the *gesture* performed by the subject. In particular, the gesture is recorded through the trace of the respective fingers on the screen. Based on such a trace, in turn, the *interaction category* used to perform the required change of the process model is identified. For each task, therefore, the interaction category has one of the following values: *picture*, *gesture*, or *menu-based (process) interaction*.

As *instrumentation* of Experiment 1, an Apple iPad and the multi-touch drawing application Doodle Buddy [311] are used. Experiment 2 applies a web application on a Google Nexus 7, which is developed for this experiment setting. In both experiments, for each task the application displays an image to the subject. This image depicts a process model serving as basis for

the respective task. In this context, the aforementioned textual explanation is displayed (cf. Table 10.2). Finger movements of subjects are captured through overlays on the image (cf. Figure 10.2 of Experiment 1).

Additionally, the gesture is captured through a video camera. The latter records the screen of the device as well as the hands of the subjects and their movements. Furthermore, subjects are asked to *think aloud* about what they are doing and what they are thinking about [312]. This information is used for classifying and interpreting results afterwards. Furthermore, the supervisor of the experiment stays in the same room as subjects, motivates them to *think aloud*, and provides assistance in case of emerging questions. Figure 10.1 gives an overview of the instrumentation applied.

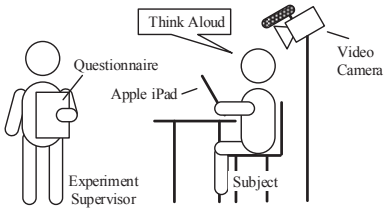


Figure 10.1: Experiment Instrumentation

10.3.2 Experiment Analysis and Results

In total, 37 subjects attended the experiments (Experiment 1: 26; Experiment 2: 11). Table 10.3 summarizes background information.

	Gender	**Profession**	**Handedness**	**Multi-Touch Experience**
37 Subjects (26/11 in Experiment 1/2)	29 Male 8 Female	26 Students 11 Res. Assist.	35 Right 2 Left	25 Yes 12 No

Table 10.3: Subjects' Background

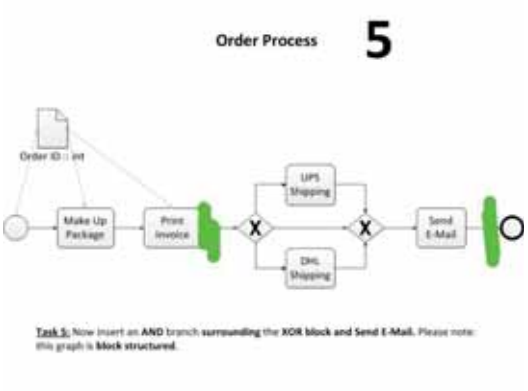
We classify the interactions a subject applies in the context of a concrete modeling step into three *interaction categories*. Note that this classification is accomplished by two persons; i.e., it constitutes a subjective measurement. As underlying basis, we choose the trace overlaying the image of the respective modeling step as well as the video capturing the actual movement of the subject's hand.

The first category groups *gesture-based interactions*. This kind of interaction uses simple movements on the multi-touch screen changing the process model (as physical gestures described in Section 10.2). Figure 10.2a exemplarily shows a gesture-based interaction, which is applied to insert an XOR block, which surrounds a given process fragment, in a process model. In this case, the subject moves two fingers simultaneously up and down to insert the surrounding block.

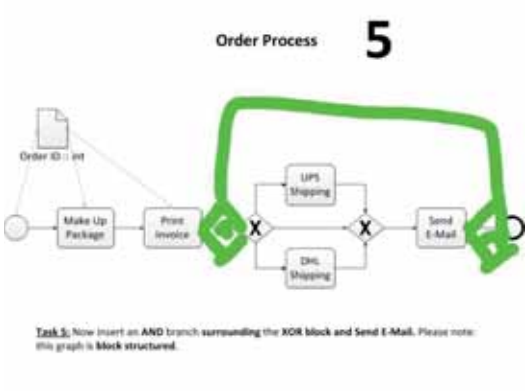
The second category comprises *picture-based interactions*, i.e., all interactions drawing a realistic representation of the final result expected from the application of the respective modeling function (as symbolic gestures, cf. Section 10.2). Figure 10.2b shows an example of this category.

The third category captures all interactions presuming the presence of a menu bar or context menu. Figure 10.2c exemplarily shows the result of a subject who is drawing a toolbar at the top

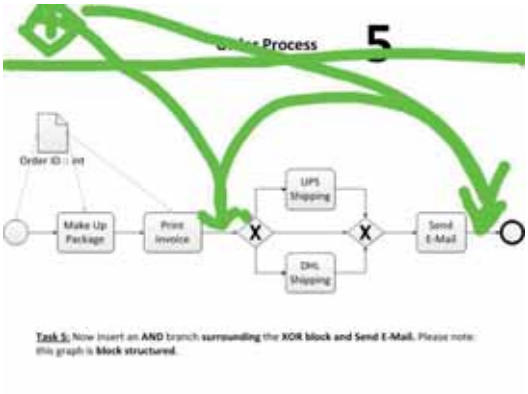
of the screen and is dragging & dropping the required process elements on the depicted process model. In general, this category covers *menu-based interactions*.



(a) Gesture-based Interaction



(b) Picture-based Interaction



(c) Menu-based Interaction

Figure 10.2: Interaction Categories of the Experiment

Figure 10.3 summarizes the results of the experiment and visualizes the distribution of the interaction categories. On the x-axis, each step of the experiment is represented through its corresponding task. In turn, the y-axis shows the number of subjects (in %) using interactions from a specific category. The raw data of the experiment can be found in Appendix D.

Obviously, there is no predominant interaction category covering all process modeling and interaction tasks. However, for certain tasks one can observe a clear preference towards a specific interaction concept. Task *T9 (Aggregate Activities)*, for example, is applied by 64% of the subjects using gesture-based interaction. In turn, for accomplishing task *T12 (Select Activity)* 100% of the subjects use gesture-based interaction. For other tasks (e.g., *T6 (Insert Data Edge)*), however, no dominating interaction category can be identified. Hence, multiple interaction concepts should be provided to optimally support different user groups in applying respective functions. In total, 47.8% of the subjects use a gesture-based interaction. Therefore, the latter predominates the other interaction categories.

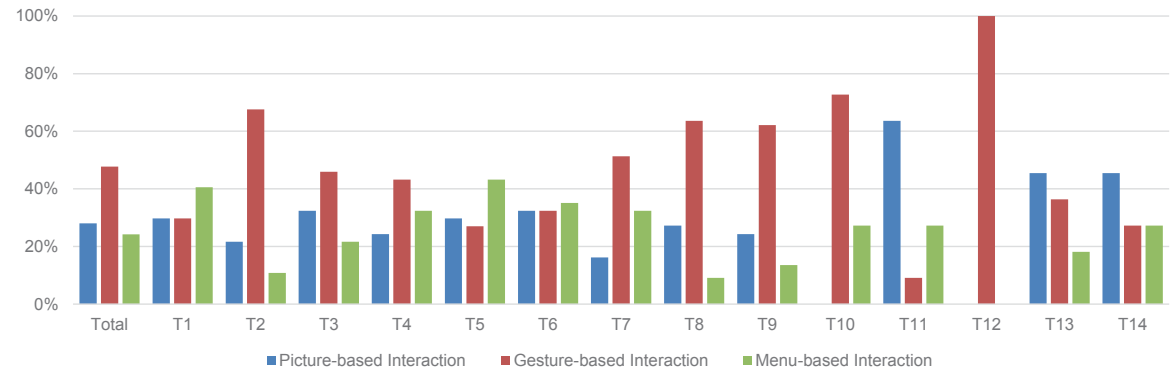


Figure 10.3: Distribution of Interaction Categories

In the following, we discuss recorded gestures in detail. Note that we analyzed recorded experiment videos as well as gesture traces to understand what subjects intend to do. Based on this discussion, we extract multi-touch gestures in Section 10.4.

T1 (Insert Activity). Inserting an activity to a process model is a frequently applied operation when creating process models. 11 subjects apply picture-based interaction and most of them draw rectangles on the respective control edge. In turn, gesture-based interaction is applied by 11 subjects. Thereby, subjects apply long tab or vertical swipe gestures on the respective control edge. Finally, 15 subjects use (context) menus to insert the activity. However, these menus are invoked in different ways: tapping, drawing lines, or pinching on the respective control edge. Altogether, all subjects interact with the control edge to insert the activity and not with the preceding or succeeding activity.

T2 (Rename Activity). Renaming an existing activity is performed by 26 subjects who utilize an on-screen keyboard. 11 subjects use their hand-writing to rename the activity. However, the way the on-screen keyboard or hand-writing window is triggered differs. Four subjects use a menu-based interaction by triggering a context menu. Eight subjects apply a picture-based interaction by drawing a window to write the activity label. Finally, 25 subjects apply a gesture to trigger an on-screen keyboard. Thereby, 11 of the 25 subjects use tab gestures. The remaining 14 subjects apply other gestures like double and long tab.

T3 (Insert Branching). Inserting a branching block, which surrounds a set of activities, is performed by 14 subjects using menu-based interaction. Thereby, they tab on a control edge and expect to open a context menu. Next, 13 subjects apply a picture-based interaction by drawing diamonds representing required gateways on the respective control edges. Afterwards, subjects draw a control edge connecting the gateways. Finally, ten subjects use gesture-based interaction; e.g., they draw two vertical lines on the control edge to insert split/join gateways.

T4 (Insert Branch). Nine subjects insert a branch to an existing branching by drawing a line between the corresponding split and join gateways (i.e., picture-based interaction). 12 subjects use a menu-based interaction by tabbing on the corresponding split gateway. Finally, 16 subjects perform a gesture-based interaction to connect on the corresponding split and join gateways.

T5 (Insert Data Element). Data elements are inserted by six subjects using a gesture-based interaction (e.g., tabbing on the background). 12 subjects apply picture-based interactions by drawing rectangles above the process model. Finally, 19 subjects prefer a menu-based interaction.

Respective menus are either located on the top of the screen (like a toolbar) or represented as a context menu.

T6 (Insert Data Edge). Four subjects use menu-based interaction to insert data edges connecting activities and data elements. In turn, ten subjects prefer picture-based interaction, i.e., lines with an arrow head are drawn. 23 subjects connect the data element and activity with a line (i.e., gesture-based interaction).

T7 (Delete Activity). Task T7 shall delete an existing activity from a process model. Six subjects drag the activity to a trash basket or out off the screen (i.e., picture-based interaction). In turn, 12 subjects delete the activity by using a context menu that is triggered by tabbing on the activity (i.e., menu-based interaction). The majority of subjects (i.e., 19 subjects) apply a gesture-based interaction by drawing a cross or a strike through line to delete the activity.

T8 (Insert Sync Edge). Inserting a synchronization edge between activities from parallel branches is considered in Experiment 2. One subject uses a context menu to insert the synchronization edge required. Three subjects apply a picture-based interaction and draw a dotted line between the two activities. Finally, seven subjects perform a drag and drop gesture to connect both activities (i.e., gesture-based interaction).

T9 (Aggregate Activities). Users shall enable a set of activities in a sub-process activity. Five subjects presume a context menu to aggregate respective activities. In turn, nine subjects apply a picture-based interaction and draw a circle or rectangle around the activities to be aggregated. Finally, 23 subjects use a gesture-based interaction. Most of them apply a pinch gesture on the "first" and "last" activity, i.e., they move fingers towards each other.

Note that tasks T10-T14 are only evaluated by Experiment 2.

T10 (Reduce Activity). Subjects are requested to hide an activity in the process model. Three subjects trigger a context menu. None of them applies a picture-based interaction. Eight subjects perform a gesture-based interaction applying a pinch gesture to the respective activity.

T11 (Create Process View). Task T11 is performed by one subject through a gesture-based interaction. Three subjects apply a menu-based interaction. Thereby, respective activities are selected through a tab gesture, afterwards a context menu is invoked. Seven subjects draw a rectangle around activities to be included in the process view (i.e., picture-based interaction).

T12 (Select Activity). Selecting an activity is performed by all subjects using a tab gesture (i.e., gesture-based interaction). An explanation might be that selections are performed in almost all mobile applications (and operation systems) through tabs [303].

T13 (Undo Action). In task T13, subjects shall perform a gesture to undo the last action performed. In turn, two subjects use a menu-based interaction. Furthermore, four subjects apply a swipe gesture either from left to right or vice versa (i.e., gesture-based interaction). Five subjects prefer a picture-based interaction, drawing a circle in counterclockwise direction or an arrow pointing to the left.

T14 (Open Help). Subjects shall open a help dialog through a gesture. Three subjects press a button on an imaginary menu bar on the top of the screen. Furthermore, three subjects swipe from the left or top border of the screen to make the help dialog visible (i.e., gesture-based interaction). Five subjects prefer a picture-based interaction and draw respective symbols on the background (e.g., "H" or "?").

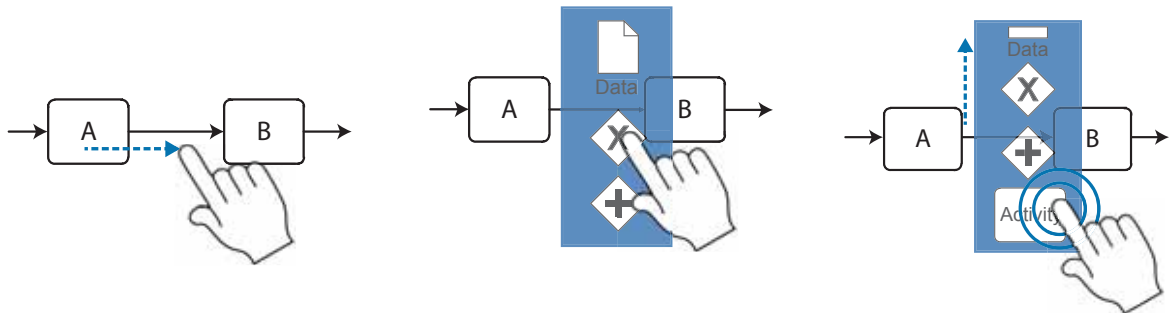
The results of the experiments are only representative to a certain degree due to the rather low number of subjects (especially for tasks T8, T10-T14) and the chosen experiment setting.
In the following, we focus on end-users being experienced with touch-enabled devices, and propose a core gesture set enabling intuitive process modeling.

10.4 Multi-Touch Gesture Set for Process Interaction

Taking the results of the experiments into account, we propose a multi-touch gesture set for creating and updating process models. In this context, we have to cope with the trade-off between commonly known interaction concepts provided by touch-enabled devices and results of the two experiments. The goal is to provide an appropriate gesture set suited for all kinds of interactions with process models on touch-enabled devices.

10.4.1 Gestures to Update Process Models

The experiments have shown that users tend to prefer a menu-based interaction when inserting elements into a process model (cf. Figure 10.3). Therefore, the suggested gesture set includes a slider menu enabling the insertion of activities, data elements, and surrounding branching blocks (cf. tasks T1, T3, and T5). To trigger the slider menu, users wipe with their finger from an existing process element from left to right (cf. Figure 10.4a). Following this, a slider menu appears at the position where the user releases his finger from the surface of the device. An alternative design choice may be the application of a pie menu as suggested in [294].



(a) Wipe to Right (b) Scroll the Menu (c) Tap on Element

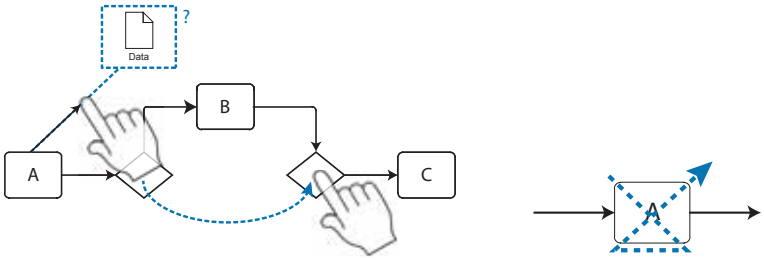
Figure 10.4: Gesture for Inserting New Elements

Through up and down movements in the slider menu, the required process element can be chosen. In particular, the top-down ordering reduces occlusion through user's hands (cf. Section 10.2.1). Finally, to insert a process element, the user taps on the respective icon in the slider menu (cf. Figure 10.4c). Afterwards, the slider menu disappears and the element is inserted. Obviously, when inserting a branching surrounding an existing process fragment, a second position needs to be chosen for adding the corresponding join gateway. For this purpose, the process modeling tool shows valid positions in the process model where the join gateway may be inserted. The user then chooses one of these positions by tapping on it. To provide more sophisticated user

support when inserting a branching, surrounding an existing process fragment, abstractions in terms of *process views* are useful, i.e., abstracting the process model to a simpler one that only comprises those activities relevant for the insertion of a join gateway.

The ordering of process elements depicted in the slider menu can be optimized by considering their relevance, i.e., frequency of their use. For example, inserting an activity might be more often required compared to the insertion of a surrounding branching block. Therefore, the respective process element should be positioned on a "central" position in the slider menu that can be quickly accessed.

Another multi-touch gesture allows inserting edges into a process model (cf. Figure 10.5a). In the context of well-structured process models (cf. Section 6.2), only three types of edges need to be "manually" added: data edges, synchronization edges, and empty branches, i.e., edges connecting split and join gateways. To insert an edge, the user has to draw a line with his finger, e.g., from the split to the join gateway in order to insert an "empty" branch between them (i.e., task T4) [313]. The gesture matches with the results of our experiments.



(a) Insert Edges (b) Delete Element

Figure 10.5: Gesture to Insert Data Edges and Delete Elements

To insert a data edge, the user has to draw a line between an activity and a respective data element or vice versa (i.e., task T6). The direction of the data edge indicates whether it represents a read or write access of the activity on the respective data element. When inserting a data edge, the gesture set supports users by suggesting a target element (cf. Figure 10.5a on the left). If the target element is the desired one, the user lifts his finger and the edge is automatically completed and inserted. The same behaviour is applied when inserting a synchronization edge between two activities of parallel branches (i.e., task T8). Note that this gesture is directly derived from experiment results—for tasks T4, T6, and T8 the majority of participants prefer gesture-based interactions.

Though 67% of the subjects use gesture-based interaction to rename a process element (i.e., task T2), most mobile operation systems suggest an on-screen keyboard for text input. The keyboard is hidden most of the time to save screen space, i.e., it is displayed solely in case a user wants to add or modify text. To comply with common interaction (de-facto) standards on touch-enabled devices, in the provided gesture set, renaming a process element can be activated by tapping on it. Afterwards, the keyboard appears and the user is able to modify the text. After confirming the text change, the keyboard disappears.

Finally, the majority of subjects delete process elements using gesture-based interactions. For realizing this function, a gesture crossing out the element to be deleted is suggested (cf. Figure 10.5b). Generally, different variants of this gesture can be envisioned (e.g., drawing two crossing lines or one line from bottom-left to top-right). In the provided gesture set, the user draws a cross on the respective element without lifting his finger. Generally, this is more accurate regarding recognition compared to the drawing of two separate lines.

10.4.2 Gestures for Creating Process Views

Generally, a multi-touch gesture is required to create a process view (i.e., task T11). 64% of the subjects prefer a picture-based interaction to create a process view. Most of them draw a rectangle (or a circle) around the respective process fragment to be included in the process view. However, with this gesture only directly connected activities can be included, i.e., activities forming a SESE block. In order to enable the creation of a process view with any activity set, a two-handed gesture is suggested. Initially, the user tabs with his first finger on an empty area on the screen. While holding this finger down, respective activities are selected with a second finger by tabbing on them (cf. Figure 10.6). Releasing the first finger, triggers the view creation based on selected activities. Obviously, this gesture is not "intuitive" to users (i.e., users do not intuitively pick such a gesture). Hence, it needs an initial training.

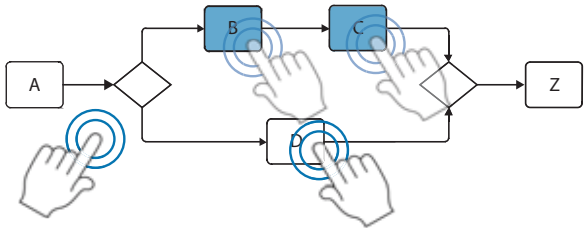


Figure 10.6: Creating a New Process View

To aggregate a set of activities to a sub-process (i.e., task T9), a two-handed gesture is introduced. The user pushes the start and end element of the process fragment towards each other (cf. Figure 10.7). While doing this, the movement needs to be animated to give direct feedback to the user. After completing the gesture, a sub-process activity is inserted at the respective position. The resulting sub-process can then be displayed through a double tap gesture. In turn, to inline a sub-process the user pulls the sides of the sub-process activity apart. When doing this, the sub-process appears and is re-inlined in the process model. Note that this gesture matches with the results of the experiments—62% of the subjects use gesture-based interaction in the context of task T9.

Reducing an activity (i.e., task T10), in turn, is performed by 73% of the subjects through a gesture-based interaction. From the perspective of a user, reducing an activity might be the same than deleting it, i.e., in both cases the activity "disappears" in the process model. However, in the context of the suggested gesture set, a gesture different from the one used for deleting is

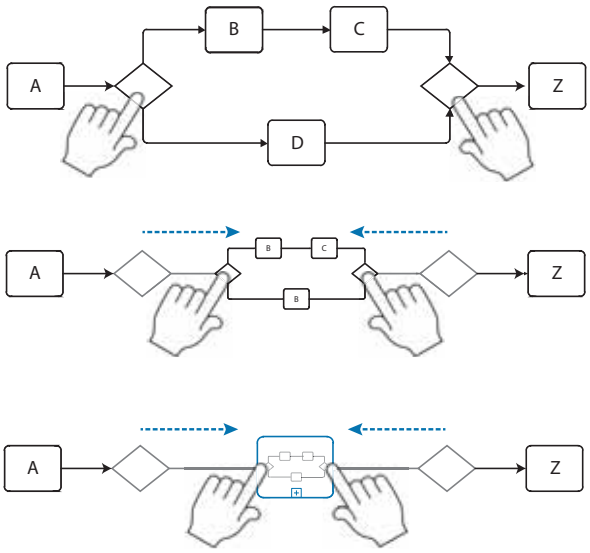


Figure 10.7: Aggregate Activities

preferred. Hence, the user may draw a rectangle around the respective activity and—without lifting his finger—a line through that rectangle (cf. Figure 10.8). As an alternative to drawing a rectangle, drawing a circle may be allowed.

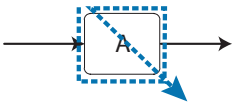


Figure 10.8: Reduce Activities

10.4.3 Gestures Triggering Supportive Functions

All subjects have chosen a gesture-based interaction to select a process element (i.e., task T12). Note that this corresponds with general gesture sets available on mobile devices (cf. Section 10.2). Hence, we suggest a tab gesture to select a process element. Instead of a single tab on the process element, a long tab or double tab might be useful to avoid a wrong detection of the gesture. [314] suggests a tactile feedback when selecting an item. However, tactile feedback is currently not supported by respective devices.

To undo the action performed last (i.e., task T13), a majority of 45% of the subjects prefer a picture-based interaction. In the provided gesture set, an arrow pointing back is suggested. To be more precise, the user draws a line pointing to the left as well as a hook at the end of the line to the top-right (cf. Figure 10.9). Drawing just the upper part of an arrow head reduces the duration of the gesture and increases the accuracy of recognition. Generally, it is desired to draw the arrow on an empty area in the editor to avoid false recognition.

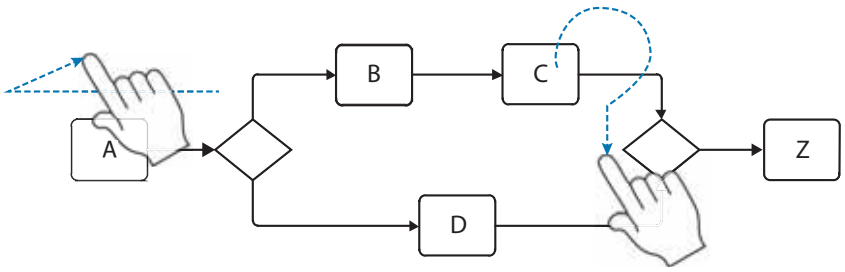


Figure 10.9: Gesture to Undo and Open Help Dialog

Finally, task T14 intends to identify a gesture to open a help dialog. As for undo, 45% of the subjects preferred a picture-based interaction. Therefore, we suggest a gesture that draws a question mark on an empty area. If the start of the gesture is located on a process element, the help dialog may provide context-sensitive information, e.g., in Figure 10.9 the help dialog could provide information about activity C and available modeling operations (e.g., delete, reduce).

10.4.4 Summary

This section introduces a multi-touch gesture set to interact with process models. The developed gesture set is based on two user experiments in order to determine how users intuitively interact with process models. The gesture set may be used in any process modeling tool. However, when applying the gesture set to other process representations (cf. Chapter 9) it must be evaluated whether the gestures are still appropriate.

10.5 Further Process Interaction Concepts

In the following, we discuss further process interaction concepts required for touchless interaction and collaborative process modeling.

10.5.1 Touchless Interaction

In certain situations, a touch-based interaction limits users when interacting with process models, e.g., when standing in front of a projector screen in a meeting room. In such a scenario, a user wants to interact directly with the projector screen instead of additionally carrying a touch-enabled device. Moreover, the maturity of touchless interaction technologies increased in the past (e.g., leap motion [315]) and might be used to equip meeting rooms [316].

Section 10.2 discusses various technologies enabling touchless interaction (e.g., Microsoft Kinect). In a business environment it might be more appropriate to select a sensor that needs be attached to the human body to decrease usage burdens. Therefore, we presume *bare hand gestures*, i.e., remote sensors supervise hand movements [317]. We omit steps required to detect bare-hand gestures (cf. [287, 318, 319]). In the following, we describe the application of the multi-touch gesture set of bare-hand gestures for process interaction.

Sensor Location. Providing an environment for recognizing hand gestures, a (monitor or projection) *screen* as well as a *sensor* are required. The sensor detects hand gestures (e.g., Microsoft Kinect [318]) and may be placed *orthogonally* or in *parallel* to the respective screen. An *orthogonal sensor location* enables the recognition of hands and fingers held orthogonally to the screen, i.e., the user points with his hand towards the screen (cf. Figure 10.10a). In case of a *parallel sensor location*, the user has to lift his hand in order to allow the sensor to detect his finger movements, i.e., hands are positioned in parallel to the screen (cf. Figure 10.10b).

