Business Process Model Abstraction

Sergey Smirnov

Business Process Technology Group
Hasso Plattner Institute, University of Potsdam
Potsdam, Germany

dissertation
zur erlangung des grades eines
doktors der naturwissenschaften
– dr. rer. nat. –

november 2011

Abstract

Business process models are used within a range of organizational initiatives, where every stakeholder has a unique perspective on a process and demands the respective model. As a consequence, multiple process models capturing the very same business process coexist. Keeping such models in sync is a challenge within an ever changing business environment: once a process is changed, all its models have to be updated. Due to a large number of models and their complex relations, model maintenance becomes error-prone and expensive. Against this background, *business process model abstraction* emerged as an operation reducing the number of stored process models and facilitating model management. Business process model abstraction is an operation preserving essential process properties and leaving out insignificant details in order to retain information relevant for a particular purpose. Process model abstraction has been addressed by several researchers. The focus of their studies has been on particular use cases and model transformations supporting these use cases.

This thesis systematically approaches the problem of business process model abstraction shaping the outcome into a framework. We investigate the current industry demand in abstraction summarizing it in a catalog of business process model abstraction use cases. The thesis focuses on one prominent use case where the user demands a model with coarse-grained activities and overall process ordering constraints. We develop model transformations that support this use case starting with the transformations based on process model structure analysis. Further, abstraction methods considering the semantics of process model elements are investigated. First, we suggest how semantically related activities can be discovered in process models—a barely researched challenge. The thesis validates the designed abstraction methods against sets of industrial process models and discusses the method implementation aspects. Second, we develop a novel model transformation, which combined with the related activity discovery allows flexible non-hierarchical abstraction. In this way this thesis advocates novel model transformations that facilitate business process model management and provides the foundations for innovative tool support.

Zusammenfassung

Geschäftsprozessmodelle werden in einer Fülle organisatorischer Initiativen eingesetzt, wobei verschiedene Stakeholder individuelle Ansprüche an die Sicht auf den jeweiligen Prozess haben. Dies führt dazu, dass zu einem Geschäftsprozess eine Vielzahl unterschiedlicher Modelle existiert. In einer sich ständig verändernden Geschäftsumgebung ist es daher schwierig, diese Vielzahl von Modellen konsistent zu halten: Ändert sich sich ein Prozess, müssen alle Modelle, die ihn beschreiben, aktualisiert werden. Aufgrund der schieren Menge an Prozessmodellen und ihrer komplexen Beziehungen zueinander, erhöhen sich Aufwand und Kosten zur Pflege aller Modelle enorm. Vor diesem Hintergrund ermöglicht die *Abstraktion von Geschäftsprozessmodellen*, die Menge der Modelle zu reduzieren und damit ihre Verwaltung zu vereinfachen. Abstraktion von Geschäftsprozessmodellen bezeichnet eine Transformation eines Prozessmodells, so dass es für einen bestimmten Zweck besonders geeignet ist. Bei der Abstraktion von Geschäftsprozessen bleiben essentielle Eigenschaften eines Modells erhalten, während irrelevante Eigenschaften verworfen werden. Mehrere Studien stellen Prozessmodellabstraktion in den Fokus und konzentrieren sich auf konkrete Anwendungsfälle, für die sie geeignete Transformationen entwickelt haben.

Diese Dissertation untersucht das Problem der Prozessmodellabstraktion und systematisiert die Lösung in einem Framework. Aktuelle Anforderungen der Industrie an die Abstraktion von Prozessmodellen wurden recherchiert und in einem Katalog von Anwendungsfällen zusammengefasst, von denen ein besonderer für die weiteren Untersuchungen ausgewählt wurde. In diesem Fall erwartet der Nutzer ein Modell niedrigeren Detailgrades, in welchem die Kontrollflussbeziehungen des Ursprungsmodells erhalten bleiben. Beginnend bei Modelltransformationen, die auf der Analyse der Prozessmodellstruktur aufbauen, entwickeln wir neuartige Abstraktionsoperationen zur Unterstützung dieses Anwendungsfalles. Darüber hinaus untersuchen wir Abstraktionsmethoden, welche die Semantik von Prozessmodellelementen berücksichtigen. Zum einen zeigen wir, wie Aktivitäten ermittelt werden können, die miteinander in semantischer Beziehung stehen – ein Problem, das bisher nur unzureichend betrachtet wurde. Die vorgeschlagenen Methoden werden mithilfe industrieller Prozessmodellsammlungen validiert und deren Umsetzung diskutiert. Zum anderen schlagen wir eine innovative Modelltransformation zur nicht-hierarchischen Abstraktion von Prozessmodellen vor. Dieser liegt die Ermittlung in Beziehung stehender Aktivitäten zugrunde. Demzufolge präsentiert diese Arbeit eine originäre Methode zur Prozessmodellabstraktion, die die Verwaltung von Geschäftsprozessmodellen vereinfacht und den Grundstein für innovative Softwarewerkzeuge legt.

Acknowledgments

At this point I would like to express gratitude to those people without whom this thesis could not be written.

I would like to thank Mathias Weske who supervised my doctoral studies at Hasso Plattner Institute. Professor Weske fostered my scientific activities throughout the master studies and encouraged my participation in the doctoral program. Being the doctoral student at his chair I enjoyed the professional spirit within the group, a great research freedom, and the unique opportunity to collaborate with the leading BPM research groups.

It was great pleasure to work with Jan Mendling. I appreciate Jan's vision and exceptional capability to inspire, develop, and complete ideas. I had the luck to work with Hajo Reijers who motivated me to revise the vision of my research and align it with the industry demand. I am thankful to Manfred Reichert and Karsten Wolf for being the reviewers of this thesis and their valuable feedback.

I was happy to collaborate on several research contributions with Matthias Weidlich, Artem Polyvyanyy, Remco Dijkman, Henrik Leopold, Ahmed Awad, Thijs Nugteren, Andreas Meyer, Christian Wiggert, and Armin Zamani Farahani. I highly value the feedback of my colleagues Evellin Cardoso, Gero Decker, Rami-Habib Eid-Sabbagh, Markus Güntert, Nico Herzberg, Jens Hündling, Matthias Kunze, Dominik Kuropka, Alexander Lübbe, Harald Meyer, Hagen Overdick, Emilian Pascalau, Nicolas Peters, Frank Puhlmann, Maria Rastrepkina, Andreas Rogge-Solti, and Juliane Siegeris. I would like to thank the anonymous reviewers of the papers and the articles I co-authored. I appreciate the expertise of the employees of Pallas Athena and AOK Nord. In particular, I am grateful to John Hoogland, Paul Eertink, Anke-Britt Möhr, Norbert Sandau, and Anja Niedersätz. I appreciate the organizational support of Katrin Heinrich. Finally, I thank my family and friends who encouraged my research initiatives. In particular, I am grateful to my wife, Alina, and parents, Tatiana and Vladimir.

Contents

List of Figures

List of Tables

1 Introduction

Chapter 1 introduces the reader to this doctoral thesis. The start of this chapter motivates the investigated research problem. Then, we explain the main research contributions and conclude with an outline of the thesis's structure.

1.1 Problem Statement

In the last decades businesses have found themselves in the volatile market with tight competition. Small- and mid-size companies are threatened by large corporations. The latter, in turn, have to cope with disruptive technologies developed by competitors [36]. To survive in this harsh environment businesses seek means to differentiate themselves from the rivals. The companies identify, secure, and develop their *core competencies*—factors that are not easy to imitate, could be applied to numerous products and markets, and contribute to the consumer benefits. Business processes are a vivid example of a company's core competency [41, 69, 71, 149]. According to [41], a business process is

> "a structured, measured set of activities designed to produce a specific output for a particular customer or market. It implies a strong emphasis on how work is done within an organization, in contrast to a product focus's emphasis on what. A process is thus a specific ordering of work activities across time and space, with a beginning and an end, and clearly defined inputs and outputs: a structure for action."

Business processes have been discussed by economic practitioners and researchers for centuries. For instance, the fundamental work of Adam Smith, [148], refers to the example of a pin production business process when arguing about the division of labor. However, the recognition of business processes as the core competency of a company has happened only recently. The prolific work of Davenport, [41], along with [69] by Hammer and Champy focused the attention on business processes as valuable artifacts. This new perspective gave birth to a new management approach—business process management (BPM). Business process management focuses on designing, enacting, managing, analyzing, adapting, and mining business processes [5]. Essentially, each of the aforementioned tasks implies that a description of the business process is available or the task creates the description. While in some cases a textual description of the process suffices, formal business process models got wide spread in industry and academia. Following [165] we assume that

> "a business process model consists of a set of activity models and execution constraints between them. A business process instance represents a concrete case in the operational business of a company, consisting of activity instances. Each business process model acts as a blueprint for a set of business process instances, and each activity model acts as a blueprint for a set of activity instances."

Among the advantages of formal models are their low ambiguity and a large choice of methods and software applications for model validation [59].

While the use of process models gives clear benefits to BPM practitioners and researchers, there is no consensus on one common business process modeling language. On the one hand, there is a large family of modeling languages that formalize business processes as graphs. The graph nodes capture activities, events, and decisions, while edges reflect the ordering constraints. This family includes such languages as ADEPT [39, 127], Business Process Model and Notation (BPMN) [113], Event-driven Process Chains (EPCs) [80], Petri nets [108, 116], UML Activity Diagrams [114], workflow nets [2, 9], and Yet Another Workflow Language (YAWL) [7]. The execution semantics of the languages in this family can be traced back to the semantics of Petri nets. Similarly, Business Process Execution Language (BPEL) puts the focus on the process control flow [111]. At the same time, business artifacts [26, 27, 76], as well as case handling [11, 130], emphasize the role of data in business processes. However, the graph-based process modeling languages got widely accepted both by industry and academia. Against this background, this thesis studies process models formalized as graphs and having the semantics resembling that of Petri nets. We assume a process model to contain a set of activity models along with the ordering constraints formalized by means of gateways and control flow relation (for technical details see Chapter 2).

Process models facilitate numerous tasks, among them configuring workflow software systems [62, 92], training new employees, identifying performance improvement opportunities [69], aligning conflicting views of stakeholders on business operations, and demonstrating an organization's compliance with external regulations [18], to name but a few. Obviously, this variety of modeling goals requires a process modeler to focus on the business process aspects relevant for a task at hand. Following this demand, companies design new process models, each supporting a particular business task, yet contributing to the overall number of maintained models. As an outcome, organizations are challenged by large process model repositories consisting of hundreds or even thousands of models. The stored models have complex interrelations:

they overlap, describe processes that subsume each other, describe one process from different perspectives or at varying levels of precision. Obviously, the owner of such a collection demands adequate means to manage this plethora of models and their interrelations. The BPM community delivered a number of methods that help managing the complexity of large process model collections. For example, there are methods to deal efficiently with process model variety [68, 86, 126, 134, 153] and algorithms to search for process models that fit a particular profile [46, 48, 52, 78, 84].

This thesis studies process models that describe *one* business process, but with different amounts of details. As we argued earlier, the demand for such models is motivated by the model pragmatics: each model supports a specific business task and, hence, has a particular modeling goal [150]. Unfortunately, *multiple* models describing *one* business process with different precision frequent to be stored independently. Since no formal relations between these models exist, the model owner is challenged by the effort of keeping them in sync. Indeed, each change of the business process needs to be propagated to all its models, which incurs a significant overhead. Hence, the storage of such models is error-prone and leads to model inconsistencies.

We illustrate the discussed challenge referring to the two examples demanding several models capturing one business process with different precision. First, consider the model of the business process “Registration form processing” presented in Fig. 1.1. The model adheres to the EPC notation and describes a business process of a German health insurance company AOK. This EPC is developed by the process modeling experts at AOK and contains well over 300 nodes with about 150 activities among them. The model is annotated with information about the average activity execution duration and the path execution probabilities. This information allows the AOK human resources department to estimate the average time needed for process realization and the amount of consumed resources. The high complexity of this model stems from its pragmatic feature—the modeling goal and the intended use. The presented model is clearly inappropriate for the communication between business users: the huge model size overwhelms the reader with exhaustive details and impedes rapid process understanding. Since AOK wants to support its management with process specifications, its designers have to deliver yet another model of this business process. According to the established business process modeling guidelines, such a model should be bounded to the size of approximately 50 elements [20, 103]. Obviously, these two models increase the maintenance effort threatening model consistency: once a business process is changed, all its models must be updated accordingly. Being already in possession of the detailed process model AOK demands a method for the derivation of an abstract process specification from the existing one in order to lower the model management burden.

The AOK scenario is real world and perfectly illustrates the complexity of industrial process models. However, we consistently reference the business process “Forecast request handling” throughout this thesis as a motivating ex-

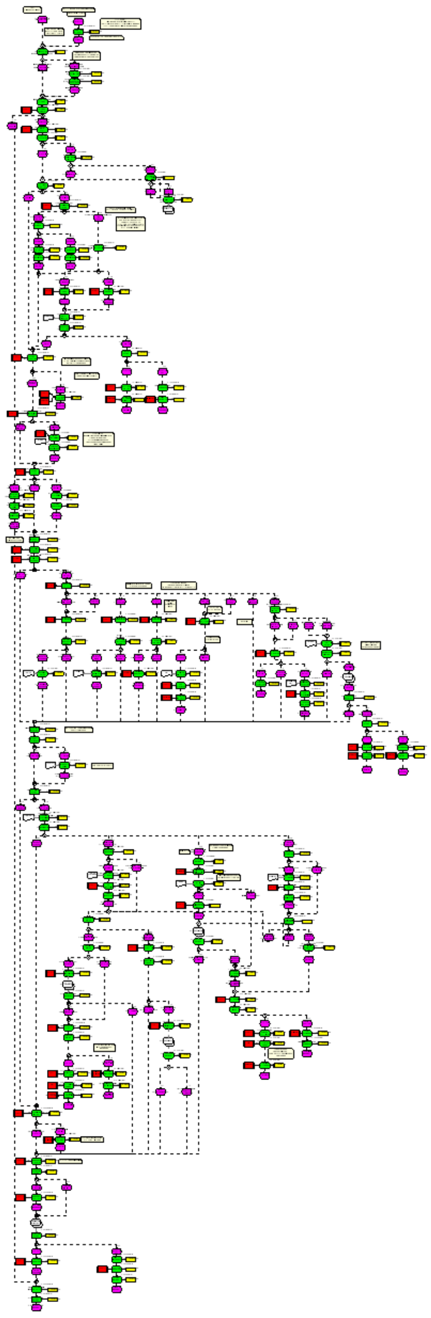


Fig. 1.1. An industrial business process model of high complexity. The model is an EPC capturing the business process “Registration form processing” by means of more than 300 nodes including around 150 activities.

The forecast request processing begins, once a clerk receives an email with a forecast request. Upon request receipt, the clerk requests gathering of data for the forecast. Then, the clerk records the request and awaits until the requested data is available. The clerk archives the received data. From this point the forecast handling has two alternative evolutions. The first option is a "quick" analysis performed by an analyst that includes a data preparatory step and the quick analysis itself. The second evolution of the process consists of a "full" data analysis and an auxiliary simulation. In this case a senior analyst creates the forecast. The senior analyst first prepares the data for the full analysis; the prepared data is used as the input for the fully fledged data analysis and the simulation. The results of the analysis and the simulation are consolidated. Disregard of the chosen analysis type, the clerk always concludes the business process generating a forecast report and sending it to the customer.

Table 1.1. Description of the business process "Forecast request handling".

ample: it is simple and easy to follow. Table 1.1 textually describes the process, while Fig. 1.2 visualizes its three models. Model PM provides the most precise process description. Model PM_a provides less details, while model PM'_a is the most abstract one of the three. Model PM contains several semantically related activities that can be aggregated together into more coarse-grained ones. In model PM the groups of related activities are marked by areas with a dashed border, for instance, group g_1 with members {*Receive request via email*, *Record request*}. Each activity set corresponds to a respective high-level activity in the abstract model PM_a, e.g., {*Receive request via email*, *Record request*} relates to *Receive forecast request*. Model PM'_a provides a more abstract view on "Forecast request handling" business process: its activities are refined by the activities of model PM_a and can be further refined with activities of PM. This constellation results in model maintenance challenges: a change of the business process implies the update of all the three models.

Against this background, *business process model abstraction* emerged as a technique reducing the number of models describing one business process at different abstraction levels. Business process model abstraction is an operation on a business process model preserving essential process properties and leaving out insignificant details in order to retain information relevant for a particular purpose. In this way, abstraction allows to derive less detailed process models from precise ones. We notice that *abstraction* is a fundamental concept in computer science as well as in software engineering. Our notion of business process model abstraction aligns with the understanding of abstraction in these disciplines. For instance, Aho and Ullman argue that "computer science is a science of abstraction, creating the right model for a problem and devising the appropriate mechanizable techniques to solve it", see [12]. In the context of software engineering Larman defines abstraction as "the

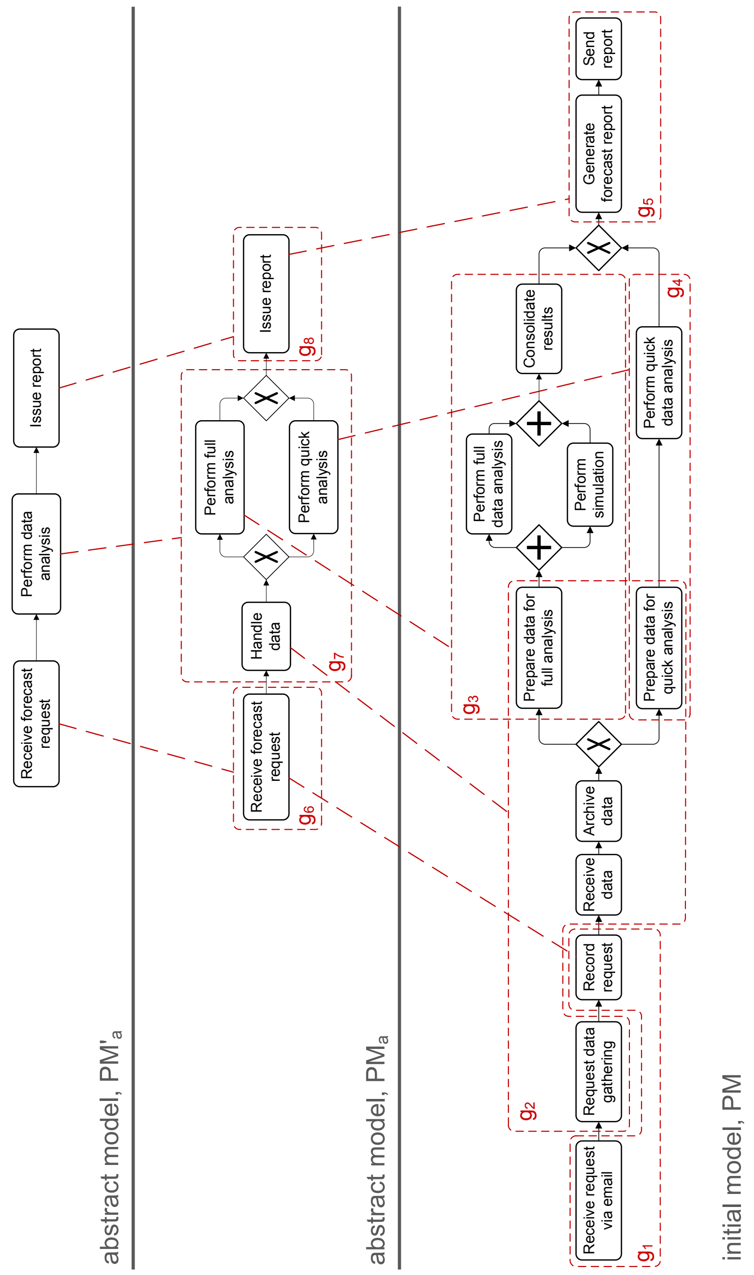


Fig. 1.2. Three models capturing business process "Forecast request handling" at different levels of abstraction.

act of concentrating the essential or general qualities of similar things. Also, the resulting essential characteristics of a thing" in [87]. Against this background, we position business process model abstraction as a specific type of abstraction related to process modeling.

Business process model abstraction has been addressed by several research endeavors. We observe that the existing contributions share two properties. First, each paper investigates one method of business process model abstraction, either addressing a specific abstraction scenario, or even leaving the application perspective out of scope. The examples of research contributions in the first category are [32, 34, 35, 55, 66, 121, 122], while [28, 30, 94, 115] advocate generic abstraction methods. This situation leads to a paradox: while technical solutions for individual subproblems are available, no coherent description of the problem's "big picture" exists. This means that the related work lacks a consistent description of abstraction concepts. In addition, the demand for various abstraction methods is explored sporadically. Second, to the best of our knowledge, the existing abstraction methods are realized as model transformations driven by process model structure and ignore the business semantics of model elements. In other words, the existing abstraction methods consider only the process model structure to conceal insignificant process details. For instance, in [94] Liu and Shen realize abstraction by means of the reduction rules developed by [135]. According to [94], an insignificant activity can be aggregated with its neighboring activity increasing the model abstraction level. Polyvyanyy et al. argue how process model decomposition can be leveraged to conceal insignificant process details, see [122, 123]. Such abstraction methods are agnostic to the business semantics of model elements: they neither take into account the semantics of abstracted elements, nor the semantics of the outcome. Hence, it is the user who assures that an abstraction delivers a process model having business meaning. This state of the art in business process model abstraction motivates the research goal of this thesis and its key contributions.

1.2 Research Contributions

The research goal of this thesis is to investigate the problem of business process model abstraction and develop abstraction methods addressing the actual user demand. To achieve this goal we first establish a framework that describes the domain of business process model abstraction. This framework distinguishes two types of model transformations that realize an abstraction: elimination and aggregation. While elimination omits model elements, aggregation puts them together into more coarse-grained ones. This thesis also extends the body of knowledge with a catalog of abstraction use cases demanded by industry. Building on the analysis of the use case catalog, we further focus on the category of use cases most demanded by practitioners. Due to this, we scope our study to abstraction realized as activity aggregation. We approach the problem from different angles. First, we investigate how analysis of the process model structure facilitates business process model abstraction. Second, we develop and evaluate advanced methods that enable non-hierarchical abstraction addressing the business semantics of model elements. Finally, we systematically investigate the related work and organize it by means of the developed framework. Notice that this thesis studies the impact of business process model abstraction on the process model activities and the ordering constraints between them. Thereafter, model elements capturing such artifacts as process data or organizational roles play only an auxiliary part. Against this state of the art, we position the main research contributions of this thesis.

Business Process Model Abstraction Framework

The BPM community tackled the research challenges of business process model abstraction by a number of papers [29, 30, 32, 55, 94]. However, typically each research endeavor focuses on one particular user demand and the related abstraction use case. Thereafter, the existing research contributions add "tiles" to the "big picture" of business process abstraction. This thesis builds on top of the existing contributions and establishes a framework of business process model abstraction. This framework decomposes the problem into three subproblems: why the abstraction is invoked, when model elements are abstracted, and how the abstraction is realized. The developed framework provides a coherent view on business process model abstraction and, hence, enables the comparison of the existing abstraction methods facilitating the identification of research gaps.

Catalog of Business Process Model Abstraction Use Cases

To the best of our knowledge there is no comprehensive study of a user demand in abstraction methods. This thesis presents a novel catalog of business process model abstraction use cases. The catalog is the outcome of an empirical study that analyzed the related research contributions, e.g., [29, 34, 35, 55, 66, 94, 123], and summarized the interviews with BPM experts from industry: consultants, software vendors, and end users. The catalog relates the abstraction use cases by means of the previously introduced framework. Furthermore, the use cases are prioritized according to the user demand. The added value of the catalog is twofold. On the one hand, the BPM practitioners and researchers benefit form the catalog, as it provides a consistent view on the industrial demand for business process model abstraction. On the other hand, the most prominent use cases determine the research direction of this thesis: the developed abstraction methods enable these use cases.

Abstraction of a Process Model According to Model Structure

This thesis engineers two approaches to process model abstraction analyzing process model structure: pattern-based and decomposition-based. The proposed pattern-based method leverages well-established model transformation rules [4, 29, 32, 55, 135]. It extends the body of knowledge with a new algorithm orchestrating the pattern application. While process model decomposition recently became a prominent topic, e.g., see [73, 124, 154, 155], this thesis develops a novel algorithm of business process model abstraction that uses process model decomposition. In addition, we discuss the pros and cons of pattern-based and decomposition-based structural abstraction methods.

Discovery of Semantically Related Activities within a Process Model

Several identified abstraction use cases synthesize a process model with the activities more coarse-grained than the activities of the initial model. Each of such coarse-grained activities abstracts a set of activities in the initial model. Thereafter, there is a demand for methods discovering sets of semantically related activities of one model. While the change of activity granularity has been tackled by a number of research papers [65, 66, 90, 128, 129, 131], this thesis introduces and empirically validates two novel methods. The first method argues how activity meronymy relation facilitates identification of related activity sets enabling a non-hierarchical abstraction. The second method implies availability of process models enriched with non-control flow information and adapts clustering analysis for activity aggregation. Both methods deliver sets of activities that semantically belong together. In contrast to the related research on process mining [65, 66], the developed methods focus on the information inherent to the process model level, rather than the instance level. Meanwhile, our contribution complements the related research on activity granularity in process models [90, 128, 129, 131] arguing how the activity sets are discovered. To evaluate the newly developed activity aggregation methods, we use the sets of industrial process models and the modeling expert judgment.

Control Flow Discovery within a Non-Hierarchical Business Process Model Abstraction

The majority of available business process model abstraction methods enable hierarchical abstraction, for instance, see [29, 55, 94, 66]. However, there is a demand for non-hierarchical refinement and generalization operations [49, 60, 82, 157]. This thesis develops a novel algorithm enabling non-hierarchical business process model abstraction. Given as inputs 1) a process model and 2) activity groups in this model, the algorithm synthesizes an abstract process model with activities corresponding to the groups. The algorithm generalizes the ideas of structural process model abstraction, since it allows free definition of a related activity set. We complement the conceptual discussion with a presentation of the algorithm's software implementation—the application FLEXAB. Along with the discussion of this concrete implementation we argue about the application aspects of the proposed algorithm.

1.3 Thesis Structure

This thesis is organized into 8 chapters. We start introducing the basic concepts and the formalization of business process model abstraction. After the thesis presents the catalog of abstraction use cases, we elaborate on methods that realize abstraction: from basic structural to advanced addressing the semantics of model elements. Finally, the thesis outlines the related work concluding with a contribution summary and an outlook of the future research.

Chapter 1: Introduction

The Introduction briefly motivates the research problem investigated by this thesis. Further, the chapter names the thesis's main research contributions and defines the thesis's structure.

Chapter 2: Preliminaries

This chapter provides the formal background for the rest of the thesis. The goal of the "Preliminaries" chapter is twofold. On the one hand, it brings the reader to the formal notion of a process model. On the other hand, this chapter introduces auxiliary concepts used throughout this thesis. For instance, we postulate the notions of a graph and its special classes, discuss two types of process model decomposition and explain the concept of behavioral profiles—a process behavioral abstraction.

Chapter 3: Business Process Model Abstraction: Theory and Practice

The "Business Process Model Abstraction: Theory and Practice" chapter argues about business process model abstraction from theoretical and practical perspectives. First, the chapter develops a framework identifying the main concepts of business process model abstraction and relations between them. We elaborate on the operations realizing abstraction and discuss prominent operation classes. Second, the chapter presents a novel catalog of use cases that reflects the demand of practitioners.

Chapter 4: Structural Methods of Business Process Model Abstraction

Chapter 4 shows how process model structure can be used for business process model abstraction. The key idea of structural abstraction is to conceal insignificant model details transforming the process model fragments containing these details. Each such fragment is either omitted or substituted for a more coarse-grained model element. While the former abstraction method is trivial, we focus on the latter one—aggregation. Building on the well established methods of process model analysis we propose two algorithms for aggregation: pattern-based and decomposition-based. We discuss the properties of each algorithm and compare them.

Chapter 5: Discovery of Related Activities in Process Models

Building on the lessons learned within the user study in Chapter 3, we investigate abstraction methods that increase activity granularity. Since the structural methods of activity aggregation discover related activity sets assuming that they belong to a particular process model fragment, they enable *hierarchical* abstraction only. This property is a strong limitation of the structural methods. To eliminate this restriction, Chapter 5 investigates alternative means to discover sets of related activities within one process model. The outcome of the study are two new methods that analyze the information about process model activities and find sets of related activities. The chapter not only engineers the two methods, but provides their empirical evaluation.

Chapter 6: Controlling Control Flow Loss

This chapter argues how the control flow information of the abstract process model is derived, once the initial process model and the sets of related activities are available. In contrast to the abstraction methods advocated in literature, e.g., see [55, 94, 123], and in Chapter 4, this original approach allows for non-hierarchical abstraction, where the related activities of the initial process model are freely distributed over the model. The model transformation approach makes use of *behavioral profiles*—an abstraction of the process behavior. Providing a thorough conceptual discussion of the approach, the chapter supplements it by a brief summary of abstraction method implementation. The proposed solution complements the contribution of Chapter 5 and overcomes the limitations of traditional structural abstraction methods.

Chapter 7: Related Work

The “Related Work” chapter elaborates on the research contributions related to business process model abstraction. We organize the related work into three streams. First, we in detail survey the existing business process model abstraction methods. Second, we describe the model transformations that can be used as process model abstraction enablers. Third, we outline the contributions on software engineering and business process management that relate to the topic of business process model abstraction. Finally, the chapter compares the works on business process model abstraction using the framework introduced in Chapter 3. As an outcome, we contrast the most investigated questions within business process model abstraction with the research gaps.

Chapter 8: Conclusion

The concluding chapter summarizes the contributions of this thesis, points to the research areas in the close proximity of business process model abstraction, and provides an outlook of the next research steps.

2

Preliminaries

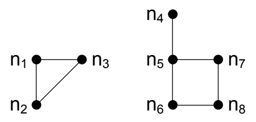
This chapter accumulates the formalisms fundamental for modeling of business processes and process model transformations advocated by this thesis. Section 2.1 briefly summarizes the concept of a graph and its special classes this thesis refers to. In Section 2.2 we familiarize the reader with Petri nets—the formalism enabling compact representation of concurrent systems. First, we explain the basic concepts of Petri nets and then elaborate on their special class—workflow nets. Workflow nets have been introduced by van der Aalst in [1] and are of particular importance for modeling working procedures of organizations. Building on the formalism of Petri nets Section 2.3 postulates the notion of a process model this thesis adheres to. Furthermore, Section 2.3 provides the formal background for process model decomposition methods and behavioral abstraction of processes—behavioral profiles. Section 2.4 concludes this chapter.

2.1 Graphs

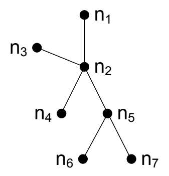
As the first step we postulate the notion of a graph, e.g., see [70].

Definition 2.1 (Graph).
A *graph* is a tuple $\mathsf{G} = (N, E)$, where N is a finite nonempty set of nodes and $E \subseteq N \times N$ is a set of edges.

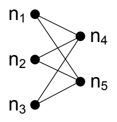
Fig. 2.1 presents several graph examples. The graphs G_1, G_2, G_3, and G_4 are *undirected* graphs, for their edges are unordered pairs. A *complete graph* is an undirected graph in which every pair of distinct nodes is connected by a unique edge, e.g., see graph G_4. A graph $\mathsf{G} = (N, E)$ is *bipartite*, iff N can be divided into two disjoint sets U and V such that for every edge $(u, v) \in E : u \in U$ and $v \in V$. Among the graphs in Fig. 2.1, G_3 is the only bipartite graph (with $U = \{n_1, n_2, n_3\}$ and $V = \{n_4, n_5\}$). If graph edges are ordered activity pairs, the graph is *directed*. The graphs G_5 and G_6 are directed graphs, see Fig. 2.1(e) and Fig. 2.1(f), respectively.



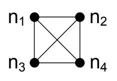
(a) Undirected not connected graph G_1.



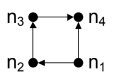
(b) Acyclic connected graph G_2.



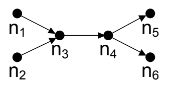
(c) Bipartite graph G_3.



(d) Complete graph G_4.



(e) Directed graph G_5.



(f) Directed graph G_6.

Fig. 2.1. Some graph examples.

We use the notions of *postset* and *preset* for the nodes in a directed graph. Let $\mathsf{G} = (N, E)$ be a directed graph. A node $n' \in N$ belongs to the *preset* of a node $n \in N$, iff there is an edge (n', n), i.e., $(n', n) \in E$. The preset of a node is denoted as $\bullet n$. The *postset* of a node $n \in N$ contains such nodes $n' \in N$ that $(n, n') \in E$. We reference a postset of node n as $n\bullet$. For instance, in the graph G_5 we observe $\bullet n_2 = \{n_1\}$ and $n_1\bullet = \{n_2, n_4\}$. For a node $n \in N$ of the directed graph $\mathsf{G} = (N, E)$ the set $in(n)$ is the *set of incoming edges* such that $in(n) = \{(n', n) | n' \in \bullet n\}$. Analogously, the *set of outgoing edges* of a node n is defined as $out(n) = \{(n, n') | n' \in n\bullet\}$. Returning to the example graph G_5, we notice $in(n_2) = \{(n_1, n_2)\}$ and $out(n_1) = \{(n_1, n_2), (n_1, n_4)\}$.

Definition 2.2 (Path).
A *path* in the graph $\mathsf{G} = (N, E)$ is a sequence of nodes $(n_1, \ldots, n_l)$ such that $(n_i, n_{i+1}) \in E$, where $i, l \in \mathbb{N}$ and $1 \leq i \leq l - 1$.

Graph G_1 exhibits paths (n_1, n_2, n_3, n_1) and $(n_4, n_5, n_6, n_8, n_7)$. A *cycle* in the graph $\mathsf{G} = (N, E)$ is a path $(n_1, \ldots, n_l)$, where $n_1 = n_l$. Paths (n_1, n_2, n_3, n_1) and $(n_5, n_6, n_8, n_7, n_5)$ in graph G_1 are cycle examples. Graph G_2 does not contain cycles.

A graph $\mathsf{G} = (N, E)$ is a *connected graph*, if for each pair of nodes $n, n' \in N$, where $n \neq n'$, there is a path from n to n'. Graph G_2 is a connected graph, since from each node of G_2 there is a path to every other node of this graph. At the same time, G_1 is not a connected graph: for instance, there is no path from n_3 to n_5. A directed graph is *weakly connected* if replacing all of its directed edges with undirected edges produces a connected undirected graph, see graph G_5 in Fig. 2.1 as an example.

Definition 2.3 (Tree).
A connected graph that has no cycles is a *tree*.

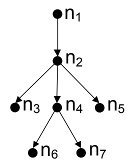


Fig. 2.2. The graph G_7 that is an arborescence.

Graph G_2 is a tree: it is acyclic and connected. As G_1 is not a connected graph and has cycles, it is not a tree. A *directed tree* is a directed graph which would be a tree, if the directions on the edges were ignored. A tree is called a *rooted tree* if one vertex is the designated root, i.e., $E \subseteq N \times (N \backslash \{r\})$. We denote a rooted tree with $\mathsf{G} = (N, r, E)$, where r is the root. For instance, tree G_2 can be rooted to nodes n_1 or n_2. We make use of a special class of rooted trees—arborescence. An *arborescence* is a directed rooted tree, where there is exactly one directed path from the root r to each node, see Fig. 2.2 for an arborescence example.

The *lowest common ancestor* (LCA) is defined between two nodes n_1 and n_2 as the lowest node in $\mathsf{G} = (N, r, E)$ that has n_1 and n_2 as descendants, where a node is a descendant of itself. In Fig. 2.2 we observe, for example, that n_2 is the LCA of n_3 and n_6, while n_5 is the LCA of n_6 and n_7. While the lowest common ancestor concept is well defined for a pair of nodes, we extend it to a node set. For a set of nodes $N' \subseteq N$ in the tree $\mathsf{G} = (N, r, E)$ its lowest common ancestor is the lowest node having all the activities of N' as descendants. For a given arborescence $\mathsf{G} = (N, r, E)$ we introduce a mapping $lca : \mathcal{P}(N) \rightarrow N$ that for a set of nodes returns its lowest common ancestor. Returning to the example graph in Fig. 2.2 we see that $n_2 = lca(\{n_3, n_6, n_7\})$

A directed graph is called *weakly connected* if a replacement of all its directed edges with undirected edges produces a connected undirected graph, for an example consider the graph G_7 in Fig. 2.3. In the sequel we are interested in a special class of directed graphs referenced as *control flow graphs.*

Definition 2.4 (Control Flow Graph).
A tuple $\mathsf{G} = (N, E, n_s, n_e)$ is a *control flow graph*, where:

– (N, E) is a weakly connected directed graph
– $n_s \in N$ is exactly one node having no incoming edges, the start node
– $n_e \in N$ is exactly one node having no outgoing edges, the end node
– every node $n \in (N \backslash \{n_s, n_e\})$ is on a path from n_s to n_e.

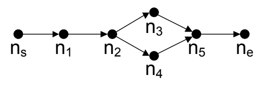


Fig. 2.3. G_8 is a control flow graph.

Fig. 2.3 exemplifies a control flow graph concept. The node n_s is the start node of G_8, while node n_e is its end node. The nodes n_1, n_2, n_3, n_4, n_5 are on the paths from n_s to n_e. For the nodes of control flow graphs the dominance and postdominance relations are introduced.

Definition 2.5 (Dominance).
Let $\mathsf{G} = (N, E, n_s, n_e)$ be a control flow graph. We say that node $n \in N$ *dominates* node $n' \in N$ if every path from n_s to n' includes n. We denote this fact as $n \triangleright n'$. Node n is a *dominator* of node n'.

Definition 2.6 (Postdominance).
Let $\mathsf{G} = (N, E, n_s, n_e)$ be a control flow graph. We say that node $n \in N$ *postdominates* node $n' \in N$ if every path from n' to n_e includes n. We denote this as $n \triangleleft n'$. Node n is a *postdominator* of node n.

In the example control flow graph in Fig. 2.3 we observe $n_2 \triangleright n_5$, since both paths from n_s to n_5 contain n_2. Meanwhile, n_3 is not a dominator of n_5, as there is path $(n_s, n_1, n_2, n_4, n_5)$ that does not contain n_3. However, $n_5 \triangleleft n_3$: the only path from n_3 to n_e contains n_5. We transfer the concepts of dominance and postdominance to edges.

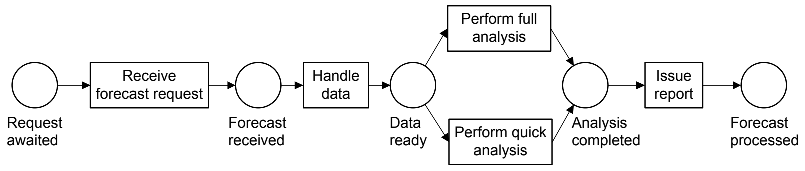
2.2 Petri Nets

Petri nets is the formalism well suited for modeling concurrent systems: they enable succinct representation of concurrency [116, 132]. Since concurrency is inherent to processes, practitioners and researchers leverage Petri nets to model the business processes [1, 2, 9, 165]. This section elaborates on the Petri net classes and properties important in the context of business process modeling. We start by defining the Petri net syntax.