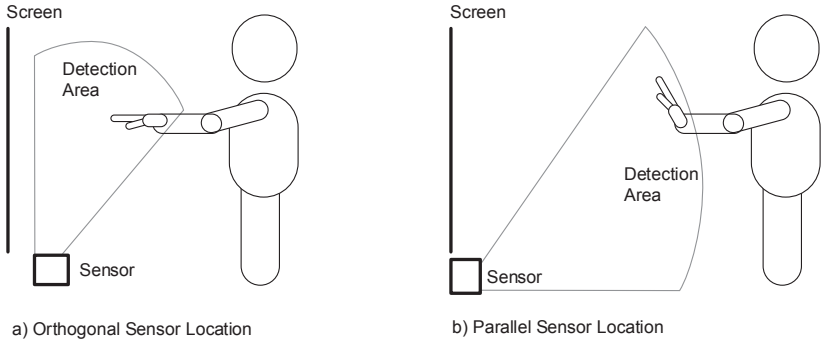


Figure 10.10: Sensor Location

Regarding process model creation and interaction, we suggest an orthogonal sensor location. Hence, the user is able to point towards the screen and process elements respectively.

Finger Pointing vs. Finger Position. When using hand gestures, it is useful to provide a reference for the user's hand position on the (projection or monitor) screen. Therefore, a pointer (comparable to a mouse pointer) should be displayed. However, calculating the position of the latter is not straightforward. A *virtual box* is specified, which is limited by the range of user's limbs (i.e., arm) or the sensor detection area. Thereby, this virtual box differs in position and size from the *real screen* (i.e., projector or monitor screen). If the user points on the *virtual screen* (i.e., an imaginary screen within the virtual box) with his hand, the position on the real screen must be determined. For such an interpretation two techniques exist: *finger pointing* or *finger position*. *Finger Position* takes x- and y-axis position of user's hand (or finger) on the virtual screen as well as the virtual screen size, and scales it to the real screen size (cf. Figure 10.11). If the user points to the center of the virtual screen, for example, he points to the center of the real screen as well.

Finger Pointing not only takes x- and y-axis positions into account, but also the angle α and the distance between the virtual and the real screen. Angle α expresses the angle of the hand in respect to the virtual screen. Based on this information and the *Pythagorean theorem* the point on the real screen, the user points to is calculated (cf. Figure 10.11).

Generally, finger pointing is more convenient for users, since all points on the real screen can be reached with minimal movements of the wrist (cf. *C3 (Fatigue of Extremities)* in Section 10.2.1). However, it requires a precise detection of the hand and its movements to enable a "smooth" pointing experience.

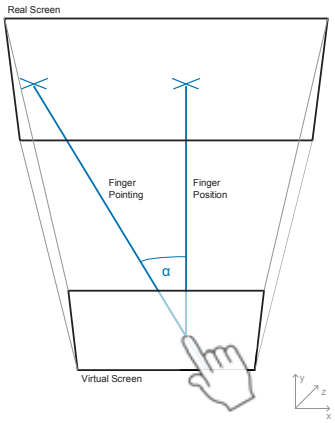


Figure 10.11: Difference between Finger Pointing and Finger Position

Applying the Gesture Set. To apply the developed gesture set, additional steps are required. Since there is no touchable surface to tab on, it has to be defined whether the user intends to perform a gesture or he is just moving his hand around (e.g., for pointing purposes). Therefore, the virtual box is divided into an *active* and *passive* area (cf. Figure 10.12). If the user moves his hand in the passive area, no gesture detection is performed, but the hand is shown as pointer on the real screen [320]. In case the user moves his hand forward towards the screen entering the active area, gesture detection starts. If a user moves his hand forward and pulls it back again, for example, this will be recognized as tab gesture.

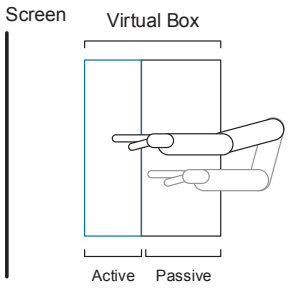


Figure 10.12: Activating Hand Gestures

As opposed to interactions on a tablet, interaction in front of a projector screen often involves several users. When detecting hand gestures, it must be recognized which hand belongs to which user to be able to detect gestures by multiple hands. Therefore, distances between the individual hands can be calculated. Those pairs of hands with lowest distances can belong to the same user.

Using this technique all suggested multi-touch gestures (cf. Section 10.4) can be applied as touchless interaction concept for process modeling. In particular, this allows us to provide an intuitive way of process modeling (cf. Requirement REQ-9).

10.5.2 Process Modeling Using Touch Tables

In certain situations it is desired to work collaboratively on a process model. Such situations include process discovery meetings or discussions among business analysts and domain experts about the appropriateness of a process model. Usually, process models are displayed on a projector screen or drawn on whiteboards. Typically, a projector screen solely enables one participant to interact with a process model (i.e., the one in front of the presentation computer). Whiteboards enable all users to document aspects on the respective business process. However, process models written on a whiteboard must then be manually transferred to a process modeling tool.

A touch table provides a large screen size as well as the possibility for multiple users to simultaneously interact with it [321]. However, designing applications for touch tables requires additional design guidelines [322, 323]. To be more precise, an application for touch tables shall provide an interaction experience like being at a desk in the office working with physical objects [323]. If a touch table provides a "physical" user interaction, it is more intuitive for users that shall interact with it. Furthermore, touch tables allow working on all sides of the table (cf. Figure 10.13), i.e., no "top" exists (despite it is mounted to the wall). This implies that applications shall provide windows and menus accessible from all sides [324]. Typically, these windows and menus are movable and rotatable. Enabling users to interact with the table from all sides might motivate multiple users to concurrently participate in the interaction, i.e., multi-user usage must be supported. In particular, users should be able to concurrently modify the content of the application.

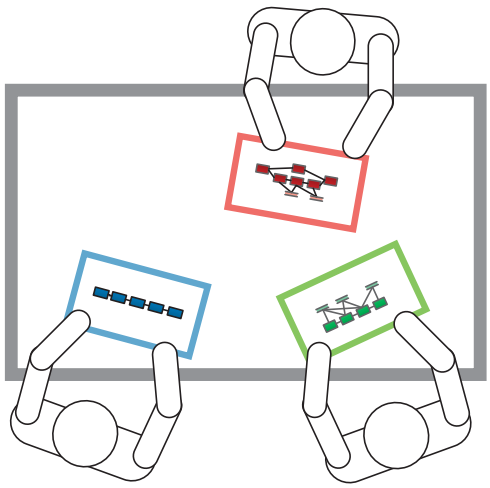


Figure 10.13: Process Modeling on a Touch Table

Working collaboratively requires separate spaces for personal or group work, i.e., users may first want to work in a personal space to prepare, for example, a process model. Then, the respective user shares results with others [325]. Moreover, the transition between personal and group work shall be seamless [326].

Applying the Developed Gesture Set. Figure 10.14 shows a user interface supporting multiple users. A *process model window* visualizes a process model. The latter is freely movable and resizable on the screen. Such windows are used on the screen to share process models with other users at the table [327] and might enable personal and group work [322].

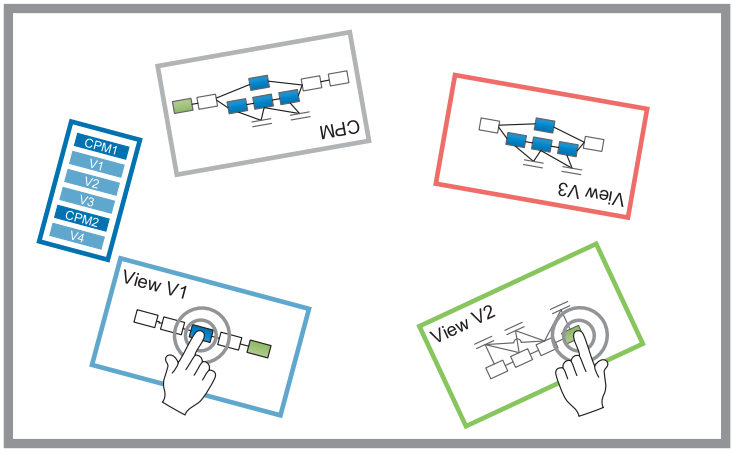


Figure 10.14: Selecting Process Elements on a Touch Table

To support users working with multiple windows and process models, process element selections should be synchronized across multiple windows. If a user selects a process element in one window, this element is highlighted in other windows as well (cf. Figure 10.14). Moreover, if the selected process element is an aggregated one, all elementary process elements are selected in the other windows. This allows users to deal with different abstraction levels (i.e., process views). In Figure 10.14, for example, a user selects an aggregated activity in process view $V1$. This element is directly highlighted in the process model windows showing process view $V3$ and the corresponding CPM. In this context, it may be useful, if each process model window has its own color to show which user has selected a respective element.

To update or interact with process models, the introduced core gesture set may be applied. However, in comparison to user interaction on tablets, windows on touch tables are movable. If the user tabs on a window and starts moving his finger, we need to distinguish whether he wants to interact with the process model or to move the surrounding window. It is common that windows are surrounded by a thick border. This border is used as handle for move gestures [323].

Finally, multi-touch tables may be combined with tablets. In particular, process models (i.e., CPMs or process views) may be exchanged between the touch table and one or more tablets. For example, a user may want to "pick up" a process model or a process fragment, update it on a tablet, and drop the updated process model back to the touch table (cf. Figure 10.15). Existing approaches for sharing screens between touch tables and tablets may be used (cf. [326, 328]). Combining touch table and multiple tablets extends the personal space to the user's tablet and increases flexibility when using such a modeling environment.

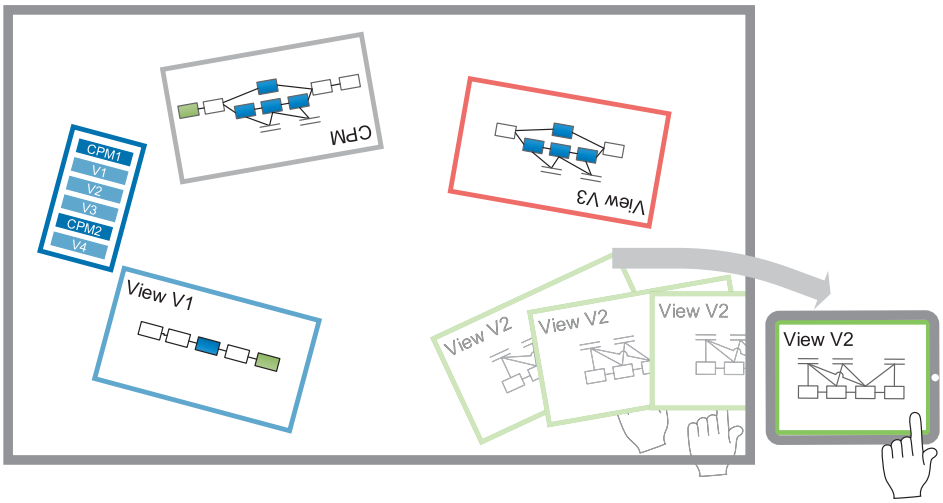


Figure 10.15: Combining Touch Table and Tablet for Process Modeling

10.6 Discussion

This chapter introduces a core gesture set for interacting with process models. The gesture set is developed based on the results of two experiments, which aim to identify an intuitive gesture set. However, results from experiments involving students and research assistants are only partially transferable to practice. Furthermore, the developed gesture set comprises only a subset of gestures required to develop a multi-touch-aware process modeling tool. However, frequent use cases for process interaction are covered.

We further provide insights how the gesture set can be applied to touchless interaction as well as to collaborative process modeling. Therefore, we discuss issues to be considered applying touchless interaction. However, further experiments are required to identify all aspects to be considered when developing a process modeling tool that provides touchless interaction or collaborative process modeling on touch tables.

10.7 Related Work

Related work may be divided in approaches related to *gesture-based interaction* and *collaborative process modeling with touch tables*.

Gesture-based Interaction. Differences between the interaction concepts of pointing, touching, and scanning are analyzed in [329]. As a result, user location (e.g., sitting, standing) is crucial for interaction concepts. A think aloud experiment evaluating a gesture set on touch tables and multi-touch devices is introduced in [293]. However, introduced gestures do not consider process interaction. [330] provides an approach for developing intuitive and ergonomic gesture sets. Thereby, the approach considers users to find appropriate gestures. Furthermore, it is

shown that in certain situations users prefer more than one gesture for one and the same task. In [303], a multi-touch gesture dictionary is suggested.

A multi-touch gesture set for interacting with graphs is provided in [331]. The approach focuses on the interaction with edges. A manipulation of the graph structure is not investigated. Finally, [280, 332] analyzes the combination of multi-touch gestures and pen input to manipulate graphs. As a result, a corresponding gesture set is suggested.

A concept to describe gestures based on examples is introduced in [333]. The latter requires an algorithm to efficiently detect hand gestures [334]. Finally, [335] compares the performance of various detection algorithms. The Microsoft Kinect 2D and 3D image can also be used for gesture recognition [336]. The resolution and accuracy of the Microsoft Kinect in the context of indoor mapping is analyzed in [337]. [338] introduces techniques and gestures for interaction remotely with large screens.

Collaborative Process Modeling. An overview on multi-user gesture-based interactions is given in [339]. In particular, special gestures for multi-user scenarios may increase usefulness of collaborative applications on touch tables. A literature review and challenges in collaborative modeling are presented in [340]. A particular challenge in this context is the modeling skills of users.

The way a process modeling tool supports the collaboratively creation of process models is crucial [327]. The difference exists between the absence of tool support and the usage of a collaborative process modeling tool is evaluated. Altogether, a collaborative process modeling tool increases model quality, but required time is also increased [327]. An approach for distributed process modeling utilizing virtual environments is provided in [341]. Users are visualized in a virtual world and are able to modify a 3D process model. In a collaborative scenario, this approach allows pointing at process elements and discusses about it.

Tangible business process modeling (TBPM) provides an approach to collaboratively document process models based on physical (i.e., tangible) elements [302]. The approach is applied in the clinical domain to demonstrate its appropriateness for non-technical users [342]. However, TBPM is only applied to initially capture processes, but not to evolve existing process models. In the context of S-BPM, [343, 344] introduces a touch table that allows users to document process models based on physical objects. As opposed to TBPM, process models are automatically stored in a process repository. Moreover, [345] shows that an adequate support of touch tables enables users with limited modeling experience to create and evolve process models.

An experiment comparing collaborative process modeling with a video projector, vertical-mounted multi-touch screen, horizontal multi-touch table, and a whiteboard are presented in [346]. As a result, a (horizontal) multi-touch table is suited best for collaborative process modeling. In particular, resulting process models are comprehensible and lively discussions become possible. However, users do not prefer on-screen keyboards for text input.

10.8 Summary

This chapter introduces user interaction concepts for process models and suggests a multi-touch gesture set, which includes gestures to interact with process models as well as to change them. This is crucial to enable users to create process models on touch-enabled devices (e.g., tablets). The developed gesture set is based on experimental results to guarantee for its appropriateness.

The gesture set is applied to touchless interaction. The latter includes technologies not requiring to touch any screen; i.e., bare-hand gestures are used to interact with a process modeling tool. In particular, it is shown that the developed gesture set can be used to interact with process models in front of a projector screen, for example, utilizing the Microsoft Kinect or Leap Motion sensor.

Finally, the developed gesture set is applied to collaborative process modeling on touch tables. The latter allow for the concurrent interaction of multiple users with process models. In this context, process views are used to reduce the complexity of process models for users as well as to allow for the concurrent change of process models.

Part IV

Validation

11 Proof-of-Concept Prototype

11.1 Introduction

The concepts and algorithms presented in Parts II and III are implemented in a proof-of-concept prototype, which aims to demonstrate the technical feasibility of the *proView* framework. In addition, it provides the basis for evaluating core concepts (cf. Chapter 12).

The following functional requirements are met by the proof-of-concept prototype:

- Enabling users to create process models (i.e., CPMs) or import existing ones.
- Supporting users in creating process views based on elementary as well as high-level view creation operations.
- Dynamically switching between different process representations and ability to easily integrate additional process representations.
- Enabling users to apply view update operations as well as to control their propagation to the related CPM.
- Enabling users to control the migration of other views to an updated CPM version.
- Enabling gesture-based process model updates and view creation.

In addition, the following non-functional requirements are met by the proof-of-concept prototype:

- Improved usability through advanced visualization and interaction concepts.
- Enabling multiple users to interact with process models and views.
- Extensibility and changeability of the implemented concepts.
- Independence from any device type and operating system respectively.

First of all, we developed a spike solution to identify required architectural components. Furthermore, we experienced impressions about potential pitfalls when implementing the concepts of this thesis [99]. Then, another prototype was implemented based on the experiences with the spike solution [239, 309, 347, 348]. In particular, this prototype provides the described functions. Initially, a server and a client component were implemented. Then, other process representations and a touch-enabled user interface were added.

Section 11.2 introduces the general architecture of the prototype. Section 11.3 describes the server component (i.e., *proViewServer*) and discusses the implementation of our concepts. Section 11.4 presents the *proViewClient* and various user interfaces. Section 11.5 discusses and concludes the chapter.

11.2 Architecture Overview

The developed proof-of-concept prototype comprises *proViewServer*—a central server component allowing for the persistent storage of CPMs, process views, and parameter settings. Furthermore, *proViewServer* supports the creation of process views and handles process view updates. In particular, the latter may be propagated to the related CPM as well as all other associated process views. Next, *proViewServer* provides a *RESTful (Representational State Transfer)* communication that enables clients to retrieve process models (i.e., CPMs or process views) as well as to evolve them over time (cf. Figure 11.1). In particular, *proViewServer* integrates AristaFlow BPM Suite[1], using its object model to represent process models as well as process model updates (i.e., to insert an edge or node).

Based on the RESTful communication provided by *proViewServer*, several clients are developed: *proViewClient*, *proViewMobile*, *proViewKinect*, and *proViewCollab*. First, *proViewKinect* allows users to interact with process models and views based on hand-gestures (i.e., touchless interaction) [320]. In turn, *proViewCollab* enables users to collaboratively model processes utilizing a touch table [321]. Both clients have been implemented using C# and *Windows Presentation Framework (WPF)*[2]. Furthermore, *proViewMobile* is an Apple iPad app that allows for interacting with process models and process views [349]. In particular, this client provides functions to create and update process views based on multi-touch gestures.

The *proViewClient* is a web client based on the Vaadin framework [350]. The latter constitutes an open source web framework for creating web applications using Java as programming language. A Vaadin application itself consists of a client and server side. Furthermore, *Google Web Toolkit (GWT)* is used to render client-side web pages, whereas *Asynchronous JavaScript and XML (Ajax)* enable the communication between client and server side. The *proViewClient* allows for a sophisticated visualization of process models (i.e., CPMs and process views) based on various process representations. Furthermore, the presented view update concept is provided to end-users. Finally, multi-touch gestures are integrated with *proViewClient*. Further details can be found in Section 11.4.

[1] See http://www.aristaflow.de and [110, 112]
[2] http://msdn.microsoft.com/en-us/library/aa970268.aspx

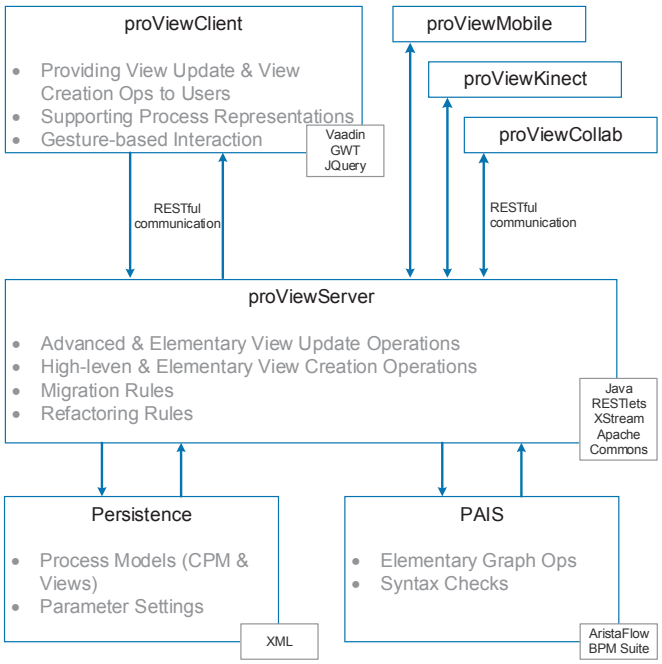


Figure 11.1: Architecture Overview

11.3 proViewServer

The *proViewServer* is responsible for managing clients as well as process models. It consists of three layers (cf. Figure 11.2).

The *Resource Layer* encapsulates all services required to provide a RESTful communication to clients. Thereby, Java objects representing process models as well as related operations are serialized with XStream[3]. *ProcessManager* supports the creation of process models as well as interactions with them. *ViewManager* allows creating, storing, and evolving process views.

The *Operational Layer* comprises services for creating and evolving process models (i.e., CPMs and process views). In this context, *CPMService* provides basic functions to create, load, and delete CPMs. In turn, *ViewService* comprises functions to load, save, refactor, and migrate process views. Furthermore, it coordinates updates on process views by triggering the *ViewUpdateService* and creates process views by triggering the *ViewCreationService*.

The *Persistence Layer* and its *PersistenceService* allow for the persistent storage of all process model and view artifacts. Process models are stored as XML files in a format supported by the AristaFlow BPM Suite [70]. The XML files, in turn, are extended with additional XML elements . In particular, the *pluginDataContainer* is used to store the *Universally Unique Identifier (UUID)* of CPM together with its version (cf. Listing 11.1). Furthermore, for each configuration parameter, an XML element is added to the respective parameter value for the CPM (cf. Listing 11.1, Lines 5-12). Note that these parameter values are inherited by each process view defined on that CPM.

[3]XStream is a library to (de-)serialize objects to and from XML. See http://xstream.codehaus.org/

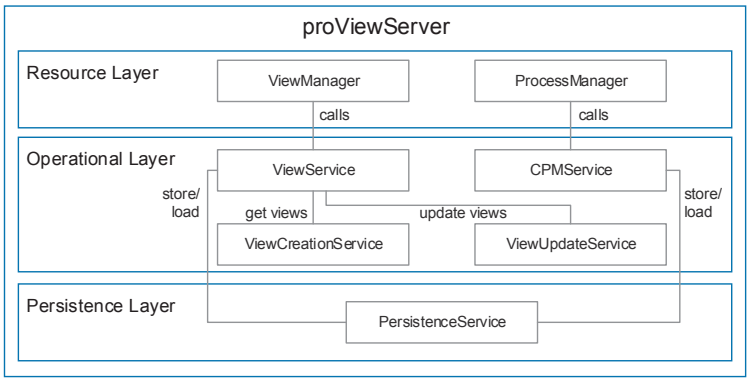


Figure 11.2: Architecture of proViewServer

```
1  <pluginDataContainer>
2    <pluginData pluginID="cpm" extensionPoint="cpm">
3      <pluginDataEntry name="cpmID">...</pluginDataEntry>
4      <pluginDataEntry name="processVersion">1</pluginDataEntry>
5      <pluginDataEntry name="AggrComplMode">AGGR</pluginDataEntry>
6      <pluginDataEntry name="AggrPartlyMode">AGGR</pluginDataEntry>
7      <pluginDataEntry name="DeleteBlockMode">INLINE</pluginDataEntry>
8      <pluginDataEntry name="InsertBlockMode">EARLY_EARLY</pluginDataEntry>
9      <pluginDataEntry name="InsertBranchMode">EARLY</pluginDataEntry>
10     <pluginDataEntry name="InsertSerialMode">EARLY</pluginDataEntry>
11     <pluginDataEntry name="RedComplMode">RED</pluginDataEntry>
12     <pluginDataEntry name="RedPartlyMode">RED</pluginDataEntry>
13   </pluginData>
14 </pluginDataContainer>
```

Listing 11.1: XML Representation of a Parameter Set

Each process view is represented by an XML document containing a reference to its CPM, its UUID, and the versions of the CPM as well as its associated process views (cf. Listing 11.2). Element *viewName* stores the name of the process view. In element *operationSet*, in turn, a set of view creation operations are stored. The latter need to be applied to the CPM to create the process view. For example, the view creation operation described in Lines 8-19 aggregates process nodes with ID 18 and ID 83 to an aggregated node with ID 84, and label it as "Preprocessing Steps". Accordingly, the view creation operation described in Lines 20-25 reduces the activity with ID 19 in the process view. Finally, element *parameterSet* describes view-specific parameter values. In Listing 11.2, no explicit parameters are defined, i.e., all parameter values are inherited from the CPM.

```
1  <view>
2    <cpmID serialization="custom">...</cpmID>
3    <viewID serialization="custom">...</viewID>
4    <cpmVersion>1</cpmVersion>
5    <viewVersion>3</viewVersion>
6    <viewName>Agent Clerk</viewName>
7    <operationSet>
8      <ViewCreateOperation>
9        <aggrNode>84</aggrNode>
10       <op class="CreateChangeOperation">AGGREGATE_SESE</op>
```

```
11          <nodeSet>
12            <int>18</int><int>83</int>
13          </nodeSet>
14          <optionSet>
15            <entry>
16              <string>nodeName</string><string>Preprocessing Steps</string>
17            </entry>
18          </optionSet>
19        </ViewCreateOperation>
20        <ViewCreateOperation>
21          <aggrNode>0</aggrNode>
22          <op class="CreateChangeOperation">REDUCE_ACTIVITY</op>
23          <nodeSet><int>19</int></nodeSet>
24          <optionSet/>
25        </ViewCreateOperation>
26      <parameterSet/>
27    </view>
```

Listing 11.2: XML Representation of a Process View

11.3.1 Updating a Process View

The sequence diagram depicted in Figure 11.3 illustrates how a view update operation is processed in *proViewServer*: the update is triggered through a REST POST sent by *proViewClient*. The REST POST is received by *ViewManager* and then de-serialized. This request is forwarded to *ViewService* together with the view update operation *viewOp* to be applied as well as the *UUID* of the view to be changed. Then, *ViewService* fetches the associated CPM and recreates—if necessary—the process view that triggered the update.