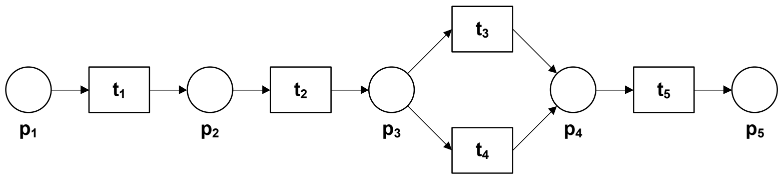
Definition 2.7 (Petri Net).
Petri net is a tuple $\mathsf{PN} = (P, T, F)$, where:

- P is a finite nonempty set of *places*
- T is a finite nonempty set of *transitions*
- $P \cap T = \emptyset$
- $F \subseteq (P \times T) \cup (T \times P)$ is the *flow relation.*

Definition 2.7 is equivalent to the definition of a Petri net as a bipartite graph. Places and transitions are the graph nodes, where places are depicted as circles and transitions—as rectangles. The flow relation is the edges of the graph. Further we assume the edges of the Petri net to have a weight of 1. Notice that in the context of process modeling transitions represent activities, while places capture conditions and the flow relation corresponds to process model control flow. As an example we consider the Petri net $\mathsf{PN}_2 = (P, T, F)$, where:



(a) Petri net PN_1.



(b) Petri net PN_2.

Fig. 2.4. Two examples of Petri nets: the net PN_1 models the business process "Forecast request handling" , while the net PN_2 is the anonymized version of PN_1.

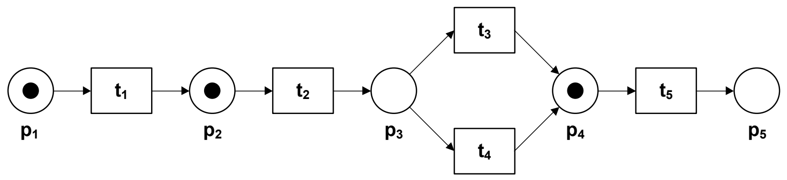
- $P = \{p_1, p_2, p_3, p_4, p_5\}$
- $T = \{t_1, t_2, t_3, t_4, t_5\}$
- $F = \{(p_1, t_1), (p_2, t_2), (p_3, t_3), (p_3, t_4), (p_4, t_5), (t_1, p_2), (t_2, p_3), (t_3, p_4), (t_4, p_4), (t_5, p_5)\}$.

An equivalent graph definition for the Petri net PN_2 is provided in Fig. 2.4(b). The net PN_2 is the anonymized version of the Petri net PN_1 in Fig. 2.4(a) that models the business process "Forecast request handling" introduced as the running example in Chapter 1. For the sake of brevity and readability the remainder of this section illustrates the Petri net concepts by means of the net PN_2.

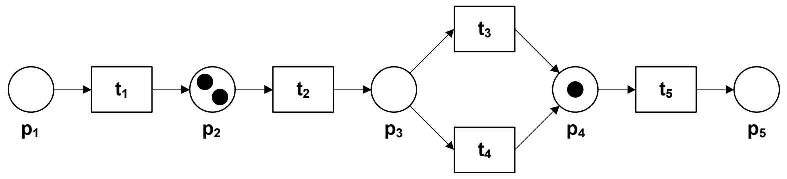
A place $p \in P$ is called an input place of a transition $t \in T$, iff there exists an edge from p to t: $(p, t) \in F$. Following the conventions introduced for directed graphs, we denote the set of input places for a transition t as $\bullet t$. A place p is an output place of a transition t, iff there exists an edge from t to p: $(t, p) \in F$. The set of output places for transition t is denoted as $t\bullet$. $p\bullet$ and $\bullet p$ denote the sets of transitions that share p as an input place and an output place, respectively. In the Petri net PN_2, we observe:

- p_1 (p_2) is the input (output) place of t_1
- $\bullet t_2 = \{p_2\}$ and $t_2\bullet = \{p_3\}$
- $\bullet p_3 = \{t_2\}$ and $p_3\bullet = \{t_3, t_4\}$.

The dynamic process evolution is captured by behavioral semantics of Petri nets. Each place of a Petri net contains zero or more tokens at any moment. A token is visualized as a black dot allocated within a circle denoting the place marked by this token, see Fig. 2.5. The process state is captured by marking—a distribution of tokens over the Petri net places.



(a) Petri net with marking $[p_1, p_2, p_4]$.



(b) Petri net with marking $[2 \cdot p_2, p_4]$.

Fig. 2.5. Two markings for the Petri net PN_2 in Fig. 2.4(b).

Definition 2.8 (Petri Net Marking).
For a Petri net $\mathsf{PN} = (P, T, F)$ *Petri net marking* is a function $M : P \to \mathbb{N}_0$ mapping a set of places onto the natural numbers with zero.

A common notation for Petri net markings is $[k_1 \cdot p_1, \ldots, k_{|P|} \cdot p_{|P|}]$, where p_i is the place of the Petri net $\mathsf{PN} = (P, T, F)$ and $k_i = M(p_i)$ is the number of tokens in place p_i. According to the notation, places without tokens are omitted, while $k_i = 1$ are skipped. Fig. 2.5 provides two examples of marking for Petri net PN_2: $[p_1, p_2, p_4]$ and $[2 \cdot p_2, p_4]$ illustrated, respectively, in Fig. 2.5(a) and Fig. 2.5(b).

Definition 2.9 (Petri Net System).
A Petri net system (PN, M_0) is a Petri net $\mathsf{PN} = (P, T, F)$ with an initial marking M_0 of PN.

For instance, $(\mathsf{PN}_2, [p_1, p_2, p_4])$ is a Petri net system. While a Petri net system describes the state at a point in time, the Petri net semantics defines the rules how one state evolves into another. Let (PN, M) be a Petri net system, where $\mathsf{PN} = (P, T, F)$ is the Petri net and M—its marking. A transition $t \in T$ is enabled, iff $\forall p \in \bullet t : M(p) > 0$. The reached marking after firing of t is M', such that:

– $M'(p) = M(p) - 1$, $\forall p \in \bullet t / t \bullet$
– $M'(p) = M(p) + 1$, $\forall p \in t \bullet / \bullet t$
– $M'(p) = M(p)$.

In Fig. 2.5(a) transitions t_1 and t_2 are enabled. At the same time, t_2, t_3, and t_4 are not enabled. Firing of the transition t_1 brings Petri net system $(\mathsf{PN}_2, [p_1, p_2, p_4])$ to system $(\mathsf{PN}_2, [2 \cdot p_2, p_4])$. Firing of the transition t is denoted as $(\mathsf{PN}, M) \xrightarrow{t} (\mathsf{PN}, M')$ shortcuted as $M \xrightarrow{t} M'$. We write $M_1 \xrightarrow{*} M_l$

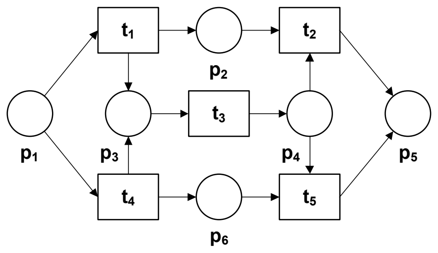


Fig. 2.6. Petri net PN_3 is not free choice: $\bullet t_2 \cap \bullet t_5 \neq \emptyset$, but $|\bullet t_2| = 2 = |\bullet t_5|$.

iff there is a sequence of transitions $t_1, t_2, \ldots, t_{l-1}$ such that $M_i \overset{t_i}{\to} M_{i+1}$, where $i, l \in \mathbb{N}$ and $i = 1, \ldots, l$. The state M' is reachable from the state M iff $M \overset{*}{\to} M'$. Using this notation we write $(\mathsf{PN}_2, [p_1, p_2, p_4]) \overset{t_1}{\to} (\mathsf{PN}_2, [2 \cdot p_2, p_4])$ and $(\mathsf{PN}_2, [p_1, p_2, p_4]) \overset{*}{\to} (\mathsf{PN}_2, [3 \cdot p_5])$.

A Petri net $\mathsf{PN} = (P, T, F)$ is *live*, iff for every reachable state M' and every transition t there is a state M'' reachable from M' which enables t. If for the transition t there is no such state M'', t is called a *dead transition.*

Further we consider a special class of Petri nets—free choice nets. Free choice nets are of practical significance, as they combine expressive power with possibilities to verify their formal properties [23, 44].

Definition 2.10 (Free Choice Petri Net).
Petri net $\mathsf{PN} = (P, T, F)$ is a *free choice Petri net*, iff $\forall t, t' \in T : \bullet t \cap \bullet t' \neq \emptyset \Rightarrow \bullet t = \bullet t'$.

Whilst the Petri nets in Fig. 2.4 are free choice nets, the net in Fig. 2.6 is not free choice. Indeed, $\bullet t_2 \cap \bullet t_5 \neq \emptyset$, but $\bullet t_2 \neq \bullet t_5$. Notice that in [23] Best references the class of models described by Definition 2.10 as *extended free choice Petri nets*. Further, [23] defined free choice nets as those where $\forall t, t' \in T, t \neq t' : \bullet t \cap \bullet t' \neq \emptyset \Rightarrow |\bullet t| = 1 = |\bullet t'|$. The latter are the subclass of nets discussed in Definition 2.10.

In the context of business process modeling another class of Petri nets, *workflow nets* (WF-nets), gets into focus [1, 3]. Such nets exhibit a distinguished start and end places that signify the beginning and the completion of the working procedure.

Definition 2.11 (Workflow Net).
A *workflow net* is a tuple $\mathsf{WFN} = (P, T, F, i, o)$, where:

– (P, T, F) is a Petri net
– (P, T, F) has a distinguished place $i \in P$, such that $\bullet i = \emptyset$
– (P, T, F) has a distinguished place $o \in P$, such that $o \bullet = \emptyset$
– $\forall n \in P \cup T$ n is located on a path from i to o.

Both Petri nets in Fig. 2.4 are the WF-nets. Let $\mathsf{WFN} = (P, T, F, i, o)$ be a WF-net and M, M' its markings. Let i be the state in which there is exactly

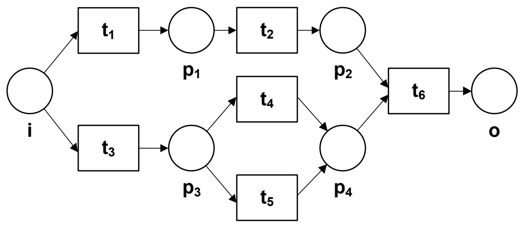


Fig. 2.7. Workflow net WFN_1 exemplifies the class of unsound WF-nets. Indeed, the transition t_6 awaits for two tokens in places p_2 and p_4, whilst only one arrives.

one token in place $i \in P$ and no token in any other place of the workflow net. Respectively, we reference as o the state in which there is exactly one token in place $o \in P$ and no token in any other place of the workflow net. From a business process modeling perspective the class of *sound* WF-nets plays an important role. For a sound WF-net three conditions hold 1) each task participates in an instance, 2) each instance terminates, and 3) when an instance terminates, there is exactly one token in the place o. Formally the class of sound WF-nets is defined as follows.

Definition 2.12 (Sound Workflow System).
A workflow system $(\mathsf{WFN}, \mathsf{i})$ with a workflow net $\mathsf{WFN} = (P, T, F, i, o)$ is *sound,* iff:

- for every state M reachable from state i there exists a firing sequence leading from M to o: $\forall M (\mathsf{i} \xrightarrow{*} M) \Rightarrow (M \xrightarrow{*} \mathsf{o})$
- state o is the only state reachable from state i with at least one token in place o: $\forall M (\mathsf{i} \xrightarrow{*} M \wedge M \geq \mathsf{o}) \Rightarrow (M = \mathsf{o})$
- there are no dead transitions in the workflow net in state i: $\forall t \in T \exists M, M' : \mathsf{i} \xrightarrow{*} M \xrightarrow{t} M'$.

We consider a workflow net to be sound iff the corresponding workflow system is sound. Thereby, the WF-net PN_2 is sound, while WFN_1 in Fig. 2.7 is not.

We conclude this section returning the example Petri net PN_1. This is a sound free choice WF-net. According to the described Petri net semantics, PN_1 models the business process “Forecast request handling”: first *Receive forecast request* is executed, then *Handle data.* Afterwards, there is a choice whether to *Perform full analysis* or *Perform quick analysis.* Finally, an *Issue report* activity is executed.

2.3 Process Models

This section introduces the notion of a process model this thesis adheres to. We also elaborate on two types of process model decomposition used in the remainder of the work. Finally, we explain the concept of behavioral profiles—a process behavioral abstraction facilitating the abstraction method developed in Chapter 6.

2.3.1 Process Model Notion

This thesis leverages the notion of a process model introduced by Definition 2.13.

Definition 2.13 (Process Model).
A tuple $\mathsf{PM} = (A, G, F, t, s, e)$ is a *process model*, where:

– A is a finite nonempty set of activities
– G is a finite set of gateways
– $N = A \cup G$ is a finite set of nodes with $A \cap G = \emptyset$
– $F \subseteq N \times N$ is the flow relation, such that (N, F) is a connected graph
– $\forall\ a \in A : |\bullet a| \leq 1 \wedge |a \bullet| \leq 1$
– $\forall\ g \in G : (|\bullet g| = 1 \wedge |g \bullet| \geq 2) \vee (|\bullet g| \geq 2 \wedge |g \bullet| = 1)$
– $s \in A$ is the only start activity, such that $\bullet s = \emptyset$
– $e \in A$ is the only end activity, such that $e \bullet = \emptyset$
– $t : G \to \{and, xor\}$ is a mapping that associates each gateway with a type.

The execution semantics of a process model is given by a translation into a Petri net following on common formalizations [3, 47]. As a process model has a dedicated start activity and a dedicated end activity, the resulting Petri net is a WF-net. All gateways are of type *and* or *xor*, such that the WF-net is free choice. Fig. 2.8 visualizes a process model example.

Further we reference the gateways with multiple incoming edges as *joins* and gateways with multiple outgoing edges as *splits*. In the Fig. 2.8 we observe that the left gateway is the XOR split, while the right gateway is the XOR join. Finally, we make use of a *short circuited process model* concept. Given a process model PM, the respective short circuited process model is obtained from PM through the introduction of a back edge leading from the end activity e to the start activity s. For instance, if we add the edge from the *Issue report* activity to the *Receive forecast request*, we obtain a short-circuited process model.

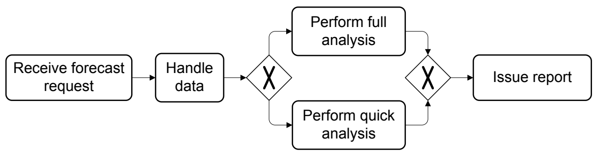


Fig. 2.8. Example of a process model. The model describes the business process "Forecast request handling" and can be mapped to a sound free choice WF-net, e.g., see the net PN_1 in Fig. 2.4(a).

2.3.2 Process Model Decomposition

This thesis refers to two process model decomposition methods. Each method results in a unique decomposition of a process model into a hierarchy of fragments. To introduce the two decomposition methods we define the notion of a process model fragment first.

Definition 2.14 (Process Model Fragment).
Let $\mathsf{PM} = (A, G, F, t, s, e)$ be a process model. A fragment f of process model PM is a tuple $\mathsf{f} = (A_f, G_f, F_f, t_f)$, where $(A_f \cup G_f, F_f)$ is the connected subgraph of the graph $(A \cup G, F)$ and function t_f is the restriction of t of PM to set G_f.

Given a process model, one can discover numerous fragments in it. Consider fragments PMF_1, PMF_2, PMF_3, and PMF_4 in Fig. 2.9. A process model can be decomposed into fragments in several ways, e.g., see [73]. In practice, however, only special kinds of process model decompositions are of value. This thesis builds upon two decomposition types. The first type is a decomposition into fragments with single entry *edge* and single exit *edge*, while the second type is the decomposition into fragments with single entry *node* and single exit *node*. The two decompositions have several important properties. First, in the context of process modeling the resulting fragments can be considered as self-contained process parts. As such fragments have single entry node and single exit node, structurally they can be isolated into a subprocess. Second, fragments do not "interleave": either one fragment fully contains another, or two fragments do not intersect. In other words, each decomposition results in a hierarchy of process model fragments according to the fragment containment relation. In the case of fragments with single entry/exit edge the decomposition results in a *process structure tree* (PST) [155], while in the case of single entry/exit node—a *refined process structure tree* (rPST) [154]. Third, these decompositions are unique. Against this background, these two decompositions are used as divide and conquer techniques in process model analysis, e.g., see [59, 154, 155, 163].

Process Structure Tree

First we elaborate on the process model decomposition into *canonical single entry single exit (SESE) fragments*. This decomposition has been proposed by Johnson, Pearson, and Pingali in the context of the program control flow analysis [79]. Vanhatallo, Voelzer, and Leymann introduced this decomposition technique to the area of business process modeling [155]. Informally, a SESE fragment is a fragment which has exactly one incoming edge and exactly one outgoing edge. Fig. 2.9 provides several examples of SESE fragments. Two SESE fragments, PMF_2 and PMF_4, are visualized as rectangles with dashed border, while the other two SESE fragments, PMF_1 and PMF_3, are highlighted with gray background. We formalize the concept of canonical SESE fragments by means of dominance and postdominance, see Definition 2.5 and Definition 2.6, respectively.

Definition 2.15 (Single Entry Single Exit (SESE) Fragment).
Let $\mathsf{PM} = (A, G, F, t, s, e)$ be a process model containing a process model fragment $\mathsf{PMF} = (A_{\mathsf{PMF}}, G_{\mathsf{PMF}}, F_{\mathsf{PMF}}, t_{\mathsf{PMF}})$. The fragment PMF is a *single entry single exit (SESE) fragment,* if it is defined by an ordered edge pair $((a_1, b_1), (a_2, b_2))$ of distinct control flow edges (a_1, b_1) and (a_2, b_2), where $(a_1, b_1) \rhd (a_2, b_2)$, $(a_2, b_2) \lhd (a_1, b_1)$, and every cycle containing (a_1, b_1) also contains (a_2, b_2) and vice versa. The sets A_{PMF}, G_{PMF}, F_{PMF} and the mapping G_{PMF} are defined as follows:

– an edge $(a, b) \in F$ belongs to the fragment PMF, i.e., $(a, b) \in F_{\mathsf{PMF}}$, if $(a, b) \lhd (a_1, b_1)$ and $(a_3, b_3) \rhd (a_2, b_2)$. The entry edge (a_1, b_1) and the exit edge (a_2, b_2) also belong to the fragment PMF
– $a \in A_{\mathsf{PMF}}$ if all incident edges of a belong to this fragment
– $g \in G_{\mathsf{PMF}}$ if all incident edges of g belong to the fragment PMF
– a function t_{PMF} is the restriction of t to set G_{PMF}.

We are interested in a specific class of SESE fragments—*canonical SESE fragments*. Formally, canonical SESE fragments are defined as follows.

Definition 2.16 (Canonical SESE Fragment).
Let $\mathsf{PM} = (A, G, F, t, s, e)$ be a process model containing a SESE fragment $\mathsf{PMF} = (A_{\mathsf{PMF}}, G_{\mathsf{PMF}}, F_{\mathsf{PMF}}, t_{\mathsf{PMF}})$. The fragment PMF is a *canonical SESE fragment,* if it is defined by the edge pair $((a_1, b_1), (a_2, b_2))$ such that:

– $(a_2, b_2) \rhd (a_4, b_4)$ for any SESE fragment defined by $((a_1, b_1), (a_4, b_4))$
– $(a_1, b_1) \lhd (a_3, b_3)$ for any SESE fragment defined by $((a_3, b_3), (a_2, b_2))$.

Fig. 2.9 references three canonical SESE fragments: PMF_0, PMF_2, and PMF_4. Notice that the model in this figure contains trivial canonical SESE fragments each including one activity only. However, Fig. 2.9 does not visualize them for the sake of readability. Given a process model PM we denote with Θ the set

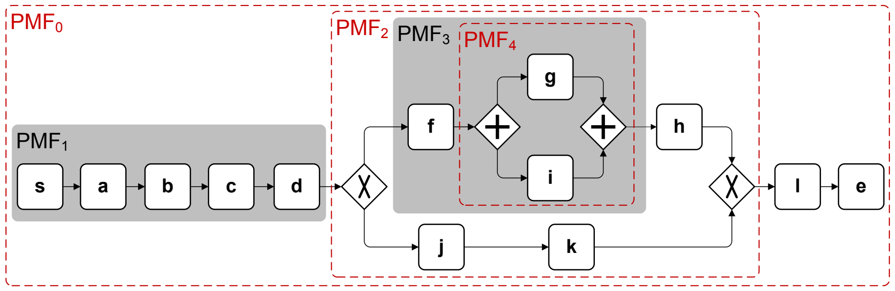


Fig. 2.9. A process model decomposed into canonical SESE fragments.

of all its canonical SESE fragments. We define two types of relations between canonical SESE fragments: parent-child and predecessor-successor.

From Definition 2.16 it follows that the node sets of two canonical SESE fragments are either disjoint or one contains the other. That is why a parent-child relation can be introduced for canonical SESE fragments. If the node set of the SESE fragment PMF is the subset of the node set of SESE fragment PMF', then PMF is the *child* of PMF' and PMF' is the *parent* of PMF. If PMF is the child of PMF' and there is no PMF'', such that PMF'' is the child of PMF' and PMF'' is the parent of PMF, PMF is the *direct child* of PMF'. In Fig. 2.9 the canonical SESE fragment PMF_2 is the direct parent of the fragment PMF_4.

Canonical SESE fragments can be organized into a hierarchy according to the parent-child relation—PST [155]. The tree nodes represent canonical SESE fragments. Let tree nodes n_1 and n_2 correspond to SESE fragments PMF_1 and PMF_2 respectively. An edge leads from tree node n_1 to n_2, once the SESE fragment PMF_1 is the direct parent of the SESE fragment PMF_2. Fig. 2.10 presents the PST for the process model from Fig. 2.9. Node PMF_0 is the root and corresponds to the whole process model. Since the canonical SESE fragment PMF_4 is the direct child of PMF_2, there is a directed edge between the corresponding nodes in the tree.

Two canonical SESE fragments can be in the predecessor-successor relation. We say that PMF precedes PMF' (and PMF' succeeds PMF) if the outgoing edge of PMF is the incoming edge of PMF'. The fact that PMF precedes PMF' is denoted as $\mathsf{PMF} \hookrightarrow \mathsf{PMF}'$, while PMF' succeeding PMF is denoted as $\mathsf{PMF}' \hookleftarrow \mathsf{PMF}$. One can observe that only the sibling nodes can be in the predecessor-successor relation. We define the PST formally using the parent-child and predecessor-successor relations.

Definition 2.17 (Process Structure Tree (PST)).
Let $\mathsf{PM} = (A, G, F, t, s, e)$ be a process model. The *process structure tree (PST)* of a process model PM is a tuple $\mathsf{PST}_{\mathsf{PM}} = (\Theta, r, \lambda, \hookrightarrow, \hookleftarrow)$, where:

– Θ is a set of all canonical SESE fragments of PM
– $\lambda \subseteq \Theta \times \Theta)$ is a parent-child relation
– (Θ, r, λ) is an arborescence rooted to the fragment r
– $\hookrightarrow \subseteq \Theta \times \Theta$ is a successor relation
– $\hookleftarrow \subseteq \Theta \times \Theta$ is a predecessor relation

In the PST we visualize sequences of nodes which are in predecessor-successor relation using dotted border rectangles. For instance, the canonical SESE fragments f and PMF_4 are put in the rectangle. The PST tree of the process model can be constructed in linear time [79].

Refined Process Structure Tree

The rPST discovers fragments with the single entry node and the single exit node. As an outcome, the decomposition is more fine grained. For instance,

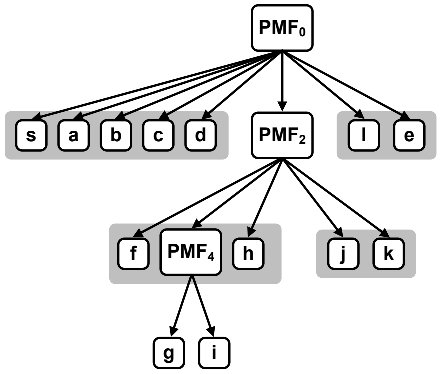


Fig. 2.10. A PST of the process model in Fig. 2.9.

the rPST directly distinguishes branches within blocks as the block's child fragments. To introduce the rPST, we start with the definition of a boundary node concept.

Definition 2.18 (Boundary Node).
Let $\mathsf{PM} = (A, G, F, t, s, e)$ be a process model with a process model fragment $\mathsf{PMF} = (A_{\mathsf{PMF}}, G_{\mathsf{PMF}}, F_{\mathsf{PMF}}, t_{\mathsf{PMF}})$. A node $n \in N_{\mathsf{PMF}}$ is a *boundary node* of PMF if $\exists e \in in(n) \cup out(n) : e \notin F_{\mathsf{PMF}}$. If n is a boundary node, it is an *entry* of PMF, if $in(n) \cap F_{\mathsf{PMF}} = \emptyset$. A node n is an *exit* of PMF, if it is a boundary node of PMF and $out(n) \cap F_{\mathsf{PMF}} = \emptyset$.

Considering the example in Fig. 2.11 we discover the two AND gateways to be the boundary nodes of the fragment B_2, while the XOR gateways bound the fragment B_1.

Definition 2.19 (Component).
Let $\mathsf{PM} = (A, G, F, t, s, e)$ be a process model with a process model fragment $\mathsf{PMF} = (A_{\mathsf{PMF}}, G_{\mathsf{PMF}}, F_{\mathsf{PMF}}, t_{\mathsf{PMF}})$. The fragment PMF is a *component* if it has exactly two boundary nodes: one entry node and one exit node.

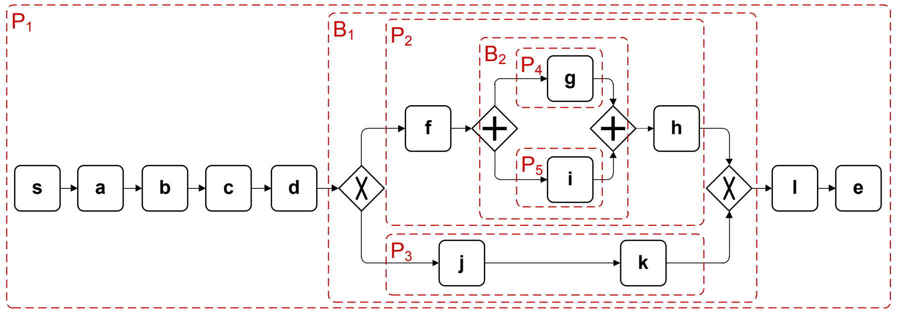


Fig. 2.11. A process model decomposed into canonical components.

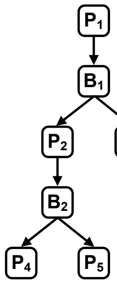


Fig. 2.12. An rPST of the process model in Fig. 2.11.

Let $\mathcal{F}$ be the set of all components in a process model PM.

Definition 2.20 (Canonical Component).
A component $\mathsf{PMF} = (A_{\mathsf{PMF}}, G_{\mathsf{PMF}}, F_{\mathsf{PMF}}, t_{\mathsf{PMF}})$ is *canonical* if $\forall \mathsf{PMF}' \in \mathcal{F}$: $\mathsf{PMF} \neq \mathsf{PMF}' \Rightarrow (F_{\mathsf{PMF}} \cap F_{\mathsf{PMF}'} = \emptyset \vee (F_{\mathsf{PMF}} \subset F_{\mathsf{PMF}'}) \vee (F_{\mathsf{PMF}'} \subset F_{\mathsf{PMF}}))$.

Fig. 2.11 shows all the canonical components in the model. Given a process model PM we denote with Ω the set of its all canonical components. As our notion of a process model allows no nodes with multiple incoming and outgoing edges, each canonical component is classified into one of the four classes: trivial, polygon, bond, and rigid. A trivial component is formed by a one edge. A polygon corresponds to a sequence of nodes or components. A set of components sharing common boundary nodes form a *bond*. A component of any other structure is a *rigid*. We shortcut the trivial type by T, polygon by P, bond by B, and rigid by R. Let $ft : \Omega \rightarrow \{T, P, B, R\}$ be a function that assigns a type to a component. In Fig. 2.11 we see that P_1, P_2, P_3, and P_4 are polygons, while B_1 and B_2 are bonds. If the two boundary nodes of a bond component are AND (XOR) gateways, we reference it as an AND-(XOR-)gateway-bordered bond component. Notice that Fig. 2.11 does not highlight the trivial components in the decomposition, i.e., the edges in the process model. Finally, we define the rPST.

Definition 2.21 (Refined Process Structure Tree (rPST)).
Let $\mathsf{PM} = (A, G, F, t, s, e)$ be a process model. The *refined process structure tree (rPST)* of a process model PM is an arborescence $\mathsf{RPST}_{\mathsf{PM}} = (\Omega, r, \chi)$ such that:

- Ω is a set of all canonical components of PM
- r is a component that is the root of the tree
- $\chi \subseteq \Omega \times \Omega$ is a relation between a component and its child component

Fig. 2.12 presents the rPST for the process model in Fig. 2.9. Finally, we note that the RPST can be constructed in time linear to the number of nodes in the process model [61, 67, 75, 124].

2.3.3 Behavioral Profiles

Behavior of systems described by our notion of a process model can be described in terms of behavioral profiles [162]. To arrive at a behavioral profile, we consider the set of all complete traces (or execution sequences) from start s to end e. The set of *complete process traces* $\mathcal{T}_{\mathsf{PM}}$ for a process model PM contains lists of the form $s \cdot A^* \cdot e$ such that a list comprises the execution order of activities. We use $a \in \sigma$ with $\sigma \in \mathcal{T}_{\mathsf{PM}}$ to denote that an activity a is a part of a complete process trace. The behavioral profile is based on *weak order* between activities. Two activities are in weak order, if there exists a complete trace in which one activity occurs after the other. This relation requires the *existence* of such a trace and does not have to hold for all traces of the model.

Definition 2.22 (Weak Order Relation).
Let $\mathsf{PM} = (A, G, F, t, s, e)$ be a process model, and $\mathcal{T}_{\mathsf{PM}}$ its set of traces. The *weak order relation* $\succ_{\mathsf{PM}} \subseteq (A \times A)$ contains all pairs (a, b), such that there is a trace $\sigma = n_1, \ldots, n_l$ in $\mathcal{T}_{\mathsf{PM}}$ with $j \in \{1, \ldots, l-1\}$ and $j < k \leq l$ for which holds $n_j = a$ and $n_k = b$.

Depending on how two activities of a process model are related by weak order, we define three relations forming the behavioral profile.

Definition 2.23 (Behavioral Profile).
Let $\mathsf{PM} = (A, G, F, t, s, e)$ be a process model. A pair $(a, b) \in (A \times A)$ is in one of the following relations:

– strict order relation $\rightsquigarrow_{\mathsf{PM}}$, if $a \succ_{\mathsf{PM}} b$ and $b \not\succ_{\mathsf{PM}} a$
– exclusiveness relation $+_{\mathsf{PM}}$, if $a \not\succ_{\mathsf{PM}} b$ and $b \not\succ_{\mathsf{PM}} a$
– interleaving order relation $||_{\mathsf{PM}}$, if $a \succ_{\mathsf{PM}} b$ and $b \succ_{\mathsf{PM}} a$.

The set of all three relations $\mathsf{BP} = \{\rightsquigarrow_{\mathsf{PM}}, +_{\mathsf{PM}}, ||_{\mathsf{PM}}\}$ is the *behavioral profile* of PM.

An example of the behavioral profile is given in Table 2.1. Every activity a in a behavioral profile is either exclusive to itself, i.e., $(a, a) \in +$, or in interleaving order with itself, i.e., $(a, a) \in ||$. The latter case is observed, once an activity is within a loop structure in the process model. We also denote the identity relation for activities in the behavioral profile BP as id_{BP}. The relations of the behavioral profile, along with the inverse strict order $\rightsquigarrow^{-1} = \{(a, b) \in (A \times A) \mid (b, a) \in \rightsquigarrow\}$, partition the Cartesian product of activities. For the class of the process models we address in this thesis the behavioral profile can be constructed in polynomial time $O(n^3)$, where n is the number of activities in a process model [162].

To conclude our discussion on behavioral profiles we make the following observation. The behavioral profile relations allow different degree of freedom for activities. While interleaving order relation allows the process activities to appear in an arbitrary order, (inverse) strict order specifies a particular execution order, and exclusiveness prohibits appearance of two activities in one

	s	a	b	c	d	f	g	h	i	j	k	l	e
s	$+_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$
a		$+_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$
b			$+_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$
c				$+_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$
d					$+_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$
f						$+_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$+_{\mathrm{PM}}$	$+_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$
g							$+_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\|_{\mathrm{PM}}$	$+_{\mathrm{PM}}$	$+_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$
h								$+_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}^{-1}$	$+_{\mathrm{PM}}$	$+_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$
i									$+_{\mathrm{PM}}$	$+_{\mathrm{PM}}$	$+_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$
j										$+_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$
k											$+_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$
l												$+_{\mathrm{PM}}$	$\rightsquigarrow_{\mathrm{PM}}$
e													$+_{\mathrm{PM}}$

Table 2.1. The behavioral profile of the model presented in Fig. 2.9.

trace. Thus, we organize the relations into a hierarchy presented in Fig. 2.13. At the top of the hierarchy the "strictest" relation appears, while at the bottom—the least restrictive.

2.4 Summary

This chapter postulated the key formal concepts of the thesis. While this thesis uses the generic concepts of graph theory in different contexts, the notion of

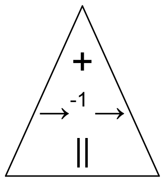


Fig. 2.13. Behavioral relation hierarchy.

Petri nets helps the reader to understand our notion of the business process model. The two introduced methods of process model decomposition serve as the mathematical foundations of the abstraction operations advocated by the thesis. Finally, the explained concept of behavioral profiles is the formal foundation of the abstraction method presented in Chapter 6.

3

Business Process Model Abstraction: Theory and Practice

Although business process model abstraction has been debated in several research papers, e.g., see [30, 55, 94, 122, 123], the term still lacks one coherent view. This chapter discusses business process model abstraction bringing together theoretical and practical perspectives. We open the discussion sketching the relations between process instance, process model, and *abstract* process model. Further, we present a framework that points out the core concepts of business process model abstraction and the relations between them. The chapter continues the theoretical discussion elaborating on the operations realizing business process model abstraction and mentioning prominent operation classes. The conceptual contribution is complemented by a catalog of use cases that reflects the demand of practitioners for business process model abstraction. The value of the catalog is twofold. One the one hand, it provides the insights into which use cases are demanded. On the other hand, the catalog indicates the user demand strength. Altogether, the chapter has two contributions: the business process model abstraction framework and the catalog of abstraction use cases.

The remainder of this chapter is structured as follows. Section 3.1 outlines the relations between the concepts of a process instance, process model, and abstract model. Section 3.2 presents the business process model abstraction framework. Section 3.3 argues about the properties of business process model abstraction operations. The designed catalog of business process model abstraction use cases is presented in Section 3.4. Finally, Section 3.5 summarizes the discussion.

3.1 Instance, Model, and Abstract Model

Chapter “Introduction” motivated the problem of business process model abstraction by the two examples. In addition, we briefly outlined the key abstraction artifacts. This section clarifies the relations between a process instance, a process model PM, and an abstract process model PM_a. First, we postulate

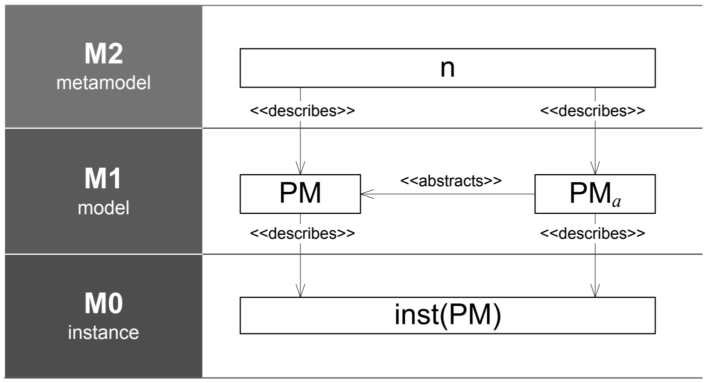


Fig. 3.1. The business process model abstraction concepts allocated to MOF levels.

a finite non-empty set of process models $\mathcal{PM}$ and an infinite non-empty set of process instances $\mathcal{I}$. A business process instance represents a concrete case in the operational business of a company consisting of activity instances. A mapping $inst : \mathcal{PM} \rightarrow \mathcal{P}(\mathcal{I})$ sets up a correspondence between a process model and the set of instances it describes. As we argued earlier, one business process can be described by multiple models. This thesis focuses on such models of one business process that differ in levels of abstraction. For instance, for a pair of models PM and PM_a it holds that *inst(PM)* = *inst(PM_a)*. We formalize the relations between models of various abstraction levels as follows. For a process model $\mathsf{PM} \in \mathcal{PM}$ there is a set of abstract process models, where each model describes the set of instances $inst(\mathsf{PM})$, but with less detail $abstr : \mathcal{PM} \rightarrow \mathcal{P}(\mathcal{PM})$. If the user has model PM, any abstract model PM_a $\in abstr(\mathsf{PM})$ provides no new information about $inst(\mathsf{PM})$.

In essence, business process model abstraction is an engineering task. To give the reader a better insight into the relations between the described business process model abstraction artifacts, we refer to the Meta Object Facility (MOF)—a standard for model-driven engineering which organizes (meta–) modeling artifacts into 4 levels [112]. We allocate the artifacts on different levels of MOF and show their relations, see Fig. 3.1. In this way we reuse the established vocabulary and the formalism of MOF. A set of process instances $inst(\mathsf{PM})$ related to process model PM is allocated to level $M0$. The business process model PM is put on level *M1*, as it describes/models a set of instances $inst(\mathsf{PM})$. Process model PM conforms to the modeling notation in which it is described—metamodel n. The process model $\mathsf{PM}_a \in abstr(\mathsf{PM})$ is an abstraction of PM and also belongs to level *M1*. Model PM_a describes the set of instances $inst(\mathsf{PM})$. Notice that we require models PM and PM_a to conform to one metamodel. For instance, if the detailed process model is created using BPMN [113], an abstract process model conforms to BPMN as well. However, in the general case, models PM and PM_a may adhere to different notations.

3.2 Business Process Model Abstraction Framework

Until now we did not address in detail what the goal of abstraction is, when the abstraction is applied, and how abstraction is exactly performed. These issues have been partially studied in [120]. The current section proposes a framework organizing these aspects systematically and enabling their formal discussion. Rather than creating the framework from scratch, we reuse the knowledge of cartographic generalization, a discipline existing for centuries. *Cartographic generalization* is the process of selecting and representing information of a map in a way that adapts to the scale of the display medium. Hence, cartographic generalization copes with a problem that resembles that of business process model abstraction.

There exist several cartographic generalization models, e.g., [31, 98, 109]. We adopt the overall structure of the first comprehensive generalization model focused on digital generalization as proposed by McMaster and Shea in [98]. McMaster and Shea claim that cartographic generalization consists of three components: a consideration of objectives of *why* to generalize; a cartometric evaluation of the conditions that indicate *when* to generalize; a selection of spatial and attribute transformations providing techniques on *how* to generalize. We will consider these components in the context of business process model abstraction, which will help us to arrive at a precise understanding of what business process model abstraction entails.