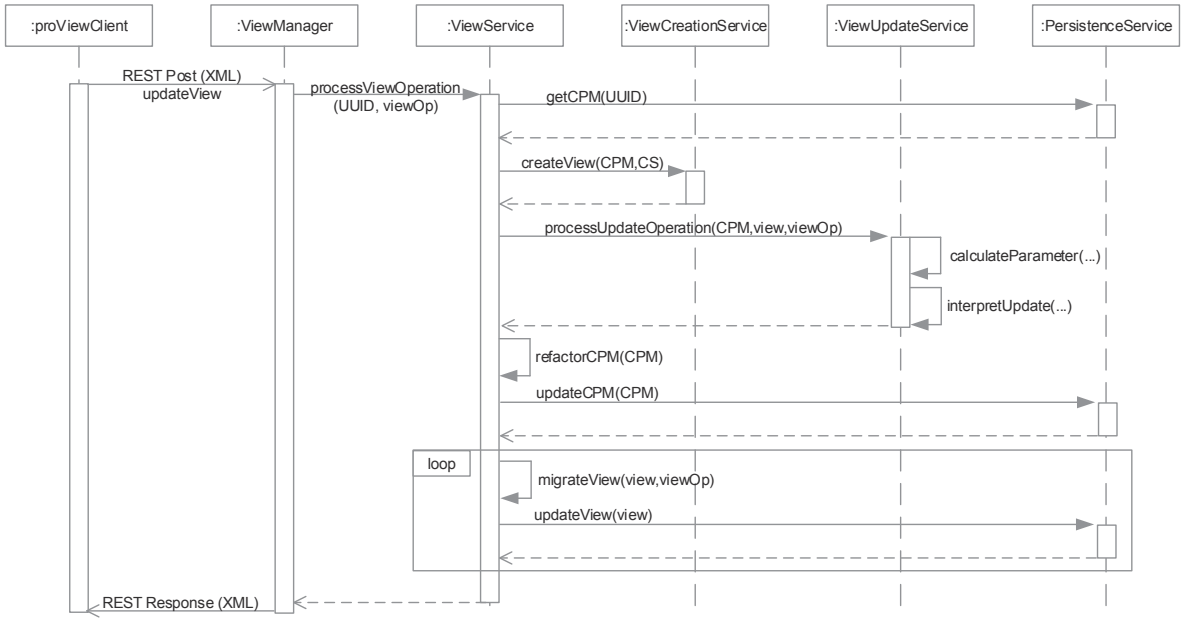


Figure 11.3: Updating a Process View

Based on the process model of the CPM and process view as well as the view update operation

to be applied, *ViewUpdateService* determines the parameter values to identify the insert position in the CPM and updates the latter (i.e., *interpretUpdate()*). Following this, the updated CPM is refactored in order to eliminate unnecessary control flow structures. The refactoring was added subsequently to the prototype in order to keep the CPM comprehensible.

The updated CPM is sent back to *PersistenceService* to ensure persistence of the applied updates. Finally, all other process views associated with the CPM are migrated to the new version of the CPM, and are then stored using the *PersistenceService*. The migrated model of the process view, which triggered the update, is sent back to *proViewClient*.

11.4 proViewClient

The *proViewClient* serves as user interface to create, view, and update process models. More precisely, this user interface consists of a side bar displaying all accessible CPMs and their associated process views. Thereby, a process view is shown as child entry of its CPM (cf. Figure 11.4A). When clicking on one of these entries, the *process modeling* window (cf. Figure 11.4B) is opened. The latter then shows the selected process model. Furthermore, a *properties* window, which provides attributes of selected process elements and their values, is displayed (cf. Figure 11.4C). Alternatively, an overview on the data elements is shown. Finally, Figure 11.4D shows an overview of applied view creation operations and allows undoing them if required.

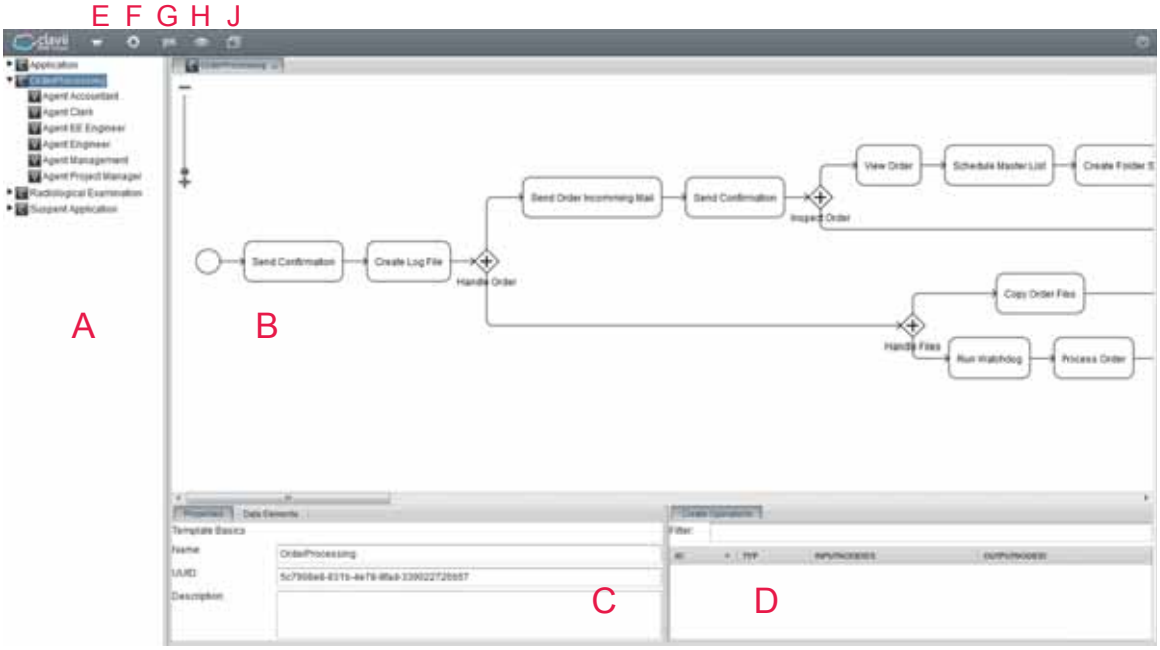


Figure 11.4: Screenshot of proViewClient

Menu bar item *market* allows loading pre-defined CPMs into the current session (cf. Figure 11.4E). Further, menu bar item *settings* (cf. Figure 11.4F) allows users to configure various settings, e.g., IP address of *proViewServer*. Furthermore, it allows configuring the style and

behaviour of process representations (e.g., to adjust vertical or horizontal distances between process elements). Finally, menu bar item *parameters* opens a dialog window for setting parameter values (cf. Chapter 7) for CPMs and process views (cf. Figure 11.4G).

Figure 11.5a shows the parameter window and the parameter values for a specific process view. In particular, parameter values with suffix "(CPM)" are inherited from the corresponding CPM. Parameter values without this suffix are explicitly set for the respective process view. If this window is opened for a CPM, it shows parameter values for the respective CPM as well as for all associated process views (cf. Figure 11.5b).

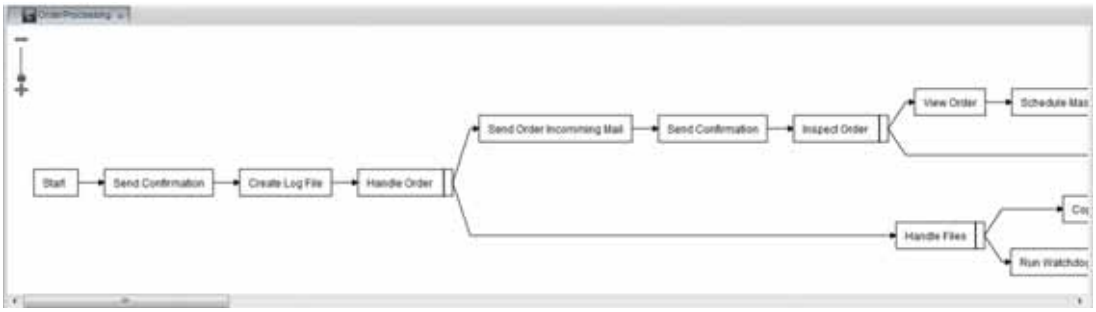
(a) Process View Parameters (b) CPM Parameter Overview

Figure 11.5: proViewClient - Parameter Settings

Menu item *representations* allows users to switch between different process representations (cf. Figure 11.4H). Currently, four process representations are provided. As default representation, BPMN can be used (cf. Figure 11.4). Furthermore, the ADEPT process modeling language is supported as alternative graph-based representation (cf. Figure 11.6a) [104]. Both graph-based process representations use specific layout algorithms. A *normal* layout positions process elements as shown in Figure 11.6a. The *simulated* layout, in turn, positions each process element at exactly the same position as in the CPM, i.e., a user may see at which parts of a process view the CPM process elements have been hidden or aggregated.

The verbalized process description introduced in Section 9.4 has been implemented as well (cf. Figure 11.6b) based on the technology described in [231]. Details about this implementation can be found in [239]. In addition, *proViewClient* allows for a form-based representation (cf. Section 9.3), which visualizes a process model in terms of nested rectangles. Further details about this implementation are provided in [348].

Finally, menu item *windows* allows users to arrange multiple process model windows right next to each other. In turn, this allows for the visual comparison of a CPM with its associated process views (cf. Figure 11.4J).



(a) ADEPT Representation

(b) Verbalized Process Representation

(c) Form-based Representation

Figure 11.6: Process Representations in proViewClient

11.4.1 Creating and Updating Process Views

Elementary and high-level view creation operations are provided by *proViewClient* to reduce process elements and to aggregate them. First of all, a process view is created based on a CPM, i.e., initially it corresponds to an exact copy of the CPM. When selecting process nodes and performing a right click on them, a list of view creation operations that may be applied on the selected node set are displayed. For example, *proViewClient* checks whether the selected node set induces a SESE block to which view creation operation *AggrSESE* can be applied. Figure 11.8 shows such a context menu, based on the selection of two activities. As can be seen, the selected activities may be reduced or aggregated. Furthermore, high-level view creation operation "View from selection" allows creating a process view that only displays selected activities. Finally, high-level view creation operation "Show my activities" and "Create process views for each agent" are supported as well.

Aggregated activities are displayed as sub-process activities (cf. Figure 11.7a). Clicking on the *plus* symbol of such an activity unfolds the sub-process, which is then displayed inline (cf. Figure 11.7b). Based on this unfolded sub-process a new process view can be created, i.e., high-level view creation operation "View from selection" is applied.

In order to update process views, various elementary view update operations are offered to users (cf. Figure 11.8). View update operations may be applied in combination with all process

(a) Closed Sub-Process (b) Open Sub-Process

Figure 11.7: proViewClient - Aggregation Operation

representations. Once a view update is applied, the respective operation is sent to *proViewServer* (cf. Section 11.3). After receiving the updated process view, *proViewClient* provides animations in order to give the user feedback, which part of the process view was changed.

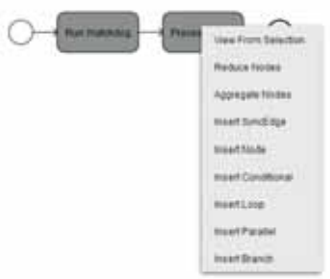


Figure 11.8: proViewClient - Triggering View Creation and Update Operations

In order to trigger view creation and update operations to all process representations an implementation of the *IModelingService* interface is required (cf. Listing 11.3). Respective methods are invoked to push selected operations to *proViewServer*. Furthermore, the interface is implemented for all representations by the *ModelingService*. Consequently, process representations are independent from the logic of view creation and update operations. To obtain the updated process view, all process representations must inherit from abstract class *AApearance*. In turn, method *updateModel* of *AApearance* is invoked if an updated process model from the server is received (cf. Section 11.3).

```
1  public interface IModelingService {
2    void renameNode(UUID uuid, ArrayList<Node> nodes);
3    void renameNode(UUID uuid, int nodeid, String newNodeName);
4    void reduceNodes(UUID viewId, ArrayList<Node> nodes, boolean showNotification);
5    void createViewFromSubprocessesAndNodes(UUID uuid,
6          Set<LayoutSubprocess>subprocessSet, Set<Integer> set);
7    void aggregateNodes(UUID uuid, ArrayList<Node> nodes);
8    void deleteNode(UUID uuid, ArrayList<Node> nodes);
9    void insertSyncEdge(UUID uuid, ArrayList<Node> nodes);
10   void insertSerial(UUID uuid, ArrayList<Node> nodes);
11   void insertLoop(UUID viewId, ArrayList<Node> selectedNodes);
12   void insertConditional(UUID viewId, ArrayList<Node> selectedNodes);
13   void insertParallel(UUID uuid, ArrayList<Node> nodes);
14   void insertBranch(UUID viewId, ArrayList<Node> selectedNodes);
15   void undoOperations(UUID viewId, Set<ViewCreateOperation> undoSet);
16 }
```

Listing 11.3: Modeling Service Interface

11.4.2 Providing Multi-Touch Gestures

In a second step, we extended *proViewClient* to be able to recognize multi-touch gestures (cf. Chapter 10) [309]. In turn, this enables us to recognize multi-touch gestures on each (mobile) device with an installed browser. Note that existing add-ons for detecting gestures within a Vaadin application (e.g., Vaadin TouchKit[4]) do not focus on multi-touch gestures. To extend *proViewClient* with the presented gesture set and, in particular, symbolic gestures, a Vaadin add-on, denoted as *proViewTouch* was developed.

As typical for any Vaadin add-on, *proViewTouch* consists of a client and server side. The client side of *proViewTouch* is represented by the *TouchPanelWidget* and *TouchPanelConnector* components. Thereby, *TouchPanelWidget* integrates itself in the process modeling window (cf. Figure 11.4B) in order to capture touch start, move, and end events triggered by the fingers of users in a browser window. Captured gestures are then sent to the server side by the *TouchPanelConnector*.

The server side part of *proViewTouch* is represented by *TouchPanel*. The latter receives the gesture from the client side and runs a recognition algorithm to detect which kind of gesture the user performed. After recognizing the gesture, the respective operation is called on *proViewServer*.

Figure 11.9 shows the interaction between the components as required to recognize a gesture in a sequence diagram. The *TouchPanelWidget* is triggered by the user when he starts touching the screen. Afterwards, class *SymbolicGesture* is triggered to capture touch positions, i.e., if the user starts moving his finger on the screen, new touch positions are sent to *SymbolicGesture*. If the user lifts his finger (i.e., a touch end event occurs), detected touch points are sent to the *TouchPanelConnector*. The latter bundles the touch points and sends them to the server side of *proViewTouch* using a RPC call. In case the user just tabs on the screen (i.e., no move event occurs), no communication between client and server side is required (i.e., tabs are handled directly by the client).

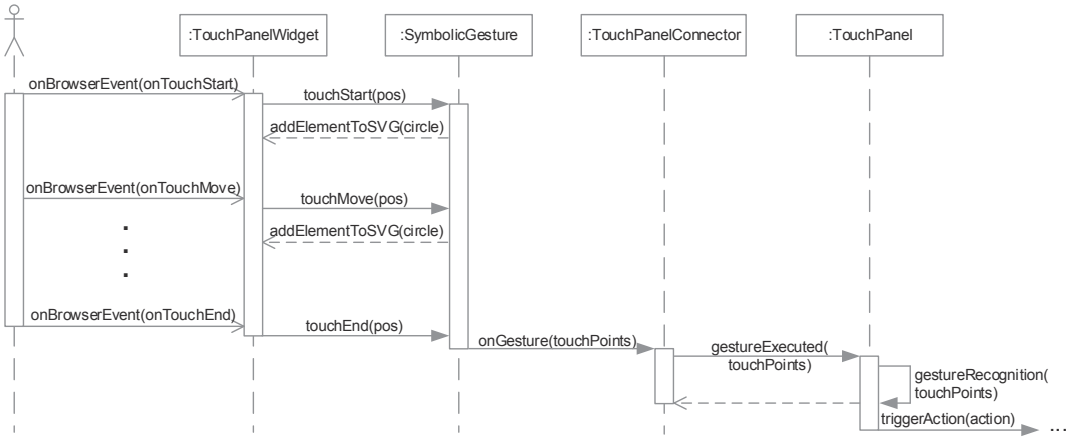


Figure 11.9: Detecting a Multi-Touch Gesture

[4]Further information about the Vaadin TouchKit: https://vaadin.com/add-ons/touchkit

When *TouchPanel* receives the touch points, it tries to recognize them. Therefore, the *$1 Gesture Recognizer* algorithm is applied. It pre-processes the received gesture information and compares detected gestures with pre-defined gesture templates [334]. The latter are defined in terms of XML documents. Listing 11.4 represents gesture template *RECTANGLE* as a set of points on a two-dimensional screen.

```
1 <templates>
2   <template name="RECTANGLE">
3     <pt y="158" x="59"/>
4     <pt y="170" x="59"/>
5     ...
6   </template>
7   ...
8 </templates>
```

Listing 11.4: XML Representation of a Gesture Template

If there is a match with a gesture template, a corresponding event is triggered based on the name of the template. Then *proViewClient* handles this event and sends a respective view update and view creation operation to *proViewServer*.

11.5 Discussion and Summary

This chapter introduces the proof-of-concept prototype that realizes the concepts and algorithms of the *proView* framework. It allows visualizing and exploring these concepts in a comprehensible and consistent manner. Furthermore, the prototype can be used in real-world case studies and experiments as will be shown in the next chapter. In particular, the proof-of-concept prototype demonstrates the feasibility of the concepts and algorithms developed in Parts II and III of this thesis. The prototype confirms that multiple users are able to interact with the *proView* framework to maintain process views. However, we do not consider scalability issues. In particular, the prototype slows and acquires a lot of memory if too many users are active. Finally, the proof-of-concept prototype lacks a number of features required for its application in a practical setting. For example, no access control mechanisms are provided. Finally, the prototype should be applied in modeling projects within companies and organizations in order to obtain further feedback.

12 Case Studies and Experiments

This chapter evaluates selected concepts and algorithms of the *proView* framework. The evaluation comprises three parts: First, Section 12.1 demonstrates the application of view creation and view update operations to practical scenarios. Second, Section 12.2 presents an experiment that validates process representations. Third, the presented multi-touch gesture set is evaluated through an experiment in Section 12.3. Note that due to the modular design of the *proView* framework, we are able to evaluate specific parts separately (cf. Figure 12.1). Finally, Section 12.4 discusses the evaluation results.

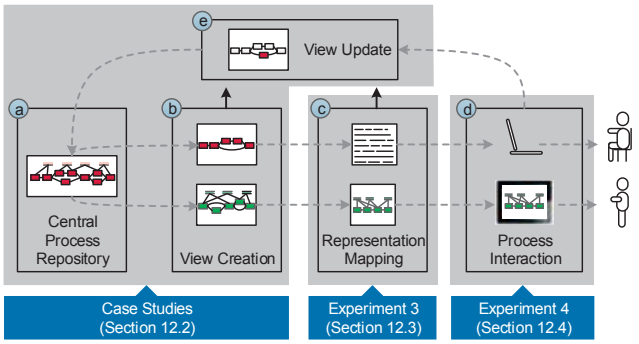


Figure 12.1: Overview of Framework Evaluation

12.1 Case Studies

In this section, we apply the presented view creation and view update operations to real-world process models in order to demonstrate their relevance, applicability, and benefit in practice. The considered process models origin from various domains, i.e., *finance* [139], *accounting* [351], and *health care* [352].

The considered process models are first created by applying the proof-of-concept prototype presented in Chapter 11. Then, the resulting models are abstracted, i.e., process views are created for each user (role) involved in the respective process. Additionally, process views are created, which represent IS contributing to the execution of a process model. Finally, for some process models additional process views are created providing an abstract overview.

In order to measure the degree of process abstraction after the application of view creation operations to a process model, we use well-known process metrics [73, 208, 353]. In detail, the total *number of activities* and *gateways* is counted. Furthermore, the *number of AND, XOR, and Loop gateways* are determined separately. Note that we not distinguish between split and corresponding join gateways in this context. Other process model metrics considered include *diameter*, *separability*, and *McCabe*. Process metric *diameter* measures the longest path through the process model [73]. In turn, process metric *separability* describes the ratio between the number of *cut-vertices*, i.e., the ratio between control edges serving as bridges between otherwise disconnected graphs (i.e., process models) and the total number of nodes in a process model. In particular, in [73] a high significance between process metric *separability* on one hand and process model understanding on the other is identified, i.e., a high process model understanding is achieved by process models having a high separability. Process metric *McCabe*, in turn, describes control flow complexity and is calculated by the weighted sum of all gateways (and their outgoing edges) in a process model [354, 355].

12.1.1 Case Study: Credit Approval

The first scenario to which we apply view creation and view update operations deals with the processing of credit applications in a Brazilian bank [139]. The process is initiated by a credit application of a customer in a branch office of the bank and terminates with the preparation of the credit contract.

In general, the process may be grouped into four phases (cf. Figure 12.2): *Preprocessing*, *Request Handling*, *Decision*, and *Contract Preparation*. Phase *Preprocessing* fetches customer data from the database, checks completeness of this data, and validates the credit application. After this phase, the credit application might be cancelled. Otherwise, phase *Request Handling* sets up a customer file, acquires customer financial reports, and prepares everything for the *Decision* phase. During the latter, all documents are revised and a final decision is made. If the decision is positive, a contract is prepared, customer records are updated, and the customer is informed in the *Contract Preparation* phase. The process comprises three user roles: *Analyst*, *Clerk*, and *Manager*. Additionally, automatic activities of the *Database* and the PAIS are documented in the respective process model of the *credit approval* process (i.e., CPM).

The bank has used five different process models to capture the process. One process model shows the (top-level) phases (i.e., it gives an *overview*) as described above (cf. Figure 12.2). Furthermore, each phase is detailed by a separate process model (i.e., sub-process). Before applying them to the proof-of-concept prototype, the process models are translated from Portuguese to English. Furthermore, the individual process models are combined into one process model. Since the process models are unstructured ones, techniques described in [117] are applied to get a well-

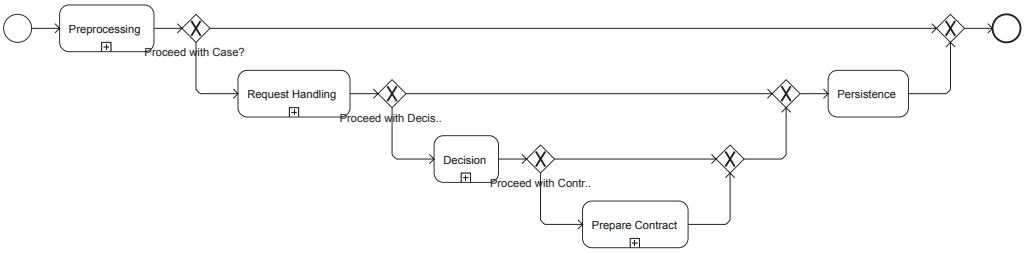


Figure 12.2: Credit Approval - Overview

structured process model matching with our definition of a process model (cf. Definition 6.1). The resulting process model (i.e., CPM) is shown in Figure 12.3.

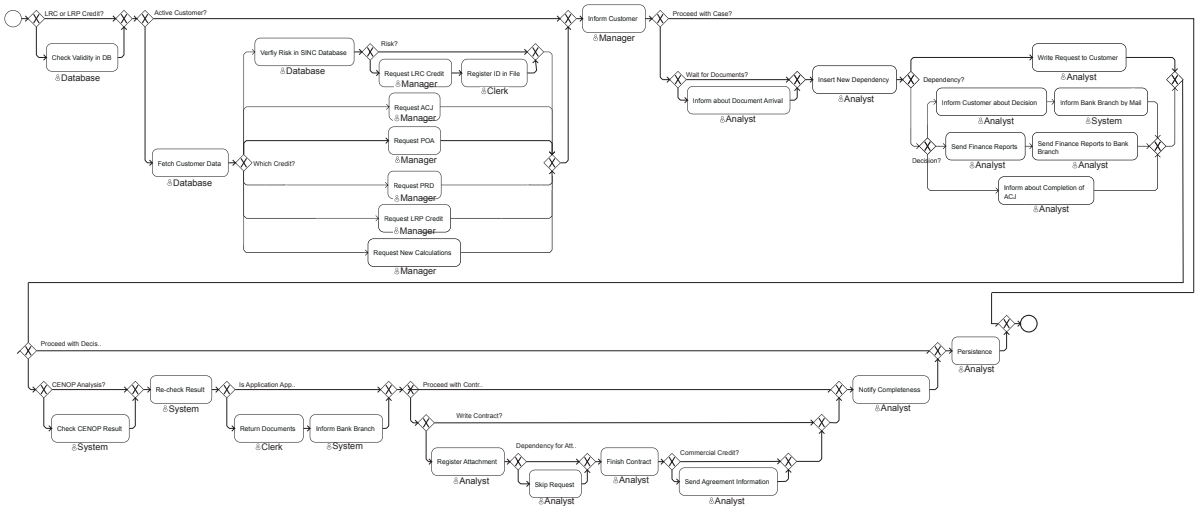


Figure 12.3: CPM Credit Approval

Taking the CPM of the *credit approval* process, we create process views for each user role. Following this, another process view is created that provides an abstract *overview* of the credit approval process, which corresponds to the high-level model of the bank (cf. Figure 12.2). Respective process views can be found in Appendix C. Table 12.1 shows the calculated process metrics for the CPM as well as for each created process view.

In comparison to the CPM, creating and using personalized process views for each user role decreases complexity to 7%–41% regarding the number of activities and 17%–46% regarding control flow complexity (i.e, McCabe metric). Furthermore, users can now look at their parts of the process at once instead of searching the respective activities in six different (sub-)process models. In particular, the diameter of process views is about 10%-45% of the diameter of the CPM. Separability is improved as well; however, values are rather small for process views. This can be explained through the high number of XOR gateways with empty branches (cf. Fig-

Process Model	# Activities	Number of Gateways				McCabe	Diameter	Separability
		Total	AND	XOR	Loop			
CPM	29	30	0	30	0	35	20	0.016
V1: Analyst	12	12	0	12	0	16	9	0.077
V2: Clerk	2	12	0	4	0	8	2	0.333
V3: Database	3	6	0	6	0	6	3	0.364
V4: Manager	7	4	0	4	0	9	2	0.231
V5: System	4	10	0	10	0	10	4	0.125
V6: Overview	5	6	0	6	0	6	5	0.231

Table 12.1: Process Metrics for Credit Approval CPM and Views

ure 12.2). In this context, it might be useful to provide refactoring rules removing empty XOR branches as well. Note that such refactorings violate control flow correctness.

Next, we apply a view update operation, which moves activity "Send Finance Reports to Bank Branch" of user *Analyst* from phase *Decision* to the start of phase *Contract Preparation*. When performing this update in the original process models provided by the bank, the respective activity has to be deleted in the process model of sub-process *Decision* and then a new one is inserted in sub-process *Contract Preparation*. Performing the same update using view update operations, the activity has to be moved in the view of the *Analyst* to its new position (cf. Figure 12.4). The other process views need not be updated. Note that this can be automatically decided utilizing the fact that all process views have been created by applying high-level view creation operation *ViewByPredicate(CPM,Role = userrole)*. In particular, the respective activity is only relevant for the *Analyst*.

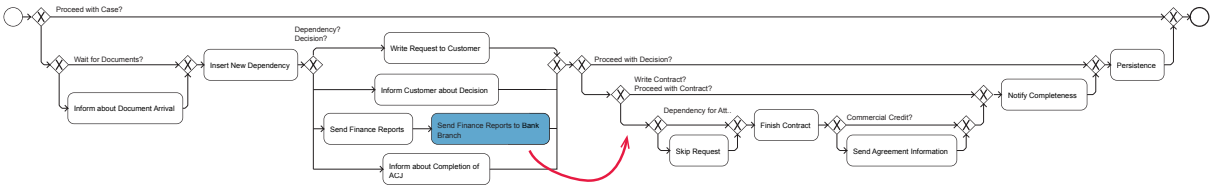


Figure 12.4: Credit Approval - Process View V1: Analyst

12.1.2 Case Study: Order-to-Delivery

The *order-to-delivery* process we consider origins from an accounting department of a medium-sized company, which produces machines customized to the specific needs of the customer. In particular, the process covers project management activities to be performed between the ordering of a machine and its delivery to the customer. Thereby, production related activities are excluded from the process model. In total, the resulting process model contains 56 activities and involves six user roles, i.e., *Accountant*, *Project Manager*, *(Mechanical) Engineer*, *Management*, *Clerk*, and *EE Engineer*.