3.2.1 Business Process Model Abstraction: Why

The *why* aspect of abstraction considers the reasons for abstracting a process model, i.e., the goal of a process model abstraction. The abstraction goal is driven by the purpose of an abstract process model and its intended audience. On the one hand, stakeholders may vary from technical specialists, interested in a particular technical perspective of a process, to managers, who are seeking for a high-level business process overview. On the other hand, even one user alone may demand a whole spectrum of abstraction scenarios. For instance, a manager may both be interested in activities which have a high execution cost *and* in the paths in the model that are executed most often. The purposes and the stakeholders of these scenarios are different, and, so are the goals.

Depending on an abstraction goal, different objects attract the user's attention. Consider an example in Fig. 3.2. Models in Fig. 3.2 describe a business process, where a forecast request is processed, see Table 1.1 in Section 1.1 for process details. Model PM_1 is the most comprehensive process description among the three. Notice that each activity is annotated with its average execution cost. The models in Fig. 3.2 illustrate two abstraction examples.

Abstraction Example 1 One abstraction scenario results from a user demand in a high-level process outline, i.e., a model describing coarse-grained activities of the process and the ordering constraints between them. Model

PM_2 is an example of such a process overview discoverable from model PM_1. In comparison to activities of model PM_1, activities of PM_2 are more abstract and each comprises a set of activities of the initial model. For instance, activity *Perform quick analysis* corresponds to the set {*Prepare data for quick analysis, Perform quick data analysis*}. Thereby, in this scenario the user focuses on the granularity change of activities.

Abstraction Example 2 In the other abstraction scenario the user wants to observe "expensive" distributed process runs by means of a model. A distributed run is a behavioral model of a distributed system that describes one complete system evolution [133]. In this scenario an abstraction mechanism has to analyze all the distributed runs defined by a process model and select those that represent expensive ones. Model PM_3 presents the result of abstraction addressing such a user demand: among two alternative runs the most expensive is preserved.

Every business process model abstraction operates with a set of objects of one type. We refer to these objects as *abstraction objects*. By applying abstraction, a decision is made for each of those objects whether it is significant or insignificant *with respect to a specific goal*. While significant abstraction objects are preserved, insignificant ones are abstracted from. For the two aforementioned abstraction examples, we identify activities (Abstraction Example 1) and model parts capturing distributed process runs (Abstraction Example 2) as abstraction objects. Formally, each abstraction object is a part of the process model, a subset of model elements.

Definition 3.1 (Abstraction Object).
Let $\mathsf{PM} = (A, G, F, t, s, e)$ be a process model. An *abstraction object* is a set $\omega \subseteq (A \cup G \cup F)$ that describes one fact about a business process. This fact is considered relevant during one act of abstraction.

Definition 3.1 formalizes the abstraction object as a subset of model elements. However, not every subset of model elements is an abstraction object: Definition 3.1 requires the set of model elements to describe a fact about a process that is either abstracted or preserved by abstraction operation. In this way an abstraction object has a pragmatic aspect, i.e., captures information about the abstraction goal. Thereafter, the set of abstraction objects may vary from one abstraction operation to another, even when the same initial model is considered. We reference the finite non-empty set of abstraction objects in model PM during one abstraction operation as Ω.

Returning to the Abstraction Example 1, demanding a process model with more coarse-grained activities, we identify activities as abstraction objects. Hence, the set of abstraction objects contains 13 activities of process model PM_1. If we consider the Abstraction Example 2, where the user is interested in expensive process runs, we discover two abstraction objects in model PM_1: the run with the lower branch and the run with the upper branch. Model PM_3 describes only one process run and, hence, has one abstraction object.

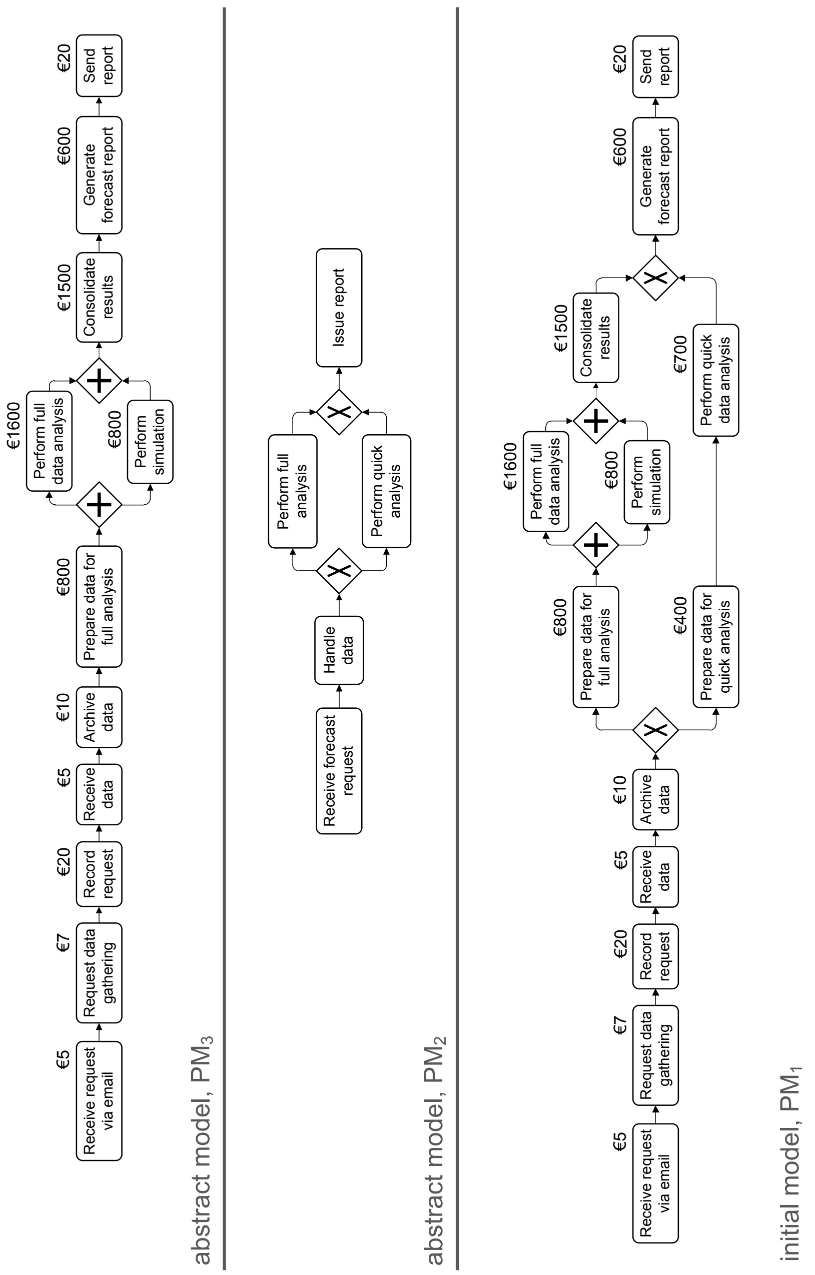

abstract model, PM3
€5 Receive request via email
€7 Request data gathering
€20 Record request
€5 Receive data
€10 Archive data
€800 Prepare data for full analysis
€1600 Perform full data analysis
€800 Perform simulation
€1500 Consolidate results
€600 Generate forecast report
€20 Send report
abstract model, PM2
Receive forecast request
Handle data
Perform full analysis
Perform quick analysis
Issue report
initial model, PM1
€5 Receive request via email
€7 Request data gathering
€20 Record request
€5 Receive data
€10 Archive data
€800 Prepare data for full analysis
€400 Prepare data for quick analysis
€1600 Perform full data analysis
€800 Perform simulation
€1500 Consolidate results
€700 Perform quick data analysis
€600 Generate forecast report
€20 Send report

Fig. 3.2. Two examples of business process model abstraction: one process model PM_1, due to two different transformations, results in models PM_2 and PM_3. Model PM_2 presents high-level activities, while model PM_3 describes the most expensive distributed process run.

This abstraction scenario evidences another phenomenon. While we distinguish two subsets of model elements that capture distributed runs, these two subsets share common model elements. Disregard of the common elements, we clearly distinguish the two abstraction objects. This phenomenon originates from the definition of an abstraction object as the subset of model elements enriched with a pragmatic aspect.

An abstraction goal defines an *abstraction criterion*—a property of an abstraction object that enables object comparison and allows the identification of objects relevant for the task at hand. For instance, in the Abstraction Example 2 the abstraction criterion is the process run execution cost. At the same time, we claim that the Abstraction Example 1 leverages user input as the abstraction criterion: the user manually selects significant activities. In particular, only activity *Generate forecast report* is considered significant and appears in model PM_2 as is.

3.2.2 Business Process Model Abstraction: When

The next component of business process model abstraction deals with the conditions under which abstraction objects are affected. An abstraction criterion allows for a comparison of abstraction objects. Subsequently, an abstraction criterion classifies abstraction objects of a process model into *significant* and *insignificant*. We formalize this classification with the function.

Definition 3.2 (Abstraction Object Significance).
Let $\mathsf{PM} = (A, G, F, t, s, e)$ be a process model and Ω—the set of abstraction objects of this model. Mapping $sign : \Omega \to \{true, false\}$ is an *abstraction object significance* function such that for every $\omega \in \Omega$:

$$sign(\omega) = \begin{cases} true, & \text{if } \omega \text{ is significant,} \\ false, & \text{else.} \end{cases}$$

For the Abstraction Example 1 we assume the abstraction object significance function to be user-defined: during abstraction the user manually specifies which activities are insignificant. The Abstraction Example 2 in Fig. 3.2 vividly illustrates the idea of abstraction significance function: while model PM_1 captures two possible distributed process runs, PM_3 describes only one process run with the highest execution cost. In this case the run of a process with the lower execution cost of 2410 € is considered insignificant. Hence, function *sign* evaluates to *true* in the former case and to *false* in the latter case. The significant abstraction object appears in model PM_3, while the insignificant object is abstracted.

If an abstraction criterion displays at least an ordinal scale, the classification into significant and insignificant elements can be realized by an *abstraction threshold value*. The threshold value partitions the set of model elements into two classes: elements with a criterion value greater or equal to the threshold, and the rest. One of these classes is considered to be significant, while the other—insignificant (the choice depends on the concrete abstraction goal). [120] proposes an *abstraction slider*, which is an implementation of the function *sign*.

3.2.3 Business Process Model Abstraction: How

The how component of business process model abstraction covers the method that enables the transformation of a process model into a more abstract process representation. While the abstraction relates process models, we makes use of an auxiliary binary relation R_α. The R_α relation sets up correspondences between abstraction objects of PM and PM_a. In this way the relation characterizes business process model abstraction on the level of process model elements.

Definition 3.3 (Abstraction Object Correspondence).
Let PM be a process model with the set of abstraction objects Ω. Let also $\mathsf{PM}_a \in abstr(\mathsf{PM})$ be a more abstract model of the same business process where the set of abstraction objects is Ω_a. An *abstraction object correspondence* is a surjective binary relation $R_\alpha \subseteq \Omega \times \Omega_a$.

Since a correspondence relation is surjective, every abstraction object of the process model PM_a corresponds to at least one abstraction object in the initial process model PM. Returning to our running examples we observe the following in the Abstraction Example 1:

– (*Generate forecast report*, *Generate forecast report*) $\in R_\alpha$
– (*Prepare data for quick analysis*, *Perform quick analysis*) $\in R_\alpha$
– (*Perform quick data analysis*, *Perform quick analysis*) $\in R_\alpha$

In the Abstraction Example 2 the distributed process run defined by the upper branch in model PM relates to the only remaining distributed process run. The abstraction object correspondence relation allows to design the business process model abstraction operation. Definition 3.4 formalizes the operation.

Definition 3.4 (Business Process Model Abstraction).
Business process model abstraction is a function $\alpha : \mathcal{PM} \rightarrow \mathcal{PM}$ that transforms a process model PM with the set of abstraction objects Ω into model PM_a with the set of abstraction objects Ω_a to conceal abstraction objects $\Omega' \subseteq \Omega$, where $\forall \omega \in \Omega' : sign(\omega) = false$ such that:

– $|\Omega| > |\Omega_a|$
– $\forall \omega_a \in \Omega_a : sign(\omega_a) = true$
– there is an abstraction object correspondence $R_\alpha \subseteq \Omega \times \Omega_a$ such that $\forall \omega \in \Omega'$:
  1. either $\nexists \omega_a \in \Omega_a$ that $\omega R_\alpha \omega_a$
  2. or $\exists \omega_a \in \Omega_a : \omega R_\alpha \omega_a \Rightarrow \exists \omega' \in \Omega : \omega \neq \omega' \wedge \omega' R_\alpha \omega_a$.

Definition 3.4 reflects several of our design decisions. First, it requires the business process model abstraction to deliver a process model with less abstraction objects, $|\Omega| > |\Omega_a|$. Second, all the insignificant abstraction objects are abstracted, see the second condition. Finally, we *design* two options to conceal an insignificant abstraction object ω. Either abstraction *eliminates* an insignificant abstraction object (option 1 in the third condition of Definition 3.4), or *aggregates* it with another abstraction object (option 2 in the third condition). This design decision implies that the two operations are sufficient to conceal any abstraction object. In other words we suggest to realize business process model abstraction by means of elimination and aggregation. Further we elaborate on the two operations in detail.

The first option is to deliver an abstract process model with no abstraction object corresponding to ω. Consider the pair of models PM_1 and PM_3 in Fig. 3.2 as an example, where PM_3 is the abstraction of PM_1. Model PM_1 describes two process runs, low cost (insignificant abstraction object) and expensive (significant abstraction object). Model PM_3 describes only one run—the expensive run. While the expensive run in PM_1 corresponds to the expensive run in PM_3, the low cost run has no correspondence.

The second option is to aggregate ω with other abstraction objects of PM_1 and absorb them into one abstraction object of PM_2. As an example we consider Abstraction Example 1 illustrated by models PM_1 and PM_2 in Fig. 3.2. In this abstraction scenario the abstraction object is an activity and several activities of PM_1 are aggregated into single activity of PM_2. Hence, each insignificant abstraction object in PM_1 corresponds to an abstraction object in PM_2 through relation R_α, while other abstraction objects in PM_1 correspond to the same abstraction object in PM_2.

The two options to conceal abstraction objects relate to two different types of abstractions: *elimination* (π) and *aggregation* (σ). Elimination produces a model that contains no information about the omitted abstraction objects Ω', while the other abstraction objects are preserved.

Definition 3.5 (Elimination).
A business process model abstraction $\pi : \mathcal{PM} \to \mathcal{PM}$ transforming a process model PM with the set of abstraction objects Ω into model PM_a with the set of abstraction objects Ω_a to conceal abstraction objects $\Omega' \subset \Omega$, where $\forall \omega \in \Omega' : sign(\omega) = false$, is an *elimination*, iff:

- $|\Omega| = |\Omega_a| + |\Omega'|$
- $\forall \omega \in \Omega'$ there is no $\omega_a \in \Omega_a$ such that $\omega R_\alpha \omega_a$
- the restriction of the abstraction object correspondence relation R_α to $(\Omega \backslash \Omega') \times \Omega_a$ is a bijection.

Elimination results in a process model with no insignificant abstraction objects (in the context of the abstraction goal). Significant abstraction objects that are not handled by elimination are preserved as is. That is why the restriction

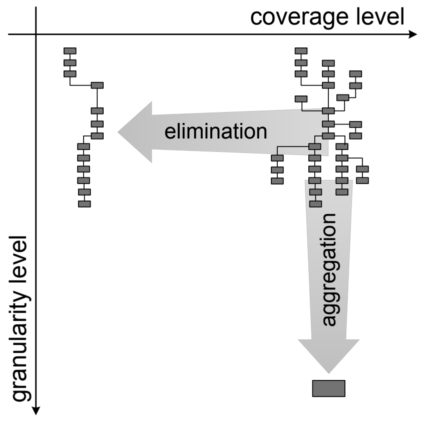


Fig. 3.3. Comparison of aggregation and elimination: elimination changes the process coverage level, while aggregation impacts granularity level of model elements.

of the auxiliary relation R_α is a bijection. Once again, as an outcome of elimination, the abstract process model provides no information about insignificant process details. In contrast, aggregation—the other type of business process model abstraction—partially preserves information about such details.

Definition 3.6 (Aggregation).
A business process model abstraction $\sigma : \mathcal{PM} \rightarrow \mathcal{PM}$ transforming a process model PM with the set of abstraction objects Ω into model PM_a with the set of abstraction objects Ω_a to conceal abstraction objects $\Omega' \subset \Omega$, where $\forall \omega \in \Omega' : sign(\omega) = false$, is an *aggregation*, iff:

– the abstraction object correspondence relation R_α is left-total
– $\forall \omega \in \Omega'$ there exist $\omega' \in \Omega$ and $\omega_a \in \Omega_a$ such that $(\omega, \omega_a), (\omega', \omega_a) \in R_\alpha$.

Aggregation produces an abstract model, where each insignificant abstraction object $\omega \in \Omega'$, together with several other abstraction objects, is represented with a newly introduced abstraction object ω_a. Object ω_a inherits the properties of objects it aggregates. For instance, if two sequential activities are aggregated into one activity, properties of the new activity comprise properties of the aggregated activities: the execution cost of an aggregating activity may be defined as the sum of execution costs of its aggregated activities.

Fig. 3.3 compares the effects of elimination and aggregation operations. Aggregation decreases the granularity of the process model, i.e., it makes a process model more coarse-grained. In the ultimate case, the whole business process is described by one high-level activity. Elimination omits model elements, but does not change their granularity level. Hence, elimination and aggregation enable navigation along two orthogonal axes: the granularity level of model elements and the coverage level of a business process by a model.

3.3 Properties of Business Process Model Abstraction

This section elaborates on properties of business process model abstraction as a model transformation. We begin discussing a prominent class of abstraction operations—hierarchical abstractions in Section 3.3.1. Sections 3.3.2 and 3.3.3 identify two other abstraction classes important in practice: order-preserving abstraction and abstraction preserving non-functional properties of model elements.

3.3.1 Hierarchical Abstraction

The study of related work, e.g., see [30, 94, 122, 123], reveals that one special class of business process model abstraction, *hierarchical abstractions*, prevails the rest. Hierarchical abstraction allows only such aggregation that each abstraction object of the initial model PM corresponds to exactly one abstraction object of PM_a. Due to this, we argue that a hierarchical abstraction can be seen as a composition of basic abstraction operations. We start formalizing the hierarchical abstraction introducing the notion of a basic abstraction operation. A basic abstraction operation allows to abstract from a single insignificant abstraction object.

Definition 3.7 (Basic Abstraction Operation).
A business process model abstraction $\alpha_\omega : \mathcal{PM} \to \mathcal{PM}$ transforming process model PM into model PM_a is a *basic abstraction operation*, if it abstracts from an insignificant abstraction object $\omega \in \Omega$, $sign(\omega) = false$, so that:

- α_ω is associated with an auxiliary surjective binary relation $R_{\alpha_\omega} \subseteq \Omega \times \Omega_a$
- the restriction of R_{α_ω} to $(\Omega \backslash \{\omega\}) \times \Omega_a$ is functional.

At this point we emphasize the difference between business process model abstraction as captured by Definition 3.4 and a basic abstraction operation. While the former conceals a set of abstraction objects, the latter handles one insignificant object at a time. Following, the hierarchical elimination is defined.

Definition 3.8 (Hierarchical Elimination).
A basic abstraction operation $\pi_\omega : \mathcal{PM} \to \mathcal{PM}$ is a *hierarchical elimination* handling insignificant abstraction object $\omega \in \Omega$, $sign(\omega) = false$, iff $|\Omega| = |\Omega_a| + 1$ and the restriction of its auxiliary relation R_{α_ω} to $(\Omega \backslash \{\omega\}) \times \Omega_a$ is a bijection.

The hierarchical aggregation is defined accordingly.

Definition 3.9 (Hierarchical Aggregation).
A basic abstraction operation $\sigma_\omega : \mathcal{PM} \to \mathcal{PM}$ is a *hierarchical aggregation* handling abstraction object $\omega \in \Omega$, iff an auxiliary relation R_{α_ω} is left-total and $\exists \omega' \in \Omega, \exists \omega_a \in \Omega_a : (\omega, \omega_a), (\omega', \omega_a) \in R_{\alpha_\omega}$.

Business process model abstraction is then realized as a composition of basic abstraction operations. Basic abstraction operations are applied until every insignificant abstraction object is handled.

Definition 3.10 (Hierarchical Business Process Model Abstraction).

Business process model abstraction is an operation $\alpha : \mathcal{PM} \to \mathcal{PM}$ transforming process model PM into model PM_a such that $\alpha = \alpha_{\omega_l} \circ \alpha_{\omega_{l-1}} \circ \cdots \circ \alpha_{\omega_1}$ is the function composition, where:

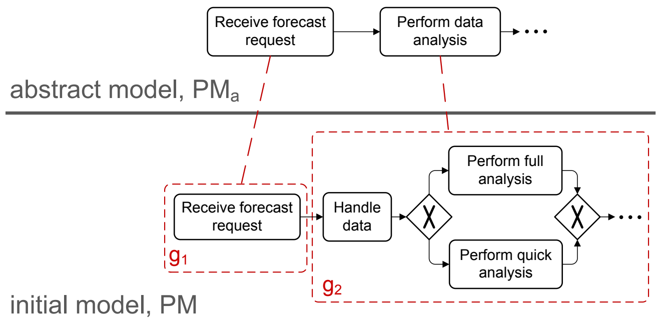
- $\forall \omega \in \Omega_a : sign(\omega) = true \wedge (\nexists k < l, \forall \omega \in \Omega_k : sign(\omega) = true)$,
- α_{ω_1} is a basic abstraction operation $\alpha_{\omega_1}(\mathsf{PM}) = \mathsf{PM}_2, \omega_1 \in \Omega \wedge sign(\omega_1) = false$,
- for $k = 2, \ldots, (l-1)$, α_{ω_k} *is a basic abstraction operation* $\alpha_{\omega_k}(PM_k) = PM_{k+1}$, $\omega_k \in \Omega_k \wedge sign(\omega_k) = false$,
- α_{ω_l} is a basic abstraction operation $\alpha_{\omega_l}(\mathsf{PM}_l) = \mathsf{PM}_a, \omega_l \in \Omega_l \wedge sign(\omega_l) = false$.

Notice that Definition 3.10 implicitly deals with the abstraction goal by referencing the abstraction objects and a significance function.

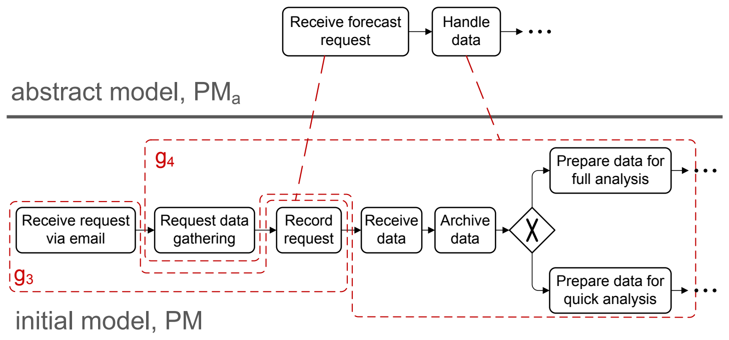
3.3.2 Order-Preserving Abstraction

As an intrinsic property of abstraction is information loss, an abstract model contains fewer ordering constraints than its detailed counterpart. Depending on the exact abstraction use case and the underlying abstraction goal, the tolerance level for the loss of ordering constraints may differ. Several works focusing on activity abstraction introduce the notion of *order-preserving* model transformation, e.g., see [30, 55, 94, 122]. The idea of order preservation is illustrated by Fig. 3.4. The example shown in Fig. 3.4(a) captures an order-preserving abstraction. Activities in the process model PM_a abstract the activity groups g_1 and g_2 in the model PM. The ordering constraints between the two coarse-grained activities coincide with the ordering constraints between the two activity groups. In the abstract model activity *Receive forecast request* is executed before *Perform data analysis*, while in the initial model activity group g_1 precedes g_2. In contrast, the abstraction in Fig. 3.4(b) is not order-preserving: activities of groups g_3 and g_4 interleave. This contradicts the sequential execution of activities *Receive forecast request* and *Handle data*. To formalize the notion of order-preserving business process model abstraction we assume that each high-level activity in $\mathsf{PM}_a = (A_a, G_a, F_a, t_a, s_a, e_a)$ is the result of aggregation of several activities in $\mathsf{PM} = (A, G, F, t, s, e)$. Then, the construction of coarse-grained activities is formalized by the auxiliary function *aggregate*.

Definition 3.11 (Function Aggregate).
Let $\mathsf{PM} = (A, G, F, t, s, e)$ be a process model and $\mathsf{PM}_a = (A_a, G_a, F_a, t_a, s_a, e_a)$— its abstract counterpart. Function $aggregate : \mathcal{A}_a \to (\mathcal{P}(A) \backslash \emptyset)$ specifies a correspondence between one activity in PM_a and the set of activities in PM.



(a) Order-preserving abstraction



(b) Not order-preserving abstraction

Fig. 3.4. Two abstractions: order-preserving and not order-preserving.

In this context, Definition 3.12 formalizes the concept of order preservation for business process model abstraction, with abstraction objects being activities. It makes use of behavioral profile relations, see Definition 2.23, to compare the behavior of initial and abstract process models.

Definition 3.12 (Order-Preserving Business Process Model Abstraction).
Let $\mathsf{PM} = (A, G, F, t, s, e)$ be a process model and business process model abstraction α maps PM to $\mathsf{PM}_a = (A_a, G_a, F_a, t_a, s_a, e_a)$, so that activities of PM are abstraction objects. Let also function *aggregate* establish a correspondence between activities of PM and PM_a. Operation α is *order-preserving business process model abstraction*, iff $\forall x, y \in A_a, x \neq y$ holds that $\forall a, b \in A$ such that $a \in aggregate(x)$ and $b \in aggregate(y)$:

– $a \rightsquigarrow_{\mathsf{PM}} b \Rightarrow x \rightsquigarrow_{\mathsf{PM}_a} y$
– $a \rightsquigarrow^{-1}_{\mathsf{PM}} b \Rightarrow x \rightsquigarrow^{-1}_{\mathsf{PM}_a} y$
– $a +_{\mathsf{PM}} b \Rightarrow x +_{\mathsf{PM}_a} y$

– $a||_{\mathsf{PM}} b \Rightarrow x||_{\mathsf{PM}_a} y$.

Definition 3.12 constraints the order of activities in models PM and PM_a related by mapping *aggregate*. Fig. 3.4 illustrates the formal definition. The abstraction presented in Fig. 3.4(a) has a mapping *aggregate* such that:

– *Receive forecast request* $\in$ *aggregate(Receive forecast request)*
– *Handle data* $\in$ *aggregate(Perform data analysis)*
– *Perform full analysis* $\in$ *aggregate(Perform data analysis)*
– *Perform quick analysis* $\in$ *aggregate(Perform data analysis).*

We notice that *Receive forecast request* $\rightsquigarrow_{\mathsf{PM}_a}$ *Perform data analysis*. Further, each pair of activities in model PM corresponding to *Receive forecast request* and *Perform data analysis* is also in strict order relation, for instance, *Receive forecast request* $\rightsquigarrow_{\mathsf{PM}}$ *Handle data*. Hence, the presented abstraction is order-preserving. Upon the other hand, abstraction in Fig. 3.4(b) is not order-preserving. This follows from the observation that *Receive forecast request* $\rightsquigarrow_{\mathsf{PM}_a}$ *Handle data*, while *Request data gathering* $\rightsquigarrow_{\mathsf{PM}}$ *Record request*.

3.3.3 Abstraction Preserving Process Non-Functional Properties

To complete our reflection on business process model abstraction, we briefly discuss preservation of process non-functional properties [77]. *Business process non-functional properties* may be more or less important to be preserved when applying abstraction [32, 53, 121]. Companies often use process models to analyze operational business processes, for example to analyze their cost or bottlenecks. Model elements of the models supporting such an analysis are annotated with additional information. If the user leverages abstract models in the analysis, a process model abstraction has to ensure that the analysis of model PM_a delivers the same results as the analysis of model PM. If the abstraction operation fulfills this requirement, we call it an abstraction that preserves process non-functional properties. Consider the setting, where an HR department leverages process models to evaluate the number of resources required for process execution. To enable this task, the process model designers enrich the model with information about activity execution time, hand-off times, activity execution probabilities, and actors executing the activities. Once a detailed process model is annotated with this information, while a more abstract model is in demand, the users require an abstraction that preserves process non-functional properties. In this context, the abstract process model should provide the same evaluation of the resources consumed by the process as the initial process model. Among the existing abstraction methods several preserve process non-functional properties, e.g., see [32, 121], while the majority leaves this aspect out of discussion.
At this point, we have arrived at a formal and complete view of business process model abstraction. Furthermore, we have stressed the context-specific importance of preserving the control-flow and non-functional properties in a

process model when applying abstraction operation. We will be using the various notions that have been introduced in this section in the structuring of a catalog of abstraction uses cases in the next section.

3.4 Catalog of Abstraction Use Cases

This section presents a catalog of business process model abstraction use cases which are identified with the help of BPM experts. First, we explain the method that has been applied to derive and validate the use cases. Next, we present the initial version of the catalog used as the input for the validation stage. Then, we discuss the feedback that we received during the validation stage and summarize the modified use case catalog.

3.4.1 Catalog Design

In order to understand the user demand for process model abstraction techniques we referred to the expertise of our industry partners. As the problem of process model abstraction is relatively new, we followed an exploratory approach and conducted a series of semi-structured interviews with BPM experts. The study was separated into the two phases of (1) generation and (2) validation, which overall involved three categories of stakeholders, i.e., end users, consultants, and software developers.

In the *first phase* we considered use cases that emerged out of a joint project with a large German health insurance company, AOK. The goal of the project was to develop abstraction techniques enabling a fast comprehension of large business process specifications containing, for example, more than 300 nodes. The abstraction use cases were retrieved and elaborated in interviews with AOK employees:

- a business process leader
- a coordinator of IT infrastructure for process management
- a business process knowledge manager
- three business process modelers.

All these employees are interested in business process model abstraction as the end users of a set of over 4,000 process models. The use cases derived from the interviews were complemented by use cases from the literature. The literature study includes papers from the reputable conferences on business process management and information systems, e.g., Business Process Management and International Conference on Service Oriented Computing, and journals, e.g., IEEE Transactions of Software Engineering and Information Systems, within the timespan of the last decade. The outcome of the first phase were 14 use cases organized into four groups.

In the *second phase* the use cases were validated by involving two further companies: Infosys, an Indian information technology services company with a specific focus on BPM, and Pallas Athena, a Dutch software vendor developing BPM systems. From these parties, ten and eight professionals participated in this study respectively. All of the involved Infosys employees fulfill a role as business process consultant; their experience with BPM had an average value of 6.5 years. The spectrum of job descriptions of the interviewees at Pallas Athena varied from that of software engineer to the chief executive officer. The BPM experience of the participants within this group had an average value of 11.5 years. The primary goal in this phase was to reflect on the relevance of the initial set of use cases. Secondly, we encouraged the interviewees to generate new use cases. The output of the second phase was a validated use case catalog. In comparison with the initial set of 14 cases, one use case was dropped and two new use cases were added leading to a total of 15 use cases in the end.

3.4.2 Initial Use Cases

The set of initial use cases that were derived from the first phase of our exploratory approach will be discussed in this section by distinguishing four groups, each of which contains use cases that have similar properties. In this discussion, we will use the notions as introduced in Section 3.2 to characterize the various groups of use cases. Specifically, the description of each group contains the central abstraction object, the used abstraction criterion, the basic abstraction operation being involved, and the importance of preserving a model's control-flow and non-functional properties.

Group 1: Preserving Relevant Activities

The user analyzes a business process captured by a process model. The model specifies numerous activities. However, the user wants to focus on activities that are significant for the task at hand. The distinction between what the significant and insignificant activities are is based on the threshold value of a non-functional property of these activities. All the activities with a value for this property that is lower than the threshold are insignificant and these are eliminated. The use cases in this group share two things in common: they have the *activity* as abstraction object and *elimination* as a basic abstraction operation. The ordering constraints between the significant activities are *preserved*, while the use of elimination leads to a *change* of the non-functional properties of the overall process. We distinguish four abstraction use cases that belong to this group.

Use Case 1: Preserve Pricey Activities
The user optimizes a business process and is interested in the activities with a high execution cost.

Use Case 2: Preserve Frequent Activities
The user improves a business process and focuses on frequently executed activities.

Use Case 3: Preserve Long Activities
The user is interested in process optimization and focuses on activities with a high duration.

Use Case 4: Show High Hand-off Times
The user optimizes a business process and focuses on activities with high hand-off times.

Group 2: Preserving Relevant Process Runs

The user analyzes a business process described by a precise model specifying the life cycle for a wide variety of process instances. The user does not want to know about each distributed process run, but needs to focus on a specific subset of runs. We call such runs significant. The significant process runs are visualized in the process model with model parts capturing distributed process runs. A business process model abstraction eliminates the parts corresponding to insignificant process runs and preserves the parts describing significant ones. To summarize, the use cases in this group have *distributed process runs* as an abstraction object, have *elimination* as a basic abstraction operation, *preserve* the ordering constraints among the significant abstraction objects, and *do not* allow to preserve the non-functional properties of the overall process. We have encountered the following use cases.

Use Case 5: Preserve Pricey Runs
The user optimizes a process and considers costly process runs as significant. She specifies a cost threshold, distinguishing significant process runs from insignificant ones: distributed process runs with an execution cost that is higher than the threshold value are significant, the rest are not.

Use Case 6: Preserve Frequent Runs
The user performs process optimization and considers frequent distributed process runs as significant. By means of a run execution frequency threshold, the user distinguishes significant runs from insignificant ones. The runs with an execution frequency higher than the threshold are considered to be significant, while the rest are insignificant.

Use Case 7: Preserve Runs with Long Duration
The user optimizes the process and considers distributed process runs with long durations as significant. She specifies a path execution duration threshold value, distinguishing significant runs from insignificant ones: the distributed runs with execution times higher than the threshold are important, while runs with lower execution times are unimportant.

Use Case 8: Trace a Case
The user is interested in the question how special cases evolve in a business process. For instance, she wants to know how orders with a cost higher than 1000 euros unfold. Hence, the user specifies a case to be traced and obtains a model capturing only the significant process evolutions.

Group 3: Filtering of Model Elements

The process model in possession of the user is overspecified for the task at hand. Only a subset of model elements is relevant and have to be disclosed. In contrast to the use cases of Group 1, the significance of model elements is determined according to their qualitative properties. To simplify model comprehension, irrelevant model elements are eliminated. The relevant elements are preserved, as well as the ordering constraints between them. The use cases of this group exhibit common properties: abstraction objects are *model elements* and a basic abstraction operation is *elimination*. The ordering constraints between significant model elements are *preserved*, while non-functional properties of the overall process are *changed*.

Use Case 9: Adapt Process Model for an External Partner
The user adapts an existing business process model for the presentation to an external partner. The available model either captures confidential, internal process details, or details which are of no interest to the partner. The user manually marks model elements, which are relevant for inter-organizational collaboration and which are significant.

Use Case 10: Trace Data Dependencies
The user modifies a data object interface. Beforehand she needs to know which data dependencies exist in the business process. Hence, the significant model elements are those that access the data object of interest.

Use Case 11: Trace a Task
The user evaluates the effect of an activity in a process model. To achieve this, a transitive closure of model elements dependent on this activity has to evaluated. Model elements of this closure are significant, while other model elements are not.

Group 4: Obtaining a Process Quick View

The user needs a business process overview for fast process comprehension. The available model is a process specification formalizing every minor detail. A study of this model is time consuming and is not necessary for the ongoing work. The user needs a representation of this business process on a higher level, capturing more coarse-grained activities and overall information about the ordering constraints. For all of the use cases in this group, *activities* are abstraction objects. *Aggregation* is the basic abstraction operation. While Use Cases 12 and 13 aim to *preserve* the ordering constraints, Use Case 14 does *not* consider the ordering constraints. Similarly, as the non-functional properties of the process are *preserved* by Use Case 12 and Use Case 13, Use Case 14 does *not* aim to preserve them. The following use cases belong to this group.

Use Case 12: Get Process Quick View Respecting Ordering Constraints
The user needs a process specification, capturing coarse-grained activities, as well as the ordering constraints between them. She does not know in advance

which abstraction level is sufficient and wants to control this level gradually. The user prefers to preserve non-functional properties of the process.

Use Case 13: Get Process Quick View Respecting Roles Activities performed by a special role, e.g., *Manager*, are considered to be significant. The rest of activities are not. Insignificant activities are aggregated into coarse-grained ones, significant activities are preserved as is, and the ordering constraints are preserved where possible. Non-functional properties of the process, e.g., execution time or execution cost, should be preserved.

Use Case 14: Retrieve Coarse-grained Activities The user wants to grasp the coarse-grained activities that appear in the business process. She does not require an abstraction mechanism to deliver the ordering constraints between the high level activities: once these activities are available, she can manually order them.

3.4.3 Use Case Validation

During the validation phase of the catalog design, each participant received a booklet that described the initial set of use cases. The participants were asked to study these descriptions and the researchers were available for clarification. Each participating BPM expert expressed her demand for each of the presented use cases. To express her opinion, each participant had three options. If the participant found the use case important and the intended abstraction approach helpful, she could mark the use case with a *yes*. If the participant saw no value in the presented use case, she could answer *no*. If the participant had doubts about the relevance of the use case, she was able to respond with *undecided*. For the evaluation we encoded the responses: positive responses correspond to *1*, negative responses were encoded with *-1*, whilst neutral answers—with *0*. Participants had the opportunity to give comments and discuss the use cases with the researchers.

Relevance and Completeness

Table 3.1 presents the aggregated values of the response codes. As can be seen, the table differentiates between the 4 groups of stakeholders: novice and experienced consultants as well as novice and experienced (software) vendors.

Use case ID / Category	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Experienced consultant (5)	2	5	3	3	3	5	1	5	3	3	3	5	3	1
Novice consultant (5)	-1	1	1	2	1	3	3	2	0	0	2	3	3	-2
Experienced vendor (2)	-1	1	1	2	2	2	1	0	1	0	1	2	2	0
Novice vendor (6)	0	6	0	5	5	6	5	0	4	3	2	6	3	-1
Total	*0*	*13*	*5*	*12*	*11*	*16*	*10*	*7*	*8*	*6*	*8*	*16*	*11*	*-2*

Table 3.1. Support of business process model abstraction use cases by interviewees.

First, we observe that the number of novice consultants involved is equal to the number of experienced consultants (5), while for the vendors the situation is uneven - the novices outnumber the experts (6 vs. 2). Second, there is a tendency notable of experts expressing a higher appreciation of the use cases than novices. This is clearly visible for the consultants, but this relation also holds for the whole group: The average score that an expert provides for a use case equals 0.60, while this is 0.38 for the novices. Finally, the opinions of the two groups, consultants and vendors, are highly consistent, with the notable exception of "Use Case 8: Trace a Case". The latter use case is favorably perceived by consultants (total score of 7), while the vendor representatives take a neutral stance (total score of 0). Overall, the use cases "Use Case 6: Preserve Frequent Runs" and "Use Case 12: Get Process Quick View Respecting Ordering Constraints" find the most outspoken support. The former is associated with finding a so-called "happy path" in the process or its "sunny day scenario". The latter use case is interpreted by most participants as the type of abstraction that is most in demand.