The *order-to-delivery* process starts with setting up a log file, which documents the progress of the order. Then, the ordered machine is constructed and a corresponding bill of material is

prepared. Under certain conditions, the customer is contacted to request additional information about his requirements. Afterwards, the order is confirmed and the production of the machine is planned and prepared. In parallel to this, electronic/electrical components are designed and the wiring of the machine is planned. Following this, all components of the machine are produced, assembled, and packaged for delivery. Finally, a delivery note as well as the invoice are created.

Initially, the process model provided by the company is translated from German to English. As opposed to the previous case study (cf. Section 12.1.1), the complete process is already documented by a single process model. Based on this process model (i.e., CPM), we create a process view for each user (role) based on high-level view creation operation *ViewByPredicate(CPM,Role = userrole)*. An overview of these process views as well as the CPM is provided by Figure 12.5. Table 12.2 shows the process metrics calculated for the process views.

Process Model	# Activities	Number of Gateways				McCabe	Diameter	Separability
		Total	AND	XOR	Loop			
CPM	56	14	12	2	0	8	25	0.042
V1: Accountant	2	0	0	0	0	0	3	0.500
V2: Project Manager	9	2	2	0	0	1	5	0.231
V3: Engineer	11	0	0	0	0	0	11	0.846
V4: Management	10	2	2	0	0	1	8	0.143
V5: Clerk	17	2	2	0	0	1	15	0.143
V6: EE Engineer	17	2	0	2	0	2	14	0.238

Table 12.2: Process Metrics for Order-to-Delivery CPM and Views

Compared to the CPM, complexity can be decreased to 4%–30% regarding to the number of activities and to 0%–25% regarding control flow complexity (i.e., McCabe metric). Although the *order-to-delivery* process has a higher number of activities than the *credit approval process*, the McCabe metric for the *order-to-delivery* process is considerably smaller (i.e., credit approval: 35, order-to-deliver: 8). This results in the small number of XOR gateways. Furthermore, since empty AND branches and AND branchings are refactored when creating a process view (as opposed to XOR branches and XOR branchings), the complexity of process views in respect to the McCabe metric is considerably smaller. Compared to the CPM, process metric diameter is decreased to 12%–60%, i.e., for process view $V1$ the longest path is 12% of the longest path in the CPM. Separability is increased about 3.4–20.1 times compared to the CPM.

Figure 12.5 shows the application of view update operation *InsertSerial* to process view $V4$: Activity *Send Confirmation* is added between activity *Send Order Incoming Mail* and aggregated activity *Preprocessing & Pricing*. The newly inserted activity is then propagated to the CPM and, afterwards, process views $V4$ and $V5$ are migrated to the resulting new CPM version to reflect this change as well as to apply the new activity *Send Confirmation*. Note that other process views will not show the inserted activity since they refer to different regions of the CPM.

12.1.3 Case Study: Planning a Chemotherapy

The process of *planning a chemotherapy* at a hospital in Germany comprises the preparatory activities required before the chemotherapy may be started [352]. The process involves three user

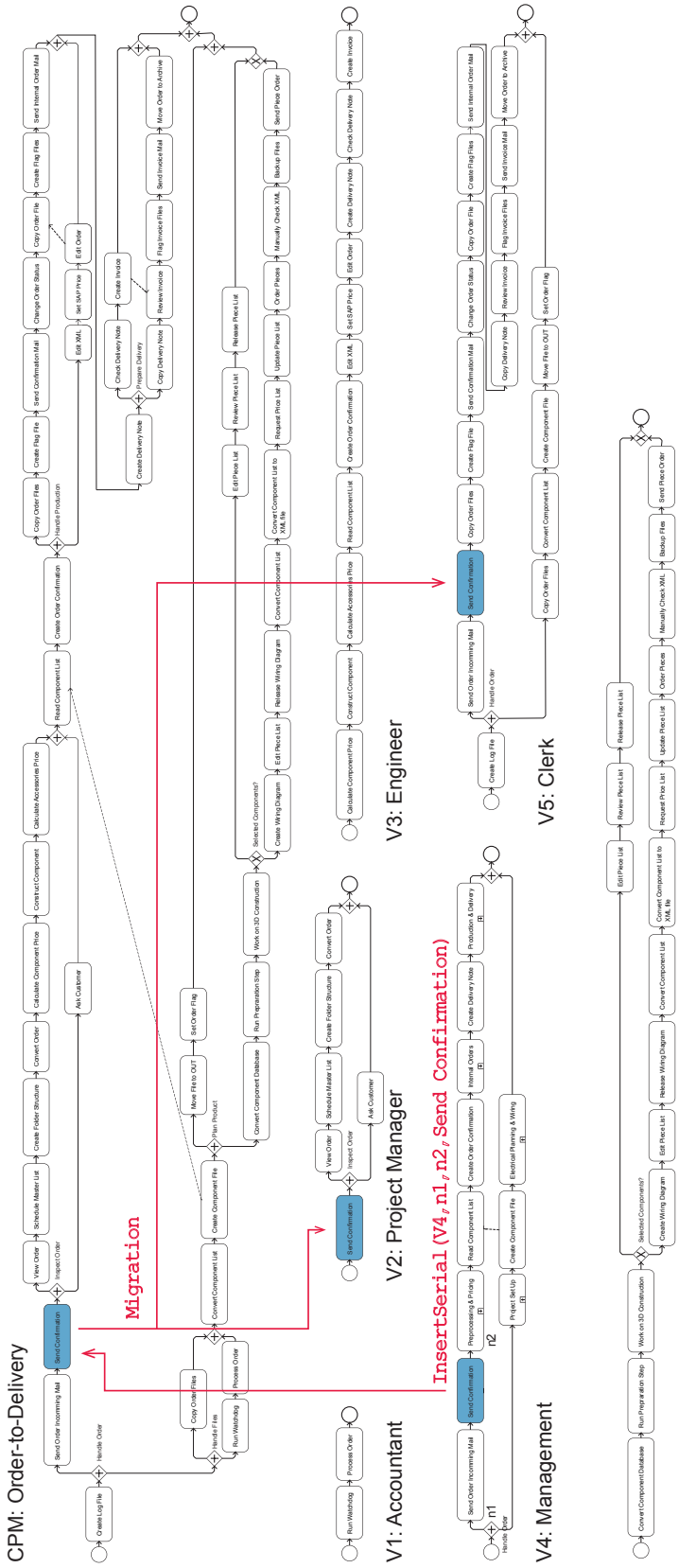


Figure 12.5: Order-to-Delivery - Applying View Update Operations

(roles), i.e., *Doctor*, *Nurse*, and *Secretary*. It starts with an activity that determines the current stage of the cancer. In parallel, the patient record is checked, the chemotherapy is organized, and a report is created. Then, an appointment for the chemotherapy is scheduled. In parallel to the latter, a record describing the procedure of the chemotherapy is created.

Initially, the original process models are merged into one process model (i.e., CPM) and well-structuredness is created [117]. Then, all activity labels are translated to English. Based on the resulting CPM, process views for each user role are created by applying the respective high-level view creation operation. Furthermore, a process view for the *Nurse* is manually created (i.e., by applying elementary view creation operations). The latter shows activities performed by the *Doctor*, in which the *Nurse* is involved in. Then, two elementary view update operations are applied to process view $V1$, and the process metrics are re-calculated. Results are shown in Table 12.3. To be more precise, the *Doctor* first inserts activity *Get Chemo Approval* by applying elementary view update operation *InsertSerial(V1, ANDjoin*$_1$*, Create Chemo Request, Get Chemo Approval)*. After propagating this update to the CPM, process views $V1$ and $V2$ are migrated to the new CPM version. User role *Secretary* is not involved in this update (i.e., process view $V3$). Afterwards, activity *Explain Risks to Patient* is inserted in parallel to activities *Get Chemo Approval* and *Create Chemo Request*. $V1$ and $V2$ are affected by this update again, whereas $V3$ is not. Figure 12.6 shows the updates and their effects on respective process views.

Process Model	#Activities*	Number of Gateways*				McCabe*	Diameter*	Separability*
		Total	AND	XOR	Loop			
CPM	10/11/12	6/6/8	4/4/6	0/0/0	2/2/2	4/4/5	7/8/8	0.278/0.316/0.227
V1: Doctor	7/8/9	4/4/6	2/2/4	0/0/0	2/2/2	3/3/4	5/6/6	0.308/0.357/0.294
V2: Nurse	4/5/6	2/2/4	2/2/4	0/0/0	0/0/0	1/1/2	3/4/4	0.500/0.556/0.417
V3: Secretary	3/3/3	4/4/4	2/2/2	0/0/0	2/2/2	3/3/3	2/2/2	0.444/0.444/0.444

*values x/y/z refer to process model before/after 1st/after 2nd view update operation

Table 12.3: Process Metrics for Order-to-Delivery CPM and Views

The process metrics in Table 12.3 show similar results as already experienced in the other case studies when abstracting the CPM for individual users. However, it is noteworthy that process metric separability increases when applying the first view update operation (i.e., sequential insertion). However, after applying the second view update operation (i.e., parallel insertion) the latter decreases again and is worse than before applying the first update.

12.1.4 Lessons Learned

View creation and update operations of the *proView* framework are applied to process models from different domains. In all scenarios, the initial process models are created and updated using the presented proof-of-concept prototype. Based on these process models, in turn, process views are created for each user role involved. Thereby, the degree of abstraction is measured with well-established process metrics. This measurement has proven that process views show a decreased complexity compared to the initial process models (i.e., CPMs). In particular, this contributes to more comprehensible process models for each user. Furthermore, we have identified the need for additional refactorings dealing with empty XOR branches as well. This is particularly helpful if process models contain a high number of XOR branchings. Note that

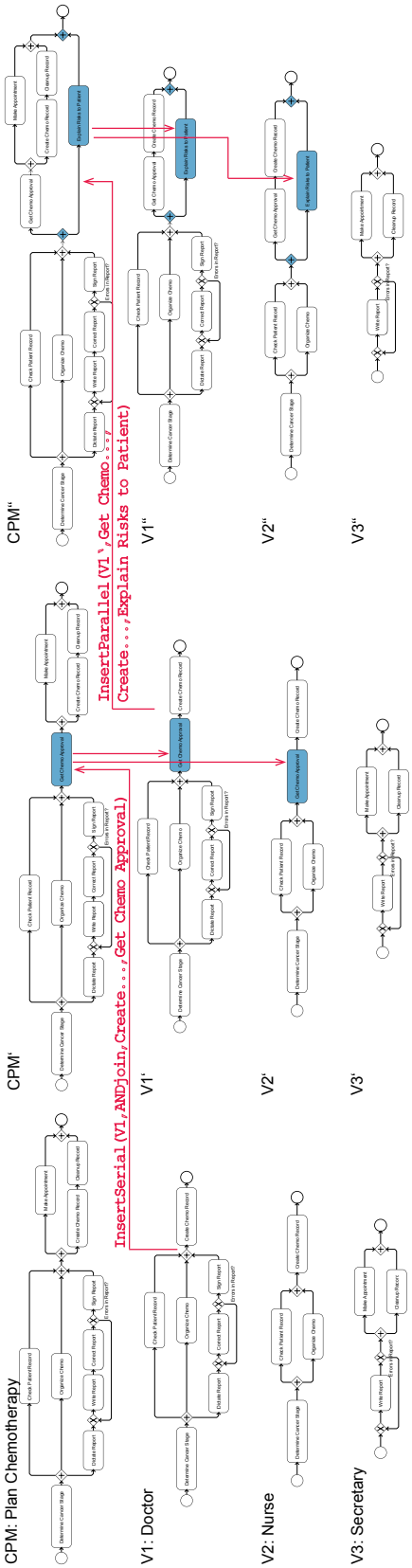


Figure 12.6: Planning a Chemotherapy - Applying View Update Operations

such a refactoring might violate control flow correctness. Finally, the case studies show that the same process models for individual user roles can be created utilizing view creation operations as manually created ones by the process designers in the case studies.

Applying view update operations show that less modeling steps are required to perform the respective update compared to changes directly applied to the original process models. Furthermore, it can be seen that the propagation and migration behaviour of the framework meets real-world requirements. Depending on the type of view update operation, process model complexity can be strongly increased. This is particularly important when automatically migrating process views since in this case the update is not triggered by that particular user. This may be confusing for end-users.

12.2 Evaluation of Process Representations

The goal of this section is to show that process models visualized in terms of the introduced process representations (cf. Chapter 9) are easier to understand, create, and update than state-of-the-art process representations. In this context, experimental research is indispensable. In particular, each presented process representation has to be evaluated against BPMN since the latter is the de-facto standard for process modeling. Furthermore, process representations have to be compared with each other in order to get insights, which fits best for users. In the context of verbalized process descriptions, [231] has already shown promising results that understanding verbalized process descriptions performs better than using BPMN. However, comparing the presented process representations with each other requires numerous experiments and a large set of subjects to get significant results.

In this thesis, an experiment (i.e., Experiment 3) is introduced that evaluates whether process models visualized through the hierarchical representation (cf. Section 9.2) are easier to understand, create, and update than process models visualized using BPMN. However, the experiment setting is designed in a generic way to be applicable to the other process representations as well. To be more precise, the following research question is investigated in Experiment 3 of this thesis:

Is the process model represented in terms of the hierarchical representation easier to understand, create, and update in comparison to BPMN?

In the following, Section 12.2.1 introduces the experiment setting applied in Experiment 3. Section 12.2.2 provides experiment analysis and results, which are further discussed in Section 12.2.3.

12.2.1 Experiment Setting

The goal of the experiment is described by the Goal Definition Template as presented in [310] and shown in Table 12.4. Taking this goal definition, the experiment evaluates both BPMN and hierarchical representation in respect to their ability to visualize and evolve (i.e., create and update) process models, i.e., the model representing CPMs or process views. The experiment is designed as a *randomized balanced single factor experiment* [310]. It is *randomized* since subjects are assigned to factor levels randomly. Furthermore, only one *single factor* varies, i.e., the process representation.

Analyze	*BPMN and hierarchical representation*
for the purpose of	*evaluating*
with respect to their	*effort to understand, create, and update*
from the point of view of	*the researchers and developers*
in the context of	*students and research assistants.*

Table 12.4: Goal Definition Template

Based on the experiment goal, three hypotheses may be derived. Hypothesis H_1 refers to the creation of a process model based on the given process representation. In turn, hypothesis H_2 addresses differences when updating a process model in either one of the two process representations. Finally, hypothesis H_3 emphasises the comprehensibility of a process model depending on the given process representation. Table 12.5 summarizes the three hypotheses.

*Does the process representation have an effect on **creating** process models*
$H_{1,0}$: There is no significant difference in creating process models either based on the hierarchical representation or BPMN. $H_{1,1}$: There is a significant difference in creating process models either based on the hierarchical representation or BPMN.
*Does the process representation have an effect on **updating** process models?*
$H_{2,0}$: There is no significant difference in updating process models either based the hierarchical representation or BPMN. $H_{2,1}$: There is a significant difference in updating process models either based the hierarchical representation or BPMN.
*Does the process representation have an effect on **understanding** process models?*
$H_{3,0}$: There is no significant difference in understanding process models either based the hierarchical representation or BPMN. $H_{3,1}$: There is a significant difference in understanding process models either based the hierarchical representation or BPMN.

Table 12.5: Experiment 3: Hypotheses Formulation

Reflecting hypotheses H_1, H_2, and H_3, the experiment is divided into Parts A-C (cf. Figure 12.7). Each of them addresses one of the three hypotheses. In the following, *subjects*, *object*, *factor levels*, and *response variables* of the experiment as well as its *instrumentation* and *data collection procedure* are described.

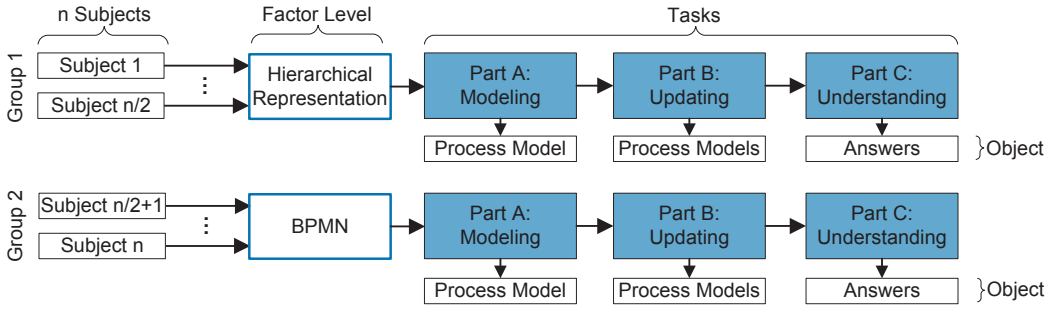


Figure 12.7: Experiment Overview

Subjects. Usually, domain experts in companies only have limited process modeling knowledge [356]. Hence, from the subjects we demand that they are at least moderately familiar with process modeling, but we do not require expert level.

Objects. As objects to be evaluated by each subject, we consider three process models. Each of them is related to one hypothesis (cf. Table 12.5). In Part A, a process model for a simplified credit application shall be created taken the process description from Table 12.6.

The credit application starts with the creation of a written credit application by the customer. After submitting the application, customer data is recorded by the clerk in three steps. Therefore, the customer residence, financial situation, and his marital status is entered in a user form. When capturing the financial situation, the customer has to state his income and expenses. *If the customer is married, it must be checked whether he has children. In this case their names must be recorded as well. Finally, the credit institute checks the creditworthiness of the customer, which has two possible results: the creditworthiness is positive or negative. In the former case, the bank may create a credit agreement, which is signed by the customer afterwards. In the latter case, the credit institute rejects the application or offers a different credit application.*

Table 12.6: Textual Description of the Credit Application Process

In Part B of the experiment, subjects must perform six updates on a given process model (cf. Figure E.1). Table 12.7 summarizes the task descriptions to perform respective updates. Each update is applied separately on the same process model, i.e., the updates do not rely on each other and can be applied independently from each other.

Task	Task Description
B-1	Insert activity X parallel to activity H.
B-2	Insert activity X as alternative to activities B and M.
B-3	Insert activity X parallel to activity B.
B-4	Delete activity L.
B-5	Move the XOR branching containing activities K, L, and M after activity C.
B-6	Parallelize activities I and J.

Table 12.7: Task Descriptions for Part B

Part C of the experiment addresses the understanding of a process model. Subjects are asked ten questions related to a given process model (cf. Figure E.1). More precisely, five questions deal with the understanding of precedence relations. In turn, the other five questions refer to the validity of execution traces[1] (cf. Table 12.8).

Factor and Factor Levels. The factor considered in our experiment is the *process representation* with levels *BPMN* and *hierarchical representation*. Accordingly, process models of Part A-C are provided utilizing the process representation of the respective factor level.

Response Variables. As response variables for all parts of the experiment, we consider the *duration* (in minutes) a subject needs to fulfill the task as well as the *cognitive effort* perceived by the subjects. The latter is rated by each subject on a 7-point Likert scale (i.e., 1=extremely low mental effort, 7=extremely high mental effort).

[1] An introduction of execution traces is given in Section 6.4.

Task	Question
C-1	Which activities are executed after activity C?
C-2	Which activities may be executed in parallel to activity F?
C-3	Which activities may be executed alternatively to activity L?
C-4	Which activities are executed after activity J?
C-5	Which activities are executed before activity G?
C-6	Is <A,L,M,N> a valid execution trace of the process model?
C-7	Is <A,D,E,F,H,G,N> a valid execution trace of the process model?
C-8	Is <A,E,B,F,C,N> a valid execution trace of the process model?
C-9	Is <A,K,L,N> a valid execution trace of the process model?
C-10	Is <A,B,H,G,E,C,F,N> a valid execution trace of the process model?

Table 12.8: Task Descriptions of Part C

Additionally, in Part A *semantic quality* is measured. The latter is measured in respect to a reference process model and is rated by two modeling experts in a consensus building process based on a 4-point scale (cf. Table 12.9). Finally, in Part A the number of modeling steps (i.e., number of process elements inserted, deleted, and moved) required to solve the task is measured. In Parts B and C, a response variable measures whether or not a task is completed correctly.

Value	Name	Description
3	optimal	Activities and control flow are both correct.
2	good	All required activities are present; control flow not entirely correct.
1	fair	Multiple activities are combined to one compared to the reference model.
0	low	Multiple activities are combined to one and control flow is not entirely correct.

Table 12.9: Values for Semantic Quality

Instrumentation. To precisely measure the response variables in a non-intrusive manner, we use the *Cheetah Experimental Platform (CEP)* [357]. The latter provides a process modeling tool designed to perform user experiments. In particular, CEP already provides a BPMN-based modeling environment that records all modeling steps together with their attributes like timestamp and type of modeling action. In the context of this thesis, CEP is extended with a modeling environment that also supports the hierarchical representation [358]. All resulting process models as well as all logged data are stored in a database. This not only allows replaying modeling sessions, but also calculating the response variables. Finally, CEP provides questionnaires to request demographic data and qualitative feedback from the subjects.
The experiment further utilizes the *think aloud* method, i.e., subjects are motivated to talk about what they are doing [312]. Additionally, the user screen is recorded with a video camera. These video recordings are then used to obtain an understanding on emerging problems as well as modeling progress. Note that this setting is the same as the one described in Figure 10.1.

12.2.2 Experiment Operation and Analysis

Students and research assistants familiar with process modeling are invited to join the experiment. Subjects are not informed about the aspects to be investigated and are randomly assigned to one of the groups representing the respective factor levels. For all subjects, anonymity is guaranteed. Before conducting the experiment, a pilot study is performed. Its results are then used to eliminate ambiguities and misunderstandings as well as to improve task descriptions.

The experiment is executed in a seminar room at Ulm University. All in all eight subjects participate in the experiment. According to the experiment setting (i.e., think aloud method) only one subject participates in the experiment at a particular point in time. Each session takes about 50 minutes and runs as follows:
First of all, the procedure of the experiment is explained to the subject. Then, the subject is randomly assigned to one of the two factor levels (cf. Section 12.2.1). For subjects assigned to the hierarchical representation a tutorial is given introducing its elements and syntax. Furthermore, an overview sheet is handed out to the subject summarizing the elements of the process representation. Following this, the subject fills out a questionnaire capturing data about the actual modeling experience as well as demographic information. Afterwards, the experiment is started and the subject works on the tasks of Parts A, B, and C (cf. Section 12.2.1). After the completion of each task, the subject is asked about the perceived mental effort.

After completing the experiment, the collected data is validated. All collected data is valid, i.e., no data of any subject has to be discarded. In particular, all subjects are familiar with process modeling, which is a requirement for the subjects (cf. Table 12.10). Subjects S1-S4 apply the hierarchical representation and subjects S5-S8 apply BPMN.

	Subject	**Since how many years have you been modeling processes?**	**How many process models did you create or edit during the last 12 months?**	**How many months ago did you start using hierarchical representation/BPMN?**
Hierarch.	S1	7	15	0
	S2	4	100	0
	S3	3	50	0
	S4	12	100	0
BPMN	S5	1	100	12
	S6	1	50	10
	S7	4	25	36
	S8	6	100	2

Table 12.10: Experience of Subjects

Results of Part A are summarized in Table 12.11. The semantic quality of all created process models is between good and fair (i.e., median is 1.5). In particular, the median of semantic quality of the process models created using the hierarchical representation is 2. Hence, it is slightly better when using the BPMN representation (i.e., median is 1). Furthermore, subjects using the hierarchical representation need more modeling steps to create a process model, which results in an average of 183.5 movements of process elements on the drawing screen. Hence, the average modeling duration to create BPMN process models is shorter (average of 13 minutes) than using hierarchical representation (average of 14.75 minutes). Finally, the median of perceived mental effort is 4 using hierarchical representation, which corresponds to *neither high nor low mental effort*. The median for BPMN is 3.5. Despite all subjects use the hierarchical representation the first time (cf. Table 12.10), they perform similarly to the subjects already familiar with BPMN. One subject applying the hierarchical representation states *"It was not too complicated, but requires a certain [mental] effort when applying it the first time"*.

Updating existing process models in Part B of the experiment shows ambivalent results (cf. Table 12.12). Updates of process models based on the hierarchical representation are performed

	Subject	Semantic Quality	# Modeling Steps				Duration	Mental Effort
			Total	Created	Moved	Deleted		
Hierarch.	S1	2	250	24	232	2	15min	4
	S2	0	213	23	184	6	19min	4
	S3	2	230	23	205	2	13min	4
	S4	3	126	13	113	0	12min	4
BPMN	S5	1	28	24	4	0	10min	4
	S6	1	77	21	54	2	17min	3
	S7	2	70	31	35	4	16min	6
	S8	1	47	21	26	0	9min	3

Table 12.11: Experiment Results for Part A

on average faster compared to BPMN. In the video recordings, for example, one subject who uses the hierarchical representation stated after reading the task description: *"Ok, this is a more complex update... hm, first, I have to insert task 'X', then, a choice relation, and it seems to be done"*. As can be seen, after reading the task description the subject thinks it requires a lot of modeling effort to perform the update and then realizes that only two model changes (i.e., *insert task 'X'* and *insert choice relation*) are required. Another subject states: *"I just have to insert an edge and the update is done"*. However, some subjects do not like the way how CEP realizes the hierarchical representation. For example, *"Placement of unconnected edges [on the modeling canvas] would make modeling easier"* or *"It is difficult to change the type of a task"*. In turn, deleting an activity (cf. Task B-4, Table 12.12) is faster when using BPMN. Furthermore, both subject groups provide solely correct solutions. Finally, perceived mental effort tends to be lower for hierarchical representation.

Task	Average Duration [min]		# Correct Solutions		Median of Mental Effort	
	Hierarch.	BPMN	Hierarch.	BPMN	Hierarch.	BPMN
B-1	61.5	77.5	4	4	2.5	2.5
B-2	85.3	116.5	4	4	3.5	4
B-3	42.8	53.8	4	4	2	3
B-4	44.0	22.3	4	4	3	2
B-5	58.5	116.8	4	4	3	4
B-6	16.0	46.0	4	4	1.5	2
C-1	59.8	78.8	3	3	3	3.5
C-2	67.3	47.3	4	4	3.5	3
C-3	71.0	51.3	3	1	4	3
C-4	32.3	23.8	4	4	2	2
C-5	50.3	36.5	4	3	4	2.5
C-6	38.0	29.3	3	4	2.5	3
C-7	47.3	39.0	4	4	3	4
C-8	51.8	53.5	3	4	3	4
C-9	28.0	21.3	4	4	2.5	4
C-10	54.3	40.8	4	4	3	4

Table 12.12: Experiment Results for Part B and Part C

Understanding process models in Part C takes more time for most of the tasks when using the hierarchical representation. Analyzing the number of correct answers does not result in a clear tendency. Video recordings indicate for the hierarchical representation that the tree structure is used by subjects to analyze the process model: *"The complete sub-tree over there is not executed"*, *"This sub-tree is executed in parallel to that sub-tree"* or *"Before task G this sub-tree is executed"*. Hence, the tree structure is utilized to create chunks in subject's mind [204].