Surprisingly, the participants seem to differentiate between use cases that exploit the *same* abstraction technique, but operate with *different* non-functional properties of model elements. This is most vividly illustrated by the contrast between the values for "Use Case 1: Preserve Pricey Activities" and "Use Case 2: Preserve Frequent Activities". Whilst the former use case is of not much interest for interviewees (score of 0), the latter is in high demand (score of 13). A less pronounced differentiation can be observed for "Use Case 5: Preserve Pricey Runs" and "Use Case 6: Preserve Frequent Runs". We conclude that *frequency* is perceived as a more natural abstraction criterion by users. Furthermore, these observations highlight the importance of an explicit choice for the abstraction criterion in question.

A study of Table 3.1 also reveals that use case 14 is a clear outlier. This use case is the only one that completely neglects control flow: it exclusively delivers a set of activities to the user. We deduce that for the abstraction stakeholders *ordering constrains* are of vital importance and belong to the essential model information to be preserved. Hence, we interpret "Use Case 14: Retrieve Coarse-grained Activities" as an example of a *false* abstraction use case and drop it from the final catalog.

During the evaluation of use cases that belong to Group 1 the participants noticed that the elimination of insignificant activities often leads to unacceptable information loss. Instead of eliminating insignificant activities, the interviewees saw benefits of aggregating them. We summarize these user requests in a new use case.

Use Case 15: Preserve Frequent Activities Summarizing Rare Activities The user analyzes a process captured in a detailed process model. She has to focus on activities relevant for the current analysis. The distinction between significant and insignificant activities bases on the threshold value of an activity frequency: the activities with a frequency value lower than the threshold are insignificant. Significant activities are preserved as-is, while insignificant activities are aggregated, when possible.

The introduction of this use case raises the issue whether a whole new family of use cases should be created that is based on the initial members of Group 1. However, despite the external similarity to the use cases of Group 1, such new use cases would heavily rely on the technique needed for “Use Case 13: Get Process Quick View Respecting Roles”. As such, we decided not to pursue this larger extension.

Interviewees also pointed to business process model abstraction scenarios where only model elements relevant for a certain perspective, e.g., a business perspective or a data flow perspective, are presented to the user. Notice that this abstraction depends on the existence of information that is relevant to make this distinction in the initial process model. Abstractions of this type belong to Group 4: Filter Model Elements. We formulate the user demand in the following use case:

Use Case 16: Get Particular Process Perspective The user analyzes a process model captured in a detailed process model. She wants to see a particular process perspective. Model elements which belong to the desired perspective are significant and preserved in the model as-is. Model elements which do not belong to this perspective are insignificant and are eliminated.

No needs for further use cases were found. In sum, this leads us to a final set of 15 use cases, which is one of the contributions of this paper.

3.4.4 Additional Insights

While the second phase in our validation approach mainly aimed at the relevance and completeness of the use case catalog, the discussions with the involved participants raised additional insights. First of all, other visualization techniques came forward as important *alternatives* to deal with some of the use cases. In particular, we can distinguish the following techniques that were brought forward:

Highlighting Instead of completely abstracting from model objects that do not need to be visualized, it is also possible to *highlight* the objects that deserve attention, for example by coloring these or changing their shape. The main advantage is that it provides the context of the highlighted objects. A good example where this could be useful is “Use Case 6: Preserve Frequent Runs”, where a “happy path” is highlighted within the process model.

Tagging Depending on the exact use case, it may be important to see *more* rather than less information in a process model, which is the objective of abstraction. Such additional information could be presented as tags, annotations, or even icons that are added to existing process model elements. For instance, in the context of "Use Case 13: Get Process Quick View Respecting Roles" it could also be useful to see relevant role information along with tasks in the model.

Animation While business process model abstraction is focused on the static representation of the process model content, for some use cases a more *dynamic* representation mode is desirable. Specifically, for the use cases in Group 2 (Preserving Relevant Process Runs) it is useful to see how a particular process evolution unfolds step-by-step.

Textual Reporting For the considered use cases, it is not always important to obtain the information that one seeks in the form of a process model. Instead, a textual or tabular enumeration can suffice. Recall that we dropped "Use Case 14: Retrieve Coarse-grained Activities" as a use case for process model abstraction, even though the participants can imagine the intended overview to be relevant in the form of a tabular visualization.

This overview is by no means meant as comprehensive, but it puts the importance of business process model abstraction into the right perspective. After all, it would be improper to consider abstraction as the *only* viable way to present relevant information in a process model. At the same time, we do argue that the value of business process model abstraction in comparison with other techniques can be explicitly found in use cases that involve very large process models. For all of the alternatives we listed, one can foresee a range of problems in such cases. For example, if highlighting is applied in an extremely large process model, it will become difficult to distinguish, let alone focus, on the emphasized objects.

A final insight relates to the specific feedback of one of the participants, who argued that he did not see value in business process model abstraction for any of the proposed use cases. He explained that in his environment a strictly hierarchical modeling approach is employed, such that each process is modeled on five different levels of granularity (using subprocesses). Therefore, according to this participant, the abstraction techniques add limited additional value with respect to navigating through these levels. Clearly, it is open to debate whether switching between subprocesses can provide exactly the same insights as the process model abstraction techniques do. Yet, it is important to realize that built-in features of process models can of course greatly contribute to an improvement of large process model understanding. This is also in line with the earlier work on the value of modularity [128].

3.5 Summary

This chapter framed the domain of business process model abstraction. On the one hand, we have conceptualized business process model abstraction:

identified the main concepts and their relations. On the other hand, we have presented the results of an empirical study illustrating the user demand in abstraction methods. While the former finding provides a theoretical perspective on business process model abstraction, the latter reveals its practical importance. In this way the two contributions complement each other and provide one coherent view on the process model abstraction.

A substantial finding of the user study is the high demand for abstraction use cases of "Group 4: Obtaining a Process Quick View". Motivated by this observation the current thesis develops methods that enable abstraction use cases within this group. Since the use cases in "Group 4: Obtaining a Process Quick View" treat activities as abstraction objects, the designed methods of abstraction also excel the role of activities. At the same time, Section 3.2.3 distinguished abstraction operations into elimination and aggregation. While elimination operation is well-studied [4, 30, 55, 94], this thesis focuses on aggregation as means of abstraction. Altogether, Chapters 4–6 elaborate on the activity aggregation and address the *how* aspect of business process model abstraction. Table 3.2 sketches the upcoming contributions: it relates the contributions of the upcoming chapters to the classes of hierarchical and order-preserving abstractions.

Thesis part \ **Abstraction class**	**Hierarchical**	**Order-preserving**
Section 4.1	•	•
Section 4.2	•	•
Section 5.1	∗	∗
Section 5.2	∗	•
Chapter 6	∗	∗

Table 3.2. Outlook of the contributions in the upcoming chapters: • denotes that the contribution belongs to the abstraction class specified in the column; ∗ indicates that the contribution belongs to the superclass of the class specified in the column.

4

Structural Methods of Business Process Model Abstraction

The commitment of each business process model abstraction operation to a particular goal explains the variety of abstraction use cases. Section 3.4 witnesses: among the 15 identified business process model abstraction use cases, the use case group "Group 4: Obtaining a Process Quick View" and, in particular, the use case "Use Case 12: Get Process Quick View Respecting Ordering Constraints", experience a high user demand. Therefore, we focus our study on this group of use cases. The investigated use cases imply that the granularity level of activities is increased. Thereby, we select activities as the abstraction object. Identifying aggregation and elimination as two primitive operations of business process model abstraction, we face the challenge of designing a concrete implementation of these operations.

We steer our research towards the design of aggregation and leave elimination out of scope. There are three reasons for this. First, we consider process models where activities have exactly one incoming and one outgoing edge. Thereby, elimination is trivial: an insignificant activity along with its adjacent edges can be substituted with one edge. Second, elimination has been extensively studied by the related work on abstraction, see, for instance, [30, 55, 45]. Finally, activity elimination is inappropriate against the use cases in focus. Elimination does not change the granularity level of model elements, while the studied use cases require that activities become more coarse-grained. To this end, aggregation increases granularity of model elements. At the same time, the principle of activity aggregation is a significant variation point in the design of the aggregation operation. This principle defines which activities belong together and constitute a more coarse-grained activity.

In this chapter we analyze how information on process model structure facilitates activity aggregation. Activity aggregation driven by structural information seeks for a process model fragment that 1) is the minimal fragment containing an insignificant activity and 2) can be securely replaced by one more coarse-grained activity. Securely here means that a model transformation must not introduce such behavioral anomalies as deadlocks or livelocks [2, 23, 165]. To guarantee the secure transformation, we seek for process

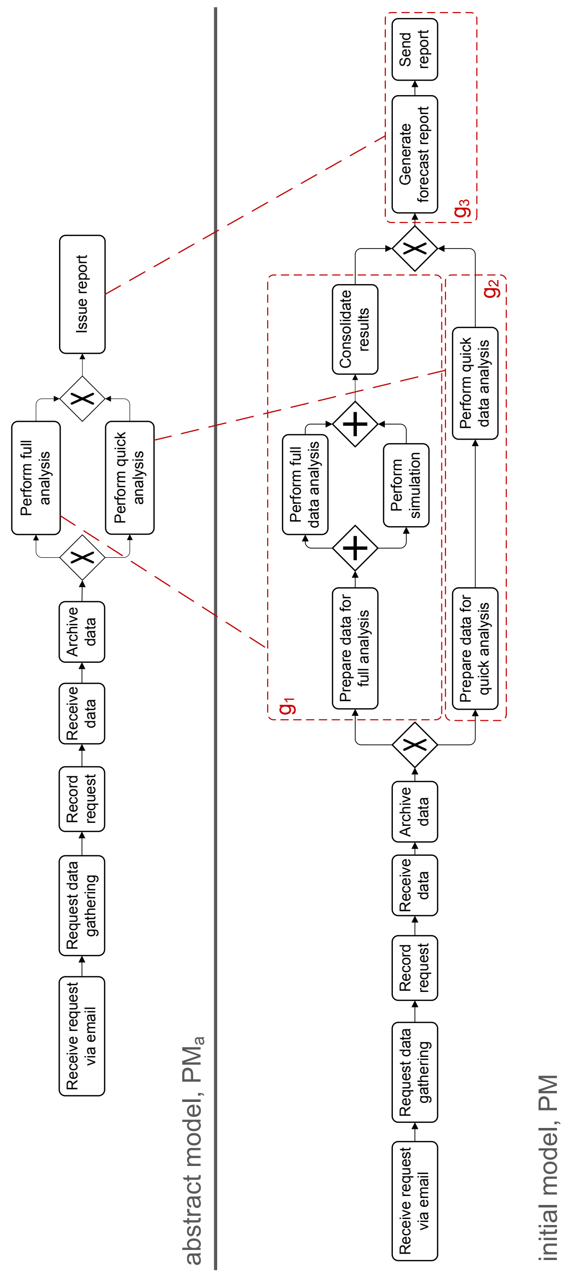


Fig. 4.1. Two models capturing "Forecast request handling" business process. Model PM captures the process with higher precision than model PM$_a$. The three activities of model PM$_a$ are refined by activity groups in PM.

model fragments that have exactly one incoming edge and one outgoing edge, see [21, 22, 101, 108, 155]. Once such a fragment is discovered, it is substituted by an aggregating activity. Thereby, structural methods of business process model abstraction assume that there exists a coarse-grained activity semantically equivalent to the replaced fragment. For an example consider the model PM capturing the business process "Forecast request handling" in Fig. 4.1. The activities of group g_1 constitute together a coarse-grained activity *Perform full analysis*. Furthermore, the activities of the group g_1 belong to a fragment with one entry edge and one exit edge—the fragment is marked by the dashed line. Hence, in the abstract process model PM_a we replace the whole fragment with the coarse-grained activity *Perform full analysis*. The activities of g_2 and g_3 form, respectively, activities *Perform quick analysis* and *Issue report*. Analyzing the initial model PM in this way and aggregating process model fragments, we arrive at the abstract model PM_a.

There are several ways to define process model fragments to be substituted. One approach is to capture the process model fragments explicitly by patterns, where a pattern prescribes fragment's structure. The majority of the existing business process model abstraction techniques build on top of this idea, e.g., see [30, 32, 55, 94]. Further, the research on process model analysis delivers a wide spectrum of reduction rules, e.g., see [21, 44, 50, 102, 107, 135, 167], that can be adopted for abstraction purposes. An alternative solution is to define process model fragments by their property. In particular, we are interested in process fragments that have single entry and single exit, as activities do. The decomposition-based approach is followed by [122, 123] and, partially, by [30]. These abstraction approaches make use of process model decomposition methods advocated in [73, 154, 155].

This chapter argues how business process model structure enables activity aggregation. In particular, we propose two novel business process model abstraction algorithms. Both algorithms adapt well established methods for process model structure analysis to the context of business process model abstraction. The first algorithm relies on structural patterns, while the second algorithm leverages process model decomposition. In addition, we systematically elaborate on the properties of the two abstraction methods and identify their pros and cons. According to the framework introduced in Section 3.2, this chapter excels the *how* aspect of abstraction, extending our body of knowledge about the abstraction *why* as well.

The remainder of this chapter is structured as follows. Section 4.1 elaborates on a pattern-based process model abstraction. Section 4.2 argues how a decomposition of process models into fragments enables abstraction. We reflect on the properties of the developed abstraction methods in Section 4.3. Finally, Section 4.4 summarizes the chapter.

4.1 Pattern-Based Methods

It is a widely used observation that process models exhibit recurrent structures [8, 14, 63, 88, 146, 147]. For instance, the model in Fig. 4.1 contains a sequence of activities *Prepare data for quick analysis* and *Perform quick data analysis* and another sequence of *Generate forecast report* and *Send report*. At the same time, model PM contains two blocks: the AND block and the XOR block. Consideration of such recurrent structures facilitates several formal model analysis methods, e.g., [59, 107, 108] argue how recurrent structures fasten the soundness checking. The first step towards operationalization of recurrent structures is describing their topology by patterns. Next, for each pattern a transformation method is specified. For example, a recurrent structure of a sequence with two nodes is captured as a pattern. The associated transformation may specify to substitute such a sequence with one node. Finally, the model is transformed by matching patterns against the model structure and subsequent corresponding transformation.

Structural patterns can be used to realize process model abstraction: patterns along with the associated transformations are natural candidates for the implementation of aggregation. We reference the combination of the structural pattern and its transformation specification as an *elementary abstraction*. This section argues about the use of elementary abstractions. Section 4.1.1 formalizes three elementary abstractions, while Section 4.1.2 presents an algorithm that coordinates their work.

4.1.1 Elementary Abstractions

We discuss three elementary abstractions: sequence, block, and loop. As it follows from the names, each elementary abstraction is associated with a particular structural pattern. Notice that the considered set of patterns is by no means "complete" and can be extended. For instance, [121] proposes a *dead end elementary abstraction*. However, disregard of the underlying structural pattern, each elementary abstraction conceals the details of the initial process model replacing model elements by more coarse-grained.

Sequence Elementary Abstraction

Business process models of high fidelity typically contain sequences of activities. *Sequence elementary abstraction* replaces a sequence of activities by one *aggregating* activity. An aggregating activity has a coarse granularity and brings a process model to a higher level of abstraction. The replaced sequence is formalized by Definition 4.1.

Definition 4.1 (Sequence Process Model Fragment).
Let $\mathsf{PM} = (A, G, F, t, s, e)$ be a process model containing a process model fragment $\mathsf{PMF} = (A_{\mathsf{PMF}}, G_{\mathsf{PMF}}, F_{\mathsf{PMF}}, t_{\mathsf{PMF}})$. The fragment PMF is a *sequence*

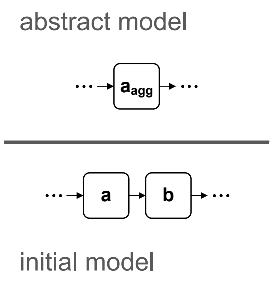


Fig. 4.2. Sequence elementary abstraction.

process model fragment defined by two sequential activities a and b if it is a process model fragment such that:

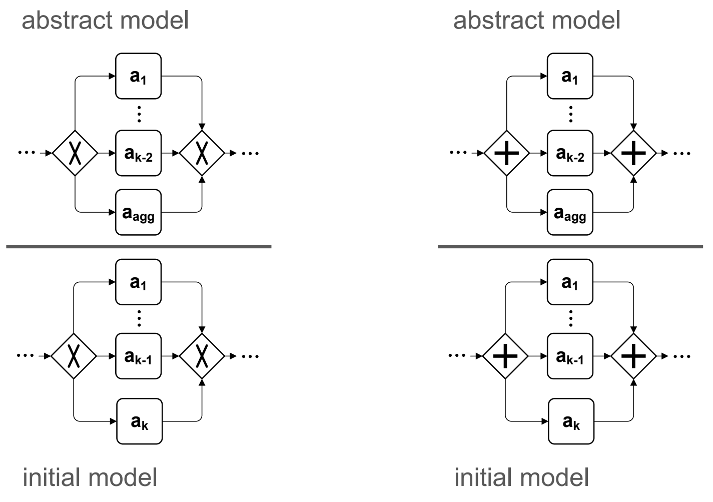
– $A_{\mathsf{PMF}} = \{a, b\}$
– $G_{\mathsf{PMF}} = \emptyset$
– $F_{\mathsf{PMF}} = \{(n, a), (a, b), (b, n')\}$, where $\{n\} = \bullet a$ and $\{n'\} = b\bullet$
– t_{PMF} is undefined, as G_{PMF} is empty.

Definition 4.1 limits a sequence process fragment to a sequence of two activities. Sequences of greater length can be abstracted through a recurrent application of sequence abstraction. Fig. 4.2 exemplifies sequence elementary abstraction. Activities a and b form a sequence. Sequence elementary abstraction substitutes a and b with an aggregating activity a_{agg}. Semantics of activity a_{agg} implies that first activity a is executed and then—activity b. Definition 4.2 introduces the sequence elementary abstraction.

Definition 4.2 (Sequence Elementary Abstraction).
Let $\mathsf{PM} = (A, G, F, t, s, e)$ be a process model. *Sequence elementary abstraction* is an aggregation operation $\sigma_s : M \to M$ that replaces a sequence process model fragment $\mathsf{PMF} = (\{a, b\}, \emptyset, \{(n, a), (a, b), (b, n')\}, t_{\mathsf{PMF}})$ with activity a_{agg} transforming process model PM into $\mathsf{PM}_a = (A_a, G_a, F_a, t_a, s_a, e_a)$ so that:

– $A_a = (A \backslash \{a, b\}) \cup \{a_{agg}\}$
– $G_a = G$
– $F_a = (F \backslash \{(n, a), (a, b), (b, n')\}) \cup \{(n, a_{agg}), (a_{agg}, n')\}$
– if $s = a$, then $s_a = a_{agg}$, i.e., a_{agg} is the start node of PM_a
– if $e = b$, then $e_a = a_{agg}$, i.e., a_{agg} is the end node of PM_a.

In addition to the structural transformation we can also specify how the non-functional properties of the aggregating activity are deduced from the properties of the aggregated activities. We consider average activity execution cost as an example of such a property. In case of sequence elementary abstraction the average execution cost of an aggregating activity a_{agg} depends on the average execution costs of functions a and b: $cost_{avg}(a_{agg}) = cost_{avg}(a) + cost_{avg}(b)$.



(a) Abstraction of a XOR block. (b) Abstraction of an AND block.

Fig. 4.3. Block elementary abstractions.

Block Elementary Abstraction

To model parallelism or to show that a decision is made in a process, a modeler encloses several branches of control flow between split and join gateways. Depending on the desired execution semantics, an appropriate gateway type is selected: AND or XOR. Replacing several branches enclosed by gateways with one branch containing a coarse-grained activity reduces the number of details. A recurrent application of this transformation abstracts the process fragment enclosed by gateways to one coarse-grained activity. *Block abstraction* enables this operation. We start specifying the transformed block process model fragment.

Definition 4.3 (Block Process Model Fragment).
Let $\mathsf{PM} = (A, G, F, t, s, e)$ be a process model containing a process model fragment $\mathsf{PMF} = (A_{\mathsf{PMF}}, G_{\mathsf{PMF}}, F_{\mathsf{PMF}}, t_{\mathsf{PMF}})$. The fragment PMF is a *block process model fragment* defined by activities $a_1, \ldots, a_k$, where $k \in \mathbb{N}, k > 1$ if it is a SESE process model fragment such that:

– $A_{\mathsf{PMF}} = \{a_1, \ldots, a_k\}$
– $G_{\mathsf{PMF}} = \{g_s, g_j\}$
– $\forall a \in A_{\mathsf{PMF}} : \bullet a = \{g_s\} \wedge a\bullet = \{g_j\}$
– $|g_s \bullet| = |\bullet g_j|$
– $|A_{\mathsf{PMF}}| \leq |g_s \bullet| \leq |A_{\mathsf{PMF}}| + 1$
– a function t_{PMF} is the restriction of t to set G_{PMF}
– $t_{\mathsf{PMF}}(g_s) = t_{\mathsf{PMF}}(g_j)$

– the fragment's entry edge is (n, g_s), where $\{n\} = \bullet g_s$
– the fragment's exit edge is (g_j, n), where $\{n\} = g_j \bullet$.

Fig. 4.3 shows an example of a block defined by activities $a_1, \dots, a_k$. After block abstraction, two branches of the original process fragment are replaced by one branch with an aggregating activity a_{agg}. Semantics of the aggregating activity conforms to the execution semantics of the abstracted block and depends on the block type. In case of a XOR block the aggregating activity a_{agg} means that only one activity of the abstracted branches is executed. In case of an AND block the aggregating activity denotes that both abstracted activities are executed.

Definition 4.4 (Block Elementary Abstraction).
Let $\mathsf{PM} = (A, G, F, t, s, e)$ be a process model. *Block elementary abstraction* associated with the block model fragment $\mathsf{PMF} = (A_{\mathsf{PMF}}, \{g_j, g_s\}, F_{\mathsf{PMF}}, t)$ and activities $a_i, a_j \in A_{\mathsf{PMF}}$, where $i, j \in \mathbb{N}$ and $i < j \leq k$ is an aggregation operation $\sigma_b : M \to M$ that transforms process model PM into $\mathsf{PM}_a = (A_a, G_a, F_a, t_a, s_a, e_a)$ so that:
if $k = 2$:

– a_{agg} is the newly introduced aggregating activity
– $A_a = (A \backslash A_{\mathsf{PMF}}) \cup \{a_{agg}\}$
– $G_a = G \backslash \{g_j, g_s\}$
– $F_a = (F \backslash F_{\mathsf{PMF}}) \cup \{(a, a_{agg}), (a_{agg}, b)\}$, where $\{a\} = \bullet g_s$ and $\{b\} = g_j \bullet$,

else:

– a_{agg} is the newly introduced aggregating activity
– $A_a = (A \backslash \{a_i, a_j\}) \cup \{a_{agg}\}$
– $G_a = G$
– $F_a = (F \backslash \{(g_s, a_i), (a_i, g_j), (g_s, a_j), (a_j, g_j)\}) \cup \{(g_s, a_{agg}), (a_{agg}, g_j)\}$.

In case of the block elementary abstraction the average execution cost of the aggregating activity is evaluated as follows. For an XOR block:

$$cost_{avg}(a_{agg}) = p(a_i) \cdot cost_{avg}(a_i) + p(a_j) \cdot cost_{avg}(a_j),$$

where $p(a_i)$ and $p(a_j)$ are the execution probabilities of activities a_i and a_j, respectively. For an AND block an average execution cost of the aggregating activity is:

$$cost_{avg}(a_{agg}) = cost_{avg}(a_i) + cost_{avg}(a_j).$$

Loop Elementary Abstraction

It is a common situation when a set of tasks is iterated to complete the business process. In a model capturing such a process the set of tasks is put into a loop construct. Loops can be modeled by means of control flow. Wide application of loops by modelers makes support of abstraction from loops an essential part of the approach. Therefore, we introduce one more elementary abstraction—*loop elementary abstraction*. Let us define what kind of process fragment is considered to be a loop fragment.

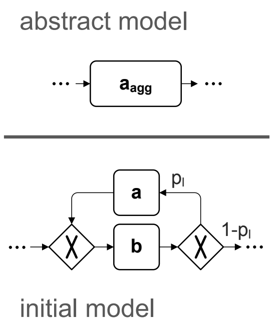


Fig. 4.4. Loop elementary abstraction.

Definition 4.5 (Loop Process Model Fragment).
Let $\mathsf{PM} = (A, G, F, t, s, e)$ be a process model containing a process model fragment $\mathsf{PMF} = (A_{\mathsf{PMF}}, G_{\mathsf{PMF}}, F_{\mathsf{PMF}}, t_{\mathsf{PMF}})$. The fragment PMF is a *loop process model fragment,* if it is a SESE fragment such that:

– $G_{\mathsf{PMF}} = \{g_j, g_s\}$
– $| \bullet g_j| = |g_s \bullet | = 2$
– there is exactly one path from g_s to g_j and exactly one path from g_j to g_s
– the entry edge of PMF is (n, g_j), where $\{n\} = \bullet g_j \backslash (A_{\mathsf{PMF}} \cup G_{\mathsf{PMF}})$
– the exit edge of PMF is (g_s, n'), where $\{n'\} = g_s \bullet \backslash (A_{\mathsf{PMF}} \cup G_{\mathsf{PMF}})$
– $|A_{\mathsf{PMF}} \cap \bullet g_s \cap g_j \bullet | + |A_{\mathsf{PMF}} \cap \bullet g_j \cap g_s \bullet | > 0$
– a function t_{PMF} is the restriction of t to set G_{PMF}
– $t_{\mathsf{PMF}}(g_j) = t_{\mathsf{PMF}}(g_s) = xor$.

Definition 4.5 allows a loop with no more than two activities, but no less than one. Loop elementary abstraction replaces a loop fragment by one aggregating function l, see Fig. 4.4. An aggregating function states that functions a and b are executed iteratively. Notice that Definition 4.5 allows either a or b to be missing. Finally, Definition 4.6 describes the mechanism of loop elementary abstraction.

Definition 4.6 (Loop Elementary Abstraction).
Let $\mathsf{PM} = (A, G, F, t, s, e)$ be a process model. *Loop elementary abstraction* is an aggregation operation $\sigma_l : M \to M$ that replaces a loop process model fragment $\mathsf{PMF} = (A_{\mathsf{PMF}}, G_{\mathsf{PMF}}, F_{\mathsf{PMF}}, t_{\mathsf{PMF}})$ with activity a_{agg} transforming process model PM into $\mathsf{PM}_a = (A_a, G_a, F_a, t_a, s_a, e_a)$ so that:

– $A_a = (A \backslash A_{\mathsf{PMF}}) \cup \{a_{agg}\}$
– $G_a = G \backslash G_{\mathsf{PMF}}$
– t_a is a restriction of t to G_a
– $F_a = (F \backslash F_{\mathsf{PMF}}) \cup \{(n, a_{agg}), (a_{agg}, n')\}$, where $\{n\} = \bullet g_j \backslash (A_{\mathsf{PMF}} \cup G_{\mathsf{PMF}})$ and $\{n'\} = g_s \bullet \backslash (A_{\mathsf{PMF}} \cup G_{\mathsf{PMF}})$.

The average execution cost of aggregating activity a_{agg} can be found as:
$cost_{avg}(a_{agg}) = \frac{1}{1-p_l} \cdot (cost_{avg}(b) + cost_{avg}(a) \cdot p_l)$,
where p_l is the probability of continuing the loop iteration.

4.1.2 Composition of Elementary Abstractions

This chapter engineers a business process model abstraction supporting the use cases in the group "Group 4: Obtaining a Process Quick View". These use cases imply that activities of a process model become more coarse-grained. According to our terminology, see Section 3.2, these coarse-grained activities are significant. Notice that each such activity corresponds to a set of insignificant activities of the initial process model. We propose to arrive at coarse-grained activities gradually, abstracting from one insignificant activity at a time. Each elementary abstraction provisions the desired effect: it conceals one insignificant activity aggregating it with other activities of the model. However, to conceal several insignificant activities within a process model, a composition of elementary abstractions is required.

While there are many ways how elementary abstractions can be composed, we provide one concrete method. The key idea of this method is to abstract from insignificant activities one by one using the available elementary abstractions. Algorithm 1 implements this idea. The inputs of the algorithm are:

Model PM	a process model to be abstracted
SortedList AL	a sorted list of insignificant activities
Set *elementaryAbstractions*	a set of available elementary abstractions

Here we assume that the algorithm's input is the sorted list of insignificant activities. As it follows from Definition 3.4, business process model abstraction conceals all insignificant abstraction objects (insignificant activities in our case). In general case these insignificant objects form a set, where each element is a process detail to be abstracted. Following our earlier observation that frequently abstraction objects can be ordered, we order the set's elements and obtain a sorted list that we use as the algorithm's input. We assume that the list of insignificant activities is sorted so that the least insignificant activity is the first element in the list. The output of the algorithm is the abstract process model.

Algorithm 1 begins creating a short circuit version of the process model to be abstracted, line 2. The algorithm iterates over the list of insignificant activities selecting the activity of the lowest significance a, lines 3–4. The activity to be abstracted is removed from the list, line 5. Algorithm 1 tries to find an elementary abstraction capable of abstracting a, line 6. If there is no such elementary abstraction, the next insignificant activity from the list is processed. If a suitable elementary abstraction is found, the corresponding process model fragment is processed, lines 7–9. First, all the activities of the fragment are removed from the sorted list AL. Then, the model PM is updated according to the elementary abstraction, line 10. If the aggregating activity a_{agg} is considered to be insignificant, it is inserted in the list AL to enable further abstraction, lines 11–12. Finally, the algorithm removes the back edge in the short circuited process model producing the abstract process model PM_a.

Fig. 4.5 illustrates the work of Algorithm 1. The starting point is the initial model PM, where activities b and i are insignificant. The figure visualizes the work of the algorithm presenting the final outcome—model PM_2 and the intermediate result of abstraction—model PM_1. Notice that to arrive at PM_1 from PM one activity is abstracted, as well as to arrive at PM_2 from PM_1. In the first step activity i is abstracted by means of block elementary abstraction, resulting activity gi. As gi is considered to be significant, the list of insignificant activities contains b only. Step 2 abstracts from b by means of sequence elementary abstraction bringing us to model PM_2 and the empty list of insignificant activities. As a result, model PM_2 is obtained.

Proposition 4.7. Algorithm 1 terminates and after termination the sorted list *activityList* is empty or the model contains exactly one activity.

Proof. First, we reason that Algorithm 1 terminates. The algorithm terminates if the *while* loop terminates, which happens if the list of insignificant activities gets empty or the model contains one activity. Hence, we need to show that at least one of the two conditions eventually holds. Consider the body of the *while* loop. Each iteration starts removing one activity from the list. After this there are two options. The first option is that there is no elementary abstraction that can handle this activity. No element is inserted into the list and within the iteration the list decreases in size by one. The model remains unchanged. The second option is that the activity can be handled. In this case an aggregating activity might be inserted into the list. Thus, the list of insignificant activities has a non-increasing size. Assuming that

Algorithm 1 Abstraction as composition of elementary abstractions

```
1: abstract(Model PM, SortedList AL, Set $elementaryAbstractions$)
2: ShortCircuitModel $\mathsf{PM}_c$ = shortCircuit(PM)
3: while !AL.isEmpty() OR $|A| == 1$ do
4:    $a$ = AL.getFirst()
5:    AL.remove($a$)
6:    if exists elementary abstraction aggregating $a$ in $\mathsf{PM}_c$ then
7:       PMF= $abstraction$.getFragment($a$)
8:       for all $fragmentActivity \in A_{\mathsf{PMF}}$ do
9:          AL.remove($fragmentActivity$)
10:      apply elementary abstraction to aggregate $a$ in $\mathsf{PM}_c$
11:      if $a_{agg}$ is insignificant then
12:         AL.insert($a_{agg}$)
13: Model $\mathsf{PM}_a$ = toProcessModel($\mathsf{PM}_c$)
14: return $\mathsf{PM}_a$
```

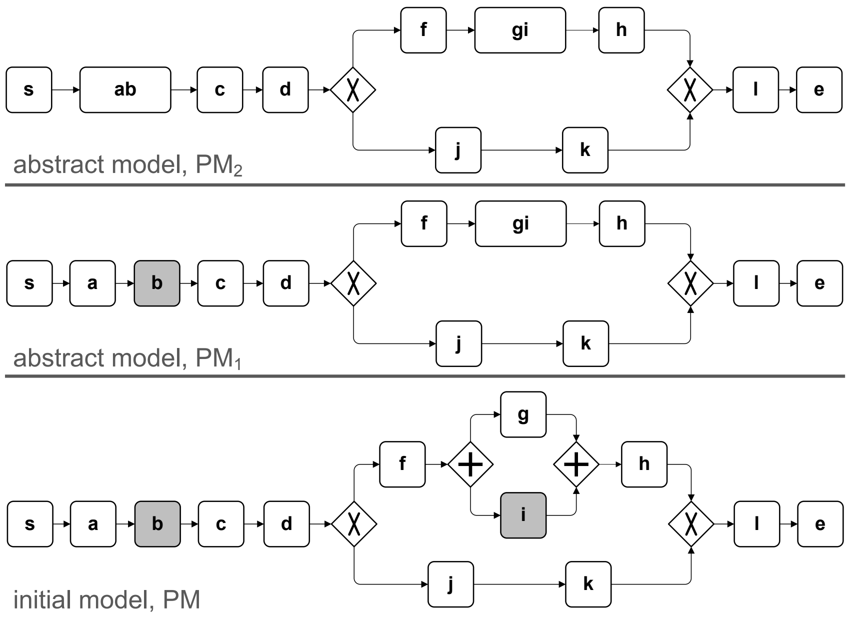


Fig. 4.5. Abstraction as a composition of elementary abstractions. Activities *b* and *i* are insignificant. First, we abstract from *i* by means of block elementary abstraction. The resulting model PM_1 contains only one insignificant activity *b*. It is aggregated using sequence elementary abstraction that brings us to model PM_2 with no insignificant activities.

elementary abstraction aggregates two activities, we observe the decrease of activities number in the model by one through each application of an elementary abstraction. Hence, the number of activities in a process model is a non-increasing number as well. Furthermore, according to the logic of the *while* loop body, each iteration decreases either the size of insignificant activity list, or the number of activities in the model. Thereby, eventually we arrive at the situation when one of the two conditions of the *while* loop holds and Algorithm 1 terminates.

The number of activities in the AL is not higher than in the process model. Indeed, each iteration removes one activity first and then may insert an aggregating activity that belongs to the model as well. As soon as the size of AL and process model activity set decrease through abstraction process, the algorithm concludes in one of the following states. Either there is nothing to abstract, i.e., the list AL gets empty, or there is one insignificant activity in the process model (AL contains exactly this activity). □

The computational complexity of Algorithm 1 is defined by two factors. First, it is the number of iterations of the *while* loop. In the worst case scenario we have to abstract each activity of a process model, i.e., do $|A|$ iterations, where A is the set of activities in the initial process model. Second, a process model fragment containing each activity has to be recognized. For the considered set of elementary abstractions, the fragments can be recognized in constant time: it is enough to analyze the adjacent nodes of an insignificant activity. Thereby, the running time of the algorithm is $O(|A|)$, where A is the set of activities in the initial process model.

The design of Algorithm 1 reflects that elementary abstractions cannot handle process models with an arbitrary topology. Namely, if no elementary abstraction can handle the fragment containing an insignificant activity, it is preserved in the model. We say that Algorithm 1 acts at the best effort principle: it *tries* to abstract from all the given insignificant activities, but does not guarantee this. The next section elaborates on the conceptual challenge explaining this phenomenon.

4.1.3 Limitations of Pattern-Based Abstraction

The identified set of process model fragment types is by no means complete with respect to the structure of process models observed in practice. As a consequence, not every process models can be abstracted by the presented set of elementary abstractions. Fig. 4.6 illustrates a process model challenging the debated abstraction approach. Consider the situation when activity f is insignificant. Among the three proposed elementary abstractions none is capable of abstracting f: no elementary abstraction prescribes the transformation of the fragment PMF. Against this background, various research endeavors suggest broader elementary abstraction sets. For instance, the study in [121] complements sequence, block, and loop elementary abstractions with the *dead end elementary abstraction.* [29, 53, 94] advocate more sophisticated elementary abstractions. However, abstraction methods based on elementary abstractions face a common challenge. Each elementary abstraction set requires an argument about the model class reducible with the given elementary abstractions. The need for such an argument is the main limitation of

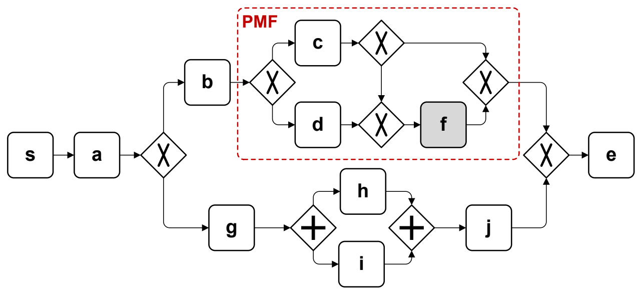


Fig. 4.6. A challenge for pattern-based abstraction. The fragment PMF cannot be handled by the elementary abstractions introduced in Section 4.1.1. Hence, either this activity is preserved, or a new elementary abstraction has to be developed.

pattern-based approaches. To abstract arbitrarily structured process models, we have to overcome this limitation. The next section explains how process model fragments can be discovered in an alternative way.

4.2 Decomposition-Based Methods

Rather than referencing process model fragments explicitly specifying their structure, we can identify them by properties. In the context of business process model abstraction, fragments with exactly one incoming edge and exactly one outgoing edge are of particular value. Such fragments are called *single entry single exit* (SESE) fragments, see Section 2.3, and can be safely replaced with one coarse-grained activity. The current section argues how we can leverage SESE fragments for business process model abstraction.

As we argued earlier, the discovery of SESE fragments in process models is well studied. In particular, we can make use of algorithms that deliver a unique hierarchy of canonical SESE fragments—a process structure tree (PST), see Section 2.3. Once a process model is decomposed into canonical SESE fragments and the corresponding PST is built, an abstraction can be realized. Our abstraction approach relies solely on aggregation of activities. This means that in every abstraction step two or more activities are aggregated. Let a be an activity to be abstracted in the current step. We aim to find the minimal canonical SESE fragment $sese_{min}$, containing a and at least one more activity. Every activity has one incoming edge and one outgoing edge. Thus, it constitutes a canonical SESE fragment represented by a leaf in the PST. We traverse all the leaves in the PST and find the one containing the activity a. Let us call it $sese_a$. Since the discovered fragment contains only the activity a, it is of no use for abstraction: $sese_a$ cannot be used as $sese_{min}$. There are two options for the selection of $sese_{min}$:

1. There is a canonical SESE fragment $sese_{a'}$ which is in the predecessor-successor relation with the fragment $sese_a$. Then $sese_{min}$ is the SESE fragment with the incoming edge of the predecessor and the outgoing edge of the successor in the pair $sese_a$, $sese_{a'}$.
2. If there is no canonical SESE fragment, which is in the predecessor-successor relation with the fragment $sese_a$, then $sese_{min}$ is a SESE fragment which is the parent of $sese_a$.

Once the fragment $sese_{min}$ is identified, it is replaced with one aggregating activity in the process model. The incoming edge of the aggregating activity is the incoming edge of $sese_{min}$, whilst its outgoing edge is the outgoing edge of $sese_{min}$. Definition 4.8 formalizes this operation.

Definition 4.8 (SESE Fragment Abstraction).
Let $\mathsf{PM} = (A, G, F, t, s, e)$ be a process model. *SESE fragment abstraction* is an aggregation $\sigma : M \to M$ that replaces a SESE process model fragment $\mathsf{PMF} =$

Algorithm 2 Abstraction based on process model decomposition

```
1: abstract(Model PM, SortedList AL)
2: ShortCircuitModel $\mathsf{PM}_c$ = shortCircuit(PM)
3: PST PST = constructPST($\mathsf{PM}_c$)
4: while !AL.isEmpty() OR $|A| == 1$ do
5:    $a$ = AL.getFirst()
6:    AL.remove($a$)
7:    using PST find $sese_a$ the minimal SESE fragment containing $a$
8:    if in PST $\exists sese'_a : sese_a \hookrightarrow sese'_a$ then
9:       $sese_{min}$ is defined by the entry edge of $sese_a$ and the exit edge of $sese'_a$
10:   else if in PST $\exists sese'_a : sese'_a \hookleftarrow sese_a$ then
11:      $sese_{min}$ is defined by the entry edge of $sese'_a$ and the exit edge of $sese_a$
12:   else
13:      $sese_{min}$ is the direct parent of $sese_a$
14:   for all $a' \in sese_{min}.A_{sese_{min}}$ do
15:      AL.remove($a'$)
16:   apply SESE fragment abstraction to $sese_{min}$ in $\mathsf{PM}_c$ and update PST
17:   if $a_{agg}$ is insignificant then
18:      AL.insert($a_{agg}$)
19: Model $\mathsf{PM}_a$ = toProcessModel($\mathsf{PM}_c$)
20: return $\mathsf{PM}_a$
```

$(A_{\mathsf{PMF}}, G_{\mathsf{PMF}}, F_{\mathsf{PMF}}, t_{\mathsf{PMF}})$ defined by a pair of edges $((a, b), (c, d))$ with activity a_{agg} transforming process model PM into $\mathsf{PM}_a = (A_a, G_a, F_a, t_a, s_a, e_a)$ so that:

– $A_a = (A \backslash A_{\mathsf{PMF}}) \cup \{a_{agg}\}$
– $G_a = G \backslash G_{\mathsf{PMF}}$
– t_a is a restriction of t to G_a
– $F_a = (F \backslash F_{\mathsf{PMF}}) \cup \{(a, a_{agg}), (a_{agg}, d)\}$.

Algorithm 2 formalizes the discussion above. The inputs of the algorithm are:

Model PM	a process model to be abstracted
SortedList AL	a sorted list of insignificant activities

Algorithm 2 returns the abstract process model. It starts with creating the short circuit process model from PM, line 2. Further, the short circuit model is decomposed resulting the PST, see line 3. The algorithm iterates over the list AL starting from the activity of the least significance a, lines 4–6. Once a SESE fragment $sese_a$ for activity a is found, line 7, $sese_{min}$ is chosen, lines 8–13. All the insignificant activities that belong to this fragment are removed from AL, lines 14–17. Furtheron, $sese_{min}$ is replaced with an aggregating activity a_{agg}. The aggregating activity a_{agg} may be added to the list of insignificant activities AL if required. Finally, the abstract process model is obtained from the short circuit model PM_c and returned as the result, lines 19–20.

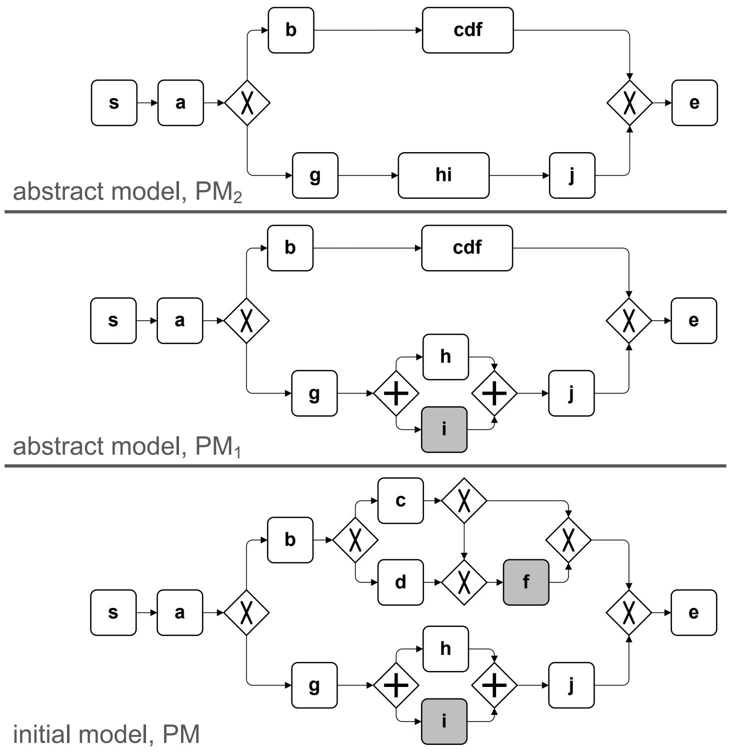


Fig. 4.7. Abstraction by means of process model decomposition. Activities f and i are insignificant. The abstraction from activity f brings us to model PM_1, while the abstraction from i results in model PM_2.

Fig. 4.7 illustrates the work of Algorithm 2. The starting point is the initial model PM, where activities f and i are insignificant. The figure visualizes the work of the algorithm presenting the final outcome—model PM_2 and the intermediate result of abstraction—model PM_1. To arrive at PM_1 from PM one activity is abstracted. Also to obtain PM_2 from PM_1 two activities are handled. In the first step activity f is abstracted, resulting activity *cdf*. Activity *cdf* is considered to be significant and the list of insignificant activities contains i only. Step 2 abstracts from i aggregating h and i. This brings us to the model PM_2 and the empty insignificant activity list.

Proposition 4.9. Algorithm 2 terminates. Upon termination the abstract process model contains no insignificant activity (list AL is empty) or the abstract model contains exactly one activity.

Proof. The termination of Algorithm 2 can be proven in the same fashion as in Proposition 4.7. Similarly, we can show that the abstract process model either contains no insignificant activity or exactly one activity, following the argumentation used in Proposition 4.7. □

The running time of Algorithm 2 is $O(|A|^2)$, where A is the set of activities in the process model to be abstracted. Indeed, we have to iterate over the *while* loop in the worst case $|A|$ times. Further, within each iteration we handle one SESE fragment, which size is also proportional to $|A|$. Although the number of iterations and the size of analyzed SESE fragments are related, we estimate the upper bound of running time for Algorithm 2 as $O(|A|^2)$.

Theorem 4.10. Algorithm 2 results in an order-preserving business process model abstraction.

Proof. Algorithm 2 realizes abstraction concealing one insignificant activity at a time. An activity is abstracted, once a SESE process model fragment containing it is replaced by one coarse-grained activity. We notice that Algorithm 2 results in a hierarchical abstraction. This implies that each process model activity is included into exactly one activity group that becomes a more coarse-grained activity. The latter can be aggregated at the later stage of abstraction, but, again, this coarse-grained activity is assigned to exactly one group. Hence, abstraction results in a hierarchy of activity groups. Since abstraction is hierarchical, Algorithm 2 is order-preserving, if abstraction of each insignificant activity is order-preserving. Indeed, the overall abstraction is then the composition of order-preserving abstractions. This composition is order-preserving, since we deal with hierarchical abstraction.

When we argue about abstraction hiding insignificant activity a, we distinguish two process models: the model, where a is the less significant activity and the model obtained as a result of aggregation of a. We reference the former model as PM and the latter model as PM_a. To arrive at PM_a from PM, a SESE process model fragment PMF is replaced by an activity x. Hence, to proof that Algorithm 2 is order-preserving, we study the behavioral relations between activity x and the rest of activities in the model PM_a. We denote an activity that does not belong to fragment PMF as b, $b \notin A_{PMF}$. Notice that every such activity appears in both models, PM and PM_a. As we discuss one abstraction step, only the activities in the fragment PMF got concealed, while the others appear in PM_a. Against this background, we adapt the Definition 3.12 of order-preserving abstraction as follows. We want to show that for x and $\forall b \in A_a, b \neq x$:

- $a \rightsquigarrow_{\mathsf{PM}} b \Rightarrow x \rightsquigarrow_{\mathsf{PM}_a} b$
- $a \rightsquigarrow^{-1}_{\mathsf{PM}} b \Rightarrow x \rightsquigarrow^{-1}_{\mathsf{PM}_a} b$
- $a +_{\mathsf{PM}} b \Rightarrow x +_{\mathsf{PM}_a} b$
- $a||_{\mathsf{PM}} b \Rightarrow x||_{\mathsf{PM}_a} b$.

To show that Algorithm 2 delivers order-preseerving abstraction, it is enough to show that these four statements hold. We prove these statements one by one. The proofs make use of the observation that for the considered class of process models there is a correspondence between the behavioral profile relations and the model structure [162]:

- $a \rightsquigarrow b$ iff aF^+b and $b\cancel{F^+}a$
- $a \rightsquigarrow^{-1} b$ iff $a\cancel{F^+}b$ and bF^+a
- $a + b$ iff $a\cancel{F^+}b$ and $b\cancel{F^+}a$
- $a||b$ iff $(aF^+b$ and $bF^+a)$ or $(a\cancel{F^+}b$ and $b\cancel{F^+}a)$.

First, we show that $a \rightsquigarrow_{\mathsf{PM}} b \Rightarrow x \rightsquigarrow_{\mathsf{PM}_a} b$. From $a \rightsquigarrow_{\mathsf{PM}} b$ it follows that $aF^+_{\mathsf{PM}}b$ and $b\cancel{F^+_{\mathsf{PM}}}a$. Since b does not belong to PMF, the exit edge *exit* of PMF is on the path from a to b. Hence, there is a path from *exit* to b. As soon as *exit* becomes the outgoing edge of x in the model PM_a, there is a path from x to b in PM_a. At the same time, there is no path from b to a in PM. As soon as there is a path from the entry edge *entry* of PMF to a, we conclude that there is also no path from b to *entry*. Subsequently, there is no path from b to x in the abstract model PM_a. Thereafter, we have shown that $x \rightsquigarrow_{\mathsf{PM}_a} b$.

Second, we show that $a \rightsquigarrow^{-1}_{\mathsf{PM}} b \Rightarrow x \rightsquigarrow^{-1}_{\mathsf{PM}_a} b$. $a \rightsquigarrow^{-1}_{\mathsf{PM}} b$ means that $a\cancel{F^+_{\mathsf{PM}}}b$ and $bF^+_{\mathsf{PM}}a$. Subsequently, there is no path from the *exit* edge of PMF to b and there is a path from b to *entry*. Hence, there is no path from x to b, while there is a path from b to x. From the latter two statements it follows that $x \rightsquigarrow^{-1}_{\mathsf{PM}_a} b$.

Following the same line of argumentation we can show that $a +_{\mathsf{PM}} b \Rightarrow x +_{\mathsf{PM}_a} b$ and $a||_{\mathsf{PM}}b \Rightarrow x||_{\mathsf{PM}_a}b$. As we have shown that the four properties of order-preserving abstraction hold, we state that abstraction of one activity according to Algorithm 2 is order-preserving. $\square$

4.3 Discussion

This chapter developed the structural methods of business process model abstraction: the pattern-based and decomposition-based. The former gives a lot of fine tuning capabilities, e.g., the user can specify methods for non-functional properties estimation. The latter handles a broader model class. Notice that both methods can be extended. In the case of the pattern-based method, new patterns can be introduced, e.g., see [53, 121]. Similarly, more advanced process model decomposition techniques, like rPST [124, 152, 154], can be used as abstraction enablers, see [123]. This section compares the two structural methods with respect to model transformation properties important from the application perspective. Section 4.3.1 argues how the user can control the abstraction methods. Section 4.3.2 elaborates on the order preservation property, Section 4.3.3—on non-functional properties evaluation, and Section 4.3.4 on the measure of information loss during abstraction. We conclude discussing the limitations of structure-based abstraction methods in Section 4.3.5.

4.3.1 User Control

Operationalization of structural abstraction methods assumes that a user has means to control the abstraction procedure. The user should be capable of specifying which abstraction objects are significant and which are not. One can notice that both algorithms, Algorithm 1 and Algorithm 2, consider the list of insignificant activities as the input. We propose to leverage this list as means to control the abstraction process. Indeed, once the list is populated, the abstraction evolves without user interference.

4.3.2 Order Preservation

Both aforementioned abstraction approaches are order-preserving, see Definition 3.12. While Theorem 4.10 formally shows that Algorithm 2 is order-preserving, the abstraction based on sequence, block, and loop elementary abstractions is order-preserving as well. This follows from two observations. As sequence and loop process model fragments are SESE process model fragments, the corresponding element abstractions are SESE fragment abstractions, see Definition 4.8. At the same time, from a structural perspective the block abstraction removes only one branch in a block. Hence, block elementary abstraction does not impact the other ordering constraints of the model. Notice that an expansion of the elementary abstraction set might yield an abstraction that is not order-preserving. Therefore, choosing the structural patterns and corresponding model transformation rules, the designer of abstraction method controls whether it is order-preserving.

4.3.3 Evaluation of Activity Non-Functional Properties

As Section 3.4 witnesses, there are several scenarios of business process model abstraction. While some scenarios demand transformation of the process model structure only, others imply transformation of additional model information. Furthermore, since such scenarios as "Use Case 15: Preserve Frequent Activities Summarizing Rare Activities" employ non-functional properties of model elements as a significance criterion, model abstraction cannot be limited to structural transformation only. Indeed, for each newly introduced model element, e.g., aggregating activity, it is essential to decide if the model element is significant or not. Hence, the abstraction should also prescribe how to transform the values of non-functional properties of model elements. For instance, consider scenario "Preserve Frequent Activities Summarizing Rare Activities", where the significance criterion is the average activity frequency. In this scenario the abstraction method should provide means to evaluate the average frequency of an aggregating activity.

Business process model abstraction based on structural methods provides a varying level of support for transformation of non-structural information.

For instance, Section 4.1.1 discusses not only the structural aspects of elementary abstractions, but also exemplifies how the average activity execution cost can be estimated. In general case, the designer of elementary abstractions may specify how non-functional properties of aggregating activities can be evaluated using the information in the initial model, e.g., see [32, 53, 121]. The abstraction approach presented in Section 4.1 evaluates such non-functional properties as the average activity execution cost and the average activity execution probability. An abstraction based on process model decomposition requires more sophisticated methods for non-functional properties evaluation: the inner structure of the transformed model fragments is unknown. Abstraction presented in Section 4.2 provides no means to evaluate activity non-functional properties.

4.3.4 Abstraction Smoothness

Abstraction leads to information loss. For instance, if a user intentionally aggregates a sequence of activities into one activity, the loss of information about particular activities and their ordering constraints is intended. An abstraction technique should provide precise mechanisms to achieve (and not to under- or overachieve) the desired level of information loss. Against this background, each abstraction method should be capable to hide only insignificant details preserving significant ones. Unfortunately, this is not always the case. Consider the example of the business process model abstraction based on the process model decomposition presented in Section 4.2. Depending on the process model structure, this abstraction may conceal significant process details. The example model fragment in Fig. 4.8 illustrates the problem. If activity f has to be concealed, activities c and d are affected as well: they are contained in the same canonical SESE fragment $sese_{min}$ to which activity f belongs. As an outcome, this abstraction operation hides three activities, c, d, and f, two of them being significant process details. From the theoretical perspective, however, to abstract from activity f by means of aggregation one more activity is enough (to aggregate f with this activity). For instance, a sequence elementary abstraction, see Section 4.1.1, always aggregates an activity with exactly one of its neighboring activities. Thereafter, an abstraction

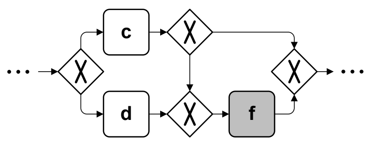


Fig. 4.8. A process model fragment resulting high abstraction smoothness if abstraction is realized according to the Algorithm 2. As the Algorithm 2 relies on process model decomposition into SESE fragments, the abstraction of activity f implies that the whole process model fragment is replaced by one activity. Hence, activities c and d are abstracted as well.

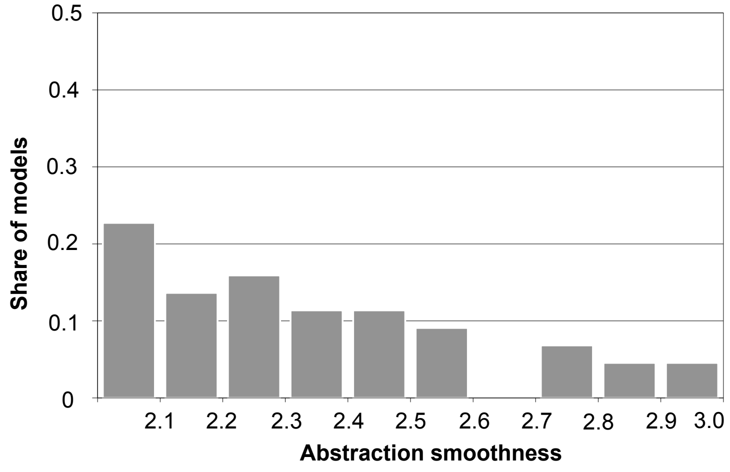
that aggregates three activities to hide one leads to undesirable information loss. In general, the "smaller" the effect of a basic abstraction operation (the less process information it abstracts), the better it caters for achievement of a precise model information level. This situation motivates us to introduce a measure that indicates how precisely aggregation performs.

In order to measure the precision of the abstraction technique quantitatively, we introduce a notion of abstraction smoothness. Abstraction smoothness quantitatively estimates the information loss produced by the application of one basic abstraction operation. In case of an abstraction which is based on activity aggregation, the abstraction smoothness reflects the difference between the number of activities in the process model before and after one basic abstraction operation. The less activities are aggregated by a single operation, the smoother it is. From the user perspective it is important to have a smooth abstraction which allows reaching required model information level and forbids undesired side effects. Notice that the purpose of smoothness is to indicate how much an abstraction technique diverges from the best theoretically possible result—aggregation of two activities only. That is why smoothness ignores if the aggregated activities belong together in terms of business semantics.

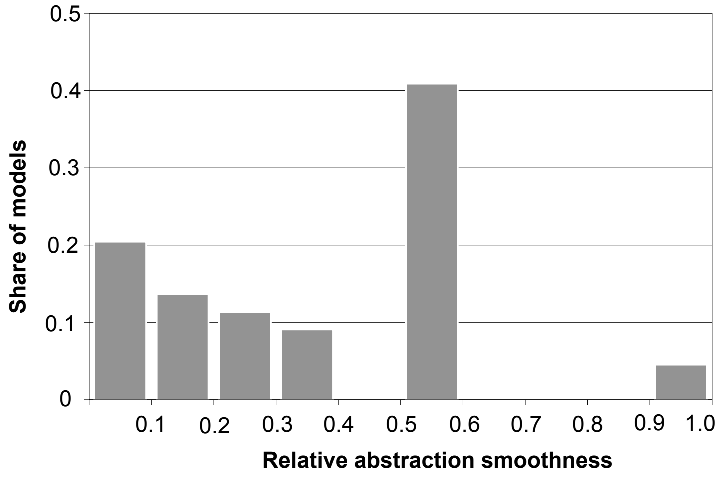
The smoothness of an abstraction technique can be measured as the mean of smoothness for every abstraction step. Theoretically abstraction demonstrates the best smoothness if every activity aggregation results in only two activities being abstracted. The elementary abstractions introduced in Section 4.1.1 fulfill this property: the sequence, block, and loop elementary abstractions aggregate exactly two activities. Thereby, the user has an opportunity to design such elementary abstractions that assure appropriate abstraction smoothness values. However, this optimal condition may not hold.

We provide empirical insights into expected abstraction smoothness values. The evaluation studies the smoothness of the abstraction approach based on process model decomposition into canonical SESE fragments. The approach is evaluated against a collection of 50 real world process models capturing business processes of a large German health insurance company. The models vary in size from 50 to 204 nodes. Each model is abstracted to one activity. While models are being abstracted, information about the smoothness is collected.

The experiment includes observation of two abstraction scenarios: "optimistic" and "pessimistic". For both scenarios we employ greedy algorithms. In the optimistic scenario the algorithm abstracts a process model in the way that the minimal number of activities is reduced in every abstraction step. In the pessimistic scenario the algorithm abstracts the maximal number of activities per step. The smoothness of model abstraction is found as the mean value of activities reduced at every step. Fig. 4.9 presents the distribution of abstraction smoothness obtained in the experiment. Results for the optimistic scenario are shown in Fig. 4.9(a) and for pessimistic—in Fig. 4.9(b). In the optimistic scenario all the models were abstracted with the smoothness between 2.0 and 3.0; more than a half—with the smoothness under 2.5. This means that very often only two activities were aggregated, which is close to



(a) Abstraction smoothness in the "optimistic" abstraction scenario: in every abstraction step the SESE fragment that contains the least number of nodes is selected.



(b) Relative abstraction smoothness in the "pessimistic" abstraction scenario: in every abstraction step the SESE fragment that contains the highest number of nodes is selected.

Fig. 4.9. An evaluation of the abstraction smoothness for the abstraction based on process model decomposition into SESE fragments. The set of 50 real world process models is used for the evaluation, where the model size varies from 50 to 204 nodes.

the best theoretically possible result. Pessimistic strategy aims to abstract the maximal number of activities in every step. Since models vary in size, in this scenario we use normalized smoothness, dividing abstraction smoothness by the number of nodes in a model. According to the diagram, around 40% of the models were abstracted in huge steps—about half of the model per step. This statistics reveals that the smoothness of the approach relying solely on activity aggregation can be poor.

4.3.5 Limitations of Structural Methods

To illustrate the shortcomings of purely structural business process model abstraction methods, we refer to the structural abstraction based on process model decomposition, see Section 4.2. We demonstrate the abstraction mechanism by a "Forecast request handling" process model example in Fig. 4.10. The model is decomposed into several fragments: the top level sequence that includes activities *Receive request via email*, *Request data gathering*, *Record request*, *Receive data*, *Archive data*, *Generate forecast report*, *Send report*, and the fragment g_1. The fragment g_1 contains two branches: one branch is a sequence of two activities, and the other branch—a sequence of activity "Prepare data for full analysis", fragment g_2, and activity "Consolidate results". The abstraction algorithm enables aggregation of either a pair of sequential activities, or branches in fragments g_1 and g_2. Activity aggregation may continue and end up with a process model containing one coarse-grained activity. The structural method exhibits the following properties.

Property 1 The model structure defines which model elements belong together.
Property 2 The model structure defines the control flow relations between model elements of the abstract process model.
Property 3 The structural abstraction enables hierarchical activity aggregation.

In this way structural abstraction methods address several issues arising within model transformation. At the same time, Properties 1–3 put strong assumptions on the abstraction approach. Property 1 implies that activities of one process model fragment in PM semantically belong together and correspond to a more coarse grained activity in PM_a. Assuming this, a structural algorithm neglects the domain semantics of activities. Property 2 follows from Property 1, as the relations between such coarse-grained activities of PM_a are directly deduced from the relations of fragments in the initial model PM. Finally, Property 3 means that each activity of the initial model PM belongs to exactly one activity of the abstract model PM_a. Together these three properties put strong limitations on the applicability of the abstraction based on structure. We exemplify the limitations of Property 1 asking the following questions with respect to activities *Receive request via email*, *Request data gathering*, and *Record request* in Fig. 4.10:

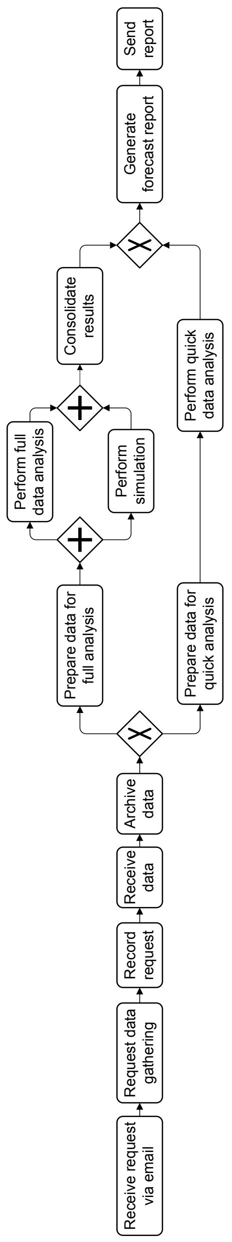

Receive request via email
Request data gathering
Record request
Receive data
Archive data
Prepare data for full analysis
Perform full data analysis
Perform simulation
Consolidate results
Prepare data for quick analysis
Perform quick data analysis
Generate forecast report
Send report

Fig. 4.10. Shortcomings of structural business process model abstraction: while the structural methods allow to aggregate activities *Receive request via email* and *Request data gathering* or *Receive request via email* and *Record Request* or all the three at once, they do not suggest which aggregation makes sense in terms of business semantics, i.e., which aggregation is meaningful.

1. Does an aggregation of *Receive request via email* and *Request data gathering* makes sense from the point of view of the domain semantics?
2. Is aggregation of activities *Receive request via email* and *Request data gathering* any better than an aggregation of *Receive request via email* and *Record Request* or than an aggregation of all the three named activities?

Apparently, structural information is insufficient to answer these questions. To overcome the limitations of the structural approach we investigate two questions separately:

1. How to discover groups of related activities within a process model?
2. How to discover the control flow relation of an abstract process model PM_a?

Decomposition of the initial problem into two subproblems requires us to cater several methods, each addressing one separate question. However, we gain an abstraction method that is more flexible and better suits the requirements of the real world. Following this path of argumentation the next chapter focuses on the discovery of groups of related activities, while Chapter 6 suggests an alternative approach to the abstract model control flow relation generation.

4.4 Summary

This chapter argued about structural methods enabling business process model abstraction. Doing so, the chapter explored the *why* and *how* of business process model abstraction with the main focus on the *how*. Building upon the findings in Section 3.4, we have designed the abstraction methods that align with the abstraction use cases demanded by practitioners. This chapter has developed two abstraction algorithms that rely on the well established techniques for process model analysis. The first algorithm makes use of structural patterns, while the second exploits process model decomposition. Both algorithms leverage process model structure to decide which activities are impacted by abstraction and how the model is transformed. The chapter provided an in depth discussion of the advocated algorithms, their advantages and limitations. We have elaborated on how the algorithms can be controlled by a user, discussed their properties, like order preservation and capability to evaluate activity non-functional properties. In particular, we have thoroughly explained the drawbacks of the presented structural abstraction, since they motivate the research contributions of chapters 5 and 6.

5 Discovery of Related Activities in Process Models

Multiple business process model abstraction use cases demand the increase of activity granularity. Vivid examples are the use cases of the group “Group 4: Obtaining a Process Quick View” demanding more coarse-grained activities to appear in the abstract process model, see Section 3.4. As we argued in Section 3.2.3, coarse-grained activities can be obtained by means of activity aggregation. In this context, each coarse-grained activity of the abstract model relates to a group of detailed activities in the initial model. Obviously, there are alternative ways to aggregate activities. From the user perspective, groups of activities that semantically belong together are of particular value. Therefore, this chapter investigates methods that aggregate activities according to their business meaning. In other words, given a business process model $\mathsf{PM} = (A, G, F, t, s, e)$, we search for such activity sets $C \subset A, C \neq \emptyset$ that each C has a self-contained business semantics. This research question can be related to both *why* and *how* of business process model abstraction. While Chapter 4 studied these two aspects by means of process model structure analysis, this chapter advocates an alternative solution considering the model element semantics.

The challenge of activity aggregation has been addressed by the BPM community in various contexts. For instance, activity aggregation is a recurrent problem in process mining where logs contain fine-grained activities. Against this backdrop, several process mining research endeavors developed methods for activity aggregation, e.g., see [65, 66, 100]. Since these solutions emerge from the process mining domain, they intensively exploit information about process instances. The analysis of the *instance* level information impedes the direct reuse of these methods in the context of business process *model* abstraction. Another approach to activity aggregation was demonstrated by [45]. Again, the authors considered the specific setting of the navigation flow in the web application. Looking back into Chapter 4, we observe that structural methods of business process model abstraction cater for identification of activity groups as well, e.g., see [30, 32, 94, 122, 123]. However, the control flow gives little if any information about the semantic relatedness of activities.

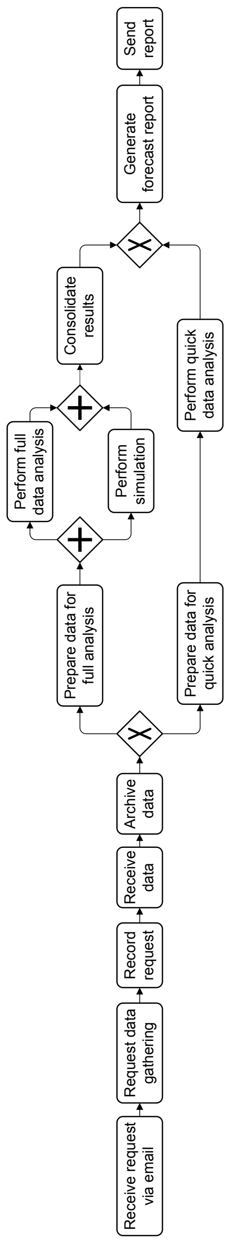

Receive request via email
Request data gathering
Record request
Receive data
Archive data
Prepare data for full analysis
Perform full data analysis
Perform simulation
Consolidate results
Prepare data for quick analysis
Perform quick data analysis
Generate forecast report
Send report

Fig. 5.1. A model of the “Forecast request handling” business process.

Indeed, [129] recently confirmed that multiple factors have to be considered, once activities are aggregated into a subprocess (which can be seen as a coarse-grained activity).

The example in Fig. 5.1 emphasizes the drawbacks of structural approaches. The figure presents the model of the "Forecast request handling" business process which detailed textual description can be found in Table 1.1. Consider the sequence of activities *Receive request via email*, *Request data gathering*, and *Record request.* According to the structural approach advocated in Chapter 4 this sequence, and its subsequences, can be considered as a coarse-grained activity. However, the structural approach does not answer the following questions:

– Does an aggregation of *Receive request via email* and *Request data gathering* makes sense from the point of view of the domain semantics?
– Is aggregation of activities *Receive request via email* and *Request data gathering* any better than an aggregation of *Receive request via email* and *Record Request* or an aggregation of all the three named activities?

We notice that these two questions challenge every structural approach. To overcome the limitations of structural abstraction, this chapter expands the scope of information considered by activity aggregation beyond the control flow. In particular, we deal with two information sources: domain ontologies and non-control flow process model elements.

This chapter develops two novel methods of activity aggregation. First, we study how domain ontologies describing activities and relations between them support aggregation. Namely, we investigate ontologies specifying activity *meronymy* (*part-of*) relation and design 1) a metric evaluating the relatedness of activities within a sample set and 2) an algorithm selecting sets of related activities in a process model. Second, we study how model elements that do not relate to the control flow, e.g., data objects and roles, help to identify groups of related activities. In this context, we adapt a well-known algorithm of cluster analysis and a model from information retrieval to realize aggregation. Notice that the discovery of related activity groups ignores the process ordering constraints. Instead, we shift the focus on the activity business semantics analysis. This allows us to overcome the limitations of traditional structural aggregation. Finally, we reason about the validity of both methods by an empirical argument.

The remainder of this chapter is structured into five sections. Section 5.1 argues about meronymy-based activity aggregation, while Section 5.2 discusses the application of cluster analysis for activity aggregation. Section 5.3 evaluates the two activity aggregation methods. The properties of the designed activity aggregations are discussed by Section 5.4. Finally, Section 5.5 concludes the chapter with a summary.

5.1 Meronymy-Based Activity Aggregation

Once we neglect process model structure as the criterion for activity aggregation, we are in the quest for alternative criteria search. One approach is to consider domain semantics of activities. We use business process domain ontologies as a formal representation of domain knowledge. The applications of ontologies in BPM have been throughly investigated by academia, e.g., see [33, 74, 85]. These research endeavors witness that while ontologies enable (semi-)automatic reasoning, the laborious step of ontology modeling impedes their wide application in industry. Thereafter, we notice that the assumption of an ontology existence limits the approach's applicability.

Ontologies may specify various relations between activities, e.g., meronymy, hyponymy, or equivalence. Meanwhile, activity aggregation explores the relations between coarse-grained activities and their parts. This context allows us to focus on the *meronymy* relation in the remainder of this section. Meronymy is a semantic relation denoting that one entity is a part of another entity. Meronymy is often referenced as a *part-of* relation and has been extensively studied in object-oriented analysis and design. In [17] Barbier et al. extend the formalization of this relation beyond the UML specification. Guizzardi focused on the modal aspects of the meronymy relation and the underlying objects in [64]. Finally, Dapoigny and Barlatier introduce a formalism that facilitates a precise and unambiguous description of a meronymy relation [40]. In the context of activity aggregation meronymy relation naturally hints on which activities belong together and which coarse-grained activity they constitute.

The meronymy relation organizes activities into a hierarchy, or a *meronymy tree*. Activities at the top of the tree are more coarse-grained than those deeper in the tree. Given an activity in the meronymy tree, its direct descendants are low-level activities to be accomplished to fulfill the given activity. Hence, each non-leaf activity can be iteratively refined down to leaves. Consider an example meronymy tree in Fig. 5.2. According to the tree, to complete activity *Receive forecast request*, activities *Receive request via email* and *Record request* have to be executed. The activity meronymy trees, or their close analogs, can be found in the works of BPM researchers and practitioners. For instance, the MIT Process Handbook specifies several activity meronymy trees [95].

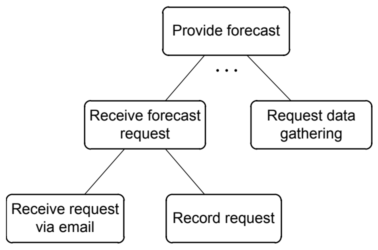


Fig. 5.2. A meronymy tree.

The *function view* of ARIS framework [138] is a vivid example of activity meronymy trees application in an industrial setting. As we leave other activity relations out of scope, each ontology can be formalized as several meronymy trees. From a practical perspective, the focus on activity meronymy relation decreases the ontology modeling effort in comparison to a full-fledged ontology.

We propose to use meronymy trees for activity aggregation. We reference a set of activities which is in question to be aggregated as an *aggregation candidate*. If all the activities of an aggregation candidate appear in a meronymy tree, they have a *lowest common ancestor* (LCA), see Section 2.1. We assume that the LCA can be used as a representative for the aggregation candidate. Returning to the example, we observe that *Receive request via email* and *Record request* are the direct descendants of activity *Receive forecast request* in the meronymy tree. According to our argument, this is a strong indication that these two activities should be aggregated. One can notice that *Request data gathering* appears in the tree as well. This allows us to consider the set *Receive request via email*, *Record request*, and *Request data gathering* as an aggregation candidate. Which of the two candidates is preferable? To answer this question we make an assumption that a good aggregation candidate comprehensively describes its LCA. In other words, we assume activities to be related, if they have a subsuming activity, LCA, and this activity is comprehensively described by the considered activities. According to this assumption, aggregation candidate *Receive request via email* and *Record request* is preferable, as it fully describes the ancestor *Receive forecast request.* At the same time, the set *Receive request via email*, *Record request*, and *Request data gathering* does not provide a comprehensive description of *Provide forecast*: there are other activities in the meronymy tree that contribute to its execution. Following this argumentation we mine activity aggregations in a process model and guide the user with recommendations on which activities belong together.

Against this background, this section develops activity aggregation that leverages meronymy relation. Once we formalize the problem, the chapter elaborates on the relation between model and ontology activities. The core contributions of this section are the metric enabling aggregation candidate comparison and the algorithm for the discovery of related activity groups.

The remainder of the current section is structured as follows. Section 5.1.1 formalizes the basic concepts, while Section 5.1.2 elaborates on the methods relating process model activities to ontology activities. Section 5.1.3 introduces a metric for aggregation candidate ranking. Finally, Section 5.1.4 develops an algorithm for the discovery of related activity groups.

5.1.1 Basic Concepts

This section formalizes the intuitive discussion sketched above. We postulate a universal alphabet of activities $\mathcal{A}$. For each process model $\mathsf{PM} = (A, G, F, t, s, e)$ it holds $A \subseteq \mathcal{A}$. According to the employed formalism an aggregation candidate $C \subseteq A$ is a subset of activities in a process model

$\mathsf{PM} = (A, G, F, t, s, e)$. The search for activity aggregations utilizes a domain ontology formalized as a meronymy forest.

Definition 5.1 (Meronymy Tree and Meronymy Forest).
A *meronymy tree* is an arborescence $\mu = (A_\mu, r_\mu, M_\mu)$, where:

- $A_\mu \subseteq \mathcal{A}$ is a finite non-empty set of activities
- $r_\mu \in A_\mu$ is a special node tree, *root*, capturing the most coarse-grained activity
- $M_\mu \subseteq A_\mu \times (A_\mu \backslash \{r_\mu\})$ is a set of edges such that $(a, b) \in M_\mu$, if b is part of a, i.e., they are in meronymy relation.

A *meronymy forest* F is a directed graph all of which weakly connected components are meronymy trees. We denote the set of activities in the meronymy forest F as $A_F = \bigcup_{\forall \mu \in F} A_\mu$.

An example meronymy tree μ is presented in Fig. 5.3. Notice that according to a meronymy forest definition, each activity appears exactly in one meronymy tree. Indeed, an activity is modeled by one node that belongs to exactly one tree. Next, Definition 5.1 does not assume the existence of one activity subsuming all the others. This is consistent, for instance, with ontologies, like the MIT Process Handbook, in which there are eight root activities [95].

We make use of the LCA for a set of nodes, see Section 2.1 for formal details. Let $\mu = (A_\mu, r_\mu, M_\mu)$ be a meronymy tree. We introduce an auxiliary function $lca_\mu : \mathcal{P}(A_\mu) \rightarrow A_\mu$, which for a set of activities $C \subseteq A_\mu$ returns a node $l \in A_\mu$ that is the lowest node having all the activities in C as descendants. The function is defined for a concrete meronymy tree and can be illustrated by the following two examples in tree μ: $lca_\mu(\{e, f\}) = n_2$ and $lca_\mu(\{e, g\}) = n_0$.

5.1.2 Matching Activities: from Process Models to Meronymy Forest

To enable aggregation we need to relate process model activities to the information in an ontology, i.e., a meronymy forest. In the trivial case, each process model activity is captured in the ontology. In practice this is true only if a process model is designed using the activities predefined by the ontology. However, we do not want to impose the restriction that a process model is constructed exclusively of ontology activities. Therefore, we use a matching

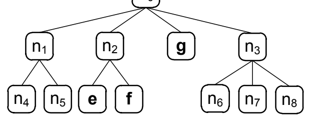


Fig. 5.3. The meronymy tree μ.

step to determine which process model activity matches which activity in the meronymy forest. The matching step is a necessity if a process model and a meronymy forest have been designed independently, but we reuse the ontology for the activity aggregation problem.