In particular, Task C-3—finding activities executed alternatively to activity L—seems to be very difficult when using BPMN (i.e., only one correct answer). In case of BPMN, perceived mental effort is lower when analyzing control flow dependencies (i.e., Tasks C-1 to C-5) and higher when checking execution traces (i.e., Tasks C-6 to C-10) compared to the hierarchical appearance.

Analyzing recorded videos in detail underlines the results reflected by the response variables, i.e., both representations perform similarly. However, some of the subjects state that a larger screen resolution or smaller icons for the hierarchical representation might be good since *"it is annoying to search the correct task on the screen [by scrolling around]"*.

12.2.3 Discussion

The experiment compares BPMN and the hierarchical representation. In particular, subjects have to create, update, and understand process models using the two process representations. A comparison of the two process representations has shown that both perform similarly regarding creating, updating, and understanding. It is noteworthy that subjects having no background with the hierarchical representation achieve similar results compared to the subjects that have been familiar with BPMN for a long time. In the video recording one subject states *"Modeling performs pretty good"*. In particular, it seems that certain updates can be performed easier in the hierarchical representation, e.g., parallelizing activities.

However, we must not generalize these results for several reasons: Students and research assistants participate in the experiment and results cannot be simply transferred to domain experts in companies. Furthermore, the instrumentation (i.e., CEP) may have an effect on the outcome. As already stated some subjects claim for improvements in the modeling environment. Due to the small number of subjects participating in the experiment we are not able to prove hypotheses $H1$, $H2$, and $H3$. However, based on the experiment results, for all hypotheses we cannot see any tendency that there is a significant difference between both process representations, i.e., null hypotheses might be confirmed.

12.3 Evaluation of Multi-Touch Gestures

Experiment 4 evaluates whether the developed multi-touch gestures (cf. Chapter 10) are suitable for users. Therefore, users shall apply the developed gesture set to interact and update process models. To be more precise, the following research question is investigated:

Is the multi-touch gesture set appropriate to update process models and to interact with them?

In order to answer this research question, we conduct a user experiment. Section 12.3.1 introduces the experiment setting. Section 12.3.2 provides experiment analysis and results, which are further discussed in Section 12.3.3.

12.3.1 Experiment Setting

The goal of the experiment is described by the Goal Definition Template as presented in [310] and is shown in Table 12.13.

Analyze	*multi-touch gesture set*
for the purpose of	*evaluating*
with respect to their	*appropriateness to update and interact with process models*
from the point of view of	*the researchers and developers*
in the context of	*students and research assistants.*

Table 12.13: Goal Definition Template

Taking this goal definition, the experiment is designed as a *single object study* [310, 359]. This means that all subjects evaluate the same object (i.e., the same gesture set). Based on the goal of the experiment, we derive the following hypothesis (cf. Table 12.14).

Is the selected multi-touch gesture set intuitive to update process models?
$H_{1,0}$: The multi-touch gesture set is not intuitively used by users to update process models.
$H_{1,1}$: The multi-touch gesture set is intuitively used by users to update process models.

Table 12.14: Experiment 4: Hypothesis Formulation

In the following, we introduce *subjects*, *object*, *factor levels*, *response variables*, and *instrumentation* of the experiment.

Subjects. Usually, domain experts in companies only have limited knowledge in process modeling [356]. Hence, from the subjects we demand that they are at least moderately familiar with process modeling, but we do not require expert level. Moreover, we do not require from subjects to be aware of multi-touch devices or to have used a (multi-touch) process modeling tool before.

Object. As object, a BPMN-based process model is used, which not only comprises control flow elements, but data elements as well. Subjects must apply 14 updates on the process model. Each update may be accomplished with one gesture from the multi-touch gesture set. Table 12.15 shows the task descriptions applied. The initial process model is shown in Figure E.2.

Task	**Task Description**
T1	Insert a synchronization edge between *Calculate Risk* and *Check Credit Protection Agency.*
T2	Aggregate the AND branching block.
T3	Insert a branch in the first XOR branching block.
T4	Insert an XOR branching block comprising activities *1st Review* and *2nd Review.*
T5	Open the help dialog.
T6	Insert an activity between activities *1st Review* and *2nd Review.*
T7	Rename the inserted activity. Choose any activity label.
T8	Delete the renamed activity.
T9	Select activity *Load Customer Data.*
T10	Reduce activity *Fill Out Credit Request.*
T11	Undo the last performed action.
T12	Insert a new data object to the process model.
T13	Insert a data edge from data object *CustomerPhone* to activity *Load Customer Data.*
T14	Create a process view including the first XOR branching block.

Table 12.15: Task Descriptions

Response Variables. Two response variables are measured during Experiment 4. First, the *duration* a subject needs to solve a task is measured. Second, it is measured whether a subject s the task *correctly.* If a subject cannot remember a multi-touch gesture and needs to ask the

investigator, the task is not considered as being correctly solved. If a subject remembers the respective gesture, however, it constitutes a correctly solved task.

Instrumentation. Initially, a paper-based questionnaire is used to document demographic data. Afterwards, subjects participate in a tutorial to practice all multi-touch gestures required. For this purpose, a special tutorial application is implemented (cf. Figure 12.8): on the left side it shows the respective gesture and on the right side subjects are able to apply the displayed multi-touch gesture on a process model. If the gesture is correct, respective feedback is given to the subject. If the subject believes that it remembers the gesture well, he or she clicks "Next" in the tutorial application and the next multi-touch gesture is presented.

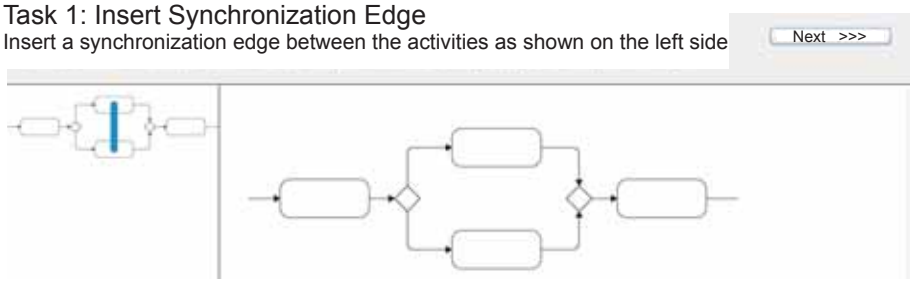


Figure 12.8: Tutorial Application

After completing the tutorial, the *proViewClient* (cf. Section 11.4) is loaded and the initial process model (cf. Figure E.2) is presented to the user. Task descriptions are displayed below the process model. Both, the tutorial application and the *proViewClient* are provided on a Google Nexus 7 tablet to subjects.

The display of the tablet as well as spoken comments are recorded by a video camera. This *think aloud* experiment setting helps us to analyze results in detail [312]. Note that the same experiment setup as shown in Figure 10.1 is used.

12.3.2 Experiment Operation and Analysis

In total, 11 students and research assistants familiar with process modeling are invited to join the experiment. Subjects are not informed about the aspects we want to investigate. Furthermore, anonymity is guaranteed to them. Additionally, three test runs are conducted, which results in minor improvements of the instrumentation as well as in task descriptions.

The experiment is conducted in a seminar room at Ulm University. Each session runs about 30 minutes. After all subjects participated in the experiment, the collected data is validated. All collected data is valid and can be further analyzed. In particular, subjects are familiar with process modeling and have per average 3.11 years experience in process modeling. Nine subjects own a smartphone and six a tablet.

As a result, in average 8.6 of 11 subjects remember the trained multi-touch gestures and, hence, solve the respective tasks correctly (cf. Table 12.16). Six gestures are remembered by all subjects (i.e., Tasks T1, T2, T5, T6, T13, and T14). In the video recordings, for example, subjects state for these gestures: *"I simply need to draw a question mark to open the help dialog"* or *"... to insert a new branch, I need to draw an edge"*. Afterwards, they directly started performing the gesture without any break.

However, only one subjects applies the gesture for reducing an activity (i.e., Task T10) correctly. In the video recordings, subjects state that the gesture is hard to remember. For example, *"I cannot remember the gesture anymore"*. After the investigator explained the gesture to that subject, the subject says: *"Well, it is actually a good gesture, but I should have remembered it"*. Other subjects apply the gesture for deleting process elements instead: *"I have to draw a rectangle with an 'X' to reduce that element"*.

Task	**# Correctly Solved**	**Average Duration [sec]**
T1	11	37.5
T2	11	39.0
T3	9	32.6
T4	6	60.9
T5	11	48.6
T6	11	29.6
T7	7	25.5
T8	8	28.7
T9	10	16.6
T10	1	40.9
T11	8	22.1
T12	5	28.1
T13	11	28.6
T14	11	56.7
All Tasks	8.6	35.6

Table 12.16: Results of Experiment 4

The average duration required to perform a task is 35.6 seconds. Note that this duration includes reading the task, remembering the gesture, and applying the latter. Applying the gestures for inserting a branching block and creating a process view take the longest time (i.e., 60.9 and 56.7 seconds respectively). In the video recordings, it can be seen that subjects first identified the process elements to be enclosed by the branching block or process view before starting the gesture. For example, one subject states *"OK, this area* [*pointing with his finger*] *is meant... then... I have to select these elements"*. Furthermore, subjects perform these gestures very carefully (and slow respectively) in order to include the correct elements.

By contrast, the gesture to undo the last action (i.e., Task T11) only requires 22.1 seconds. One subject states *"Undo is symbolized by a line like this..."* and draws a gesture on the desk to recall it. Furthermore, drawing a data edge requires 28.1 seconds and seems to be easy for subjects to perform. For example, *"Read data edges... is a line from this to that element"*.

Some subjects struggle when drawing the question mark to open the help dialog or when drawing the 'X' to delete an element: *"OK, this way the detection works"* or *"when I start here, it works"*. These subjects remember the gesture, but the detection algorithm performs not as expected.

12.3.3 Discussion

Experiment 4 evaluates the presented multi-touch gesture set for interacting with process models. We first introduce the gesture set to subjects. Then, we ask them to apply the gestures to a process model. Most of the gestures are well remembered by subjects. Some subjects enjoyed updating process models using multi-touch gestures: *"I am absolutely ecstatic"* or *"pretty cool"*. However, the gesture to reduce activities should be revised since only one subject applied it correctly. Furthermore, creating a process view and inserting a branching block requires much time to apply. Hence, the respective gestures should also be revised since gestures should not take too much time. Furthermore, the detection algorithm and gesture templates should be improved to increase the detection accuracy.

In order to obtain generalizable results, the experiment needs to be repeated with domain experts from practice. Furthermore, a higher number of subjects are required to obtain statistical evidence, i.e., the hypothesis can neither be rejected nor confirmed. However, this experiment indicates that the presented multi-touch gesture set is appropriate for users and can be intuitively applied by them.

12.4 Summary

This chapter evaluates the concepts and algorithms developed in this thesis. This evaluation is based on the proof-of-concept prototype described in Chapter 11. To be more precise, we first perform case studies based on process models from practice to evaluate view creation and view updates concepts. In order to measure the abstraction of view creation and effects of view updates, well-known process metrics are applied. As a result, we identify the need for further refactoring rules. Furthermore, process model understandability may be decreased when applying certain view update operations. Finally, for some updates less modeling effort is required compared to original process models of the case studies.

An experiment evaluating process representations is presented as well. For this purpose, the hierarchical representation and BPMN are compared regarding their appropriateness to create, update, and understand process models. Although subjects apply the hierarchical representation the first time, they perform similar to the ones using BPMN. However, further experiments are required to compare process representations among each other and with BPMN in more detail.

Finally, another experiment is conducted to evaluate introduced multi-touch gestures. In particular, users are trained in using the presented multi-touch gesture set on a tablet. In a second phase, they have to recall the respective gestures and apply them on a process model. As a result, we identify a few gestures, which have to be replaced since they are too hard to perform and remember. However, most of the defined gestures seem to be suitable for users.

Altogether, we show that the concepts and algorithms are working well and also are suitable to practical application scenarios.

Part V

Conclusion

13
Summary and Outlook

This thesis contributes the *proView* framework enabling domain experts to create and evolve large and complex process models based on process views, alternative process representations, and process interaction concepts. Process views provide personalized abstractions on process models and allow domain experts for process updates. Then, a set of process representations are introduced to assist domain experts in comprehending and evolving process models. Finally, a set of multi-touch gestures is presented to provide an intuitive way for interacting with process models. The concepts and algorithms are designed in a modular way, i.e., process views, process representations, and multi-touch gestures are orthogonal. In the following, we discuss the contribution in more detail:

Creating Process Views. View creation operations allow hiding (i.e., reducing) process elements of the original process model. In addition, process elements may be aggregated to provide an abstracted view on them. In particular, respective operations consider the control flow perspective as well as the data flow and process attributes. Introduced view creation operations guarantee control flow correctness, i.e., after applying respective operations, a correct process model results. This is crucial to enable domain experts to perform updates, while not distorting the "meaning" of a process model. Introduced view creation operations may be combined to high-level view creation operations providing a semantically enriched way to create process views. Utilizing these operations, a more convenient way for domain experts is provided to create process views. Finally, the *proView* framework enables to create flexible process hierarchies based on introduced view creation operations. This enables us to define a set of process hierarchies based on the same underlying process model.

Updating Process Views. View update operations are introduced to allow domain experts to evolve complex process models based on process views. Consequently, domain experts need not to understand complex process models, but may directly perform updates on related process views. Furthermore, view updates are automatically *propagated* to the corresponding process model. Then, all associated process views are *migrated* to the new process model version.

View-based process updates consider the control flow perspective as well as data flow and process attributes. Furthermore, elementary view update operations may be combined to advanced view update operations to enable high-level changes.

Visualizing Process Models. Process representations are presented to visualize and update process models. Further, domain experts are able to dynamically switch between the process representations on the fly. In total, three process representations are introduced: *hierarchical*, *form-based*, and *verbalized*.

The *hierarchical representation* visualizes a process model in terms of a tree and allows for a top-down exploration of process models. Furthermore, in certain cases less modeling steps are required to perform updates based on the hierarchical representation than for BPMN.

The *form-based representation* visualizes a process model in terms of nested rectangles. Particularly, the number of different symbols required to visualize a process model can be decreased. Furthermore, due to its structure updates preserve the control flow correctness.

Finally, the *verbalized process description* represents process models as textual documents. Particularly, domain experts must not be aware of any process modeling language. Furthermore, changes in the textual description are interpreted as updates on the corresponding process model.

Interacting with Process Models. Process interaction is supported by a set of multi-touch gestures, which allow domain experts to interact with process models utilizing touch-enabled devices. In particular, this gesture set results from empirical research. The presented multi-touch gestures enable domain experts to perform updates on process models as well as to create process views. Further gestures allow, for example, to undo modeling actions or select process nodes.

This thesis has unveiled aspects that may be addressed by further research:

- View-based process updates require proper concepts to handle concurrent updates. Thereby, concurrency control concepts known from database management systems may be applied. Particularly, as opposed to most database applications updates on process models are rather rare and it may be appropriate to lock a process model or parts of it during such an update.
- Process views may be applied in the context of process variants [26]. A process variant, for example, describes a business process of a specific country or product. All variants of a business process may be documented in one process model (i.e., CPM) and process views may be used to visualize individual process variants. Therefore, specific operations to create process views are required. The specific semantic of a process view might have an impact on update propagation and migration.
- Run-time data (e.g., progress of process execution) must be visualized in process views as well as process representations when executing underlying process models. In this context, it has to be considered that a lack of execution states (e.g., through reduced activities) in process views might exist. Furthermore, applying view update operations to process instances (i.e., executed process models) arise new challenges. For example, it has to be checked whether or not an update is allowed. Finally, in order to support domain experts to perform ad-hoc adaptions [104], process views may be applied to reduce complexity of such adaptation tasks.

- Navigation approaches provide interaction support for domain experts to determine required information in process model collections and process models respectively [248]. Process views may complement such approaches to provide personalized abstractions on process model collections and allow for process updates. Furthermore, process-oriented information logistics may be applied to automatically create such personalized process model collections by utilizing context-specific information of domain experts [360].
- An initial set of gestures for process interaction is introduced. However, further gestures must be evaluated and defined, for example, in order to pan process models. Further process interaction concepts are required to support collaborative process modeling.
- Further user studies are required to evaluate the presented concepts with practitioners. In particular, domain experts with limited or no process modeling knowledge should be encouraged to apply the presented view creation and update operations. Feedback gained from this should be used to evolve the concepts. Moreover, user studies comparing process representations may help for a deeper understanding of their strengths and weaknesses.

Bibliography

[1] Kolb, J., Rudner, B., Reichert, M.: Towards Gesture-based Process Modeling on Multi-Touch Devices. In: Proc. 1st Int'l Workshop on Human-Centric Process-Aware Information Systems (HC-PAIS'12), Gdansk, Poland (2012) 280–293

[2] Kolb, J., Huebner, P., Reichert, M.: Model-Driven User Interface Generation and Adaptation in Process-Aware Information Systems. UIB 2012-04, Technical Report, Ulm University (2012)

[3] Kolb, J., Reichert, M., Weber, B.: Using Concurrent Task Trees for Stakeholder-centered Modeling and Visualization of Business Processes. In: Proc. 4th Int'l Conf. S-BPM ONE 2012, CCIS 284. (2012) 237–251

[4] Reichert, M., Kolb, J., Bobrik, R., Bauer, T.: Enabling Personalized Visualization of Large Business Processes through Parameterizable Views. In: Proc. 26th Symposium On Applied Computing (SAC'12), Riva del Garda (Trento), Italy, ACM (2012) 1653–1660

[5] Kolb, J., Kammerer, K., Reichert, M.: Updatable Process Views for User-centered Adaption of Large Process Models. In: Proc. 10th Int'l Conf. on Service Oriented Computing (ICSOC'12), Shanghai, China (2012) 484–498

[6] Kolb, J., Kammerer, K., Reichert, M.: Updatable Process Views for Adapting Large Process Models: The proView Demonstrator. In: Proc. 10th Int'l Conf. on Business Process Management (BPM'12), Demonstration Track, Tallinn, Estonia (2012) 6–11

[7] Kolb, J., Hübner, P., Reichert, M.: Automatically Generating and Updating User Interface Components in Process-Aware Information Systems. In: Proc. 10th Int'l Conf. on Cooperative Information Systems (CoopIS'12). (2012) 444–454

[8] Kolb, J., Reichert, M.: Data Flow Abstractions and Adaptations through Updatable Process Views. In: Proc. 27th Symposium on Applied Computing (SAC'13), Coimbra, Portugal, ACM (2013) 1447–1453

[9] Kolb, J., Reichert, M.: Supporting Business and IT through Updatable Process Views: The proView Demonstrator. In: Proc. 10th Int'l Conference on Service Oriented Computing (ICSOC'12), Demonstration Track, Shanghai, China (2013) 460–464

[10] Kolb, J., Reichert, M.: A Flexible Approach for Abstracting and Personalizing Large Business Process Models. ACM Applied Computing Review **13**(1) (2013) 6–17

[11] Lanz, A., Kolb, J., Reichert, M.: Enabling Personalized Process Schedules with Time-aware Process Views. In: Proc. 25th CAiSE 2013 Workshops, 2nd Int'l Workshop on Human-Centric Information Systems (HCIS 2013), Valencia, Spain (2013) 205–216

[12] Kolb, J., Leopold, H., Mendling, J., Reichert, M.: Creating and Updating Personalized and Verbalized Business Process Descriptions. In: Proc. 6th IFIP WG 8.1 Working Conference on the Practice of Enterprise Modeling (PoEM'13), Riga, Latvia (2013) 191–205

[13] Kolb, J., Rudner, B., Reichert, M.: Gesture-based Process Modeling Using Multi-Touch Devices. Int'l Journal of Information System Modeling and Design (IJISMD) **4**(4) (2013) 48–69

[14] Kolb, J., Zimoch, M., Weber, B., Reichert, M.: How Social Distance of Process Designers Affects the Process of Process Modeling: Insights From a Controlled Experiment. In: Proc. 29th Symposium on Applied Computing (SAC'14), Gyeongju, Korea, ACM (2014) 1364–1370

[15] Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies. Springer (2012)

[16] Weber, B., Reichert, M., Wild, W., Rinderle, S.: Life Cycle Support in Process-Aware Information Systems. International Journal of Cooperative Information Systems **18**(1) (2009) 115–165

[17] Davies, I., Green, P., Rosemann, M., Indulska, M., Gallo, S.: How Do Practitioners Use Conceptual Modeling in Practice? Data & Knowledge Engineering **58**(3) (September 2006) 358–380

[18] Weber, B., Reichert, M.: Refactoring Process Models in Large Process Repositories. In: Proc. 20th Int'l Conf. Advanced Information Systems Engineering (CAiSE'08), Montpellier, France (2008) 124–139

[19] Mendling, J.: Detection and Prediction of Errors in EPC Business Process Models. PhD Thesis, Vienna University of Economics and Business Administration (WU Wien) (2007)

[20] Sedera, W., Gable, G.G., Rosemann, M., Smyth, R.: A Success Model for Business Process Modeling: Findings From a Multiple Case Study. In: Proc. 8th Pacific Asia Conference on Information Systems, Shanghai, China (2004) 485–498

[21] Davis, R.: Business Process Modelling with ARIS - A Practical Guide. Springer (2003)

[22] Scheer, A.W.: ARIS-House of Business Engineering. IWI Hefte **133** (1996) 1–34

[23] Buchwald, S., Bauer, T., Reichert, M.: Bridging the Gap Between Business Process Models and Service Composition Specifications. In: Service Life Cycle Tools and Technologies: Methods, Trends and Advances. IGI Global (2011) 124–153

[24] Buchwald, S.: Erhöhung der Durchgängigkeit und Flexibilität prozessorientierter Applikationen mittels Service-Orientierung. PhD Thesis, Ulm University (2012)

[25] Weidlich, M., Barros, A., Mendling, J., Weske, M.: Vertical Alignment of Process Models - How Can We Get There? In: Proc. 10th Int'l Workshop on Enterprise, Business-Process and Information Systems Modeling. Volume 29. Springer (2009) 71–84

[26] Hallerbach, A.: Management von Prozessvarianten. PhD Thesis, Ulm University (2009)

[27] Ayora, C., Torres, V., Weber, B., Reichert, M., Pelechano, V.: VIVACE: A Framework for the Systematic Evaluation of Variability Support in Process-Aware Information Systems. Information & Software Technology **57** (2015) 248–276

[28] Hallerbach, A., Bauer, T., Reichert, M.: Managing Process Variants in the Process Life Cycle. Technical report, Centre for Telematics and Information Technology, University of Twente, Enschede (2007)

[29] Hallerbach, A., Bauer, T., Reichert, M.: Guaranteeing Soundness of Configurable Process Variants in Provop. In: Proc. 11th IEEE Conf. on Commerce and Enterprise Computing (CEC'09), Vienna, Austria (July 2009) 98–105

[30] DeBruin, T.: Insights into the Evolution of BPM in Organisations. In: Proc. 18th Australasian Conference on Information Systems, Toowoomba, Australia (2007) 632–642

[31] Smirnov, S.: Business Process Model Abstraction. PhD Thesis, HPI Potsdam (2012)

[32] Bobrik, R.: Konfigurierbare Visualisierung komplexer Prozessmodelle. PhD Thesis, Ulm University (2008)

[33] Streit, A., Pham, B., Brown, R.: Visualization Support for Managing Large Business Process Specifications. In: Proc. 3rd Int'l Conf. Business Process Management (BPM'05). (2005) 205–219

[34] CapGemini: Global Business Process Management Report. Technical report (2012)

[35] Harmon, P., Wolf, C.: The State of Business Process Management 2014. Technical report, BPTrends.com (2014)

[36] Indulska, M., Recker, J., Rosemann, M., Green, P.: Business process modeling: Current Issues and Future Challenges. In: Proc. 21st Int'l Conf. on Advanced Information Systems Engineering (CAiSE'09). Number June, Amsterdam, Netherlands (2009) 501–514

[37] Hammer, M., Champy, J.: Reengineering the Corporation: A Manifesto for Business Revolution. Business Horizons **36**(5) (1993) 90–91

[38] Hammer, M., Champy, J.: Reengineering the Corporation. Collins Business Essentials (1993)

[39] Rinderle, S., Reichert, M., Dadam, P.: Flexible Support of Team Processes by Adaptive Workflow Systems. Distributed and Parallel Databases **16**(1) (2004) 91–116

[40] Smith, H., Fingar, P.: Business Process Management: The Third Wave. Meghan-Kiffer Press. (2003)

[41] Hill, J.B., Cantara, M., Plummer, D.: Magic Quadrant for Business Process Management Suites. Technical report, Gartner Research (2009)

[42] Logica Management Consulting: Securing the Value of Business Process Change. Technical report, Whitepaper (2008)

[43] Eikebrokk, T.R., Iden, J., Olsen, D.H., Opdahl, A.L.: Exploring Process-Modelling Practice: Towards a Conceptual Model. In: Proc. 41st Annual Hawaii Int'l Conf. on System Sciences (HICSS 2008), Waikoloa, HI (January 2008) 376–376

[44] Tan, D., Wandke, H.: Process-Oriented User Support for Workflow Applications. In: Proc. 12th Int'l Conf. on Human-Computer Interaction. HCI Applications and Services, Beijing, China (2007) 752–761

[45] Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer Berlin / Heidelberg (2013)

[46] Jones, T., Schulte, W.R., Cantara, M.: Magic Quadrant for Intelligent Business Process Management Suites. Technical Report March, Gartner Inc. (2014)

[47] Barnett, A.G., Holt, M.: Ovum Decision Matrix : Selecting a Business Process Management Solution, 2014. Technical report, Ovum (2014)

[48] Uselmann, A.: Enterprise Mobility mit der SAP Mobile Infrastructure: Untersuchung der Sybase Unwired Platform anhand einer Fallstudie im Bereich Instandhaltung unter Einbezug geografischer Daten. Diploma Thesis, Ulm University (2013)

[49] Hevner, A.R., March, S.T., Park, J., Ram, S.: Design Science in Information Systems Research. MIS Quarterly **28**(1) (2004) 75–105

[50] Bobrik, R., Reichert, M., Bauer, T.: Requirements for the Visualization of System-Spanning Business Processes. Proc. 1st Int'l Workshop on Business Process Monitoring and Performance Management (BPMPM'05) in conjunction with (DEXA'05) (2005) 948–954