Definition 5.2 (Activity Match).
Let $\mathsf{PM} = (A, G, F, t, s, e)$ be a process model and F be a meronymy forest. The function $match_{\mathsf{PM}} : A \rightarrow \mathcal{P}(A_F)$ maps a process model activity to a set of activities in the meronymy forest. We extend *match* to sets such that $match_{\mathsf{PM}} : \mathcal{P}(A) \rightarrow \mathcal{P}(\mathcal{P}(A_F))$ and that for $Q \subseteq A$ it is defined as $match_{\mathsf{PM}}(Q) = \{match_{\mathsf{PM}}(q) | \; q \in Q\}$, which returns a set of match-sets, each corresponding to an element of Q.

Activity match function can be illustrated by mapping of a process model activity *Receive request via email* to meronymy forest activities *Receive email*, *Get email*, and *Receive message* and activity *Record request* to meronymy forest activities *Record request* and *Log request*:

match(Receive request via email) = {*Receive email, Get email, Receive message*}

match(Record request) = {*Record request, Log request*}

Then, the extension of function *match* to an activity set looks as follows.

match({*Receive request via email, Record request*}) = {{*Receive email, Get email, Receive message*}, {*Record request, Log request*}}

Further we use function *activity mixmatch* to capture the possible mappings of an aggregation candidate on the meronymy forest activities.

Definition 5.3 (Activity Mixmatch).
Let $\mathsf{PM} = (A, G, F, t, s, e)$ be a process model and F be a meronymy forest. Function $mixmatch_{\mathsf{PM}}$ returns all potential combinations of matches for each process model activity from an input set. This function $mixmatch_{\mathsf{PM}} : \mathcal{P}(A) \rightarrow \mathcal{P}(A_F)$ is defined so that for a set of activities $Q \subseteq A$ holds that $S \in mixmatch_{\mathsf{PM}}(Q)$, if $|S| = |Q|$ and $\forall u, v \in S$ holds that $\exists_{a_1, a_2 \in A}[a_1 \neq a_2 \wedge u \in match_{\mathsf{PM}}(a_1) \wedge v \in match_{\mathsf{PM}}(a_2)]$.

The activity mixmatch function is illustrated by the following example.

mixmatch({*Receive email, Record request*}) = {*(Receive email, Record request), (Receive email, Log request), (Get email, Record request), (Get email, Log request), (Receive message, Record request), (Receive message, Log request)*}

The *match* mapping enables activity mapping in both cases: if the process model was designed in the presence of a meronymy forest, or independently. In the former case function *match* maps an activity to a trivial set, containing only this activity. In the latter case *match* maps a process model activity to a set of similar activities in the meronymy forest.

One can foresee several implementations of functions *activity match* and *activity mixmatch*. In the trivial case the mapping can be realized by a user.

However, various methods for automation can be used. The reader can consult [143] for further insights.

5.1.3 Aggregation Candidates Ranking

Without any prior knowledge, every subset of a process model activity set might be considered as a potential aggregation candidate. However, we aim to select only those aggregation candidates which activities are strongly semantically related. There are various options for defining semantic relations among activities, for instance, based on operation on the same data object or execution by the same resource. In this section we utilize meronymy relations between activities in order to judge on their semantic relatedness. We say that activities in an aggregation candidate are strongly related, if together they comprehensively describe another activity—their LCA in a meronymy tree. The comprehensiveness depends on the existence of the LCA descendants that do not belong to the aggregation candidate. The larger share of the LCA descendants belongs to the aggregation candidate, the more comprehensive is the description. For example, activity set $\{e, f\}$ in Fig. 5.3 fully describes its LCA n_2. In contrast, activity set $\{e, g\}$ describes only a small part of its LCA, activity n_0.

We define a metric to measure how comprehensively a set of activities describes its LCA. We impose the following requirements on the metric. The metric must reflect, whether the activities of an aggregation candidate describe the LCA comprehensively. The more descendants, which do not belong to the aggregation candidate, the LCA has, the smaller share of the LCA is described by the aggregation candidate. The metric must be neutral to the distance between activities of an aggregation candidate and the LCA, as the distance has no direct influence on how comprehensively activities describe their ancestor. Similarly, the relative position of the LCA to the tree root is not characteristic in this context. This position reflects the abstraction level of an activity. However, we have no interest in LCA abstraction level. We also require the metric to be neutral to the size of an aggregation candidate. In other words, the metric enables comparison of aggregation candidates of different sizes and even comparison of an aggregation candidate with aggregation candidates, which are its subsets. Finally, it is handy, if a metric has a value between 0 and 1. We summarize this discussion as a list of requirements.

R1 Reflect, if the LCA has other descendents, except aggregation candidate activities.
R2 Be neutral to the depth of aggregation candidate in the LCA-rooted subtree.
R3 Be neutral to the depth of the LCA in the meronymy tree.
R4 Be neutral to the size of the aggregation candidate.
R5 Have a value between 0 and 1.

To present the designed function we first introduce an auxiliary function *meronymy leaves*. The function sets up a correspondence between a meronymy tree node and its descending leaves.

Definition 5.4 (Meronymy Leaves).
Let $\mu = (A_\mu, r_\mu, M_\mu)$ be a meronymy tree in a meronymy forest F. A *meronymy leaves function* $w_\mu : A_\mu \to \mathcal{P}(A_\mu)$ maps an activity $a \in A_\mu$ to a set of nodes that are the leaves of the subtree rooted to activity a.

Returning to the example tree μ, consider $w_\mu(g) = \{g\}$ and $w_\mu(n_2) = \{e, f\}$. Thereon, we propose the following metric for aggregation candidate ordering.

Definition 5.5 (Degree of Aggregation Coverage).
Let $\mu = (A_\mu, r_\mu, M_\mu)$ be a meronymy tree in a meronymy forest F and $C \subseteq A_\mu$ be an aggregation candidate. A function $cover : (\mathcal{P}(A_\mu)\backslash\emptyset) \to (0, 1]$ describes the degree of aggregation coverage, defined as:

$$cover(C) = \frac{\left| \bigcup\limits_{\forall a \in C} w_\mu(a) \right|}{|w_\mu(lca_\mu(C))|}.$$

The function captures the extent to which the activity set covers the LCA activity. The larger the share, the more "comprehensive description" provides the activity set. For the motivating example in the tree μ, see Fig. 5.3, the metric has values $cover(\{e, f\}) = 1$ and $cover(\{e, g\}) = 0.25$, i.e., $cover(\{e, f\}) > cover(\{e, g\})$. Due to this we conclude that $\{e, f\}$ is a better aggregation than $\{f, g\}$. As the aggregation metric makes use of *meronymy leaves* function, it considers the presence of other LCA descendents rather than those in the aggregation candidate. As the metric makes no use of distance measures, it is neutral to the depth of aggregation candidates in the LCA-rooted subtree, as well as the depth of the LCA in the meronymy tree. The metric is indifferent to the size of the aggregation candidate, but considers the number of leaves in the tree "covered" by the candidate. Finally, the metric value is always greater than 0, and reaches 1 at most. We conclude that the proposed aggregation metric satisfies requirements R1–R5.

5.1.4 Activity Aggregation Mining Algorithm

Building on the designed activity matching function and the metric for aggregation candidate ranking, we propose an algorithm for mining of activity aggregations from a process model. The mining algorithm has two subproblems: generation of aggregation candidates out of a process model and selection of aggregations from aggregation candidates. While the later problem exploits the developed aggregation metric *cover*, the former requires a discussion.

Generation of aggregation candidates from the model can be approached in a brute force fashion, if all the possible activity combinations are considered. However, the number of combinations in this case is $\mathcal{P}(|A|)$, where A is the set of activities in a process model. As business process model abstraction addresses complex process models with a large number of activities, this brute force method is insufficient. We need a method for coping with the computational complexity. A wholesome observation is that related activities are co-located within a process model [129]. According to this observation, we assume that for a given activity, the related activities can be found within a fixed graph distance. In this way we effectively manage the computational complexity problem. The computational complexity is further reduced, if we iteratively construct aggregation candidates pruning redundant ones. First, all the aggregation candidates of size two are created and analyzed. Candidates, which matches do not appear in one meronymy tree, are pruned. In the next iteration aggregation candidates of size three are constructed from the candidates of size two. Hence, the construction of aggregation candidates of size $k+1$ makes use of aggregation candidates of size k and their pruning.

Algorithm 3 formalizes the discussion above. The input of the algorithm is a process model $\mathsf{PM} = (A, G, F, t, s, e)$, a meronymy forest F, an aggregation metric threshold value $cover_0$, and *dist*—the graph node distance. The threshold value $cover_0$ and distance *dist* allow to set up the algorithm. The values can be selected by the user or empirically obtained, see Section 5.3.2. The output of the algorithm is the set of aggregations. The iterative construction of aggregations of increasing size is realized by two functions: *mine* and *kStep*. The entry point of the algorithm is function *mine*. For each activity in a process model, line 3, the algorithm finds a set of neighboring activities within a specified distance *dist*. Function $findNeigbours(activity, dist)$ returns the set of activities allocated within a distance not greater than *dist* from *activity* in the process model, line 5. Within this set all the subsets of size two are considered as aggregation candidates, line 6. Each candidate is evaluated against the ontology. If the candidate has no mappings to the ontology activities that belong to one tree, it is pruned, lines 7–8. Otherwise, the candidate mappings are evaluated against the specified metric threshold value $cover_0$. If there is at least one mapping of an aggregation candidate, for which the value of *cover* is greater than m_0 the candidate is considered to be an aggregation, lines 10–11. All the aggregation candidates that have not been pruned are used as the input for function *kStep*, line 12. Function *kStep* iteratively increases the size of aggregation candidates by one, pruning and evaluating them, lines 16–29. The pruning and evaluation of candidates follows the same principle as in function *mine*.

Algorithm 3 terminates, since functions *mine* and *kStep* iterate over finite sets. Upon termination the algorithm delivers set *aggregations*. The set *aggregations* contains those activity sets that fulfill the requirements imposed by the threshold $cover_0$ and are allocated within the radius of *dist* in the process model PM. As we argued earlier, in general case function *mine* iterates over

Algorithm 3 Activity aggregation mining

```
1: mine(Model PM = $(A, G, F, t, s, e)$, Forest $F$, double $cover_0$, int $dist$)
2: Set $aggregations = \emptyset$
3: for all $activity \in A$ do
4:    Set $candidates = \emptyset$
5:    for all $activityPair \in$ findNeighbours($activity$, $dist$) do
6:       $candidate = \{activityPair[1], activityPair[2]\}$
7:       for all $ontologyCandidate \in mixmatch_m(candidate)$ do
8:          if $\exists \mu \in F, \mu = (A_\mu, r_\mu, M_\mu) : ontologyCandidate \subseteq A_\mu$ then
9:             $candidates = candidates \cup \{candidate\}$
10:            if $cover(ontologyCandidate) \geq cover_0$ then
11:               $aggregations = aggregations \cup \{candidate\}$
12:   $aggregations = aggregations \cup$ kStep($candidates$, PM, $F$, $cover_0$, $dist$)
13: return $aggregations$
14:
15: \\Inductive step of aggregation mining
16: kStep(Set $kCandidates$, Model PM = $(A, G, F, t, s, e)$, Forest $F$, double $cover_0$, int $dist$)
17: Set $aggregations = \emptyset$
18: Set $(k+1)Candidates = \emptyset$
19: int $k = kCandidates[1]$.size
20: for all $candidatePair$ from $kCandidates$ do
21:    $newCandidate = candidatePair[1] \cup candidatePair[2]$
22:    if $newCandidate$.size == k + 1 then
23:       for all $ontologyCandidate \in mixmatch_{PM}(newCandidate)$ do
24:          if $\exists \mu \in F, \mu = (A_\mu, r_\mu, M_\mu) : ontologyCandidate \subseteq A_\mu$ then
25:             $(k+1)Candidates = (k+1)Candidates \cup \{newCandidate\}$
26:             if $cover(ontologyCandidate) \geq cover_0$ then
27:                $aggregations = aggregations \cup \{newCandidate\}$
28: $aggregations = aggregations \cup$ kStep($(k+1)Candidates$, PM, $F$, $cover_0$, $dist$)
29: return $aggregations$
```

all activity tuples in the model, which is exponential to the activity number. Due to the application of divide and conquer technique, see line 5, the size of the search space decreases rapidly. However, the search is still exponential to the number of activities within a graph distance *dist*.

5.2 Activity Aggregation as Cluster Analysis Problem

Alternatively, we interpret activity aggregation as a problem of cluster analysis. In this case we do not inspect the information external to the model, but study an activity environment within a process model. Examples of elements constituting such an environment are data objects accessed by activities and roles supporting activity execution, e.g., see model in Fig. 5.4. The list of such

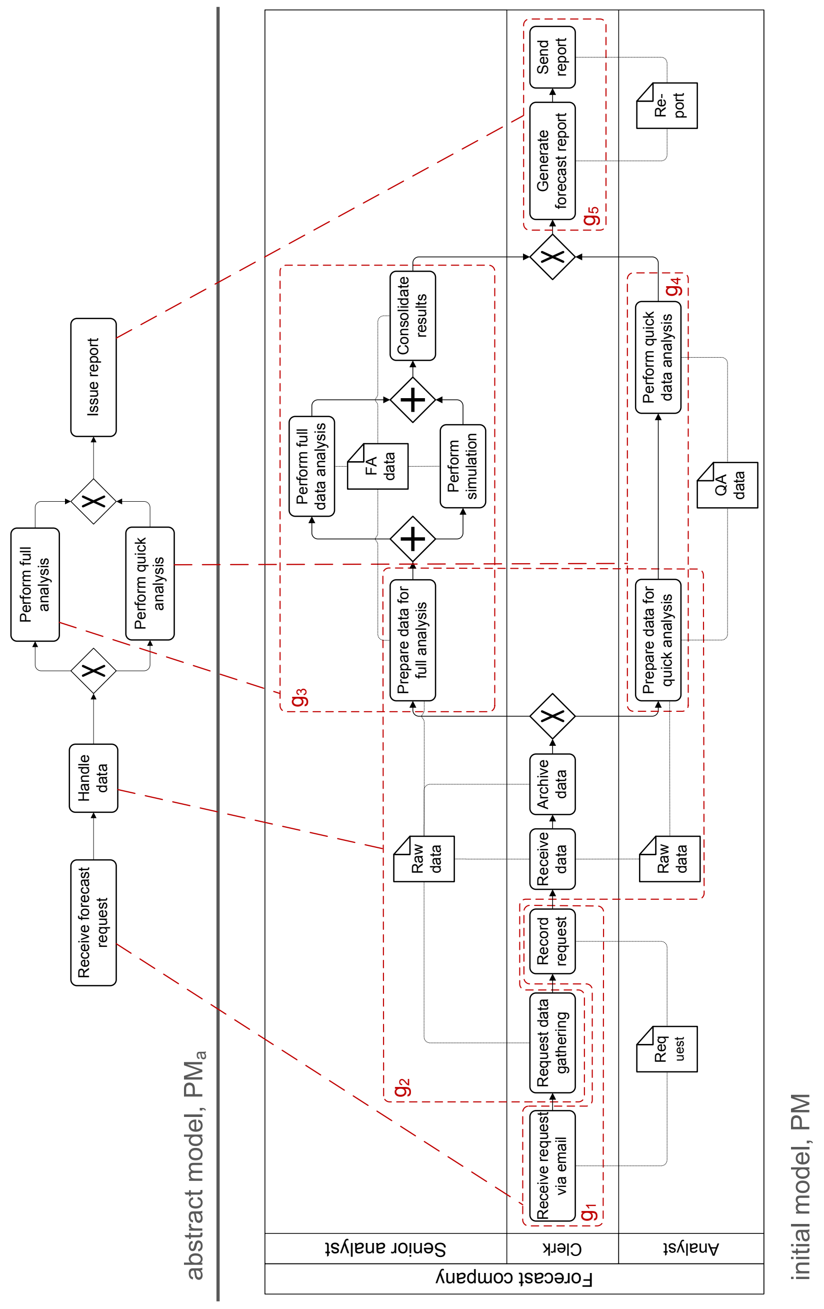


Fig. 5.4. Two models of the "Forecast request handling" business process at different levels of abstraction (BPMN notation).

model element types varies depending on the process modeling language, the tool at hand, modeling procedures taken into account, and the modeler's style. We formulate the clustering problem as follows. Let $\mathsf{PM} = (A, G, F, t, s, e)$ be a process model to be abstracted. The set of objects to be clustered is the set of activities A. The cluster analysis outcome, activity clusters, correspond to coarse-grained activities of the abstract process model.

The core contribution of this section is a method for activity aggregation based on cluster analysis. We argue how activity aggregation can be interpreted as the cluster analysis problem, choose a clustering algorithm that suits our context, and introduce a distance measure between the activities.

The remainder of this section is structured as follows. Section 5.2.1 introduces the notion of an annotated process model. Section 5.2.2 interprets activity aggregation as a cluster analysis problem.

5.2.1 Towards Annotated Process Model

In this section we design an activity aggregation neutral to control flow information, but considering other information contained in the model. To reason about this information formally, we introduce several auxiliary concepts and the notion of an annotated process model.

Definition 5.6 (Activity Property Value and Activity Property Type).

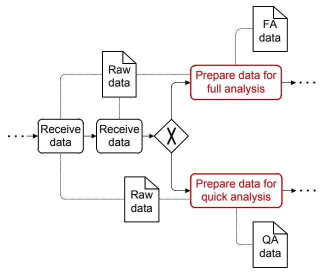
Let $\mathcal{V}$ be a finite nonempty set of activity property values. Alongside, $\mathcal{T}$ is a finite nonempty set of activity property types. Mapping $type : \mathcal{V} \to \mathcal{T}$ assigns a type to each value.

The process model in Fig. 5.4 illustrates Definition 5.6. *Raw data*, *FA data*, and *Analyst* are examples of activity property values. The process model presents two activity property types: *Role* and *Data object*. For instance, *type(Raw data) = Data object*, *type(FA data) = Data object*, and *type(Analyst) = Role*. Then, we define an *annotated process model* as follows.

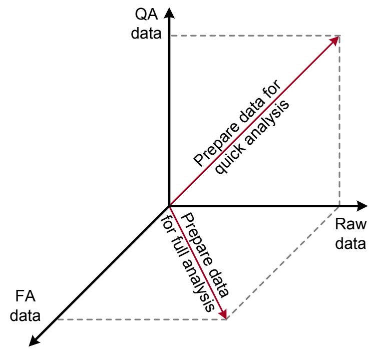
Definition 5.7 (Annotated Process Model).
A tuple $\mathsf{APM} = (A, G, F, t, s, e, V, props)$ is an *annotated process model,* where:

– (A, G, F, t, s, e) is a process model, see Definition 2.13
– $V \subseteq \mathcal{V}$ is a set of activity property values
– $props : A \to \mathcal{P}(V)$ is a mapping that assigns property values to an activity.

Mapping *props* assigns activity property values to model activities. Referring to model PM in the motivating example of Fig. 5.4, mapping *props* can be illustrated as *props(Collect data)* = {*Clerk, Raw data*}. Definitions 5.6 and 5.7 allow to manage the considered activity property types in a flexible fashion: it is enough to introduce a new activity property type to set $\mathcal{T}$, the values to set $\mathcal{V}$, and respectively update mapping *type*. Thereafter, new activity properties can be easily considered within the activity aggregation.


FA data
Raw data
Prepare data for full analysis
Receive data
Receive data
Raw data
Prepare data for quick analysis
QA data

(a) Excerpt of the process model in Fig. 5.4 emphasizing the activities and their properties of type "Data object".


QA data
Prepare data for quick analysis
Raw data
Prepare data for full analysis
FA data

(b) Representation of the activities as activity vectors in the vector space.

Fig. 5.5. A vector space with dimensions *FA data, QA data,* and *Raw data.*

5.2.2 Activity Clustering using K-means Algorithm

Cluster analysis provides a large variety of algorithms, e.g., see [137]. In this chapter we engineer activity aggregation motivated by the abstraction use cases in the group "Obtaining a Process Quick View". In the considered use cases, the user demands control over the number of activities in the abstract process model. For example, a practical guideline is that five to seven activities are displayed on each level in the process model [140]. Provided a fixed number the clustering algorithm has to assure that the number of clusters equals the request by the user. We turn to the use of k-means clustering algorithm, as it is simple to implement and typically exhibits good performance [72]. K-means clustering partitions an activity set into k clusters. The algorithm assigns an activity to the cluster, which centroid is the closest to this activity. To evaluate an activity distance, we analyze activity property values V. We foresee a number of alternative activity distance measures and elaborate on them in this section.

To introduce the distance measure among activities we represent activities as vectors in a vector space. Such an approach is inspired by the vector space model, an algebraic model widely used in information retrieval [136]. While this thesis makes use of the vector space model, activity clustering may profit from advanced models building on the basic idea of the vector space. Examples are the generalized vector space model, see [166], and enhanced topic vector space model, see [19], that allow to capture relations between activity properties. We leverage the vector space model, where the space dimensions correspond to activity property values V and the vector space can be captured as vector $(v_1, \dots, v_{|V|})$, where $v_j \in V$ for $j = 1, \dots, |V|$. Consider an example set of property values $V' = \{$*FA data, QA data, Raw data*$\}$ and the corresponding vector space presented in Fig. 5.5(b). The vector $\mathbf{a}$ representing an activity $a \in A$ in the annotated process model $\mathsf{APM} = (A, G, F, t, s, e, V, props)$ is constructed as follows. If activity a is associated with a property value $v_j \in V$, the corresponding vector dimension $\pi_j(\mathbf{a})$ has value 1; otherwise, the dimension $\pi_j(\mathbf{a})$ has value 0:

$$\pi_j(\mathbf{a}) = \begin{cases} 1, & \text{if } v_j \in props(a); \\ 0, & \text{otherwise.} \end{cases}$$

For the annotated process model APM in Fig. 5.4, activities *Prepare data for quick analysis* and *Prepare data for full analysis* correspond, respectively, to vectors $\mathbf{a_1} = (0, 1, 1)$ and $\mathbf{a_2} = (1, 0, 1)$ in the vector space with dimensions *FA data, QA data,* and *Raw data*, see Fig. 5.5.

Similarity of two vectors in the space is defined by the angle between these vectors: the larger the angle, the more distant the vectors are. Typically, the cosine of the angle between two vectors is used as a vector similarity measure. Let θ be the angle between the vectors $\mathbf{a_1}$ and $\mathbf{a_2}$. Then, the similarity of $\mathbf{a_1}$ and $\mathbf{a_2}$ is:

$$sim(a_1, a_2) = cos\theta = \frac{\mathbf{a_1} \cdot \mathbf{a_2}}{\|\mathbf{a_1}\| \, \|\mathbf{a_2}\|} \tag{5.1}$$

For instance, the similarity of activities *Prepare data for quick analysis* and *Prepare data for full analysis* is 0.5 according to Equation 5.1. The distance between two activities is:

$$dist(a_1, a_2) = 1 - sim(a_1, a_2) \tag{5.2}$$

By construction the vector dimension values are non-negative. Hence, the activity similarity and activity distance measures vary within the interval $[0, 1]$. According to Equation 5.2 we obtain the distance between activities *Prepare data for quick analysis* and *Prepare data for full analysis* to be 0.5.

We distinguish two types of vector spaces. On the one hand, a vector space can be formed by the dimensions corresponding to the activity property values disregard their type, i.e., all elements of V. We reference such spaces as *heterogeneous vector spaces.* An example of a heterogeneous vector space is a space with 6 dimensions *Analyst, Clerk, FA data, QA data, Raw data*, and *Senior analyst.* On the other hand, a vector space can be formed by the dimensions corresponding to the activity property values of a particular type. Given an activity property type t, such a space is formally defined by the set $V_t = \{\forall v \in V : type(v) = t\}$. We refer to such spaces as *homogeneous vector spaces.* Fig. 5.5(b) provides an example of a homogeneous vector space formed by activity properties of type *Data object.* We denote the activity distance in a heterogeneous space with $dist_h(a_1, a_2)$ and in a homogeneous vector space with $dist_t(a_1, a_2)$, where t is the respective activity property type. Both distance measures can be employed for activity aggregation. If the user wants to make use of one activity property type t only, the distance is defined by $dist_t$. To cluster activities according to several activity property types, $dist_h$ can be employed. In addition, we introduce an alternative distance measure $dist_{agg}$ that aggregates multiple homogeneous distance measures $dist_t$:

$$dist_{agg}(a_1, a_2) = \frac{1}{|T|} \sum_{\forall t \in T} w_t \cdot dist_t(a_1, a_2) \qquad (5.3)$$

In Equation 5.3, the set T corresponds to the activity property types that appear in an annotated process model. Then, function $dist_{agg}$ is the weighted average value of distance measures in the vector spaces corresponding to the available activity property types. Coefficient w_t is the weight of $dist_t$ indicating the impact of the activity distance according to property type t. We reference all the weights in Equation 5.3 as $\mathbf{W} = (w_{t_1}, \ldots, w_{t_n})$, where $n = |T|$. In the remainder of this section we will explain the role of vector $\mathbf{W}$.

The application of different abstraction operations to one process model leads to various abstract representations of the modeled business process. The differences between abstraction operations are explained by their pragmatics, i.e., various abstraction purposes. If the abstraction is realized by a human, the modeling habits of the designer are reflected in the abstraction operation as well. Hence, abstraction pragmatics and modeling habits of the designer are inherent properties of the abstraction operation and together form an *abstraction style*. We use vector $\mathbf{W}$ in Equation 5.3 to model an abstraction style.

From the user perspective vector $\mathbf{W}$ is the tool to express the desired abstraction style. We foresee two scenarios how vector $\mathbf{W}$ can be obtained. In the first scenario, the user explicitly specifies $\mathbf{W}$. This approach is useful if the user wants to introduce a new abstraction style. However, coming up with an appropriate value for $\mathbf{W}$ may be challenging. Hence, the second scenario implies that vector $\mathbf{W}$ is mined from a process model collection enriched with subprocess relation. The discovered vector is a “fingerprint” of the process model collection with respect to the used abstraction style. An approach to discover a fingerprint for a process model collection is elaborated and validated in [145].

5.3 Evaluation

The presented methods for activity aggregation call for an evaluation. This section empirically evaluates the advocated activity aggregation methods by means of industrial process models. We start defining the evaluation goal and the sketching the evaluation method in Section 5.3.1. Then, Section 5.3.2 evaluates meronymy-based activity aggregation. Subsequently, Section 5.3.3 presents an evaluation of aggregation based on cluster analysis. We conclude reflecting on the key properties of the two activity aggregation methods in Section 5.3.4.

5.3.1 Goal and Method

The goal of the evaluation is to estimate the quality of activity aggregation methods developed in Section 5.1 and Section 5.2. To achieve this goal we compare the activity aggregations delivered by the advocated methods (retrieved activity aggregations) against the aggregations specified by humans (relevant activity aggregations). As quantitative measures we use the standard notions of recall, precision, and F-score [16]. A precision indicates the share of retrieved aggregations that are relevant, while a recall is the fraction of relevant aggregations that are retrieved. An F-score is the harmonic mean of the precision and the recall.

Precision, recall, and F-score assume the comparison of retrieved and relevant activity aggregations. The reader may notice that activity aggregations are of different size, often containing more than two activities. This fact complicates the comparison of activity aggregations. Consider an example of two activity aggregations, one of size five and the other of size six. The two aggregations share four common activities. If we compare the two aggregations as sets, we learn that the aggregations are unequal. Meanwhile, the two activity aggregations are very similar, as they have several activities in common. To consider the intersection of activity aggregations, we compare activity pairs only. Given an activity aggregation we decompose it into a set of all its subsets of size two. In this context the comparison of two activity aggregations turns into comparison of two sets of activity pairs. Consider, for instance, the set $\{a, b, c\}$, where activity c is weakly related to a and b. This set can be decomposed into pairs $\{a, b\}$, $\{a, c\}$, $\{b, c\}$. The fact that c is weakly related to a and b can be easily pointed out by claiming (a, c) and (b, c) irrelevant. Against this background, we evaluate the similarity of two activity aggregation sets comparing each pair from the first activity pair set with each pair in the second activity pair set. We claim that two pairs coincide if they contain the same elements. Further we evaluate both activity aggregation methods against industrial process model collections.

5.3.2 Meronymy-Based Activity Aggregation

This section focuses on the evaluation of meronymy-based activity aggregation. We start by introducing the experiment setting and then present the observed results.

Experiment Setting

We evaluated the meronymy-based activity aggregation by applying it to a set of models capturing the business processes of a large electronics manufacturer. The model collection considered in the experiment includes 6 business process models. Each model contains on average 42 activities, with a minimum of

18 activities and a maximum of 81 activities. On average, an activity label contains 4.1 words.

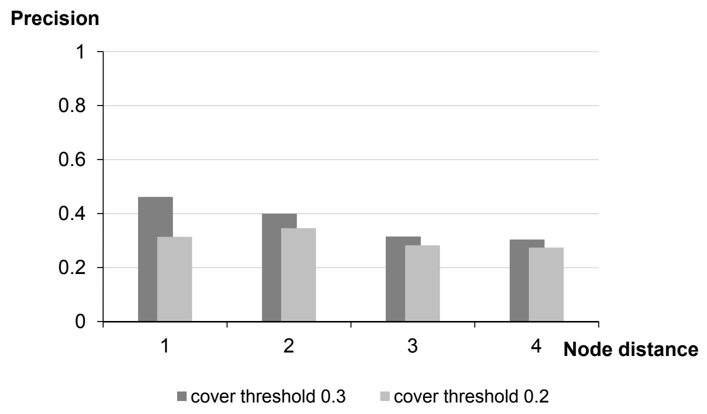
A meronymy forest is represented by the MIT Process Handbook [95]. The MIT Process Handbook describes business processes elicited by researchers in the interviews with business process experts. It spans several business domains, like sales, distribution, and production. The handbook describes about 5 000 activities and specifies hyponymy and meronymy relations between them. We make use of activities and a meronymy relation only. The process models were not aligned with the Handbook in advance: no relations between process model activities and the MIT Process Handbook activities were established. We matched process model activities to the activities of the Handbook according to the semantics of their labels, as discussed in Section 5.1.2.

To obtain relevant activity aggregations we asked a process modeling expert from TU Eindhoven, who was unfamiliar with the technique, to select sets of related activities within the model collection. We gave the instruction to consider an activity aggregation relevant, if a given set of activities could be reasonably "abstracted" into a single activity. The means for abstraction that could be considered were: aggregating the activities in the abstraction, generalizing the activities, putting the activities into a common subprocess, or any other means that the evaluator considered relevant.

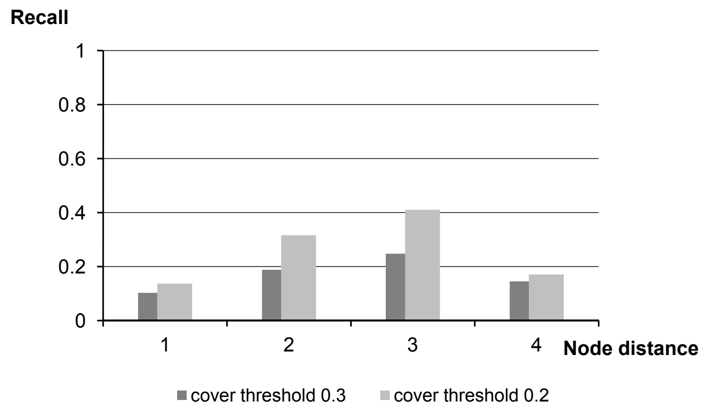
Observed Results

We have conducted a series of experiments in which we varied the parameters of our aggregation technique. In each run of the experiments we have fixed the parameters of *match* function (each process model activity was mapped to at most 10 activities in the Handbook). At the same time we varied the node distance and *cover* threshold value. The node distance runs the values from 1 to 4, while the *cover* threshold values were 0.2 and 0.3. Within the experiment we observed the precision, recall, and F-score values.

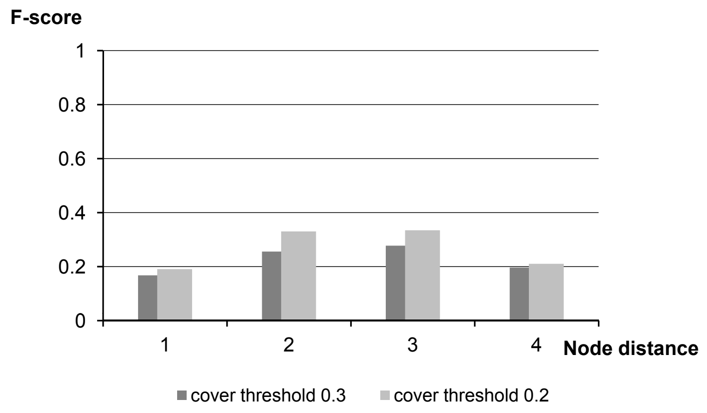
Fig. 5.6 illustrates the observed results. The precision value varies between 0.27 (the node distance equals to 4 and the cover threshold value is of 0.2) and 0.46 (the node distance equals to 1 and the cover threshold value is of 0.3), see Fig. 5.6(a). One can see two tendencies in the experiment. First, a higher $cover_0$ threshold value leads to a higher precision. Indeed, a high threshold prunes more aggregation candidates, as it imposes more strict search conditions. The total number of aggregations declines, increasing the precision. Second, the increase of node distance leads to a precision decrease. This observation can be explained by the fact that a node distance increase brings more activities into algorithm's consideration. As [129] argues the greater the distance is, the less related activities appear in the set. Thereby, the precision decrease is expected. The recall value varies between 0.10 and 0.41, but behaves differently, see Fig. 5.6(b). First, we observe that the higher $cover_0$ threshold value leads to a lower recall. As a high $cover_0$ value signifies a strict selection of activity aggregations, the probability that a relevant aggregation is



(a) Variation of activity aggregation precision with respect to the selected node distance



(b) Variation of activity aggregation recall with respect to the selected node distance



(c) Variation of activity aggregation F-score with respect to the selected node distance

Fig. 5.6. Precision, recall, and F-score observed within the evaluation of meronymy-based activity aggregation.

retrieved gets low. Meanwhile, the recall grows and reaches its local maximum at the value of node distance of 3. We explain this phenomenon as follows: until the node distance value reaches the value of 3, the number of retrieved aggregations increases, along with the number of relevant aggregations. However, at the value of 4 the overall number of aggregations increases while the share of relevant aggregations decreases. Finally, the F-score is presented in Fig. 5.6(c). The F-score varies between 0.17 and 0.33 and summarizes the co-evolution of recall and precision.

While the technique returns a considerable amount of helpful suggestions, there is still quite a number of irrelevant aggregations proposed. Hence, we aim to improve the technique precision. Further, the conducted experiment evaluated both aggregation mining algorithm and *match* function. Since the process models and the used ontology were not aligned beforehand, there is also a contribution to a gap in precision by the *match* technique. To study the behavior of aggregation mining algorithm further, we need a setting, where process models are created using activities from a domain ontology. We perceive such an evaluation as the future work.

5.3.3 Activity Aggregation based on Cluster Analysis

Section 5.2 has demonstrated how cluster analysis enables activity aggregation. However, the practical utility of the proposed solutions has not been estimated. This section evaluates the usefulness of the designed activity aggregation based on K-means clustering. We perform an empirical evaluation of the approach by conducting an experiment with a real world business process model collection. This section describes the evaluation experiment setting, and presents the observed results.

Experiment Setting

We evaluate the activity aggregation based on cluster analysis by means of a set of business process models from a large telecommunication service provider. This organization is currently in the process of setting up a repository with high-quality process models, which are brought together for the purpose of consultation and re-use by business users. The model set includes 48 elaborate models, enriched with activity properties of the following two types *roles* and *data objects*. Notice that the models used for evaluation in the previous could not boast such an elaborate modeling approach. In addition to these non-control flow types of information, we also study the impact that activity labels have on the decision to aggregate activities into the same subprocess. To compare activities with respect to their labels, the corresponding vector space is formed by the words that appear in the labels. Against this background, finding the distance between activities becomes an information retrieval task as labels can be treated as documents in information retrieval. Table 5.1 outlines the relevant properties of the process models. In the model

	Nodes	**Activities**	**Role**	**Data object**
Average	15.5	6.3	0.76	0.76
Minimum	5	1	0	0
Maximum	48	20	2	17

Table 5.1. Properties of business process models used in the evaluation.

set we have identified 28 models where activities are refined by a subprocess. Within the experiment we consider each subprocess as a relevant activity aggregation.

Observed Results

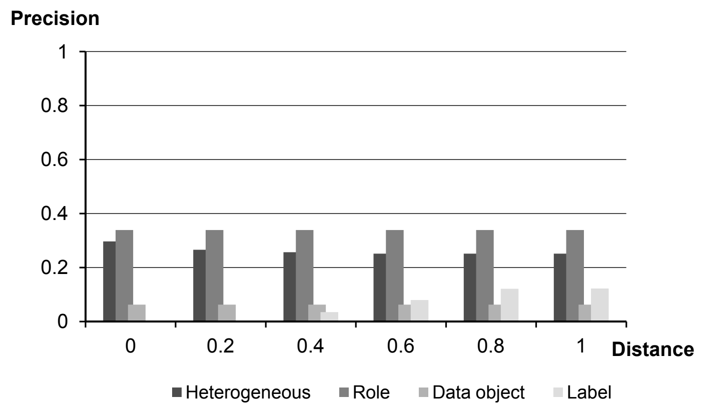
Within the evaluation we have varied two parameters: the distance value and the vector space type. The distance value varies between 0 and 1 with the step of 0.2. As the vector space types we have explored homogeneous spaces with types *Role*, *Label*, and *Data object*, as well as the heterogeneous vector space. The obtained precision, recall, and F-score values are plotted in Fig. 5.7. The precision value varies between 0.00 and 0.34, the recall is between 0.00 and 0.86, and the F-score varies from 0.00 to 0.40. First, we notice that the values moderately depend on the distance value. However, the graphs vividly illustrate the large difference in the performance of clustering in heterogeneous vector space and homogeneous vector space with type *role* on the one hand, and the homogeneous vector spaces with types *label* and *data object* on the other hand. In particular, the recall of aggregation according to the heterogeneous space stands out: it lies between 0.64 and 0.86. Meanwhile, the recall value in a data object vector space is 0. Finally, we mention that the activity aggregation based on clustering demonstrates a significant difference between the precision and recall values, where recall excels precision.

Another look at the obtained results show the advantages of heterogeneous vector space and a homogeneous space of type “Role”. We attribute this phenomenon to two facts. First, the model designers consider information about roles when they identify subprocesses. Second, a high performance of the heterogeneous vector space can be explained by the fact that it encapsulates information about roles.

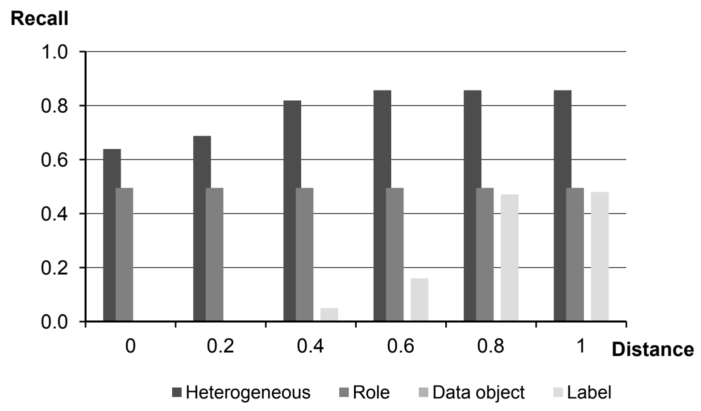
5.3.4 Key Observations

To summarize the evaluation of the activity aggregation we conclude with several key observations:

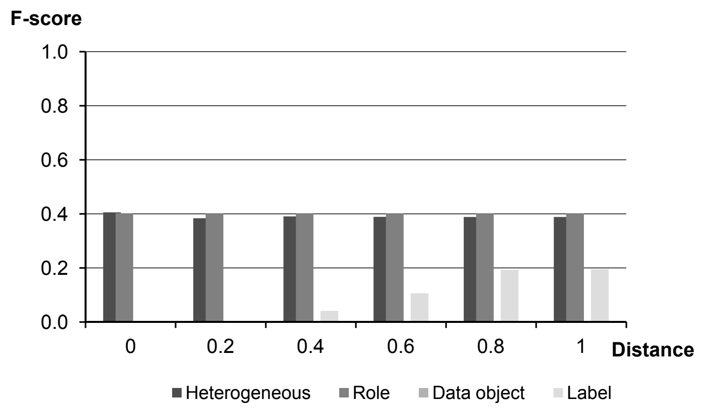
Recall vs. precision First, we compare the obtained recall and precision values. In the case of meronymy-based activity aggregation we see there is a tendency that the precision exceeds the recall. In contrast, for activity



(a) Variation of activity aggregation precision with respect to the distance



(b) Variation of activity aggregation recall with respect to the distance



(c) Variation of activity aggregation F-score with respect to the distance

Fig. 5.7. Precision, recall, and F-score observed within the evaluation of activity aggregation based on cluster analysis.

aggregation based on clustering we notice that the recall dominates the precision.

F-score We continue contrasting the F-score of the two activity aggregation methods. The F-score of meronymy-based activity aggregation is inconsistent: it varies between 0.17 and 0.34. At the same time, activity aggregation based on clustering exhibits high consistency: its F-score fluctuates around the value of 0.40. We conclude that with respect to the F-score values there is a slight advantage of the clustering method.

Possible applications The two activity aggregation methods can be differentiated with respect to their precision and recall. If the application context implies that the recall is more significant, activity aggregation based on clustering is preferable. If the application requires high precision, meronymy-based aggregation outperforms aggregation based on clustering. Finally, the obtained precision, recall, and F-score values witness that the developed aggregation methods can be barely used in a fully automatic fashion. Against this background, we propose to use the methods to support the user with suggestions on activity aggregations.

5.4 Discussion

This chapter addressed the challenge of finding groups of related activities in process models. The two developed methods for activity aggregation are independent of the control flow information. Against this background, both methods expect information-rich business process models as the input. Notice that this implies elaborate business process modeling. However, both designed activity aggregation methods are free of the limitations inherent to abstraction methods based on the process model structure. For instance, the meronymy-based activity aggregation explicitly enables non-hierarchical activity aggregation. The advocated activity clustering approach results in a hierarchical activity aggregation. Indeed, as k-means clustering partitions the observation into k clusters, the activity aggregation that builds on clustering approach partitions the activity set of an initial process model. Hence, no activity of the initial model PM is assigned to several clusters corresponding to coarse-grained activities of PM_a. However, as cluster analysis exhibits numerous algorithms, non-hierarchical aggregation can be realized as well. One possible direction is fuzzy clustering, where fuzzy c-means algorithm, see [25] can be seen as an analogue of K-means clustering.

Both presented activity aggregation methods equip the user with efficient means to control the abstraction. Meronymy-based aggregation can be used to provide the user with suggestions on which activity groups can be aggregated [143]. Potentially, the system can also recommend the name of an aggregating activity derived from the ontology. Activity aggregation based on k-means clustering provides the user a direct control over of the abstraction level in a process model: specifying the value of k the user changes the number of process model activities.

The practical applicability of both methods depends on their running time. In general case the k-means clustering problem is NP-hard. However, various heuristics allow to devise an efficient solution. Similarly, the meronymy-based aggregation explores the power set of a model activity set. However, we suggested heuristics that decrease the state space.

5.5 Summary

This chapter has engineered two methods for activity aggregation. In contrast to the existing abstraction approaches, both methods focus on the business meaning of the aggregated activities and aim to deliver activity groups that have a self-contained business semantics. In this way each obtained activity group can be related to a coarse-grained activity of the abstract process model. Since control flow information does not suffice for meaningful activity aggregation, both methods study additional information. The first method considers domain ontologies enriched with activity meronymy relation, while the second method analyzes non-control flow model elements, like roles and data objects. Within the framework of meronymy-based activity aggregation we have proposed a novel metric enabling activity set comparison and an algorithm for selecting strongly related activity sets. The analysis of process model non-control flow elements adapts the well-established algorithm of cluster analysis and the model of information retrieval. We have supported the conceptual discussion by an empirical evidence witnessing that the designed aggregation approaches are applicable in a real world setting. Finally, we have compared the properties of the developed aggregation methods relevant in an application context. As soon as the developed activity aggregation methods take into account the semantics of model elements, the chapter addresses the *why* of abstraction. In addition, the methods suggest ways to aggregate activities contributing to the *how* aspect.

The focus of this chapter is on methods increasing the coarse-granularity of process model activities. Hence, the proposed solutions allow to synthesize the set of activities for the abstract process model PM_a. However, PM_a lacks the control flow relation. To complete business process model abstraction, we lack a method for the discovery of the ordering constraints in PM_a. The upcoming Chapter 6 advocates an advanced method for identification of ordering constraints in the abstract process model that complements the findings of this chapter.

6

Controlling Control Flow Loss

The desired effect of business process model abstraction is information loss [30, 120, 151]. Each abstraction use case has an implication on the kind of information allowed to be lost. Following the user demand, see Section 3.4, we concentrate on the abstraction use cases in the group "Group 4: Obtaining a Process Quick View". These use cases motivate two types of information loss. First, they require process models with more coarse-grained activities, i.e., the details about activities are lost. Second, the use cases demand models with the overall ordering constraints between activities, i.e., the precise ordering constraints of the initial model are to be "generalized". The former type of information loss has been addressed by Chapter 4 and Chapter 5: the chapters argued how to arrive at coarse-grained activities by means of aggregation. The latter type of information loss has been discussed in Chapter 4. However, the solution relied on the strong assumptions. This chapter revises the problem of abstracting the process ordering constraints and proposes a novel solution. In this way the chapter concentrates on the *how* aspect of abstraction (for details see the framework introduced in Section 3.2). This introduction revises the abstraction problem, argues about the drawbacks of the methods in place, and sketches the contribution of this chapter.

We revisit the problem describing it is a black box with inputs and outputs and specifying the desired properties of the transformation. Naturally, the problem input is the process model to be abstracted. In addition, we know methods telling which activities belong together within the abstracted model, see Chapters 4–5. Hence, groups of related activities become the second component of the input. The origin of these groups is out of scope of this chapter. They can be manually created by the user, discovered according to the meronymy-based activity aggregation, see Section 5.1, or obtained through activity clustering, see Section 5.2. The abstraction output is a process model with coarse-grained activities and their ordering constraints. Fig. 6.2 visualizes the problem's inputs and outputs.

Following the idea of the abstraction use cases, every coarse-grained activity in the output model corresponds to a group of detailed activities in

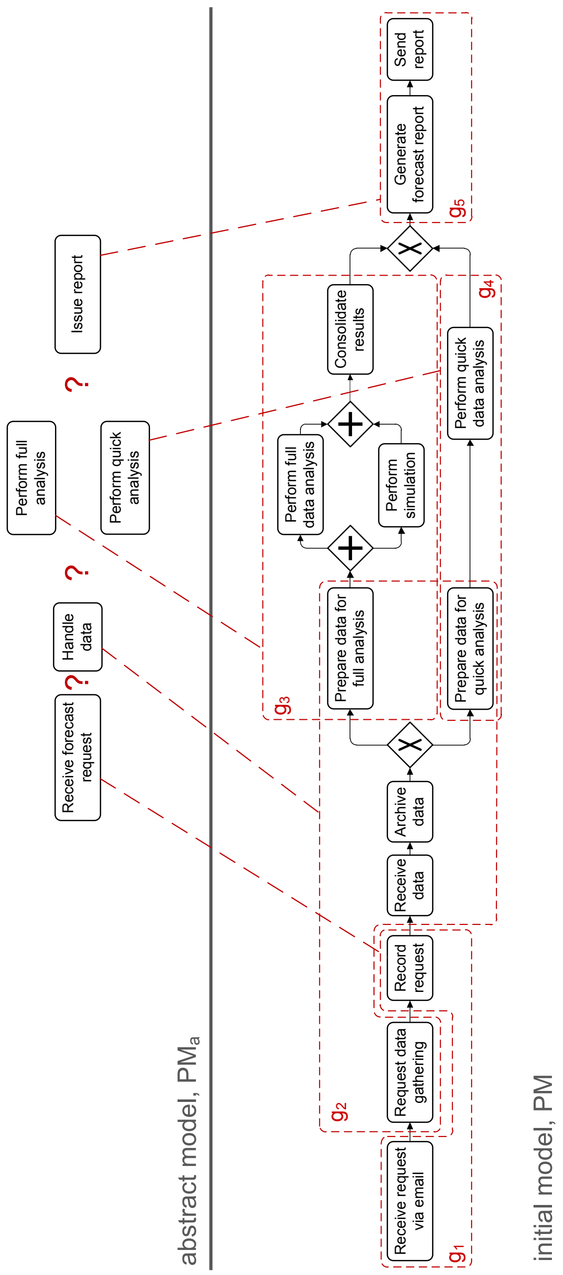


Fig. 6.1. Two descriptions of the "Forecast request handling" business process. Model PM describes the process in detail. Partial model PM_a defines coarse-grained activities, but lacks the control flow. Each activity of PM_a is refined by an activity group in PM. Notice that the activity refinement is not hierarchical.

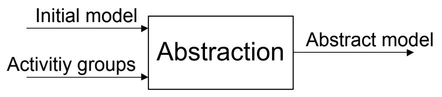


Fig. 6.2. Abstraction as a block box.

the input model. Unfortunately, the related research, e.g., [32, 55, 94, 123], as well as Chapter 4, puts strong assumptions on such activity groups. The first assumption is that abstraction is hierarchical: every activity of the input model is assigned to exactly one activity group, see Section 3.3.1. The second assumption is that related activities always belong to one process model fragment. Two observations motivate us to discard both of these assumptions. On the one hand, the interviews with practitioners indicate that these assumptions are too strong in practice, see Section 3.4. Upon the other hand, the research on software engineering, as well as on business process modeling, [60, 82, 110, 157] motivates the demand for abstraction free of these limitations.

The example in Fig. 6.1 motivates why the two aforementioned assumptions are too restrictive. The figure presents two descriptions of the business process "Forecast request handling". Model PM specifies fine-grained activities and the control flow relation. Specification PM_a strictly speaking is not a model: it contains coarse-grained activities, but no ordering constraints. Fig. 6.1 also visualizes the activity groups in PM and relates them to activities in PM_a. We notice that activity groups of PM make sense from a business perspective, yet go beyond the borders of both limitations. For instance, activity *Prepare data for quick analysis* belongs to groups g_2 and g_3, i.e., the abstraction is not hierarchical. The group g_1 contains activities *Receive request via email* and *Record request* split by activity *Request data gathering*, i.e., one activity group does not constitute a process model fragment. Fig. 6.1 once more illustrates the abstraction setting. While precise process ordering constraints and coarse-grained activities are known, the control flow relation of the abstract model is missing. This is reflected by the partial model PM_a that lacks the control flow. To the best of our knowledge, none of the existing abstraction methods allows to derive the ordering constraints between the activities of PM_a given the specified activity grouping.

The contribution of this chapter is a business process model abstraction overcoming the aforementioned limitations and capable of delivering the abstract model control flow relation in the described setting. The abstraction in question allows such activity groups that:

Property 1 one activity belongs to several activity groups
Property 2 activity groups are distributed over the process model in an arbitrary fashion.

Inspecting the existing abstraction methods we admit that they directly study process model structure, i.e., the model control flow relation. Since all these methods suffer from the discussed limitations, we conclude that the control flow relation is too strict to support an abstraction, where Properties 1 and 2 hold. Thereby, we develop an alternative abstraction approach leveraging a process behavioral abstraction—*behavioral profiles*, see Section 2.3.3. Behavioral profiles describe the process ordering constraints, yet in a less detailed way than the process model control flow relation. The use of behavioral profiles helps us to deliver an abstraction featuring Properties 1 and 2. The proposed method contains the following four steps:

Step 1 derive the behavioral profile $\mathsf{BP}_{\mathsf{PM}}$ for model PM
Step 2 construct the behavioral profile $\mathsf{BP}_{\mathsf{PM}_a}$ for model PM_a
Step 3 **if** the behavioral profile $\mathsf{BP}_{\mathsf{PM}_a}$ is well-structured
Step 4 **then** synthesize PM_a, **else** report a not well-structuredness.

The chapter not only provides a conceptual discussion of the abstraction approach, but also its implementation. We briefly outline Flexab—the tool enabling business process model abstraction based on behavioral profiles.

The structure of this chapter aligns to a large extend with the above mentioned steps. Section 6.1 argues how to construct a behavioral profile for a process model. Section 6.2 shows how an abstract model behavioral profile is derived, while Section 6.3 elaborates on abstract model synthesis. The software implementation of the approach is presented in Section 6.4. Section 6.5 discusses the practical and theoretical properties of the proposed solution. Finally, Section 6.6 summarizes the chapter.

6.1 Deriving Behavioral Relations from a Process Model

There exist several methods for construction of a behavioral profile describing the process behavior. Each of these methods implies a system description by means of a Petri net. A generic approach enabling behavioral profile derivation for a Petri net relies on the computation of a net unfolding [57, 99], which is NP-complete [161]. An efficient algorithm has been proposed for the class of sound free-choice WF-nets [162]. The algorithm deduces the behavioral profile relations from the WF-net structure. It enables the derivation of the behavioral profile in $O(n^3)$ time with n being the number of nodes of the WF-net. In this thesis we operate with process models that can be mapped to sound free-choice WF-nets. Hence, we are able to reuse techniques for the derivation of behavioral profiles introduced for this model class. Leveraging a behavioral profile construction method introduced in [162] we derive a behavioral profile for model PM in Fig. 6.1, see Table 6.1. For the sake of compact representation we abbreviate the names of activities using the first letters of each word in the label, e.g., *Perform full data analysis* is acronymed to *PFDA*.

	RE	*RDG*	*RR*	*RD*	*AD*	*PDFA*	*PFDA*	*PS*	*CR*	*PDQA*	*PQDA*	*GFR*	*SR*
RE	$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
RDG		$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
RR			$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
RD				$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
AD					$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
PDFA						$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$+_{\mathsf{PM}}$	$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
PFDA							$+_{\mathsf{PM}}$	$\|_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$+_{\mathsf{PM}}$	$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
PS								$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$+_{\mathsf{PM}}$	$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
CR									$+_{\mathsf{PM}}$	$+_{\mathsf{PM}}$	$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
PDQA										$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
PQDA											$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
GFR												$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
SR													$+_{\mathsf{PM}}$

Table 6.1. The behavioral profile of model PM in Fig. 6.1.

6.2 Construction of Abstract Model Behavioral Profile

Every high-level activity in $\mathsf{PM}_a = (A_a, G_a, F_a, t_a, s_a, e_a)$ is the result of aggregation of several activities in $\mathsf{PM} = (A, G, F, t, s, e)$. We formalize the construction of coarse-grained activities by the function *aggregate*, see Definition 3.11. The function *aggregate* can be illustrated by the example in Fig. 6.1, where *aggregate(Perform quick analysis)* = {*Prepare data for quick analysis, Perform quick data analysis*} and *aggregate(Handle data)*={*Collect data, Archive data, Prepare data for full analysis, Prepare data for quick analysis*}. The behavioral profile of model PM_a defines the relations between each pair of activities in PM_a. To discover the behavioral profile for PM_a we analyze the relations among activities in PM and consider the function *aggregate*. For each pair of coarse-grained activities x, y, where $x, y \in A_a$, we study the relations between a and b, where $(a, b) \in aggregate(x) \times aggregate(y)$. This study reveals a dominating behavioral relation between elements of $aggregate(x)$

Algorithm 4 Derivation of a behavioral relation for an activity pair

```
1: deriveBehavioralRelation(Activity $x$, Activity $y$, Double $w_t$)
2: $w(x \succ_{\mathsf{PM}_a} y) = |\{\forall(a,b) \in aggregate(x) \times aggregate(y) : a \rightsquigarrow_{\mathsf{PM}} b \vee a||_{\mathsf{PM}} b\}|$
3: $w(y \succ_{\mathsf{PM}_a} x) = |\{\forall(a,b) \in aggregate(x) \times aggregate(y) : a \rightsquigarrow^{-1}_{\mathsf{PM}} b \vee a||_{\mathsf{PM}} b\}|$
4: $w(x \not\succ_{\mathsf{PM}_a} y) = |\{\forall(a,b) \in aggregate(x) \times aggregate(y) : a \rightsquigarrow^{-1}_{\mathsf{PM}} b \vee a +_{\mathsf{PM}} b\}|$
5: $w(y \not\succ_{\mathsf{PM}_a} x) = |\{\forall(a,b) \in aggregate(x) \times aggregate(y) : a \rightsquigarrow_{\mathsf{PM}} b \vee a +_{\mathsf{PM}} b\}|$
6: $w_{prod} = |aggregate(x)| \cdot |aggregate(y)|$
7: $w(x +_{\mathsf{PM}_a} y) = \frac{min(w(x \not\succ_{\mathsf{PM}_a} y), w(y \not\succ_{\mathsf{PM}_a} x))}{w_{prod}}$
8: $w(x \rightsquigarrow_{\mathsf{PM}_a} y) = \frac{min(w(x \succ_{\mathsf{PM}_a} y), w(y \not\succ_{\mathsf{PM}_a} x))}{w_{prod}}$
9: $w(x \rightsquigarrow^{-1}_{\mathsf{PM}_a} y) = \frac{min(w(y \succ_{\mathsf{PM}_a} x), w(x \not\succ_{\mathsf{PM}_a} y))}{w_{prod}}$
10: $w(x||_{\mathsf{PM}_a} y) = \frac{min(w(x \succ_{\mathsf{PM}_a} y), w(y \succ_{\mathsf{PM}_a} x))}{w_{prod}}$
11: if $w(x +_{\mathsf{PM}_a} y) \geq w_t$ then
12:     return $x +_{\mathsf{PM}_a} y$
13: if $w(x \rightsquigarrow_{\mathsf{PM}_a} y) \geq w_t$ then
14:     if $w(x \rightsquigarrow^{-1}_{\mathsf{PM}_a} y) > w(x \rightsquigarrow_{\mathsf{PM}_a} y)$ then
15:         return $x \rightsquigarrow^{-1}_{\mathsf{PM}_a} y$
16:     else
17:         return $x \rightsquigarrow_{\mathsf{PM}_a} y$
18: if $w(x \rightsquigarrow^{-1}_{\mathsf{PM}_a} y) \geq w_t$ then
19:     return $x \rightsquigarrow^{-1}_{\mathsf{PM}_a} y$
20: return $x||_{\mathsf{PM}_a} y$
```

and $aggregate(y)$. We assume that the behavioral relations between activity pairs of PM_a can be discovered independently from each other, i.e., the relation between x and y, where $x, y \in A_a$ depends on the relations between activities in $aggregate(x)$ and $aggregate(y)$, but does not depend on the relations between $aggregate(x)$ and $aggregate(z), \forall z \in A_a$, where $z \neq x$ and $z \neq y$.

Algorithm 4 formalizes the derivation of behavioral relations. The input of the algorithm is a pair of activities, x and y, and w_t—the user-specified threshold telling significant relation weights from the rest and, hence, managing the ordering constraints loss. The output of the algorithm is the behavioral profile relation between x and y. Algorithm 4 derives behavioral profile relations between x and y from the frequencies of relations between activities a and b, where $(a,b) \in aggregate(x) \times aggregate(y)$. According to Definition 2.23, each of the behavioral profile relations is specified by the corresponding weak order relations. Thereby, to conclude about the behavioral profile relation between x and y, we first evaluate the frequencies of weak order relations for x and y. The latter are found in the assumption that each weak order relation holding for $(a,b) \in aggregate(x) \times aggregate(y)$, contributes to the weak order relation between x and y. This rationale helps to find the weight for each weak order relation between x and y (lines 2–5). The overall number of relations is stored in variable w_{prod} (line 6). Algorithm 4 continues finding the

	RFR	HD	PFA	PQA	IR
RFR	$+_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$
HD		$+_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$
PFA			$+_{\mathsf{PM}_a}$	$+_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$
PQA				$+_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$
IR					$+_{\mathsf{PM}_a}$

Table 6.2. The behavioral profile of PM_a constructed given model PM and function *aggregate* as informally defined in Fig. 6.1. The used weight threshold is $w_t = 0.5$.

relative weight for each behavioral profile relation (lines 7–10). The relative weights of behavioral relations together with the relation hierarchy are used to choose the dominating relation (lines 11–20). The behavioral relations are ranked according to their relative weights. Threshold w_t selects significant relations, omitting those for which the relative weights are less than w_t. Finally, the relation hierarchy allows us to choose the strictest relation among the significant ones. The input parameter w_t implements the slider concept: using w_t a user expresses the preferred ordering constraint loss level to obtain the respective behavioral relations for model PM_a.

To illustrate Algorithm 4 we refer to the example in Fig. 6.1 and derive the behavioral relations between activities of model PM_a. As before, we acronym the names of activities. Assuming the threshold $w_t = 0.5$, abstraction results in a behavioral profile presented in Table 6.2. As relations of the behavioral profile are derived independently, we illustrate the construction of the behavioral profile in Table 6.2 looking at one activity pair. We elaborate on the derivation of a behavioral relation for activities *Handle data* (HD) and *Perform quick analysis* (PQA). Following Algorithm 4, $w(HD \succ_{\mathsf{PM}_a} PQA) = 6$, $w(PQA \succ_{\mathsf{PM}_a} HD) = 0$, $w(HD \not\succ_{\mathsf{PM}_a} PQA) = 4$, $w(PQA \not\succ_{\mathsf{PM}_a} HD) = 10$, and $w_{prod} = 10$. Then, $w(HD +_{\mathsf{PM}_a} PQA) = 0.4$, $w(HD \rightsquigarrow_{\mathsf{PM}_a} PQA) = 0.6$, $w(HD \rightsquigarrow^{-1}_{\mathsf{PM}_a} PQA) = 0$, and $w(HD ||_{\mathsf{PM}_a} PQA) = 0$. The constellation of behavioral relation weights is shown in Fig. 6.3. Each relation weight w_r defines a segment $[0, w_r]$, where the respective behavioral relation r is valid. If the maximum weight of the relations w_{max} is less than 1, we claim that the interleaving order relation is valid in segment $[w_{max}, 1]$ (it provides most freedom in execution of two activities). While the resulting segments overlap, the relation hierarchy defines the dominating relation in a particular point of $[0, 1]$. For $w(HD \rightsquigarrow_{\mathsf{PM}_a} PQA) \geq 0.5$ we state *Handle data* $\rightsquigarrow_{\mathsf{PM}_a}$ *Perform quick analysis* according to the behavioral relation hierarchy.

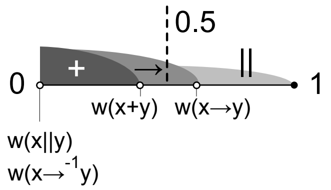


Fig. 6.3. Discovery of a behavioral relation for an activity pair *HD* and *PQA* of model PM_a Fig. 6.1. The weights of behavioral relations are evaluated according to the Algorithm 4, assuming $w_t = 0.5$.

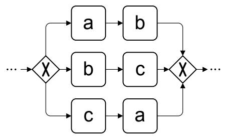
The Algorithm 4 terminates: it iterates over finite sets $aggregate(x)$ and $aggregate(y)$ and then compares the discovered relations. Given a pair of activities x and y in the abstract model PM_a, the time complexity of the Algorithm 4 is $O(k \cdot l)$, where $k = |aggregate(x)|$ and $l = |aggregate(y)|$. To construct the behavioral profile of model PM_a we need to derive the behavioral relations for each pair of activities in PM_a. Thereby, we need to investigate $\frac{|A_a|^2}{2}$ relations.

6.3 Abstract Process Model Synthesis

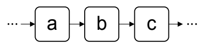
The creation of the behavioral profile as introduced in Section 6.2 might yield a profile for which we cannot generate a process model. We use the notion of a *well-structured* behavioral profile to distinguish a class of behavioral profiles for which we construct a process model. This section introduces the notion of a well-structured behavioral profile before we target process model synthesis.

6.3.1 Well-Structured Behavioral Profiles

The notion of a well-structured behavioral profile is coupled with the existence of a process model that satisfies the profile constraints. Whether such a process model exists depends on the applied notion of a process model and

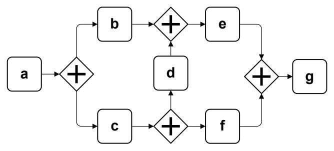


(a) The fragment fulfills the constraints $a \rightsquigarrow b$, $b \rightsquigarrow c$, and $c \rightsquigarrow a$ at the expense of activity duplication.

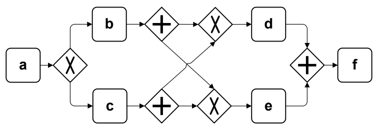


(b) The fragment contains exactly one occurrence of a, b, and c. While the fragment complies with $a \rightsquigarrow b$ and $b \rightsquigarrow c$, it violates $c \rightsquigarrow a$.

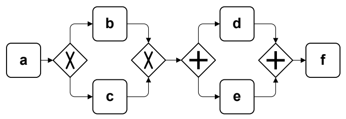
Fig. 6.4. Process model fragments restricting the execution of a, b, and c.



(a) Model PM_1 is not well-structured.



(b) Model PM_2 is not well-structured.



(c) Model PM_3 is well-structured.

Fig. 6.5. The process models PM_1 and PM_2 are not well-structured. While the process model PM_1 cannot be structured according to [117], PM_2 can be structured, resulting the behavior equivalent model PM_3.

its structural and behavioral characteristics. For instance, the strict order relation may define a cyclic dependency between three activities a, b, and c: $a \rightsquigarrow b$, $b \rightsquigarrow c$, and $c \rightsquigarrow a$. The process model fragment in Fig. 6.4(a) satisfies these behavioral constraints at the expense of activity duplication. The result is clearly inappropriate against the background of our use case: an abstract model should provide a concise and compact view on the process. For our notion of a process model, the aforementioned behavioral constraints cannot be satisfied as exemplified by the model in Fig. 6.4(b), where $c \rightsquigarrow a$ is violated.

For the model synthesis, we focus on *well-structured* process models. Notice that our notion of a process model implies that models can be mapped to sound free-choice WF-nets. While soundness means the absence of behavioral anomalies, well-structuredeness refers to model topology. In a well-structured process model every split gateway has a corresponding join gateway, whereas both gateways bound a process model fragment with one entry node and one exit node [81]. The class of well-structured process models is of high practical importance. On the one hand, such models are easy to understand for humans [89]. On the other hand, well-structured process models can be efficiently handled by various analysis techniques, e.g., the computation of temporal constraints [37]. The class of well-structured process models can be defined by means of process model decomposition, the rPST, discussed in Section 2.3.2.

Definition 6.1 (Well-Structured Process Model).
Let $\mathsf{PM} = (A, G, F, t, s, e)$ be a process model. The model PM is *well-structured*, iff the set of canonical components of the rPST of PM contains no rigid component.

Fig. 6.5 exemplifies well-structured process models. Models PM_1 and PM_2 are not well-structured, as both contain rigids, while PM_3 is well-structured. Recently, [117, 118] developed an algorithm enabling process model structuring—construction of behaviorally equivalent well-structured process models for not well-structured process models. The behavioral equivalence is understood in terms of fully concurrent bisimulation [24]. However, not every process model can be structured. For instance, the algorithm of [117, 118] delivers no well-structured process model that is behavior equivalent to PM_1. However, the algorithm structures model PM_2 delivering PM_3.

We design the synthesis of a well-structured process model following the structuring algorithm introduced by [117, 118]. The structuring bases on the relations induced by a complete prefix unfolding and guarantees the preservation of a rather strong behavior equivalence. In the following we show how the model synthesis defined for these relations is adapted to the behavioral profile relations.

To decide whether a well-structured process model can be constructed for a behavioral profile, we use the notion of an order relations graph. [117] introduced an order relations graph capturing the order relations of a complete prefix unfolding. We construct an order relations graph for the behavioral profile relations.

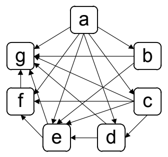
Definition 6.2 (Order Relations Graph). Let $\mathsf{BP} = \{\rightsquigarrow, +, ||\}$ be a behavioral profile over a finite set of activities A_{BP}. A tuple $\mathsf{G} = (V, E)$ is an *order relations graph* of BP such that:

- $V = A_{\mathsf{BP}}$, i.e., the nodes are activities within A_{BP}
- $E = \rightsquigarrow \cup + \setminus id_{A_{\mathsf{BP}}}$, i.e., the edges correspond to the strict order relation and exclusiveness relation without self-relation of activities.

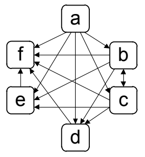
Edges in the order relations graph denote strict order and exclusiveness relations. We assume the strict order relation to be asymmetric and the exclusiveness relation to be symmetric. Thereafter, the strict order and exclusiveness relations are denoted in the graph as unidirectional or bidirectional edges, respectively. Fig. 6.6 shows the order relations graphs for the behavioral profiles of the models depicted in Fig. 6.5. As models PM_2 and PM_3 have equivalent behavior, they share one order relations graphs. That is due to the fact that the notion of equivalence assumed for structuring, fully concurrent bisimulation, is much stronger than behavioral profile equivalence, see [162].

The topology of a well-structured process model relates to the order relations graph structure. According to Definition 6.1 all the canonical components of the rPST of a well-structured process model are of types trivial, polygon, or bond. Such components are represented in the order relations graph by node subsets that have uniform relations with all the remaining graph nodes. We refer to such node subsets as *modules*. Definition 6.3 formalizes the notion of a module and module types following [117].

Definition 6.3 (Module).
Let $\mathsf{G} = (V, E)$ be an order relations graph.



(a) The order relations graph for process model PM_1.



(b) The order relations graph for process models PM_2 and PM_3.

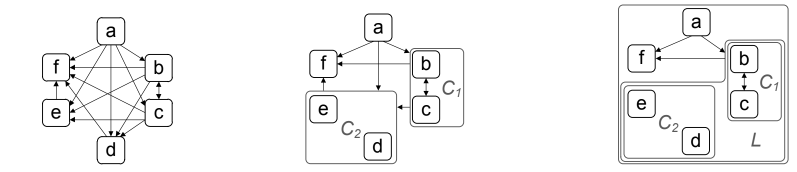
Fig. 6.6. The order relations graphs of the models in Fig. 6.5.

- A *module* $M \subseteq V$ is a non-empty set of nodes that have uniform relations with nodes in $V \setminus M$, i. e., $\forall\, x, y \in M, z \in (V \setminus M)$ it holds $(x,z) \in E \Leftrightarrow (y,z) \in E$ and $(z,x) \in E \Leftrightarrow (z,y) \in E$.
- Two modules $M, M' \subseteq V$ *overlap*, iff they intersect and neither is a subset of the other.
- A module $M \subseteq V$ is *strong*, iff there is no module $M' \subseteq V$, such that M and M' overlap.
- The empty set of nodes $\emptyset$, V, and the node sets of the from $\{v\}, \forall v \in V$ are *trivial* modules.
- A non-trivial module $M \subseteq V$ is *complete*, iff M induces the subgraph of G that is either complete or edgeless. If the subgraph is complete, we say that M is *XOR-complete*. If the subgraph is edgeless, we say that M is *AND-complete*.
- A non-trivial module $M \subseteq V$ is *linear*, iff there exists a linear order $(v_1, \ldots, v_{|M|})$ of elements of M, such that $(v_i, v_j) \in E$ and $(v_j, v_i) \notin E$ for $i, j \in \mathbb{N}$, $1 \leq i, j \leq |M|$ and $i < j$.
- A non-trivial module $M \subseteq V$ is *primitive*, iff it is neither complete nor linear.

To discover modules we leverage the modular decomposition [97]. Modular decomposition of a graph results in a unique arborescence of maximal non-overlapping modules.

Definition 6.4 (Modular Decomposition).
Let $\mathsf{G} = (V, E)$ be an order relations graph. The *modular decomposition tree* is a tuple $\mathsf{MDT}_\mathsf{G} = (\Omega, \xi)$, such that Ω is a set of all strong modules and $\xi : \Omega \rightarrow \mathcal{P}(\Omega)$ is a function that assigns child modules to modules with $\forall\, \omega, \gamma \in \Omega\ [\ (\xi(\omega) \cap \xi(\gamma) \neq \emptyset) \Rightarrow \omega = \gamma\ ]$.

Fig. 6.7 exemplifies the modular decomposition for the order relations graph in Fig. 6.6. The order relations graph is stepwise decomposed into a hierarchy of strong modules. Two sets of nodes, $\{b, c\}$ and $\{d, e\}$, are identified as strong modules that have equal relations to all other nodes in the graph. Both modules constitute together another module, as the former modules are of equal relations to the nodes a and f. Finally, we return to the illustrative



(a) Order relations graph.

(b) Complete modules C_1 and C_2 are discovered.

(c) Nodes a and f with modules C_1 and C_2 constitute a linear module L.

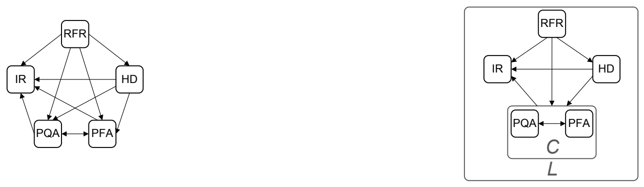
Fig. 6.7. The step-wise modular decomposition of the order relations graph in Fig. 6.6(b). In the initial order relations graph node sets {b, c} and {d, e} are discovered as strong modules. Module C_1 is XOR-complete, while C_2 is AND-complete. Nodes a and f with modules C_1 and C_2 constitute linear module L.

example. Fig. 6.8 shows the order relations graph and its decomposition for the behavioral profile in Table 6.2.

The modular decomposition of an order relations graph characterizes behavioral profiles for which we construct a well-structured process model. That is, we check for the absence of a primitive module in the modular decomposition. Note that we implicitly assume that the relational properties of a behavioral profile are satisfied.

Definition 6.5 (Well-Structured Behavioral Profile).
Let $\mathsf{BP} = \{\rightsquigarrow, +, ||\}$ be a behavioral profile over a finite set of transitions A_{BP} and G—the order relations graph of BP. The behavioral profile BP is *well-structured*, iff the modular decomposition tree of G, $\mathsf{MDT_G}$, contains no primitive module.

In the example with the three activities a, b, and c, where $a \rightsquigarrow b$, $b \rightsquigarrow c$, and $c \rightsquigarrow a$ the profile is not well-structured. The modular decomposition of the respective order relations graph comprises a primitive module covering the three activities. Fig. 6.6 visualizes the order relations graphs for the process models in Fig. 6.5. The graph in 6.6(a) does not represent a well-structured



(a) The order relations graph for the behavioral profile in Table 6.2.

(b) Decomposition of the order relations graph.

Fig. 6.8. The order relations graphs for the behavioral profile in Table 6.2 and its modular decomposition.

behavioral profile since the modular decomposition tree contains a primitive module. The modular decomposition of the graph in Fig. 6.6(b) is shown in Fig. 6.7. As the modular decomposition contains no primitive module, graph in Fig. 6.6(b) represents a well-structured behavioral profile. We conclude that *1)* model PM_1 in Fig. 6.5 has a non-well-structured behavioral profile, and *2)* the behavioral profile of models PM_2 and PM_3 is well-structured. Returning to the running example, we consider Fig. 6.8. Fig. 6.8(a) shows the relations graph corresponding to the behavioral profile of the abstract process model. The modular decomposition of this order relations graph is presented in Fig. 6.8(b). The decomposition has two modules: the complete module C and the linear module L.

We use the existing methods for graph modular decomposition to decide if a behavioral profile is well-structured. Verification of a behavioral profile well-structuredeness can be done in linear time. According to Definition 6.5, we create the modular decomposition tree of the order relations graph of the validated behavioral profile. The modular decomposition tree can be constructed in linear time [97]. The number of strong modules in the modular decomposition tree is linear to the size of the graph [97].

Finally, we show that well-structuredness of a behavioral profile is a necessary condition for the existence of a well-structured process model exhibiting this profile.

Lemma 6.6. The behavioral profile of a sound well-structured process model is well-structured.

Proof. Let $\mathsf{RPST}_{\mathsf{PM}} = (\Omega, r, \chi)$ be the rPST of a sound process model $\mathsf{PM} = (A, G, F, t, s, e)$. According to our notion of a process model each activity is a boundary node of at most two trivial components of $\mathsf{RPST}_{\mathsf{PM}}$. Let $\alpha, \beta \in \Omega$ be two trivial components for which the fragment entries are two distinct activities $a, b \in A$, $a \neq b$. Since PM is well-structured, $\mathsf{RPST}_{\mathsf{PM}}$ does not contain any rigid component. According to the Proposition 4.1 in [163], the profile relation for a pair of transitions in a sound free-choice WF-net with no rigids can be deduced from *1)* the type of the lowest common ancestor (LCA), see Section 2.1, component of the respective rPST, $\gamma = lca(\alpha, \beta)$, and *2)* the existence of a loop fragment on the path from the root of the tree to the LCA component γ. As we consider process models that can be mapped to sound free-choice WF-nets, we apply this observation to our notion of a process model. If the trivial components $\alpha, \beta \in \Omega$ related to two distinct activities $a, b \in A$ are part of a loop fragment, the corresponding module is AND-complete. If they are not a part of the loop fragment, the type of the LCA component, $\gamma = lca(\alpha, \beta)$, determines the type of the module.

For any fragment of the rPST, there is a module in the respective modular decomposition tree $\mathsf{MDT}_{\mathsf{G}}$ of the order relations graph G of the behavioral profile. A polygon yields a linear module, an AND-gateway-bordered bond component—an AND-complete module, a XOR-gateway-bordered acyclic

Algorithm 5 Synthesis of a sound well-structured process model from a well-structured behavioral profile

```
1: synthesizeModel(BP = {$\rightsquigarrow, +, ||$})
2: G = constructOrderRelationsGraph(BP)
3: MDT ($\Omega, \xi$) = modularDecomposition(G)
4: for all $\omega \in \Omega$ following on a postorder traversal using $\xi$ do
5:    if $\omega$ is trivial then
6:       add activity to PM
7:    if $\omega$ is AND-complete then
8:       construct AND-bordered bond in PM
9:    if $\omega$ is XOR-complete then
10:      construct XOR-bordered acyclic bond in PM
11:   if $\omega$ is linear then
12:      construct trivial or polygon in PM
13: if PM misses start or end activity then
14:    add start and/or end activity to PM
15: for all $a \in A_{PM}$ such that $a||a$ do
16:    insert control flow cycle around $a$ in PM
17: return PM
```

bond component—a XOR-complete module. Hence, MDT_G does not contain any primitive module and the behavioral profile is well-structured. □

6.3.2 Synthesis of a Process Model from a Well-Structured Behavioral Profile

Once well-structuredness of a behavioral profile is verified, we proceed with the model synthesis. The synthesis algorithm iteratively constructs a model from the modules identified by the modular decomposition. We largely rely on the synthesis algorithm presented in [117, 118].