[51] vom Broke, J., Thomas, O.: Reference Modeling for Organizational Change: Applying Collaborative Techniques for Business Engineering. In: Proc. of 12th Americas Conference on Information Systems (AMCIS'06), Acapulco, Mexico (2006) 680–688

[52] Scheer, A.W., Nüttgens, M.: ARIS Architecture and Reference Models for Business Process Management. In: Proc. 1st Int'l Conf. on Business Process Management (BPM 2000). (2000) 376–389

[53] Scheer, A.W.: ARIS - Business Process Modeling. Springer (2000)

[54] Moody, D.L.: What Makes a Good Diagram? Improving the Cognitive Effectiveness of Diagrams in IS Development. Advances in Information Systems Development **2** (2007) 481–492

[55] Moody, D.L.: Theoretical and Practical Issues in Evaluating the Quality of Conceptual Models: Current State and Future Directions. Data & Knowledge Engineering **55**(3) (December 2005) 243–276

[56] Spath, D., Weisbecker, A., Kopperger, D., Nägele, R.: Business Process Management Tools 2011. Fraunhofer IAO, Stuttgart, Germany (2011)

[57] Jim Sinur, Schulte, W.R., Hill, J.B., Jones, T.: Magic Quadrant for Intelligent Business Process Management Suites. Technical Report September, Gartner Inc. (2012)

[58] Peffers, K., Tuunanen, T., Rothenberger, M.A., Chatterjee, S.: A Design Science Research Methodology for Information Systems Research. Journal of Management Information Systems **24** (2007) 45–77

[59] Österle, H., Becker, J., Frank, U., Hess, T., Karagiannis, D., Krcmar, H., Loos, P., Mertens, P., Oberweis, A., Sinz, E.J.: Memorandum on Design-oriented Information Systems Research. European Journal of Information Systems **20** (2010) 7–10

[60] Gregor, S.: The Nature of Theory in Information Systems. MIS Quarterly **30** (2006) 611–642

[61] Cole, R., Purao, S., Rossi, M., Sein, M.K.: Being Proactive : Where Action Research meets Design Research. In: Proc. 26th Int'l Conf. on Information Systems (ICIS 2005). (2005) 325–336

[62] Robey, D.: Research Commentary: Diversity in Information Systems Research: Threat, Promise, and Responsibility. Information System Research **7**(4) (1996) 400–408

[63] Kerlinger, F.N.: Behavioral Research: A Conceptual Approach. Holt, Rinehart and Winston New York (1979)

[64] March, S.T., Smith, G.F.: Design and Natural Science Research on Information Technology. Decision Support Systems **15** (1995) 251–266

[65] Silver, M.S., Markus, M.L., Beath, C.M.: The Information Technology Interaction Model: A Foundation for the MBA Core Course. MIS Quarterly **19**(3) (1995) 361–390

[66] Mutschler, B., Reichert, M., Bumiller, J.: Unleashing the Effectiveness of Process-oriented Information Systems: Problem Analysis, Critical Success Factors and Implications. IEEE Transactions on Systems, Man, and Cybernetics, Part C **38**(3) (2008) 280–291

[67] Weber, B., Reichert, M., Mendling, J., Reijers, H.A.: Refactoring Large Process Model Repositories. Computers in Industry **62**(5) (2011) 467–486

[68] Simon, H.A.: The Sciences of the Artificial. MIT Press (1996)

[69] ISO 9000: Quality management systems – Requirements

[70] Kreher, U.: Konzepte, Architektur und Implementierung adaptiver Prozeßmanagementsysteme. PhD Thesis, Ulm University (2014)

[71] Vossen, G., Weske, M.: The WASA Approach to Workflow Management for Scientific Applications. In: Workflow Management Systems and Interoperability. (1998) 145–164

[72] Jablonski, S., Bussler, C.: MOBILE: A Modular Workflow Model and Architecture. In: Proc. of 4th Int'l Working Conference on Dynamic Modelling and Information Systems. (1994)

[73] Mendling, J., Strembeck, M.: Influence Factors of Understanding Business Process Models. In: Proc. 11th Int'l Conf. Business Informations Systems (BIS'08). (2008) 142–153

[74] Mendling, J., Reijers, H.A., Cardoso, J.: What Makes Process Models Understandable? In: Proc. 5th Int'l Conf. on Business Process Management (BPM 2007). (2007) 48–63

[75] Chittaro, L.: Visualizing Information on Mobile Devices. Computer **39**(3) (2006) 40–45

[76] Tran, H.: View-Based and Model-Driven Approach for Process-Driven, Service-Oriented Architectures. PhD Thesis, TU Wien (2009)

[77] Günther, C.: Process Mining in Flexible Environments. PhD Thesis, Technische Universiteit Eindhoven (2009)

[78] Norton, D., Blechar, M., Jones, T.: Magic Quadrant for Business Process Analysis Tools 2010. In: Gartner RAS Core Research Note. Volume 2. (2010) 1–15

[79] Ahukanna, D., de Almeida, V.P.A., Gucer, V., Narain, S., Pham, B., Salem, M., Warkentin, M., Wood, J.K., Xie, Z.Q., Zhang, C.: IBM Business Process Manager Version 8.0 Production Topologies. Technical report (2013)

[80] Seemann, J.: Innovator for Business Analysts. Modeling Magazine **4** (2009) 4–9

[81] Garcia, F., Vizcaino, A., Ebert, C.: Process Management Tools. IEEE Software **28**(2) (2011) 15–18

[82] IBM: IBM Blueworks Live. www.blueworkslive.com (2013)

[83] Kunze, M., Weske, M.: Signavio-Oryx Academic Initiative. In: Proc. 8th Int'l Conf. on Business Process Management, Demonstration Track. (2010) 6

[84] Wyllie, D.: Cubetto BPMN - Geschäftsprozesse auf dem iPad modellieren (in German). Computerwoche (2012)

[85] Orchard ebusiness Pty Ltd.: Orchard Mobile Process Designer, http://orchardprocess.mobi/ (2013)

[86] Tabtou Ltd.: Process Craft, www.showgen.com (2013)

[87] OMG: Business Process Management Notation (BPMN) 2.0 (2010)

[88] Krogstie, J., Sindre, G., Jø rgensen, H.v.D.: Process Models Representing Knowledge for Action: A Revised Quality Framework. European Journal of Information Systems **15**(1) (February 2006) 91–102

[89] Chi, E.H.h.: A Taxonomy of Visualization Techniques using the Data State Reference Model. In: Proc. IEEE Symp. on Information Visualization (InfoVis 2000). Number Table 2, IEEE Press (2000) 69–75

[90] Chi, E.H.h., Riedl, J.T.: An Operator Interaction Framework for Visualization Systems. In: Proc. IEEE Symp. on Information Visualization (InfoVis'98). (1998) 63–70

[91] Schumann, H., Müller, W.: Information Visualization: Techniques and Perspectives (in German). it - Information Technology **46**(3) (2004) 135–141

[92] Gröller, E.: Insight into Data through Visualization. In: Graph Drawing, Vienna, Austria (2002) 352–366

[93] Rosemann, M.: Potential pitfalls of process modeling: part A. Business Process Management Journal **12**(2) (2006) 249–254

[94] Li, C., Reichert, M., Wombacher, A.: Discovering Reference Process Models by Mining Process Variants. In: Proc. 6th Int'l Conf. Web Services. Number 2, Beijing, IEEE (2008) 45–53

[95] van der Aalst, W.M.P., van Dongen, B., Herbst, J., Maruster, L., Schimm, G., Weijters, A.: Workflow Mining: A Survey of Issues and Approaches. Data & Knowledge Engineering **47**(2) (November 2003) 237–267

[96] Li, C., Reichert, M., Wombacher, A.: Mining Business Process Variants - Challenges, Scenarios, Algorithms. Data & Knowledge Engineering, Special Issue on Best BPM'09 **70**(5) (2011) 409–434

[97] La Rosa, M., Dumas, M., Käärik, R., Dijkman, R.: Merging Business Process Models (Extended Version). Technical report, Queensland University of Technology, Brisbane, Australia (2009)

[98] Silberschatz, A., Korth, H., Sudarshan, S.: Database System Concepts. McGraw-Hill (2005)

[99] Ihlenfeld, M.: Implementierung einer Komponente zur Erstellung und Visualisierung von Prozesssichten. Master Thesis, Ulm University (2011)

[100] Weidlich, M., Weske, M., Mendling, J.: Change Propagation in Process Models using Behavioural Profiles. Proc. 6th IEEE Int'l Conf. Services Comp. (2009) 33–40

[101] zur Muehlen, M., Recker, J.: How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation. In: Advanced Information Systems Engineering. Volume 5074 of Lecture Notes in Computer Science. 5074 edn. Springer Berlin Heidelberg, Berlin, Heidelberg (2008) 465–479

[102] Mendling, J., van Dongen, B., van der Aalst, W.M.P.: Getting Rid of OR-joins and Multiple Start Events in Business Process Models. Enterprise Information Systems **2**(4) (2008) 403–419

[103] Reichert, M., Dadam, P.: ADEPTflex - Supporting Dynamic Changes of Workflows Without Losing Control. Journal of Intelligent Information Systems **10**(2) (1998) 93–129

[104] Reichert, M.: Dynamische Ablaufänderungen in Workflow-Management-Systemen. PhD Thesis, Ulm University (2000)

[105] Johnson, R., Pearson, D., Pingali, K.: Finding Regions Fast: Single Entry Single Exit and Control Regions in Linear Time. In: Proc. Conf. on Programming Language Design and Implementation (ACM SIGPLAN'94). (1993)

[106] Gambini, M.: The Design of Graphical Process Modeling Languages: from Free Composition to Modular Construction. PhD Thesis, University of Verona (2012)

[107] Combi, C., Gambini, M., Migliorini, S.: The NestFlow Interpretation of Workflow Control-Flow Patterns. In: Proc. 15th Int'l Conf. Advances in Databases and Information Systems (ADBIS 2011). (2011) 316–332

[108] Dijkstra, E.W.: Notes on Structured Programming. Technological University Eindhoven Netherlands (1972)

[109] Curbera, F., Leymann, F., Storey, T., Ferguson, D.F., Weerawarana, S.: Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More. Prentice Hall (2005)

[110] Dadam, P., Reichert, M.: The ADEPT Project: A Decade of Research and Development for Robust and Flexible Process Support. Computer Science - Research and Development **23**(2) (April 2009) 81–97

[111] Minor, M., Tartakovski, A., Schmalen, D.: Agile Workflow Technology and Case-based Change Reuse for Long-Term Processes. International Journal of Intelligent Information Technologies (IJIIT) **4**(1) (2008) 80–98

[112] Lanz, A., Kreher, U., Reichert, M., Dadam, P.: Enabling Process Support for Advanced Applications with the AristaFlow BPM Suite. In: Proc. 10th Int'l Conf. on Business Process Management 2010 Demonstration Track, Hoboken, NJ, USA (2010)

[113] Koehler, J., Vanhatalo, J.: Process Anti-Patterns: How to Avoid the Common Traps of Business Process Modeling. Technical report, IBM, Zurich (2007)

[114] Reijers, H.A., Mendling, J.: Modularity in Process Models: Review and Effects. In: Proc. 6th Int'l Conf. on Business Process Management (BPM'08), Milan, Italy (2008)

[115] Mendling, J., Neumann, G., van der Aalst, W.M.P.: Understanding the Occurrence of Errors in Process Models based on Metrics. In: Proc. 15th Int'l Conf. Cooperative Information Systems (CoopIS'07), Amantea, Clabria (2007) 113–130

[116] Mendling, J., Reijers, H.A., van der Aalst, W.M.P.: Seven Process Modeling Guidelines (7PMG). Information & Software Technology **52**(2) (February 2010) 127–136

[117] Liu, R., Kumar, A.: An Analysis and Taxonomy of Unstructured Workflows. In: Proc. 5th Int'l Conf. on Business Process Management (BPM 2005). (2005) 268–284

[118] Kiepuszewski, B., ter Hofstede, A.H.M., Bussler, C.: On Structured Workflow Modelling. In: Proc. 12th Int'l Conf. on Advanced Information Systems Engineering, Stockholm, Sweden (2000) 431–445

[119] Mundbrod, N.: Abbildbarkeit unstrukturierter Prozessmodelle auf strukturierte Workflows Eine Untersuchung am Beispiel BPMN und ADEPT. Bachelor Thesis, Ulm University (2008)

[120] Reichert, M., Dadam, P.: A Framework for Dynamic Changes in Workflow Management Systems. In: Proc. 8th Int'l Workshop on Database and Expert Systems Applications, Toulouse, France (1997) 42–48

[121] Reichert, M., Rinderle, S., Dadam, P.: On the Modeling of Correct Service Flows with BPEL4WS. In: Informationssysteme im E-Business und E-Government (EMISA 2004). (2004) 117–128

[122] Leopold, H., Mendling, J., Reijers, H.A.: On the Automatic Labeling of Process Models. In: Proc. 23rd Int'l Conf. on Advanced Information Systems Engineering (CAiSE 2011), London, UK (2011) 512–520

[123] Gruhn, V., Laue, R.: Reducing the Cognitive Complexity of Business Process Models. In: Proc. 8th IEEE Int'l Conf. on Cognitive Informatics, Kowloon, Hong Kong (2009) 339–345

[124] Li, C.: Mining Process Model Variants: Challenges, Techniques, Examples. PhD Thesis, University of Twente, Enschede, The Netherlands (2010)

[125] van der Aalst, W.M.P., Weijters, A.: Process Mining: A Research Agenda. Computers in Industry **53**(3) (2004) 231–244

[126] van Glabbeek, R., Goltz, U.: Refinement of Actions and Equivalence Notions for Concurrent Systems. Acta Informatica **37**(4-5) (January 2001) 229–327

[127] Rinderle, S., Reichert, M., Dadam, P.: Correctness Criteria for Dynamic Changes in Workflow Systems - A Survey. Data & Knowledge Engineering **50**(1) (July 2004) 9–34

[128] Jungnickel, D.: Graphs, Networks and Algorithms. Volume 5. Springer (2005)

[129] Reijers, H.A., Mendling, J., Dijkman, R.: On the Usefulness of Subprocesses in Business Process Models. Technical report, BPM Center Report BPM-10-03, BPMcenter.org (2010)

[130] IEEE: IEEE 1471-2000 - Recommended Practice for Architectural Description for Software-Intensive Systems

[131] Schumm, D., Leymann, F., Streule, A.: Process Viewing Patterns. In: Proc. 14th IEEE International EDOC Conference, EDOC 2010, IEEE Computer Society (2010) 89–98

[132] Chebbi, I., Dustdar, S., Tata, S.: The View-based Approach to Dynamic Inter-Organizational Workflow Cooperation. Data & Knowledge Engineering **56**(2) (2006) 139–173

[133] Chiu, D.K., Cheung, S., Till, S., Karlapalem, K., Li, Q., Kafeza, E.: Workflow View Driven Cross-Organizational Interoperability in a Web Service Environment. Information Technology and Management **5**(3/4) (July 2004) 221–250

[134] Kafeza, E., Chiu, D.K., Kafeza, I.: View-Based Contracts in an E-Service Cross-Organizational Workflow Environment. In: Techn. E-Services. (2001) 74–88

[135] Schulz, K.A., Orlowska, M.E.: Facilitating Cross-Organisational Workflows with a Workflow View Approach. Data & Knowledge Engineering **51**(1) (2004) 109–147

[136] Knuplesch, D., Reichert, M., Pryss, R., Fdhila, W., Rinderle-Ma, S.: Ensuring Compliance of Distributed and Collaborative Workflows. In: Proc. 9th IEEE Int'l Conf. on Collaborative Computing: Networking, Applications and Worksharing (CollborateCom'13), IEEE Computer Society Press (2013) 133–142

[137] Fdhila, W., Rinderle-Ma, S., Reichert, M.: Change Propagation in Collaborative Processes Scenarios. In: Proc. 8th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom'12), IEEE Computer Society Press (2012) 452–461

[138] Fdhila, W., Indiono, C., Rinderle-Ma, S., Reichert, M.: Dealing with Change in Process Choreographies: Design and Implementation of Propagation Algorithms. Information Systems **49** (2015) 1–24

[139] Branco, M.C., Troya, J., Czarnecki, K., Küster, J., Völzer, H.: Matching Business Process Workflows Across Abstraction Levels. In: Proc. 15th Int'l Conf. Model Driven Engineering Languages and Systems (MODELS'12), Innsbruck, Italy (2012)

[140] Favre, C., Küster, J., Völzer, H.: The Shared Process Model. In: Demo Track of the 10th Int'l Conf. on Business Process Management (BPM'12). (2012) 12–16

[141] Sadiq, W., Orlowska, M.E.: Analyzing Process Models Using Graph Reduction Techniques. Information systems **25**(2) (2000) 117–134

[142] Polyvyanyy, A., Smirnov, S., Weske, M.: The Triconnected Abstraction of Process Models. In: Proc. 7th Int'l Conf. on Business Process Management. (2009)

[143] Smirnov, S., Reijers, H.A., Weske, M.: A Semantic Approach for Business Process Model Abstraction. In: Proc. 23rd Int'l Conf. on Advanced Information Systems Engineering, Springer Berlin / Heidelberg (2011) 497–511

[144] Eshuis, R., Grefen, P.: Constructing Customized Process Views. Data & Knowledge Engineering **64**(2) (2008)

[145] Schumm, D., Latuske, G., Leymann, F., Mietzner, R., Scheibler, T.: State Propagation for Business Process Monitoring on Different Levels of Abstraction. In: Proc.of the 19th European Conf. on Information Systems, Helsinki, Finland (2011)

[146] Shan, Z., Yang, Y., Li, Q., Luo, Y., Peng, Z.: A Light-Weighted Approach to Workflow View. APWeb 2006 (2003) (2006) 1059–1070

[147] Reijers, H.A., Mans, R., van der Toorn, R.: Improved Model Management with Aggregated Business Process Models. Data & Knowledge Engineering **68**(2) (February 2009) 221–243

[148] Fleischmann, A.: What Is S-BPM? In: Proc. 2nd Int'l Conf. on S-BPM ONE, Karlsruhe, Germany (2010) 85–106

[149] Börger, E.: The Subject-Oriented Approach to Software Design and the Abstract State Machines Method. In: Proc. Conceptual Modeling and Its Theoretical Foundations. (2012) 52–72

[150] Fettke, P., Loos, P.: Refactoring von Ereignisgesteuerten Prozessketten. In: EPK. (2002) 37–49

[151] Sunyé, G., Pollet, D., Traon, Y.L., Jézéquel, J.M.: Refactoring UML Models. In: UML 2001 - The Unified Modeling Language. Modeling Languages, Concepts, and Tools. Springer Berlin / Heidelberg (2001) 134–148

[152] Boger, M., Sturm, T., Fragemann, P.: Refactoring Browser for UML. In: Tagungsband Net.ObjectDays 2002. (2002) 364–376

[153] Mens, T., Tourwe, T.: A Survey of Software Refactoring. IEEE Transactions on Software Engineering **30**(2) (2004) 126–139

[154] Fowler, M., Beck, K., Brant, J., Opdyke, W.: Refactoring: Improving the Design of Existing Code. Addison-Wesley (1999)

[155] Freund, J., Rücker, B., Henninger, T.: Praxishandbuch BPMN. Hanser Verlag (July 2010)

[156] Reijers, H.A., Mendling, J., Dijkman, R.: Human and Automatic Modularizations of Process Models to Enhance their Comprehension. Information Systems **36**(5) (2011) 881–897

[157] Lohrmann, M., Reichert, M.: Effective application of process improvement patterns to business processes. Software & Systems Modeling (2014) 1–23

[158] Weber, B., Reichert, M., Rinderle-Ma, S.: Change Patterns and Change Support Features - Enhancing Flexibility in Process-Aware Information Systems. Data & Knowledge Engineering **66**(3) (2008) 438–466

[159] Weber, B., Pinggera, J., Torres, V., Reichert, M.: Change Patterns in Use : A Critical Evaluation. In: Proc. 14th Int'l Conf. BPMDS 2013. (2013) 261–276

[160] van der Aalst, W.M.P.: Formalization and Verification of Event-Driven Process Chains. Information & Software Technology **41**(10) (1999) 639–650

[161] Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M., Wohed, P.: On the Suitability of UML 2.0 Activity Diagrams for Business Process Modelling. In: Proc. 3rd Asia-Pacific Conf. on Conceptual Modelling. Volume 53. (2006) 95–104

[162] Weerawarana, S., Curbera, F., Leymann, F., Storey, T., Ferguson, D.F.: Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More. Prentice Hall (2005)

[163] van der Aalst, W.M.P.: The Application of Petri Nets to Workflow Management. Journal Circuits, Sys. & Comp. **8**(1) (1998) 21–66

[164] van der Aalst, W.M.P.: Verification of Workflow Nets. Application and Theory of Petri Nets1 **1248** (1997) 407–426

[165] Wohed, P., van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M., Russell, N.: On the Suitability of BPMN for Business Process Modelling. In: Proc. 4th Int'l Conf. Business Process Management (BPM'06), Vienna, Austria (2006) 161–176

[166] van der Aalst, W.M.P., Pesic, M., Schonenberg, H.: Declarative Workflows: Balancing Between Flexibility and Support. Computer Science - Research and Development **23**(2) (March 2009) 99–113

[167] Zugal, S., Pinggera, J., Weber, B.: The Impact of Testcases on the Maintainability of Declarative Process Models. In: BMMDS/EMMSAD 2011. (2011) 163–177

[168] Igler, M., Jablonski, S., Moura, P., Zeising, M.: ESProNa: Constraint-Based Declarative Business Process Modeling. In: Proc. 3rd WS Dynamic & Declarative Bus. Proc., Vitória, ES, Brazil (2010)

[169] Weber, B., Reijers, H.A., Zugal, S., Wild, W.: The Declarative Approach to Business Process Execution: An Empirical Test. In: Proc. 21st Int'l Conf. on Advanced Information Systems Engineering (CAiSE'09). (2009) 470–485

[170] Zugal, S., Soffer, P., Haisjackl, C., Pinggera, J., Reichert, M., Weber, B.: Investigating Expressiveness and Understandability of Hierarchy in Declarative Business Process Modeling. Software & Systems Modeling **4102** (2013) 161–176

[171] Haisjackl, C., Barba, I., Zugal, S., Soffer, P., Hadar, I., Reichert, M., Pinggera, J., Weber, B.: Understanding Declare Models: Strategies, Pitfalls, Empirical Results. Software & Systems Modeling (2014) 1–28

[172] Fahland, D., Mendling, J., Reijers, H.A., Weber, B., Weidlich, M., Zugal, S.: Declarative versus Imperative Process Modeling Languages : The Issue of Maintainability. In: Proc. 1st Workshop ER-BPM, Ulm (2009) 65–76

[173] Hull, R., Llirbat, F., Siman, E., Su, J., Dong, G., Kumar, B., Zhou, G.: Declarative Workflows That Support Easy Modification and Dynamic Browsing. SIGSOFT Softw. Eng. Notes **24**(2) (1999) 69–78

[174] van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.: Workflow Patterns. Distributed and Parallel Databases **14**(1) (2003) 5–51

[175] Russell, N., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Workflow Data Patterns: Identification, Representation and Tool Support. In: Proc. Conceptual Modeling - ER 2005. (2005) 353–368

[176] van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: Yet another workflow language. Information Systems **30** (2005) 245–275

[177] Thom, L.H., Lau, J.M., Iochpe, C., Mendling, J.: Extending Business Process Modeling Tools With Workflow Patterns Reuse. In: Proc. 9th Int'l Conf. on Enterprise Information Systems (ICEIS'07). (2007) 447–452

[178] Gschwind, T., Koehler, J., Wong, J.: Applying Patterns during Business Process Modeling. In: Lecture Notes in Computer Science. 5240 edn. Springer Berlin Heidelberg, Berlin, Heidelberg (2008) 4–19

[179] Becker, J., Rosemann, M., Uthmann, C.V.: Guidelines of Business Process Modeling. In: Proc. 1st Int'l Conf. on Business Process Management (BPM 2000). Volume 1806., Springer-Verlag (2000) 30–49

[180] Rinderle, S., Reichert, M., Weber, B.: On the Formal Semantics of Change Patterns in Process-Aware Information Systems. In: Proc. 27th Int'l Conf. Conceptual Modeling (ER'08). (2008) 279–293

[181] Weilbach, P.: Implementierungsaspekte zur Verwaltung und Synchronisation dynamischer Änderungen in prozeßorientierten Workflow-Management-Systemen. Diploma Thesis, Ulm University (1997)

[182] Koschmider, A., Song, M., Reijers, H.A.: Advanced Social Features in a Recommendation System for Process Modeling. In: Proc. 12th Int'l Conf Business Information Systems (BIS'09), Poznan, Poland (2009) 109–120

[183] Hornung, T., Koschmider, A., Lausen, G.: Recommendation Based Process Modeling Support : Method and User Experience. In: Proc. 27th Int'l Conf. on Conceptual Modeling (ER'08). (2008) 265–278

[184] Schnabel, F., Gorronogoitia, Y., Radzimski, M., Lecue, F.: Empowering Business Users to Model and Execute Business Processes. In: Proc. 8th Int'l Conf. on Business Process Management Workshops, Hoboken, NJ, USA (2010)

[185] Leopold, H., Smirnov, S., Mendling, J.: Refactoring of Activity Labels in Business Process Models. In: Proc. 15th Int'l Conf. on Applications of Natural Language to Information Systems (NLDB'10), Cardiff, Wales (2010)

[186] Weidlich, M., Mendling, J., Weske, M.: Propagating Changes between Aligned Process Models. Journal of Systems and Software **85** (2012) 1885–1898

[187] Küster, J., Völzer, H., Favre, C., Castelo Branco, M., Czarnecki, K.: Supporting different process views through a shared process model. Software & Systems Modeling (2015) 20–36

[188] Küster, J., Völzer, H., Favre, C., Branco, M.C., Czarnecki, K.: Supporting Different Process Views through a Shared Process Model. Technical report, Technical Report, IBM Research, RZ3823 (2012)

[189] Roulaux, R.: Dynamic Changes and Process Views. Master Thesis, Technische Universiteit Eindhoven (2012)

[190] Beidl, B.: Betrachtung von S-BPM und die Abbildbarkeit auf BPMN. PhD Thesis, Ulm University (2012)

[191] Date, C.J., Darwen, H.: A Guide to SQL. 4 edn. Addison-Wesley Longman, Amsterdam, Amsterdam (1997)

[192] Kusiak, A.: Concurrent Engineering: Automation, Tools and Techniques. John Wiley & Sons (1993)

[193] Quan, W., Jianmin, H.: A Study on Collaborative Mechanism for Product Design in Distributed Concurrent Engineering. In: Proc. 7th Int'l Conf. on Computer-Aided Industrial Design and Conceptual Design, Hangzhou, China (2006) 1–5

[194] Bernstein, P.A., Hadzilacos, V., Goodman, N.: Concurrency Control and Recovery in Database Systems. Addison-Wesley (1987)

[195] Bernstein, P.A., Newcomer, E.: Principles of Transaction Processing, Second Edition (The Morgan Kaufmann Series in Data Management Systems). Morgan Kaufmann (Elsevier) (2009)

[196] Casati, F., Castano, S., Fugini, M.: Managing Workflow Authorization Constraints through Active Database Technology. Information Systems Frontiers **3** (2001) 319–338

[197] Weber, B., Reichert, M., Wild, W., Rinderle, S.: Balancing Flexibility and Security in Adaptive Process Management Systems. In: Proc. 13th Conf. Cooperative Inf. Sys., Agia Napa, Cyprus (2005) 59–76

[198] Leitner, M., Rinderle-Ma, S., Mangler, J.: AW-RBAC: Access Control in Adaptive Workflow Systems. In: Proc. 6th Int'l Conf on Availability, Reliability and Security, Vienna, Austria, IEEE (August 2011) 25–37

[199] Leitner, M., Miller, M., Rinderle-Ma, S.: An Analysis and Evaluation of Security Aspects in the Business Process Model and Notation. In: Proc. 8th Int'l Conf. on Availability, Reliability and Security (ARES 2013), Regensburg, Germany (2013)

[200] Predeschly, M., Dadam, P., Acker, H.: Security Challenges in Adaptive e-Health Processes. In: Proc. 27th Int'l Conf. on Computer Safety, Reliability and Security (SAFECOM 2008). (2008) 181–192

[201] Reichert, M., Bassil, S., Bobrik, R., Bauer, T.: The Proviado Access Control Model for Business Process Monitoring Components. Enterprise Modelling and Information Systems Architectures - An International Journal **5**(3) (2010) 64–88

[202] Bassil, S., Reichert, M., Bobrik, R., Bauer, T.: Access Control for Monitoring System-Spanning Business Processes in Proviado. In: Proc. 3rd Int'l Workshop on Enterprise Modelling and Information Systems Architectures (EMISA'09), Ulm, Germany (2009)

[203] Weske, M.: Business Process Management - Concepts, Languages, Architectures. Springer (2007)

[204] Moody, D.L.: The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. IEEE Transactions on Software Engineering **35**(6) (November 2009) 756–779

[205] Norman, D.: The Design of Everyday Things. The MIT Press (1988)

[206] Hipp, M., Michelberger, B., Mutschler, B., Reichert, M.: A Framework for the Intelligent Delivery and User-Adequate Visualization of Process Information. In: Proc. 28th Symposium on Applied Computing (SAC'13), Coimbra, Portugal (2013)

[207] Gündogdu, F.: Analyse zur Verwendung der Workflow Pattern und der Business Process Modelling and Notation bei der Modellierung von Prozessen. Bachelor Thesis, Ulm University (2014)

[208] Reijers, H.A., Mendling, J.: A Study into the Factors that Influence the Understandability of Business Process Models. IEEE Transactions on Systems, Man, and Cybernetics, Part C **41**(3) (2011) 449–462

[209] Lemon, K., Allen, E.B., Carver, J.C., Bradshaw, G.L.: An Empirical Study of the Effects of Gestalt Principles on Diagram Understandability. In: Proc. 1st Int'l Symposium on Empirical Software Engineering and Measurement (ESEM 2007). (2007) 156–165

[210] Cruz-Lemus, J.A., Maes, A., Genero, M., Poels, G., Piattini, M.: The Impact of Structural Complexity on the Understandability of UML Statechart Diagrams. Information Sciences **180**(11) (2010) 2209–2220

[211] Nysetvold, A.G., Krogstie, J.: Assessing Business Processing Modeling Languages Using a Generic Quality Framework. Advanced Topics in Database Research **5** (2006) 79–93

[212] Baldwin, C., Clark, G.: Design Rules Volume 1: The Power of Modularity. MIT Press, Cambridge, Massachuesetts, USA (2000)

[213] Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley (1974)

[214] Paternò, F., Mancini, C., Meniconi, S., Maria, V.S.: ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In: Proc. IFIP TC13 Int'l Conf. on Human-Computer Interaction. (1997) 362–369

[215] Lieberman, H., Paternò, F., Klann, M., Wulf, V.: End-User Development: An Emerging Paradigm. In: Human-Computer Interaction Series. (2006) 1–8

[216] Nordbotten, J., Crosby, M.: The Effect of Graphic Style on Data Model Interpretation. Information Systems Journal **9**(2) (1999) 139–156

[217] Nassi, I., Shneiderman, B.: Flowchart Techniques for Structured Programming. SIGPLAN Not. **8**(8) (1973) 12–26

[218] Agbulak, Y.: Adaptation und Definition von Prozessmodellen auf Basis von Concurrent Task Trees. PhD Thesis, Ulm University (2011)

[219] Paternò, F., Santoro, C., Spano, L.D.: Model-Based Design of Multi-device Interactive Applications Based on Web Services. In: Human-Computer Interaction (INTERACT 2009). (2009) 892–905

[220] Paternò, F.: ConcurTaskTrees : An Engineered Approach to Model-based Design of Interactive Systems. The Handbook of Analysis for Human Computer Interaction (1999) 1–18

[221] Mori, G., Paternò, F., Santoro, C.: CTTE : Support for Developing and Analyzing Task Models for Interactive System Design. IEEE ToSE **28**(8) (2002) 797–813

[222] Liebermann, H., Paternò, F., Wulf, V.: End-User Development (Human-Computer Interaction Series). Springer, Dordrecht (2006)

[223] Amann, K., Fleischmann, A.: Bewertung der Verständlichkeit graphischer Modelle. In: Modellierung, Innsbruck, Tirol, Austria (2006) 281–284

[224] Britton, C., Jones, S.: The Untrained Eye: How Languages for Software Specification Support Understanding by Untrained Users. Human Computer Interaction **14**(1) (1999) 191–244

[225] ISO 8807: Information Processing Systems - Open Systems Interconnection - LOTOS: A Formal Description Technique based on the Temporal Ordering of Observational Behaviour. (1989)

[226] Barner, J.: Formular-basierte Modellierung, Ausführung und Änderung von Prozessmodellen. Bachelor Thesis, Ulm University (2011)

[227] Hübner, P.: Transformation of Activity-based Business Process Models to Complex User Interfaces : A Model-Driven Approach. Master Thesis, Ulm University (2012)

[228] Sousa, K., Mendonça, H., Vanderdonckt, J., Rogier, E., Vandermeulen, J.: User Interface Derivation from Business Processes: A Model-Driven Approach for Organizational Engineering. In: Proc. ACM Symp. on Applied Computing (SAC'08). (2008) 553–560

[229] Goodman, N.: Languages of Art: An Approach to a Theory of Symbols. Bobbs-Merrill Co. (1968)

[230] Frederiks, P., van der Weide, T.: Information Modeling: The Process and The Required Competencies of its Participants. Data & Knowledge Engineering **58**(1) (2006) 4–20

[231] Leopold, H.: Natural Language in Business Process Models. PhD Thesis, Humbolt Universität zu Berlin (2013)

[232] Leopold, H., Mendling, J., Polyvyanyy, A.: Generating Natural Language Texts from Business Process Models. In: Proc. 24th Int'l Conf. on Advanced Information Systems Engineering (CAiSE'12). (2012) 64–79

[233] Miller, G., Fellbaum, C.: WordNet: An Electronic Lexical Database. MIT Press Cambridge (1998)

[234] Stanford: Stanford Tagger

[235] Vanhatalo, J., Völzer, H., Koehler, J.: The Refined Process Structure Tree. Data & Knowledge Engineering **68**(9) (2009) 793–818

[236] Polyvyanyy, A., Vanhatalo, J., Völzer, H.: Simplified Computation and Generalization of the Refined Process Structure Tree. In: Proc. 7th Int'l Workshop on Web Services and Formal Methods (WS-FM 2010), Hoboken, NJ, USA (2011) 25–41

[237] Mel'čuk, I.A., Polguere, A.: A Formal Lexicon in the Meaning-Text Theory: (or How to do Lexica with Words). Computational Linguistics **13**(3-4) (1987) 261–275

[238] Lavoie, B., Rambow, O.: A Fast and Portable Realizer for Text Generation Systems. In: Proc. 5th Conf. on Applied Natural Language Processing (ANLC 1997), Washington, USA (1997) 265–268

[239] Wipp, W.: Natural Language-based Visualization and Modeling for Updatable Process Views. Bachelor Thesis, Ulm University (2013)

[240] Klein, D., Manning, C.D.: Accurate Unlexicalized Parsing. In: Proc. 41st Meeting of the Association for Computational Linguistics, Sapporo, Japan (2003) 423–430

[241] Bertin, J., Berg, W.J.: Semiology of Graphics: Diagrams, Networks, Maps. Esri Press (2011)

[242] Larkin, J.H., Simon, H.A.: Why a Diagram is (Sometimes) Worth Ten Thousand Words. Cognitive Science **11**(1) (1987) 65–99

[243] Cohn, M.: User Stories Applied: For Agile Software Development. Addison-Wesley Professional (2004)

[244] Lanz, A., Weber, B., Reichert, M.: Time Patterns for Process-aware Information Systems. Requirements Engineering **19**(2) (2014) 113–141

[245] Becker, J., Pfeiffer, D., Räckers, M.: Domain Specific Process Modelling in Public Administrations - The PICTURE-Approach. In: Proc. 6th Int'l Conf. on Electronical Government (EGOC 2007), Regensburg, Germany (2007) 68–79

[246] Thom, L.H., Reichert, M., Iochpe, C.: Activity Patterns in Process-aware Information Systems: Basic Concepts and Empirical Evidence. Int'l Journal of Business Process Integration and Management **4**(2) (2009) 93

[247] Simoes, D., Antunes, P., Pino, J.A.: Humanistic Approach to the Representation of Business Processes. In: Proc. 16th Int'l Conf. on Computer Supported Cooperative Work in Design (CSCWD), Ieee (May 2012) 655–665

[248] Hipp, M.: Navigating in Complex Process Model Collections. PhD Thesis, Ulm University (2015)

[249] Hipp, M., Mutschler, B., Reichert, M.: Navigating in Process Model Collections: A new Approach Inspired by Google Earth. In: Proc. Workshops of 9th Int'l Conf. on Business Process Management (BPM'11), Clermont-Ferrand, France (2011)

[250] Hipp, M., Mutschler, B., Reichert, M.: Navigating in Complex Business Processes. In: Proc. 23rd Int'l Conf. on Database and Expert Systems Applications (DEXA'12), Part II. Number 7447 in LNCS, Springer (2012) 466–480

[251] Hipp, M., Strauss, A., Michelberger, B., Mutschler, B., Reichert, M.: Enabling a User-Friendly Visualization of Business Process Models. In: Proc. 3rd Int'l Workshop on Theory and Applications of Process Visualization (TaProViz'14), BPM 2014 Workshops. (2014)

[252] Michelberger, B., Mutschler, B., Hipp, M., Reichert, M.: Determining the Link and Rate Popularity of Enterprise Process Information. In: Proc. 21st Int'l Conf. on Cooperative Information Systems (CoopIS 2013), Graz, Austria (2013) 112–129

[253] Ballegooij, A.V., Schönhage, B., Elliëns, A.: 3D Gadgets for Business Process Visualization - A Case Study. In: Proc. 5th Symposium on Virtual Reality Modeling Language, Monterey, USA, ACM (2000) 131–138

[254] Eichhorn, D., Koschmider, A., Yu, L., Sturzel, P., Oberweis, A., Trunko, R.: 3D Support for Business Process Simulation. In: Proc. 33rd Computer Software and Applications Conference (COMPSAC '09). (2009) 73–80

[255] Brown, R., Eichhorn, D., Herter, J.: Virtual World Process Perspective Visualization. In: Proc. 4th Int'l Conference on Information, Process, and Knowledge Management (eKNOW'12). Volume 2011., Valencia, Spain (2012)

[256] Guo, H., Brown, R., Rasmussen, R.: A Theoretical Basis for Using Virtual Worlds as a Personalised Process Visualisation Approach. In: Proc. 2nd Int'l Workshop on Human-Centric Information Systems, Valencia, Spain (2013) 229–240

[257] Hildebrandt, T., Kriglstein, S., Rinderle-Ma, S.: Beyond Visualization: On using Sonification Methods to Make Business Processes More Accessible to Users. In: Proc. 18th Int'l Conf. on Auditory Display (ICAD 2012), Atlanta, USA (2012) 248–249

[258] Hildebrandt, T., Kriglstein, S., Rinderle-Ma, S.: On Applying Sonification Methods to Convey Business Process Data. In: Proc. CAiSE'12 Forum. (2012) 74–81

[259] Nóbrega, L., Nunes, N.J., Coelho, H.: Mapping ConcurTaskTrees into UML 2.0. In: Interactive Systems. Design, Specification, and Verification. (2006) 237 – 248

[260] Brüning, J., Dittmar, A., Forbrig, P., Reichart, D.: Getting SW Engineers on Board: Task Modelling with Activity Diagrams. In: Proc. of Engineering Interactive Systems 2007 (EIS'07), Salamanca, Spain (2008) 175–192

[261] Pintus, A., Paternò, F., Santoro, C.: Modelling User Interactions in Web Service-based Business Processes. In: WEBIST'10. (2010) 175–180

[262] Brüning, J., Forbrig, P.: TTMS : A Task Tree Based Workflow Management System. In: Proc. BPMDS'11, Springer Berlin / Heidelberg (2011) 186–200

[263] Vanhatalo, J., Hagen, V., Leymann, F.: Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition. (2007) 43–55

[264] Vanhatalo, J., Völzer, H., Koehler, J.: The Refined Process Structure Tree. Lecture Notes in Computer Science **5240** (2008) 100–115

[265] Polyvyanyy, A., García-Bañuelos, L., Dumas, M.: Structuring Acyclic Process Models. In: Proc. 8th Int'l Conf. on Business Process Management, New York, USA (2010) 276–293

[266] Holl, A., Valentin, G.: Structured Business Process Modeling (SBPM). Information Systems Research in Scandinavia (IRIS 27) (2004)

[267] Seyfang, A., Kaiser, K., Gschwandtner, T., Miksch, S.: Visualizing Complex Process Hierarchies during the Modeling Process. In: Proc. 10th Int'l Conf. on Business Process Management, Workshops (BPM'12), Tallinn, Estonia (2013) 768–779

[268] Neubauer, M., Oppl, S., Stary, C.: Towards Intuitive Modeling of Business Processes : Prospects for Flow- and Natural-Language Orientation. In: Task Models and Diagrams for User Interface Design. Volume 5963. (2010) 15–27

[269] Goldberg, E., Driedger, N., Kittredge, R.: Using Natural-Language Processing to Produce Weather Forecasts. IEEE Expert **9**(2) (1994) 45–53

[270] McKeown, K., Kukich, K., Shaw, J.: Practical Issues in Automatic Documentation Generation. In: Proc. 4th Conf. on Applied Natural Language Processing. (1994) 7–14

[271] Lavoie, B., Rambow, O., Reiter, E.: The ModelExplainer. In: Proc. 8th Int'l Workshop on Natural Language Generation. (1996) 9–12

[272] Meziane, F., Athanasakis, Nikos Ananiadou, S.: Generating Natural Language Specifications from UML Class Diagrams. Requirements Engineering **13**(1) (2008) 1–18

[273] Rolland, C., Proix, C.: A Natural Language Approach for Requirements Engineering. In: Proc. 4th Conf. Advanced Information System Engineering (CAiSE 1992), Manchester, UK (1992) 257–277

[274] OMG: Semantics of Business Vocabulary and Business Rules (SVBR). (2008)

[275] Friedrich, F., Mendling, J., Puhlmann, F.: Process Model Generation from Natural Language Text. In: Proc. 23rd Int'l Conf. on Advanced Information Systems Engineering (CAiSE 2011), London, UK, Springer Berlin / Heidelberg (2011) 482–496

[276] Ottensooser, A., Fekete, A., Reijers, H.A., Mendling, J., Menictas, C.: Making Sense of Business Process Descriptions: An Experimental Comparison of Graphical and Textual Notations. Journal of Systems and Software **85**(3) (2012) 596–606

[277] Schobel, J., Schickler, M., Pryss, R., Maier, F., Reichert, M.: Towards Process-Driven Mobile Data Collection Applications: Requirements, Challenges, Lessons Learned. In: Proc. 10th Int'l Conference on Web Information Systems and Technologies (WEBIST 2014), Special Session on Business Apps, Barcelona, Spain (2014) 371–382

[278] North, C., Dwyer, T., Lee, B., Fisher, D., Isenberg, P.: Understanding Multi-touch Manipulation for Surface Computing. In: Proc. of 12th IFIP TC13 Int'l Conf. on Human-Computer Interaction (INTERACT 2009), Uppsala, Sweden (2009) 236–249

[279] Moscovich, T.: Principles and Applications of Multi-touch Interaction. PhD Thesis, Brown University, Rhode Island, USA (2007)

[280] Frisch, M., Heydekorn, J., Dachselt, R.: Diagram Editing on Interactive Displays Using Multi-Touch and Pen Gestures. In: Proc. 6th Int'l Conf. on Diagrammatic Representation and Inference (Diagrams'10). (2010) 182–196

[281] Schneider, D., Seifert, J., Rukzio, E.: MobIES: Extending Mobile Interfaces Using External Screens. In: Proc. 11th Int'l Conf. on Mobile and Ubiquitous Multimedia (MUM'12), New York, New York, USA (2012) 59:1–59:2

[282] Gong, J., Tarasewich, P.: Guidelines for Handheld Mobile Device Interface Design. In: Proc. DSI 2004 Annual Meeting. (2004) 3751–3756

[283] Wang, X., Ghanam, Y., Maurer, F.: From Desktop to Tabletop: Migrating the User Interface of AgilePlanner. In: Proc. 2nd Conf. on Human-Centred Software Engineering (HCSE'08). (2008) 263–270

[284] Pickering, J.: Touch-sensitive screens: the technologies and their application. Int'l Journal of Man-Machine Studies **25**(3) (September 1986) 249–269

[285] Cournia, N., Smith, J.D., Duchowski, A.T.: Gaze- vs. Hand-based Pointing in Virtual Environments. In: Proc. CHI '03 Extended Abstracts on Human Factors in Computer Systems (CHI '03). Volume 2., New York, USA, ACM Press (2003) 772

[286] Tanriverdi, V., Jacob, R.J.K.: Interacting with Eye Movements in Virtual Environments. In: Proc. SIGCHI Conference on Human Factors in Computing Systems (CHI '00). (2000) 265–272

[287] von Hardenberg, C., Bérard, F.: Bare-hand human-computer interaction. In: Proc. 2001 Workshop on Percetive User Interfaces (PUI '01), Orlando, USA (2001) 1–8

[288] Thalmic Labs: Myo Armband, www.thalmic.com (2013)

[289] Minker, W., Lee, G.G., Mariani, J., Nakamura, S.: Spoken Dialogue Systems Technology and Design. Springer, Boston, USA (2011)

[290] Nilsson, E.G.: Design patterns for user interface for mobile applications. Advances in Engineering Software **40**(12) (December 2009) 1318–1328

[291] Bailly, G., Lecolinet, E., Guiard, Y.: Finger-Count & Radial-Stroke Shortcuts : Two Techniques for Augmenting Linear Menus on Multi-Touch Surfaces. In: Proc. SIGCHI Conf. on Human Factors in Computing Systems (CHI'2010), Atlanta, GA, USA (2010) 591–594

[292] Mitra, S., Member, S., Acharya, T.: Gesture Recognition : A Survey. **37**(3) (2007) 311–324

[293] Wobbrock, J.O., Morris, M.R., Wilson, A.D.: User-Defined Gestures for Surface Computing. In: Proc. 27th Int'l Conf. on Human Factors in Computing Systems (CHI'09), New York, USA (2009) 1083–1092

[294] Brandl, P., Leitner, J., Seifried, T., Haller, M., Doray, B., To, P.: Occlusion-Aware Menu Design for Digital Tabletops. In: Proc. 27th Int'l Conf. on Human Factors in Computing Systems (CHI EA'09), Boston, MA, USA (2009) 3223–3228

[295] Bieber, G., Rahman, E.A.A., Urban, B.: Screen Coverage: A Pen-Interaction Problem for PDA's and Touch Screen Computers. In: Proc. 3rd Int'l Conf. on Wireless and Mobile Communications (ICWMC'07), Guadeloupe, French Caribbean, Ieee (March 2007) 87–87

[296] Benko, H., Wilson, A.D., Baudisch, P.: Precise Selection Techniques for Multi-Touch Screens. In: Proc. SIGCHI Conf. on Human Factors in Computing Systems (CHI'06). (2006) 1263–1272

[297] Wang, F., Ren, X.: Empirical Evaluation for Finger Input Properties in Multi-Touch Interaction. In: Proc. 27th Int'l Conf. on Human Factors in Computing Systems (CHI'09). (2009) 1063–1072

[298] Pfauth, M., Priest, J.: Person-Computer Interface Using Touch Screen Devices. In: Proc. 25th Annual Meeting on Human Factors and Ergonomics Society. (1981) 500–504

[299] Yee, W.: Potential Limitations of Multi-touch Gesture Vocabulary: Differentiation, Adoption, Fatigue. In: Proc. 13th Int'l Conf. on Human-Computer Interaction (HCI'09). (2009) 291–300

[300] Gemmell, M.: iPad Multi-Touch. In: http://mattgemmell.com/2010/05/09/ipad-multi-touch/, last visited: 06.02.2012 (2010)

[301] Hornecker, E., Marshall, P., Dalton, N.S., Rogers, Y.: Collaboration and Interference: Awareness with Mice or Touch Input. In: Proc. 2008 ACM Conf. on Computer Supported Cooperative Work (CSCW'08). (2008) 167–176

[302] Edelman, J., Grosskopf, A., Weske, M., Leifer, L.: Tangible Business Approach Process Modeling: A New Approach. In: Proc. Int'l Conf. on Engineering Design (ICED'09). (2009) 489–500

[303] Elias, J.G., Westerman, W.C., Haggerty, M.M.: Multi-Touch Gesture Dictionary. In: US Patent & Trademark Office, US7840912. (2007)

[304] Weber, B., Mutschler, B., Reichert, M.: Investigating the Effort of Using Business Process Management Technology: Results from a Controlled Experiment. Science of Computer Programming **75**(5) (May 2010) 292–310

[305] Recker, J., Dreiling, A.: Does It Matter Which Process Modelling Language We Teach or Use? An Experimental Study on Understanding Process Modelling Languages without Formal Education. In: Proc. 18th Australasian Conference on Information Systems, Toowoomba, Australia (2007) 356–366

[306] Recker, J., Safrudin, N., Rosemann, M.: How Novices Model Business Processes. In: Proc. 9th Int'l Conf. Business Process Management (BPM'10), New York, USA (2010) 29–44

[307] Mutschler, B., Weber, B., Reichert, M.: Workflow Management versus Case Handling: Results from a Controlled Software Experiment. In: Proc. 23rd Annual ACM Symposium on Applied Computing (SAC'08), Special Track on Coordination Models, Languages and Architectures. (2008) 82–89

[308] Rudner, B.: Fortgeschrittene Konzepte der Prozessmodellierung durch den Einsatz von Multi-Touch-Gesten. Bachelor Thesis, Ulm University (in German) (2011)

[309] Just, A.: Multi-Touch Gestures for Process Modeling. Master Thesis, Ulm University (2014)

[310] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslen, A.: Experimentation in Software Engineering - An Introduction. Kluwer Academic Publishers (2000)

[311] Pinger Inc.: Doodle Buddy. In: http://itunes.apple.com/de/app/doodle-buddy/id313232441?mt=8, last visited: 06.02.2012 (2012)

[312] Jaspers, M.W., Steen, T., van den Bos, C., Geenen, M.: The Think Aloud Method: A Guide to User Interface Design. Int'l Journal of Medical Informatics **73**(11) (2004) 781–795

[313] Cheema, S., Jr, J.J.L.: QuickDraw : Improving Drawing Experience for Geometric Diagrams. In: Proc. SIGCHI Conf. on Human Factors in Computing Systems (CHI'12), Austin, TX, USA (2012) 1037–1046

[314] MacKenzie, S.I., Oniszczak, A.: A Comparison of Three Selection Techniques for Touchpads. In: Proc. SIGCHI Conf. on Human Factors in Computer Systems (CHI'98), Los Angeles, CA, USA (1998) 336–343

[315] Lau, O.: Ätherisch - Handbewegungen erkennen mit dem Leap-Motion-Sensor. c't **2013**(20) (2013) 196–199

[316] Bolt, R.A.: Put-That-There: Voice and Gesture at the Graphics Interface. In: Proc. 7th Annual Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH'80)1, Seattle, WA, USA (1980) 262–270

[317] Lee, J., Kunii, T.: Model-based Analysis of Hand Posture. IEEE Computer Graphics and Applications **15**(5) (1995) 77–86

[318] Ren, Z., Meng, J., Yuan, J., Zhang, Z.: Robust Hand Gesture Recognition with Kinect Sensor. In: Proc. 19th ACM Int'l Conf. on Multimedia (MM'11), Scottsdale, USA (2011) 759–760

[319] Pavlovic, V., Sharma, R., Huang, T.: Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review. IEEE Transactions on Pattern Analysis and Machine Intelligence **19** (1997)

[320] Hess, H.: Hand Gesture-based Process Modeling for Updatable Processes. Bachelor Thesis, Ulm University (2013)

[321] Burkhardt, J.: Collaborative Process Modelling with Multi-Touch Tables. Diploma Thesis, Ulm University (2013)

[322] Scott, S.D., Grant, K.D., Mandryk, R.L.: System Guidelines for Co-located, Collaborative Work on a Tabletop Display. In: Proc. 8th European Conference on Computer Supported Cooperative Work (ECSCW'03). Number September, Helsinki, Finland (2003) 159–178

[323] Microsoft: Microsoft Surface 2.0 Design and Interaction Guide - Principles and Guidelines for Designing and Developing Surface Applications. Technical Report July, Microsoft (2011)

[324] Rogers, Y., Lindley, S.: Collaborating Around Vertical and Horizontal Large Interactive Displays: Which Way is Best? Interacting with Computers **16**(6) (2004) 1133–1152

[325] Lauwers, J.C., Lantz, K.A., Park, M.: Collaboration Awareness in Support of Collaboration Transparency: Requirements for the Next Generation of Shared Window Systems. In: Proc. SIGCHI Conf. on Human Factors in Computing Systems (CHI'90), Seattle, WA, USA (1990) 303–311

[326] Ronis, S.: Collaborative Modeling of Business Processes on Co-Located TableTop Systems. Master Thesis, Ulm University (2014)

[327] Rittgen, P.: Collaborative Modeling of Business Processes - A Comparative Case Study. In: Proc. of ACM Symposium on Applied Computing (SAC'09), Honolulu, Hawaii (2009) 225–230

[328] Schmidt, D., Seifert, J., Rukzio, E., Gellersen, H.: A Cross-Device Interaction Style for Mobiles and Surfaces. In: Proc. ACM Symposium on Designing Interactive Systems (DIS '12), Newcastle, UK, ACM Press (2012)

[329] Rukzio, E., Leichtenstern, K., Callaghan, V., Holleis, P., Schmidt, A., Chin, J.: An Experimental Comparison of Physical Mobile Interaction Techniques: Touching, Pointing and Scanning. In: UbiComp 2006: Ubiquitous Computing. (2006) 87–104

[330] Nielsen, M., Störring, M., Moeslund, T.B., Granum, E.: A Procedure for Developing Intuitive and Ergonomic Gesture Interfaces for HCI. In: Proc. 5th Int'l Gesture Workshop (GW'03), Genova, Italy (2003) 409–420

[331] Schmidt, S., Nacenta, M.a., Dachselt, R., Carpendale, S.: A Set of Multi-Touch Graph Interaction Techniques. In: Proc. ACM Int'l Conf. on Interactive Tabletops and Surfaces (ITS '10), New York, New York, USA, ACM Press (2010) 113

[332] Frisch, M., Heydekorn, J., Dachselt, R.: Investigating Multi-Touch and Pen Gestures for Diagram Editing on Interactive Surfaces. In: Proc. ACM Int'l Conf. on Interactive Tabletops and Surfaces (IST'09). (2009) 167–174

[333] Rubine, D.: Specifying Gestures by Example. ACM SIGGRAPH Computer Graphics **25** (1991) 329–337

[334] Wobbrock, J.O., Wilson, A.D., Li, Y.: Gestures Without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes. In: Proc. 20th Annual ACM Symposium on User Interface Software and Technology, New York, USA (20007) 159–168

[335] Myers, C.S., Rabiner, L.R.: Comparative Study of Several Dynamic Time-Warping Algorithms for Connected-Word Recognition. The Bell System Technical Journal **60**(7) (1981) 1389–1409

[336] Xu, W., Lee, E.J.: Gesture Recognition based on 2D and 3D Feature by using Kinect Device. In: Proc. 6th Int'l Conf. on Information Security and Assurance (ISA'12), Shanghai, China (2012) 77–79

[337] Khoshelham, K., Elberink, S.O.: Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications. Sensors **12**(2) (2012) 1437–1454

[338] Malik, S., Ranjan, A., Balakrishnan, R.: Interacting with Large Displays from a Distance with Vision-Tracked Multi-Finger Gestural Input. In: Proc. 18th ACM Symposium on User Interface Software and Technology (UIST '05), ACM Press (2005) 43–45

[339] Morris, M.R., Huang, A., Paepcke, A., Winograd, T.: Cooperative Gestures: Multi-User Gestural Interactions for Co-located Groupware. In: Proc. SIGCHI Conf. on Human Factors in Computing Systems (CHI'06). (2006) 1201–1210

[340] Renger, M., Kolfschoten, G.L., de Vreede, G.J.: Challenges in Collaborative Modeling: A Literature Review & Research Agenda. Int'l Journal of Simulation and Process Modelling **4**(3/4) (2008) 248–263

[341] Poppe, E., Brown, R., Recker, J., Johnson, D.: Improving Remote Collaborative Process Modelling using Embodiment in 3D Virtual Environments. In: Proc. Conferences in Research and Practice in Information Technology, Adelaide, Australia (2012)

[342] Scheuerlein, H., Rauchfuss, F., Dittmar, Y., Molle, R., Lehmann, T., Pienkos, N., Settmacher, U.: New Methods for Clinical Pathways-Business Process Modeling Notation (BPMN) and Tangible Business Process Modeling (t.BPM). Langenbeck's Archives of Surgery **397**(5) (June 2012) 755–61

[343] Oppl, S., Stary, C.: Tabletop Concept Mapping. In: Proc. 3rd Int'l Conf. on Tangible and Embedded Interaction, Cambridge, United Kingdom (2009) 275–282

[344] Oppl, S., Stary, C.: Facilitating Shared Understanding of Work Situations using a Tangible Tabletop Interface. Behaviour & Information Technology (October 2013) 1–17

[345] Döweling, S., Tahiri, T., Schmidt, B., Nolte, A., Khalilbeigi, M.: Collaborative Business Process Modeling on Interactive Tabletops. In: Proc. 21st European Conference on Information Systems. (2013)

[346] Wittern, H.: Empirical Study Evaluating Business Process Modeling on Multi-touch Devices. In: Proc. IEEE International Conference on Software Science, Technology and Engineering, Ieee (June 2012) 20–29

[347] Büringer, S.: Development of a Business Process Abstraction Component based on Process Views. Bachelor Thesis, Ulm University (2012)

[348] Hofmann, J.: Implementierung einer Visualisierungskomponente für Prozesssichten. Bachelor Thesis, Ulm University (2012)

[349] Dapper, M.: Implementation of a Multi-Touch, Gesture-based Process Modeling Component for Apple iPad. Bachelor Thesis, Ulm University (2012)

[350] Frankel, N.: Learning Vaadin 7. Second edi edn. Packt Publishing (2013)

[351] Nahm, M.: Dokumentation, Analyse und Optimierung der Auftragsbearbeitung-Geschäftsprozesse zur Einführung eines BPM-Systems bei der ThyssenKrupp Aufzugswerke GmbH. Master's Thesis, Ulm University (2012)

[352] Schultheiss, B., Meyer, J., Mangold, D.J., Zemmler, T., Reichert, M.: Prozessentwurf für den Ablauf einer stationären Chemotherapie. Technical report, Department of Databases and Information Systems, Ulm University, Ulm (1995)

[353] Mebrahtu, S.: Metrics for Updatable Process Views on Large Process Models. Bachelor Thesis, Ulm University (2012)

[354] McCabe, T.J.: A Complexity Measure. IEEE Transactions on Software Engineering **4** (1976) 308–320

[355] Cardoso, J.: Evaluating Workflows and Web Process Complexity. In: Workflow Handbook. (2005) 284–290

[356] Wolf, C., Harmon, P.: The State of Business Process Management. Technical report, BPTrends Report (2012)

[357] Pinggera, J., Zugal, S., Weber, B.: Investigating the Process of Process Modeling with Cheetah Experimental Platform. In: Proc. 1st Int'l WS Empirical Research Proc.-Oriented Inf. Sys., Hammamet (2010)

[358] Grees, T.: Modeling and Changing Business Process Models with Concurrent Task Trees. Master Thesis, University of Innsbruck (2012)

[359] Basili, V.R., Green, S.: Software Process Evolution at the SEL. IEEE Software **11**(4) (1994) 58–66

[360] Michelberger, B.: Process-Oriented Information Logistics: Aligning Process Information with Business Processes. PhD Thesis, Ulm University (2015)

Part VI

Directories

List of Acronyms

ADEPT	Application DEvelopment Based on Pre-Modeled Process Templates
Ajax	Asynchronous JavaScript and XML
BPM	Business Process Management
BPMN	Business Process Model and Notation
CPM	Central Process Model
CPR	Central Process Repository
CTT	Concurrent Task Trees
CTTE	Concurrent Task Tree Environment
DBMS	Database Management Systems
DSynT	Deep-Syntactic Tree
EPC	Event-driven Process Chains
ERP	Enterprise Resource Planning
EUD	End-User Development
GWT	Google Web Toolkit
IS	Information System
ISO	International Organization for Standardization
IT	Information Technology
PAIS	Process-aware Information System
PST	Process Structure Tree
REST	Representational State Transfer
RPST	Refined Process Structure Tree (See PST)
RPC	Remote Procedure Call

SaaS	Software as a Service
S-BPM	Subject-oriented Business Process Management
SESE	Single Entry Single Exit
UML	Unified Modeling Language
UUID	Universally Unique Identifier
VCD	Value-Added Chain Diagram
WS-BPEL	Web Service Business Process Execution Language
XML	Extensible Markup Language
YAWL	Yet Another Workflow Language
WPF	Windows Presentation Framework

Part VII

Appendix

A
Additional Functions

Algorithm A.1: InitializeView(CPM)

Input: Process model $CPM = (N, D, NT, CE, EC, ET, DE, DET)$
Output: Process view $V = (N', D', NT', CE', EC', ET', DE', DET')$

```
1  begin
2  |  N' := N;
3  |  D' := D;
4  |  CE' := CE;
5  |  DE' := DE;
6  |  NT' : N' → NodeType, ∀n ∈ N : NT'(n) = NT(n);
7  |  ET' : CE' → EdgeType, ∀e ∈ CE : ET'(e) = ET(e);
8  |  DET' : DE' → DEdgeType, ∀de ∈ DE : DET'(de) = DET(de);
9  |  return V;
10 end
```

Let $CPM = (N, D, NT, CE, EC, ET, DE, DET)$ be a process model. Then:
Function

$$updateControlFlowEdges(CE, N_a, n_{new}) \tag{A.1}$$

removes control edges connected to nodes in N_a and adds new control edges connecting node n_{new} with its new predecessor and successor. If applicable, synchronization edges are reconnected to node n_{new}.

For $N_a \subseteq N$ and $d \in D$, we define:

$$isOnlyAccessedInSet(d, DE, N_a) := \begin{cases} false & \exists de \in DE : de = (d, n) \vee de = (n, d) \wedge n \notin N_a \\ true & else \end{cases} \tag{A.2}$$

$$isReadAccess(d, DE, N_a) := \begin{cases} true & \exists (d, n) \in DE : n \in N_a \\ false & else \end{cases} \tag{A.3}$$

$$isWriteAccess(d, DE, N_a) := \begin{cases} true & \exists (n, d) \in DE : n \in N_a \\ false & else \end{cases} \tag{A.4}$$

Function

$$accessedByAllTraces(d, DE, N_a) \tag{A.5}$$

checks whether data element $d \in D$ is accessed by all branches in the SESE block induced by node set $N_a \subseteq N$

B
Optimization Rules

Optimization Rule OR4 (Data Element Aggregation on Aggregation)
Let CPM be a process model. Further, let $Op_V = \langle op_1, op_2 \rangle$ be an operation sequence to create process view V on CPM, i.e., $CPM = P_0 \xrightarrow{op_1} P_1 \xrightarrow{op_2} P_2 = V$ with $P_i = (N_i, D_i, NT_i, CE_i, EC_i, ET_i, DE_i, DET_i)$.
***Precondition**:*
$op_1 = AggrDataElement(P_0, D') \wedge op_2 = AggrDataElement(P_1, D'') \wedge$
$D' \cap CPMNode(P_1, D'') \neq \emptyset \ \wedge \ D' \subseteq D_0 \ \wedge \ D'' \subseteq D_1$

***Postcondition**:*
$op_{new} := AggrDataElement(P_0, D' \cup CPMNode(P_1, D''))$
$Op_V := Op_V \ominus \langle op_1, op_2 \rangle \oplus \langle op_{new} \rangle$

Optimization Rule OR5 (Data Element Reduction on Aggregation)
Let CPM be a process model. Further, let $Op_V = \langle op_1, op_2 \rangle$ be an operation sequence to create process view V on CPM, i.e., $CPM = P_0 \xrightarrow{op_1} P_1 \xrightarrow{op_2} P_2 = V$ with $P_i = (N_i, D_i, NT_i, CE_i, EC_i, ET_i, DE_i, DET_i)$.
***Precondition**:*
$op_1 = AggrDataElement(P_0, D') \ \wedge \ op_2 = RedDataElement(P_1, d) \wedge$
$D' = CPMNode(P_1, d) \ \wedge \ D' \subseteq D_0 \ \wedge \ d \in D_1$

***Postcondition**:*
$Op_D := \langle op_{d_1}, \ldots, op_{d_k} \rangle$ *with*
$op_{d_i} := RedDataElement(P_0, d_i), d_i \in D', \forall i \in 1, \ldots, k$
$Op_V := Op_V \ominus \langle op_1, op_2 \rangle \oplus Op_D$

Optimization Rule OR6 (Process Attribute Aggregation on Aggregation)
Let CPM be a process model. Further, let $Op_V = \langle op_1, op_2 \rangle$ be an operation sequence to create process view V on CPM, i.e., $CPM = P_0 \xrightarrow{op_1} P_1 \xrightarrow{op_2} P_2 = V$ with $P_i = (N_i, D_i, NT_i, CE_i, EC_i, ET_i, DE_i, DET_i)$.
***Precondition**:*
$op_1 = AggrAttr(P_0, elem, AS', attrFunc_1) \wedge$
$op_2 = AggrAttr(P_0, elem, AS'', attrFunc_2) \wedge$
$attrFunc_1 = attrFunc_2 \wedge$
$AS' \cap CPMAttr(P_1, elem, AS'') \neq \emptyset \wedge$
AS', AS'' *subset of attributes of process element* $elem \in N_0 \dot\cup D_0 \dot\cup CE_0 \dot\cup DE_0$

***Postcondition**:*
$op_{new} := AggrAttr(P_0, elem, AS' \cup CPMAttr(P_1, elem, AS''), attrFunc_1)$
$Op_V := Op_V \ominus \langle op_1, op_2 \rangle \oplus \langle op_{new} \rangle$

Accordingly to function $CPMNode(V, N)$, $CPMAttr(V, n, AS)$ returns the corresponding set of attributes for all process attributes in AS of process node n in process view V.

Optimization Rule OR7 (Process Attribute Reduction on Aggregation)
Let CPM be a process model. Further, let $Op_V = \langle op_1, op_2 \rangle$ be an operation sequence to create process view V on CPM, i.e., $CPM = P_0 \xrightarrow{op_1} P_1 \xrightarrow{op_2} P_2 = V$ with $P_i = (N_i, D_i, NT_i, CE_i, EC_i, ET_i, DE_i, DET_i)$.
***Precondition**:*
$op_1 = AggrAttr(P_0, elem, AS', attrFunc_1) \ \wedge \ op_2 = RedAttr(P_1, elem, attr) \wedge$
$AS' = CPMAttr(P_1, elem, a) \ \wedge \ elem \in N_0 \dot\cup D_0 \dot\cup CE_0 \dot\cup DE_0 \wedge$
AS' *subset of attributes of process element* $elem \in N_0 \dot\cup D_0 \dot\cup CE_0 \dot\cup DE_0$

***Postcondition**:*
$Op_A := \langle op_{a_1}, \ldots, op_{a_k} \rangle$ *with* $op_{a_i} := RedAttr(P_0, d_i), a_i \in AS', \ \forall i \in 1, \ldots, k$
$Op_V := Op_V \ominus \langle op_1, op_2 \rangle \oplus Op_A$

C
Process Models of Case Study

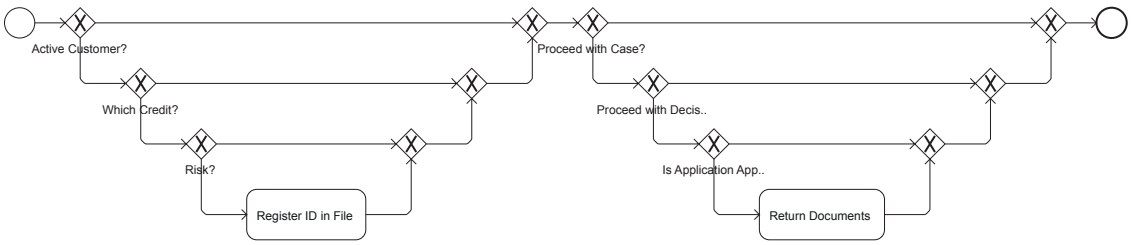


Figure C.1: Credit Approval - View V2 Clerk

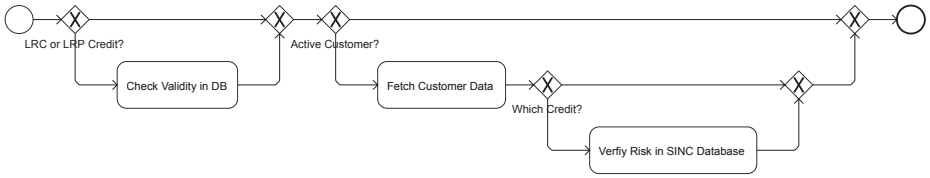


Figure C.2: Credit Approval - View V3 Database

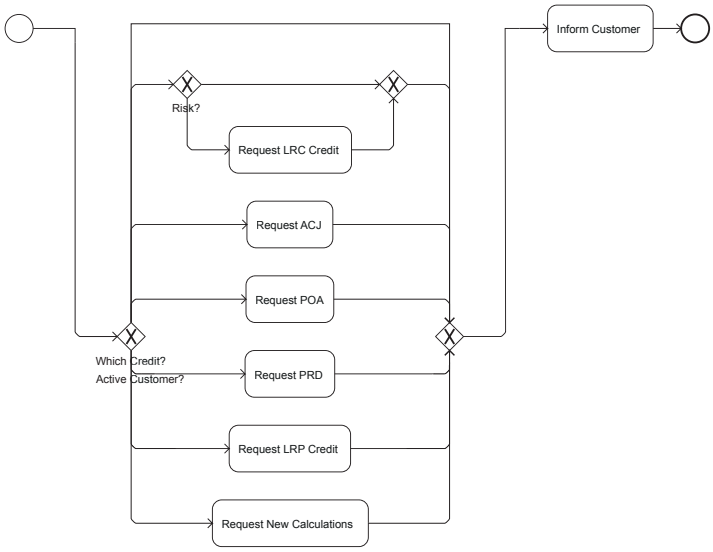


Figure C.3: Credit Approval - View V4 Manager

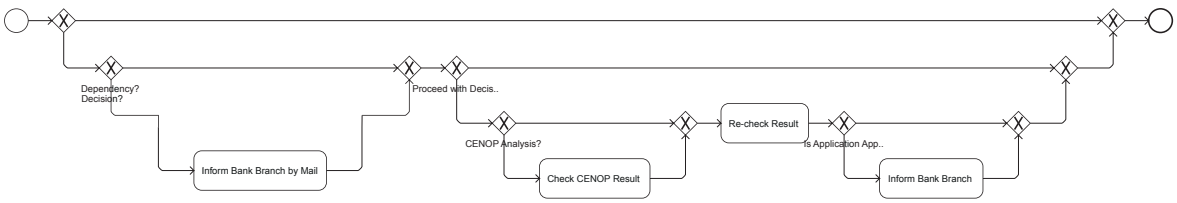


Figure C.4: Credit Approval - View V5 System

D
Results of Multi-Touch Experiment

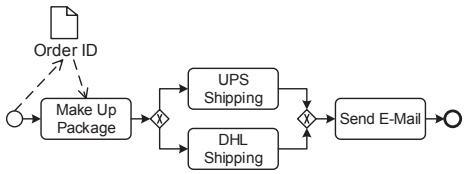


Figure D.1: Initial Process Model of Experiment 1

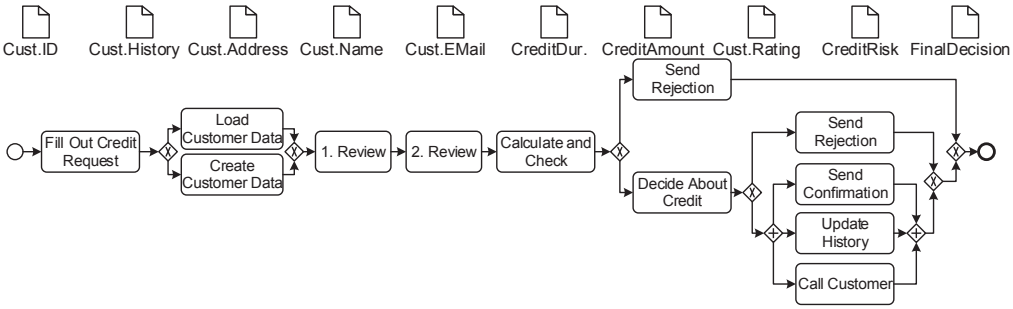


Figure D.2: Initial Process Model of Experiment 2

Results of individiual subjects are provided at:
https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.030/downloads/experiments_1_2.zip

Subject	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	Profession	Experience
1	M	G	G	G	G	G	G		G						ST	yes
2	P	G	P	P	P	P	G		P						ST	yes
3	P	G	P	P	P	P	G		P						ST	no
4	M	G	G	G	G	M	G		G						ST	yes
5	G	G	G	G	G	M	G		G						ST	yes
6	P	G	P	P	P	P	G		G						RA	no
7	G	P	P	P	P	P	P		G						RA	no
8	M	G	G	G	G	M	G		G						RA	no
9	G	M	G	G	G	G	P		G						RA	no
10	M	P	G	G	P	P	M		G						ST	no
11	M	G	G	G	M	M	G		G						RA	yes
12	M	P	M	M	M	M	M		G						RA	yes
13	M	M	G	M	M	M	M		G						RA	yes
14	M	P	G	G	G	G	M		G						ST	no
15	P	G	P	P	P	P	G		G						ST	no
16	P	P	P	P	P	P	G		G						RA	no
17	G	G	G	G	G	G	M		G						ST	yes
18	P	G	P	P	P	P	M		P						ST	no
19	M	P	M	M	M	M	M		M						RA	no
20	P	G	P	P	P	P	M		G						ST	yes
21	G	G	G	G	G	P	G		G						ST	yes
22	G	G	G	M	M	M	M		M						ST	yes
23	M	P	M	M	M	M	M		M						ST	yes
24	P	P	G	M	M	M	M		G						ST	yes
25	M	G	G	G	M	M	G		M						ST	yes
26	P	G	G	G	M	M	G		G						ST	no
27	M	G	M	M	M	M	M	M	P	G	P	G	M	M	ST	yes
28	P	G	M	G	M	P	G	P	P	G	M	G	P	P	ST	yes
29	G	G	P	G	P	G	P	P	G	G	P	G	P	P	ST	yes
30	M	M	M	M	M	G	G	G	P	M	M	G	M	G	RA	yes
31	G	G	P	M	P	P	P	G	G	G	P	G	P	P	RA	yes
32	M	M	P	G	G	G	P	G	G	G	P	G	G	M	ST	yes
33	G	G	P	M	M	G	G	G	P	M	P	G	P	P	ST	yes
34	G	G	G	M	M	G	G	G	P	G	P	G	G	G	ST	yes
35	G	G	G	G	M	G	G	G	M	G	M	G	G	P	ST	yes
36	P	G	M	P	M	G	P	P	P	G	P	G	P	M	ST	yes
37	M	G	M	M	G	G	G	G	G	M	G	G	G	G	ST	yes

Legend

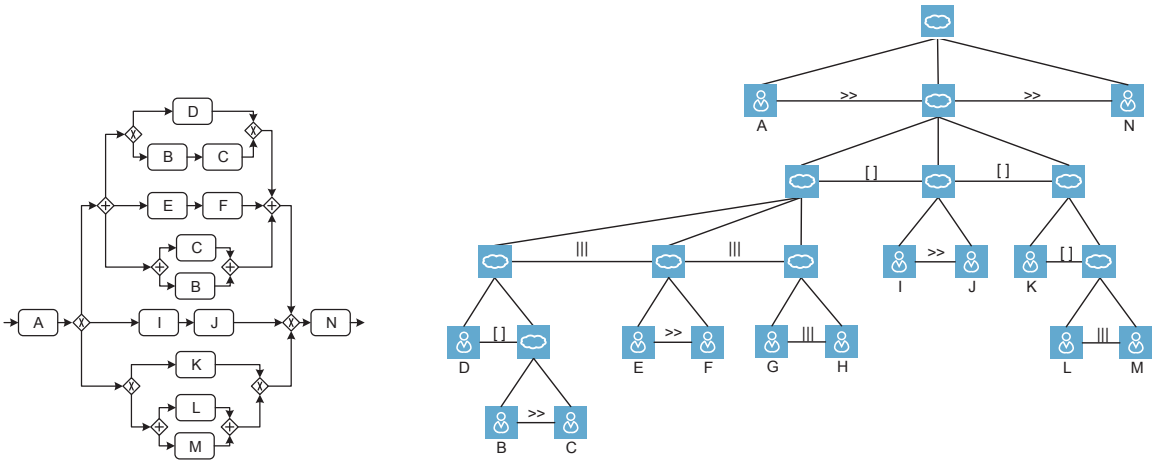
P Picture-based Interaction
G Gesture-based Interaction
M Menu-based Interaction

ST Student
RA Research Assistant

T1 Insert Activity
T2 Rename Activity
T3 Insert Branching
T4 Insert Branch
T5 Insert Data Element
T6 Insert Data Edge
T7 Delete Activity
T8 Insert Sync Edge
T9 Aggregate Activities
T10 Reduce Activity
T11 Create Process View
T12 Select Element

Figure D.3: Raw Data of Multi-Touch Experiment

E
Experiment Results



(a) BPMN Representation

(b) Hierarchical Representation

Figure E.1: Experiment 3: Process Models of Part B

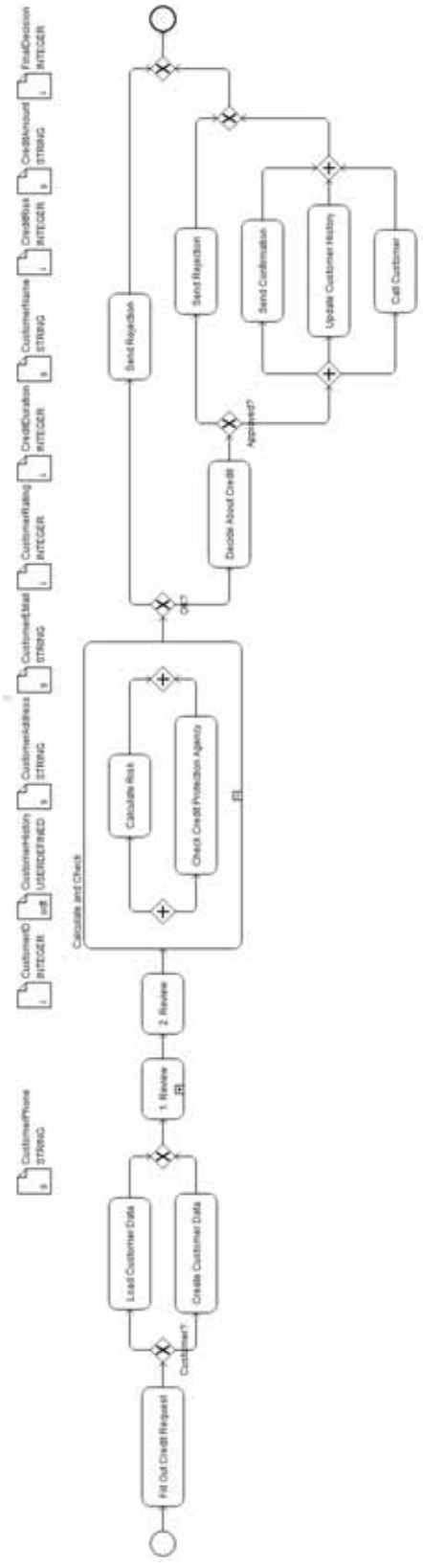


Figure E.2: Experiment 4: Process Model