Algorithm 5 outlines the steps of the model synthesis. First, we construct the order relations graph of the behavioral profile (line 1). Modular decomposition discovers modules in the order relations graph (line 2). The algorithm iterates over all the identified modules to construct the model skeleton (lines 2–14). Trivial modules contribute only single activities to the model. A complete module leads to the creation of an AND-gateway-bordered bond or a XOR-gateway-bordered acyclic bond. Such a bond comprises all activities or model fragments encapsulated by this complete module. A linear module leads to the creation of a polygon connecting the respective activities or model

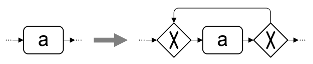


Fig. 6.9. Insertion of a XOR-bordered cyclic bond for an activity with interleaving order as a self-relation.

fragments. If the resulting model structure is gateway-bordered, it is normalized to satisfy the structural requirements of the process model (lines 13–14). For all activities that have interleaving order as their self-relation according to the behavioral profile, we insert circuits into the created process model structure (lines 15–26). Those comprise a XOR-gateway-bordered cyclic bond and polygons. This transformation step is illustrated in Fig. 6.9. As the final step, the algorithm returns the process model.

We prove the correctness of the Algorithm 5 as follows.

Proposition 6.7. Algorithm 5 terminates and after termination the sound well-structured process model $\mathsf{PM} = (A, G, F, t, s, e)$ shows the behavioral profile used as the algorithm's input.

Proof. First, we show that the resulting model is indeed a sound well-structured model. Second, we prove that the behavioral profile used as the input coincides with the behavioral profile of the created model for the respective activities.

Termination The set of activities of the behavioral profile is finite. Thus, the order relations graph and the number of modules identified in the decomposition are finite. Once we iterate over all activities and modules, the algorithm terminates.

Result We first prove the correctness of syntax, then of semantics. Finally, we consider the behavioral profile correctness.

Syntax The algorithm creates a process model $\mathsf{PM} = (A, G, F, t, s, e)$ by a postorder traversal of the modular decomposition tree. Hence, for all nodes there is a path from (to) the node that represents the entry (exit) of the component created for the root module. If those nodes are gateways, the algorithm adds a start and end activities. Hence, PM aligns with the process model notion described in Definition 2.13. As the algorithm constructs only trivial, polygon, and bond components (the trivial circuit is a bond as well), model PM can be mapped to a free-choice WF-net.

Semantics As it follows from step Syntax the Algorithm 5 delivers a process model that can be mapped to a WF-net. The model is constructed by nesting trivial, polygon, and bond components. The constructed bond fragments are acyclic, bordered by either AND- or XOR-gateways. The construction of a trivial circuit inserts polygons and cyclic XOR-bordered bonds. Trivial and polygon components do not cause unsoundness. Further, Lemma 1 and Lemma 2 in [163] argue that place- and transition-bordered bonds do not cause unsoundness. As XOR- and AND-gateway-bordered bonds are mapped, respectively, to place- and transition-bordered bonds the created model satisfies the soundness requirements. Thereafter, the produced process model is sound.

Behavioral Profile The constructed process model $\mathsf{PM} = (A, G, F, t, s, e)$ is well-structured and can be mapped to a sound WF-net. According to Lemma 6.6, the behavioral profile of PM is well-structured. This behavioral profile coincides with the behavioral profile used as the algorithm's input. The latter follows from the types of the constructed fragments. Neglecting the trivial circuits inserted at the end, the algorithm creates a trivial, polygon, or bond component depending on the type of the module, i.e., depending on the relations observed between the activities of the module in the order relations graph. For two distinct activities $a, b \in A$, $a \neq b$, this component is the LCA of the trivial components $\alpha, \beta \in \Omega$ for which the component entries are a and b, respectively. Neglecting the trivial circuits, the type of the LCA component determines the profile relation, see Proposition 1 in [163]. A trivial circuit includes only one activity causing the interleaving order as the self-relation for this activity. Therefore, an insertion of trivial circuits has no impact on the relation between two distinct activities. Trivial circuits are introduced only for activities with interleaving order as the self-relation. Thus, the behavioral profile of the constructed process model coincides with the behavioral profile used as the algorithm input. □

Corollary 6.8. Given a well-structured behavioral profile, the construction of a process model exhibiting this behavioral profile can be solved in linear time.

Proof. We represent the relations used in the process model synthesis as bi-dimensional arrays that map to zero or one. Against this background, adding an entry to a relation and checking a tuple membership is done in constant time. The construction of order relations graph takes linear time to the size of the behavioral profile. Further, the modular decomposition tree for order relations graph is realized in linear time [97]. We proceed iterating the strong modules in the modular decomposition tree. The number of strong modules is linear to the graph size [97]. The construction of a respective model fragment takes linear time to the behavioral profile size. Process model normalization implies the check for the start and end activities. This operation also takes linear time. Finally, insertion of trivial circuits takes linear time to the size of the behavioral profile. □

Now we can make the following statement.

Theorem 6.9. There exists a sound well-structured free-choice WF-system, if and only if, the behavioral profile is well-structured.

Proof. $\Rightarrow$ follows from Lemma 6.6, $\Leftarrow$ from Proposition 6.7. □

We conclude this section returning to the motivating example presented in Fig. 6.1. Fig. 6.10 illustrates the complete abstract model derived from the initial model according to the developed abstraction technique. Following Algorithm 5 the model in Fig. 6.10 is obtained from the modular decomposition presented in Fig. 6.8.

6.4 Software Implementation

In this section, we elaborate on FLEXAB—an application enabling the presented abstraction and presented in [164]. FLEXAB extends the Oryx framework [42], which we introduce first. Then, we describe the FLEXAB architecture and illustrate the usage to demonstrate the capabilities of FLEXAB.

6.4.1 Oryx

We implemented the proposed business process model abstraction approach within the Oryx Framework. Oryx is an extensible modeling framework bringing Web 2.0 technologies to business process designers. It allows for web-based modeling following a zero-installation approach. Oryx identifies each model by a URL, so that models can be shared by passing references rather than by exchanging model documents in email attachments. The framework can be extended in various directions. New languages are added by stencil sets that define explicit model element typing, rules of the composition and connection of elements, and the visualization of elements. Further, Oryx features a plugin infrastructure to add new functionality.

Oryx is organized into client and server components. The client component, the Oryx editor, realizes the modeling functionality. The editor is a JavaScript application running in a web-browser. The server component, the Oryx backend, stores process models, stencil sets, and fulfills other tasks, e.g., user management and rendering of various model representations (SVG, PNG, or PDF). The backend is implemented in Java.

6.4.2 Oryx Mashup Framework

The Oryx editor addresses use cases that center around a single model, i.e., a designer edits one model at a time and does not need to trace dependencies with other models. However, several use cases, and process model abstraction is one of them, require the designer to observe several models simultaneously. The Oryx Mashup Framework provides an API for developing applications in which several models are manipulated on one screen. Similar to the

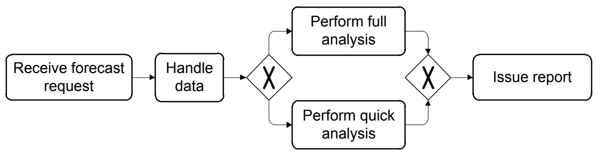


Fig. 6.10. Abstract model of the process "Forecast request handling". This model is obtained from the model PM in Fig. 6.1 using the abstraction algorithm based on behavioral profiles with the threshold of 0.5 and the activity groups as defined in Fig. 6.1.

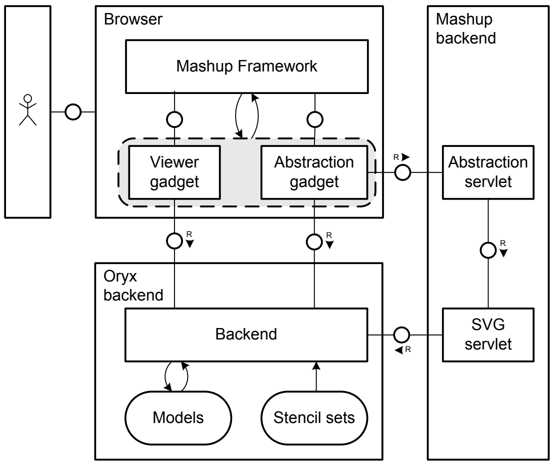


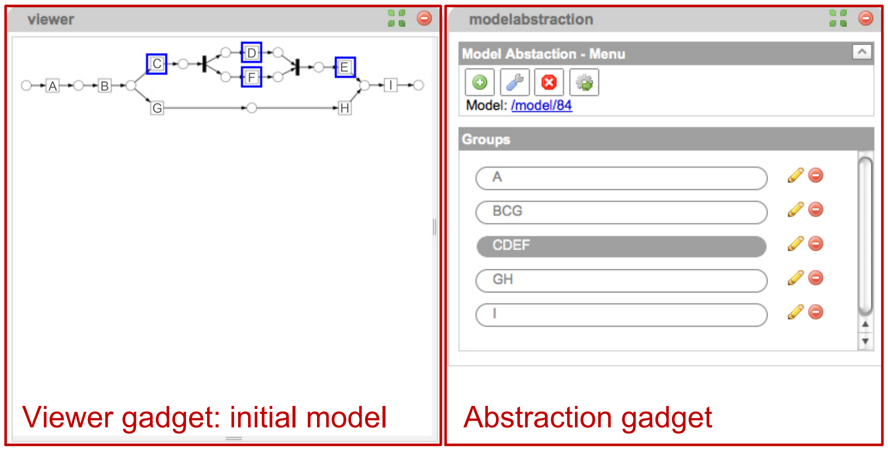
Fig. 6.11. FLEXAB architecture overview (FMC notation)

Oryx Editor, the Mashup Framework is written in JavaScript and runs within a browser. The framework organizes functionality by *gadgets* and provides means to support communication between different gadgets. Each gadget not only accumulates business logic, but also has a UI representation. The UI components of gadgets are allocated on a dashboard. Typical gadgets provide model viewing functionality or enable selection of model elements. Hence, the Oryx Mashup Framework enables developers to create mashups for analyzing existing Oryx models and for concurrent interaction with several models.

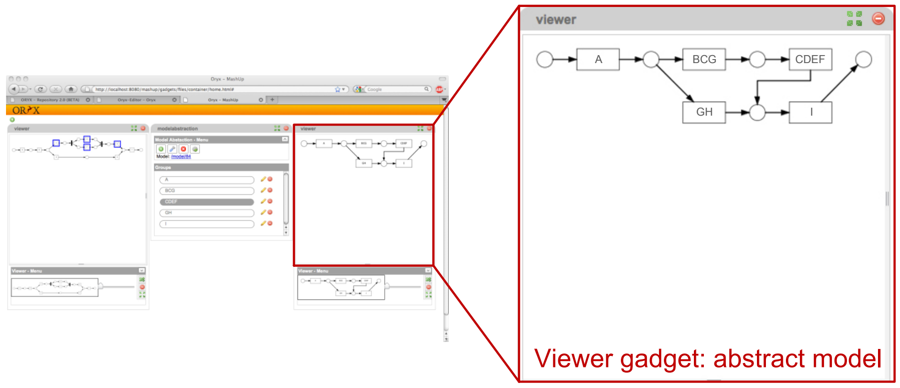
6.4.3 FLEXAB

We have used the Oryx Mashup Framework as the basis for FLEXAB. Logically, the application is decomposed into the client-side and server-side components. The client-side component is built as an extension of the Oryx Mashup Framework. The server-side component is further distributed into the Oryx backend and Mashup backend, see Fig. 6.11. The communication between these three components is established by HTTP requests. The client-side component renders the user interface of the application. A viewer gadget presents the initial model that should be abstracted. The abstraction gadget, in turn, enables the user to define activity groups. This is supported by the viewer gadget to allow for populating groups with activities by simply selecting the activities in the viewer. Finally, another instance of a viewer gadget is used to show the abstract model.

Once the abstraction is triggered, the abstraction gadget sends the user-defined activity groups along with the initial process model to the abstraction servlet on the server side. Given this input, the abstraction servlet performs the abstraction algorithm and produces an abstract model. The abstraction



(a) Screenshot of Flexab at the stage of activity group creation.



(b) Flexab presents the process model emerging from abstraction.

Fig. 6.12. The screenshots presenting the user interface of the Flexab.

servlet is supported by an SVG servlet that is responsible for the generation of a SVG representation of the abstract model. To this end, it needs to retrieve the respective stencil set from the Oryx backend.

From a user perspective, abstracting a process model in Flexab works as follows. The user starts selecting the model to be abstracted. In response, the application caters two gadgets: a viewer gadget and an abstraction gadget, see Fig. 6.12(a). The user creates named activity groups, edits, and deletes the groups using the controls of the abstraction gadget. The viewer gadget not only renders the process model and provides zoom functionality, but also supports activity group creation: the user populates groups selecting activities directly in the model. Once the groups are finalized, the user initiates model transformation clicking the abstraction button in the abstraction gadget. Then, FLEXAB abstracts the model in the background and instantiates a new viewer gadget to visualize the result of abstraction. Fig. 6.12(b) presents the UI constellation in terms of the complete Mashup dashboard once model abstraction completes.

6.5 Discussion

This section summarizes the properties of the developed business process model abstraction. First, we argue about the application context of the proposed control flow relation discovery method. Second, we position the developed approach against the research on process model synthesis.

6.5.1 Application Perspective

The developed abstraction exhibits two features that distinguish it from the already available methods, e.g., see [30, 94, 122, 123]:

Property 1 one activity may belong to several activity groups

Property 2 activity groups may be distributed over the process model in an arbitrary fashion.

This turns the proposed abstraction into a powerful tool, yet having side effects that limit its applicability. First, the process model abstraction based on behavioral profiles has a high degree of information loss. This phenomenon follows from the fact that behavioral profile relations provide much less information about the process behavior than the process model control flow relation. Consider, for instance, the interleaving order relation, which stems from structures capturing concurrency or cyclic structures. Second, process model abstraction based on behavioral profiles is not order preserving. This observation directly follows from Property 1 and Property 2. However, if activity groups are obtained from structural process model decomposition, see Chapter 4, the abstraction based on behavioral profiles is order preserving. Against this background, the advocated approach is the generalization of structural abstraction methods. Third, the complexity of activity groups impedes the evaluation of the non-functional property values for coarse-grained activities. For instance, it follows from Property 1 that one activity of the initial process model may contribute to several coarse-grained activities of the abstract model. Hence, the non-functional properties of this single activity impact the properties of multiple coarse-grained activities. Thereafter, the deduction of properties becomes non-trivial.

The three identified features limit the application of the abstraction approach advocated in this chapter. However, they are balanced by the advantage of non-hierarchical process model abstraction. Furthermore, the three limiting factors are the direct consequences of non-hierarchical abstraction. Thereafter, we conclude that the user has to consider the pros and cons of the proposed model transformation during the design of the abstraction approach. For instance, if the user tolerates a considerable information loss for the sake of non-hierarchical abstraction, the approach based on behavioral profile relations is suitable. This approach, for instance, fits well the use cases of the group "Group 4: Obtaining a Process Quick View".

6.5.2 Related Work on Process Model Synthesis

We outline the streams of work related to the synthesis of an abstract process model form a behavioral profile. On the one hand, this discussion includes the synthesis of Petri nets. On the other hand, behavioral profile relations is only one of the many approaches to describe the process behavior. Hence, we present the alternative relation families and argue about their advantages and disadvantages in the abstraction context.

The developed method for the construction of an abstract process model extends the family of process model synthesis techniques. As our notion of a process model is tightly coupled with the Petri nets formalism, we consider the problem of Petri net synthesis [54]. In [56] Esparza discusses the reduction rules for live and bounded free choice Petri nets. The article argues that the rules inverse to these reduction rules address the problem of model synthesis. A number of research endeavors considered generation of Petri nets from finite state machines [38, 43]. A series of works studied the generation of a Petri net as a composition of several other nets [96, 139].

While this thesis proposed process model synthesis from the behavioral profile relations, there are other relation families that abstract behavior of systems with concurrency. For many of these families the researchers defined methods for the synthesis of models. Here we discuss the applicability of these relations in the context of business process model abstraction. In process mining the alpha algorithm is used for the construction of a process model from event logs [10]. The primary difference between the alpha relations and the behavioral profile relations is the definition of the strict order. Two activities belong to the *alpha* strict order relation, if one activity is the *direct* successor of the other. In this setting, the exclusiveness relation quickly dominates the behavioral abstraction of the initial process model. This fact impedes the behavior generalization procedure defined by Algorithm 4. Another set of relations suggested by Li, Reichert, and Wombacher in [93] describes the model behavior in the form of an *order matrix*. While order matrix relations are semantically very close to behavioral profile relations, they are defined for the class of well-structured process models only. Meanwhile, the order matrix relations provide more detailed information about the activities within cyclic structures than the behavioral profile relations. Finally, in [117] Polyvyanyy et al. use the family of relations with a similar semantics, but defined for the complete prefix unfoldings of a Petri net, see [57, 99]. These relations preserve more information about the process ordering constraints. Unfortunately, the unfolding mechanism implies duplication of activities within a process model, which is undesirable in the context of process model abstraction.

The literature study witnesses of other numerous abstractions of process behavior. For instance, [51] advocate the concept of *causal footprints* to judge about process model similarity. However, the construction of causal footprints is rather inefficient. This circumstance impedes the use of causal footprints, as business process model abstraction deals with large process models. Another example are *precedes* and *leads to* relations used for checking the compliance of process models to regulations [13, 15]. However, these two relations provide less information about the process ordering constraints. This limits the application of *leads to* and *precedes* in the context of our problem.

The outlook of process behavior abstractions shows that each technique has its pros and cons, while there is no clear "leader". As some relation families preserve the behavioral constraints in a precise manner, they fall short on the class of models that can be handled or have side effects that contradict the abstraction use case.

6.6 Summary

This chapter introduced a novel approach to business process model abstraction. As an input the approach takes a process model and groups of related activities in this model. Each activity group corresponds to one coarse-grained activity of the abstract process model. In this setting, the advocated abstraction focuses on the synthesis of the control flow relation in the abstract process model. In comparison to the existing business process model abstraction methods, our approach allows 1) one activity to belong to multiple groups and 2) an activity group to be freely spread over the model. Because of this, the advocated method overcomes the limitations of the structural approaches presented in Chapter 4 and in the related work, see, for instance, [30, 55, 66, 94, 123]. Due to the flexibility of the novel abstraction, it can be combined with various methods for the discovery of related activities, both manual and automatic, see Chapter 5. Among the limitations of the proposed method, one can name the capability to synthesize well-structured process models only. In addition, the abstraction is sensible to the presents of cycles in process models. Finally, we have complemented the theoretical discussion of the abstraction approach with an overview of its software implementation.

The current chapter focused on the *how* of business process model abstraction. Against this background, its contribution complements the findings of Chapter 5. Furthermore, the novel abstraction approach can be seen as the generalization of the abstraction proposed in Chapter 4.

7

Related Work

This chapter outlines the research contributions related to the topic of business process model abstraction. We organize the related work in three tiers, each providing a different perspective on business process model abstraction.

The first tier includes papers and articles that directly discuss the problem of business process model abstraction. We have discovered eight methods that are devoted to this topic and provide a detailed description for each of them. The second tier of the related work is formed by the papers developing methods and algorithms that *might* become abstraction enablers. While these papers do not discuss the problem of business process model abstraction directly, their contributions can be reused in the abstraction context. Therefore, the papers of this tier may be used as a theoretical foundation for new abstraction methods. The chapter complements the discussions of individual abstraction methods and potential abstraction enablers by a summarizing comparison. The comparison presents the "big picture" of the existing abstraction methods: highlights the most explored fields and reveals the research gaps. The third tier discusses the aspects of model management that are similar to business process model abstraction problem. Among other topics this stream of work comprises the studies of intermodel relations and the problem of process modularization. Notice that this tier puts the problem of business process model abstraction in a wider context and compares it to other business process model management tasks.

Against this background, the contribution of this chapter is twofold. First, it provides a discussion of the related work organized into three tiers. Second, the chapter provides a comparison of the existing methods distinguishing the already explored areas from those with the research opportunities.

The remainder of this chapter is organized as follows. Section 7.1 presents the state-of-the-art business process model abstraction methods. Section 7.2 mentions the research contributions that can support new abstraction methods. Section 7.3 discusses problems similar to business process model abstraction. Finally, Section 7.5 concludes this chapter with a summary.

7.1 Abstraction State of the Art

Scientific papers that discuss business process model abstraction by no means always use this exact label, but rather refer to *process views*, see [30, 55], or focus on *process simplification*, see [66]. However, the essential purpose of these methods is in line with the way we characterized abstraction in this thesis. While a number of papers, e.g., [30, 94, 141], discuss generic process model abstraction methods, others address concrete use cases, e.g., see [55]. This section presents an overview of the available business process model abstractions preceded with their short summary.

Cardoso et al. [32] Propose a quality of service evaluation method for workflows enriched with information on transition probability and activity execution time/cost.

Liu and Shen [94, 141] Suggest an order-preserving abstraction approach making use of reduction rules developed in [135].

Chiu et al. [34, 35] Focus on the business process model abstraction in the context of cross-organizational interaction, where the generic model captures overall interaction, while abstract models are partner-specific.

Pankratius and Stucky [115] Adapt the principles of views in relational databases to the context of business process models delivering operations for constructing abstract process specification.

Günther and van der Aalst [66] Develop an abstraction technique for process models mined from logs, where the technique exploits metrics based on log information.

Bobrik et al. [28, 29, 30] Propose a process model abstraction with an emphasize on *how* aspect, specifying abstraction operations and their composition rules.

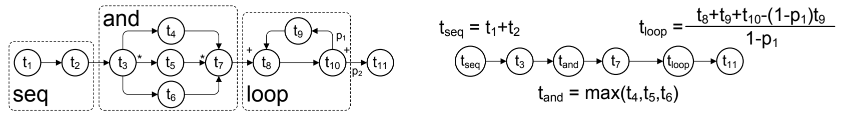
Eshuis and Grefen [55] Address the abstraction scenario, where an internal process model is adapted for an external partner in two steps: 1) the private details are concealed, 2) excessive information is hidden.

Polyvyanyy et al. [122, 123] Argue that process model decomposition can be employed in business process model abstraction and develop abstraction algorithms based on decomposition.

In the remainder of this section, we elaborate on each approach using the business process model abstraction framework introduced in Section 3.2. Each approach is positioned against the use cases elaborated in Section 3.4. To illustrate the approaches we employ examples in the notations of the original papers.

7.1.1 Cardoso et al.

In [32] Cardoso et al. evaluate the workflow quality of a service. The authors assume that every workflow activity is annotated with a non-functional property value, e.g., execution time or cost, and an execution probability.


and
seq
loop
t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11}
p_1
p_2
$t_{seq} = t_1+t_2$
$t_{loop} = \frac{t_8+t_9+t_{10}-(1-p_1)t_9}{1-p_1}$
t_{seq} t_3 t_{and} t_7 t_{loop} t_{11}
$t_{and} = \max(t_4,t_5,t_6)$

(a) Initial model PM (b) Abstract model PM_a

Fig. 7.1. Illustration of the business process model abstraction approach developed by Cardoso et al. in [32].

Quality of service for a workflow is evaluated through aggregation of workflow activities, where non-functional properties of an aggregating activity are determined by the properties of the aggregated ones. The paper considers activities as abstraction objects and utilizes aggregation as the basic abstraction operation. The proposed solution addresses the business process model abstraction problem, namely the use cases of the group "Group 1: Preserving Relevant Activities" and the use cases "Use Case 12: Get Process Quick View Respecting Ordering Constraints" and "Use Case 13: Get Process Quick View Respecting Roles".

The paper concentrates on the *how* aspect of abstraction. While the employed model is formalized as a graph with nodes being tasks and edges being transitions between tasks, the process model is well-structured, see Definition 6.1. In such a setting the authors specify the abstraction algorithm based on patterns and corresponding reduction rules. Once a workflow fragment is matched against a pattern, it is reduced according to the reduction rule. Four patterns are identified: sequence, AND-block, XOR-block, and two types of loop blocks. Fig. 7.1 demonstrates the application of these rules to model PM in Fig. 7.1(a). The reduction rules for sequence, AND-block, and the loop block are applied to fragments *seq*, *and*, and *loop*, respectively. The resulting abstract model is presented in Fig. 7.1(b). The reduction rules specify not only the structural transformations, but also the non-functional properties evaluation method. In this way the approach advocates a hierarchical order-preserving abstraction that preserves process non-functional properties.

7.1.2 Liu and Shen

In [94, 141] Liu and Shen study the construction of process views and, in particular, order-preserving process views. The paper regards activities as the abstraction objects and employs aggregation as the basic abstraction operation. The proposed business process model abstraction can support the use cases of the group "Group 1: Preserving Relevant Activities" along with the use cases "Use Case 9: Adapt Process Model for an External Partner", "Use Case 12: Get Process Quick View Respecting Ordering Constraints", "Use Case 13: Get Process Quick View Respecting Roles", and "Use Case 15: Preserve Frequent Activities Summarizing Rare Activities".

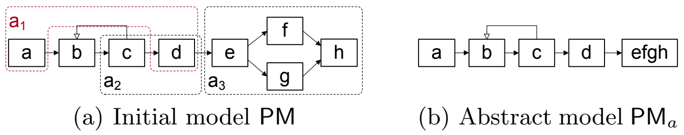


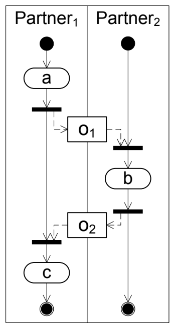
Fig. 7.2. Illustration of the business process model abstraction approach developed by Liu and Shen in [94, 141].

The authors provide a formal definition of a process model and specify a process execution semantics. The formalism allows for one type of model nodes, activities, that may realize the splitting and joining logic of ANDs and XORs. Loop dependency is considered as a special dependency type and process models may contain only single entry-single exit loops. Fig. 7.2 presents process model examples that adhere to the notation used in [94]. The paper elaborates on an algorithm for abstract process model construction based on the reduction rules proposed in [135]. The algorithm obeys three principles: activity membership, activity atomicity, and order preservation. The latter principle is of great practical importance and is thoroughly discussed in the paper. The example in Fig. 7.2(a) illustrates capabilities of the abstraction approach. While aggregations a_1 and a_2 violate the declared principles, a_3 is valid. Fig. 7.2(b) presents the result of order-preserving abstraction: activities e, f, g, and h in the initial model are aggregated, see Fig. 7.2(a). While the paper elaborates on the abstraction *how*, it does not discuss the *why* and *when* questions. The developed abstraction is hierarchical, but does not discuss preservation of non-functional properties.

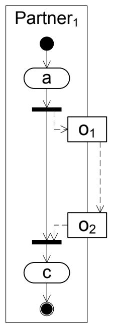
7.1.3 Chiu et al.

Chiu et al. discuss the business process model abstraction in the context of cross-organizational interaction in a web service environment [34]. The authors concentrate on the use case, where an existing process model describes interorganizational interaction, while partner-specific models hiding confidential details are in demand. Effectively, this approach maps to “Use Case 9: Adapt Process Model for an External Partner”. The abstraction approach selects activities as abstraction objects and uses elimination as the basic abstraction operation to deliver partner-specific views and narrow the scope within a party. The paper argues about the *how*, delegating the *why* and *when* questions to the human user.

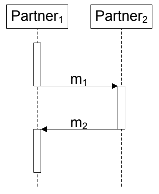
Chiu et al. employ UML activity diagrams to capture processes and introduce a metamodel for workflow views. The metamodel defines a view as a graph with activities that can be organized into sequences enriched with choice logic. Each activity can either be optional or iterative. Fig. 7.3 illustrates the approach. Model PM in Fig. 7.3(a) is the initial model, while abstract model PM_a in Fig. 7.3(b) captures the process from the point of view of one participant. The paper informally discusses the transition from the initial model



(a) Initial model PM



(b) Abstract model PM_a



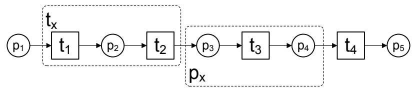
(c) Abstract model PM'_a

Fig. 7.3. Illustration of the business process model abstraction approach developed by Chiu et al. in [34].

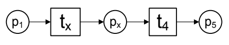
to the abstract one. The authors argue that the partner interaction specified in the initial UML activity diagram can be captured on the high level by means of a sequence diagram. The paper gives the general idea of such a transformation. Fig. 7.3(c) presents an example of a sequence diagram that can be mined from the initial model PM. Finally, the paper specifies consistency criteria for process model views with respect to the initial model. While the discussion of model transformation is rather informal we attribute it to hierarchical order-preserving abstraction that does not address process non-functional properties.

7.1.4 Pankratius and Stucky

Pankratius and Stucky relate process views to views in relational databases, see [115]. The authors define process models as Petri nets and adapt relational database operations to the Petri net formalism. This results in eight operations enabling process view creation: selection, difference, place projection, transition projection, place join, transition join, theta join, and union. The designed operations may support the business process model abstraction, where activities and, potentially, events are abstraction objects. Elimination and aggregation can be realized through place projection and transition pro-



(a) Initial model PM



(b) Abstract model PM_a

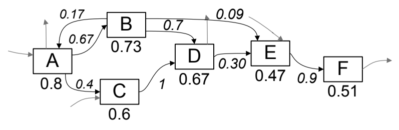
Fig. 7.4. Illustration of the business process model abstraction approach developed by Pankratius and Stucky in [115].

jection, respectively. Fig. 7.4 exemplifies an application of place projection and transition projection for an abstraction realization. Fig. 7.4(a) shows process model PMwith two fragments, t_x and p_x, to be abstracted. Application of a transition projection to fragment t_x and a place projection to fragment p_x in PM results in transition t_x and place p_x in model PM_a, respectively. Fig. 7.4(b) exhibits the abstract model PM_a.

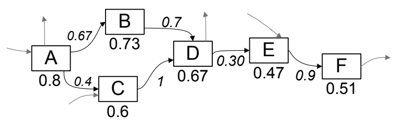
It should be noted that the developed operations consider only the structure of Petri nets, ignoring the execution semantics. As a consequence, important properties, e.g., soundness, of delivered process models may be violated [156]. In essence, the paper addresses the business process model abstraction *how*. The developed abstraction supports the use cases of the group "Group 1: Preserving Relevant Activities" together with the use cases "Use Case 9: Adapt Process Model for an External Partner", "Use Case 12: Get Process Quick View Respecting Ordering Constraints", "Use Case 13: Get Process Quick View Respecting Roles", "Use Case 15: Preserve Frequent Activities Summarizing Rare Activities", and "Use Case 16: Get Particular Process Perspective".

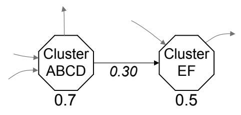
7.1.5 Günther and van der Aalst

In [66] Günther and van der Aalst investigate the simplification of "spaghetti-structured" process models as mined from event logs. The paper addresses all three aspects of process model abstraction: *why*, *when*, and *how*. The authors choose activities and edges as abstraction objects. The abstraction mechanism assumes the availability of substantial process logs enriched with activity and transition frequencies. The authors elaborate on how this information can be used within abstraction. They suggest metrics, e.g., activity frequency, distinguishing significant abstraction objects from insignificant ones. The metrics are classified into *significance* and *correlation* metrics and orchestrate abstraction operations. Fig. 7.5 exemplifies the abstraction approach. The elements of



(a) Initial model PM





(b) Result of edge elimination PM_a (c) Result of activity aggregation PM'_a

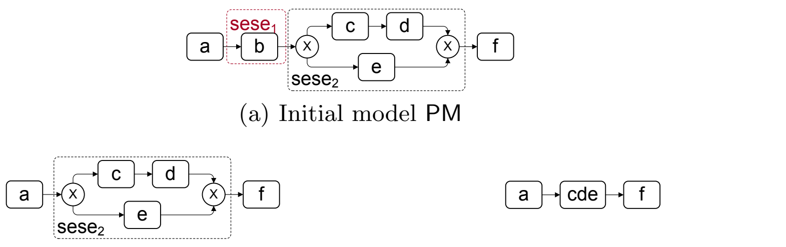
Fig. 7.5. Illustration of the business process model abstraction approach developed by Guenther and van der Aalst in [66].

the initial model PMare annotated with execution frequencies, see Fig. 7.5(a). In Fig. 7.5(b) the edges with low frequencies are eliminated, while Fig. 7.5(c) shows the outcome of activity aggregation. The designed business process model abstraction is potentially capable of supporting the use cases of the groups "Group 1: Preserving Relevant Activities" and "Group 2: Preserving Relevant Process Runs" and the use cases "Use Case 12: Get Process Quick View Respecting Ordering Constraints", "Use Case 13: Get Process Quick View Respecting Roles", "Use Case 15: Preserve Frequent Activities Summarizing Rare Activities", and "Use Case 16: Get Particular Process Perspective".

The authors formalize a process model as a graph, where nodes are activities, and edges—control flow relation. Unfortunately, such a simplistic model limits the applicability of the approach to some extent, as process modeling notations typically specify more than one node type. Moreover, the assumed availability of rich process logs is rather restrictive: models are rarely enriched with such detailed execution information. On the other hand, the approach preserves process non-functional properties and enables hierarchical process model abstraction.

7.1.6 Bobrik et al.

Bobrik et al. study process views in [28, 29, 30]. The authors concentrate on the abstraction's *how* component, leaving the *why* and *when* out of scope. These works consider activities as the abstraction objects. The basic abstraction operations are aggregation and elimination. The developed abstraction method addresses the use cases of the group "Group 1: Preserving Relevant Activities" along with the use cases "Use Case 9: Adapt Process Model for an External Partner", "Use Case 12: Get Process Quick View Respecting Ordering Constraints", "Use Case 13: Get Process Quick View Respecting Roles", "Use Case 15: Preserve Frequent Activities Summarizing Rare Activities", and "Use Case 16: Get Particular Process Perspective".



(a) Initial model PM

(b) Result of $sese_1$ elimination PM_a (c) Result of $sese_2$ aggregation PM'_a

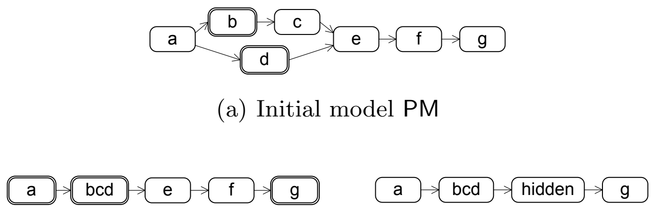
Fig. 7.6. Illustration of the business process model abstraction approach developed by Bobrik et al. in [28, 29, 30].

The process model is formalized as a graph with two node types, activities and gateways (subsequently typed to ORs, XORs, and ANDs), and the edges representing the control flow. Aggregation and elimination make use of SESE fragments—fragments with exactly one incoming and exactly one outgoing edge. Elimination substitutes a SESE fragment with an edge, while aggregation—with an activity. The paper studies the abstraction properties, paying attention to control-flow preservation. The resulting abstraction is hierarchical, but does not address the aspect of non-functional properties. The abstraction proposed by Bobrik et al. is not order-preserving in general case.

Along with the model views, the authors discuss view construction for visualizations of process instances. The distinction of completed and not completed activities allows to extend basic abstraction operations beyond SESE fragment transformations. The completed activities are abstracted in a more flexible fashion. Fig. 7.6 illustrates the proposed abstraction. The initial model PM is shown in Fig. 7.6(a). Two basic abstraction operations are sequentially applied. The SESE fragment *sese*1 is eliminated, resulting in model PM_a, see Fig. 7.6(b). Then, the fragment *sese*2 is aggregated resulting model PM'_a, see Fig. 7.6(c).

7.1.7 Eshuis and Grefen

In [55] Eshuis and Grefen are challenged by the adaptation of process models to support interorganizational communication. The designed approach has two steps: 1) the process owner specifies internal activities to be aggregated, 2) the process consumer omits and hides unnecessary activities. The selection of activities to be abstracted is manual. Thereby, the paper focuses on the *how*, ignoring *why* and *when*. The paper selects activities as the abstraction object. Aggregation and elimination are employed as abstraction operations. The abstraction is hierarchical, order-preserving, yet ignores preservation of process non-functional properties. While the paper directly addresses the use case "Use Case 9: Adapt Process Model for an External Partner", the developed business process model abstraction indirectly supports the use cases



(a) Initial model PM

(b) Abstract model PM_a (c) Customized model PM'_a

Fig. 7.7. Illustration of the business process model abstraction approach developed by Eshuis and Grefen in [55].

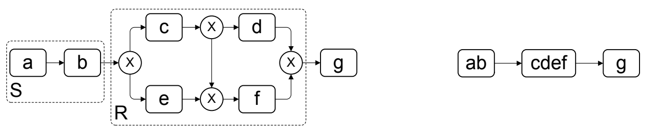
of the group “Group 1: Preserving Relevant Activities” and the use cases “Use Case 9: Adapt Process Model for an External Partner”, “Use Case 12: Get Process Quick View Respecting Ordering Constraints”, “Use Case 13: Get Process Quick View Respecting Roles”, “Use Case 15: Preserve Frequent Activities Summarizing Rare Activities”, and “Use Case 16: Get Particular Process Perspective”.

The processes are captured in UML Activity Diagrams. The formalization of a process model restricts models to well-structured ones, allowing AND- and XOR-blocks along with loops. The approach ensures that the resulting abstract models are order-preserving. The first phase is based solely on activity aggregation, concealing the private activities. Fig. 7.7(a) captures the initial model PM, where the user selects to aggregate activities b and d. Fig. 7.7(b) shows the abstraction result: activities b, c, and d are aggregated into bcd. Activity c is aggregated, as the abstraction method constructs order-preserving views. The second phase, customization, employs aggregation and elimination to preserve only activities demanded by the consumer. In the example model PM_a, see Fig. 7.7(b), the user preserves activities a, bcd, and g. Model PM'_a in Fig. 7.7 shows the final result of abstraction, where irrelevant activities are hidden.

7.1.8 Polyvyanyy et al.

In [122, 123] Polyvyanyy et al. study how process model decomposition supports business process model abstraction. The developed abstractions make use of aggregation as an abstraction operation and choose activities as abstraction objects. The papers superficially discuss *why* and *when*, bringing *how* in the focus. The developed abstraction supports the use cases of the group “Group 1: Preserving Relevant Activities”, as well as the use cases “Use Case 9: Adapt Process Model for an External Partner”, “Use Case 12: Get Process Quick View Respecting Ordering Constraints”, “Use Case 13: Get Process Quick View Respecting Roles”, “Use Case 15: Preserve Frequent Activities Summarizing Rare Activities”, and “Use Case 16: Get Particular Process Perspective”.

The paper formalizes a process model as a graph, where nodes are activities and gateways (ANDs and XORs), and edges correspond to the control flow.



(a) Initial model PM (b) Abstarct model PM_a

Fig. 7.8. Illustration of the business process model abstraction approach developed by Polyvyanyy et al. in [122, 123].

However, the obtained results can be extended for BPMN, see [142]. While [122] decomposes the model into fragments with exactly one incoming and exactly one outgoing edge, [123] seeks for fragments having exactly one entry node and exactly one exit node. The latter approach results in more fine-grained decomposition making the abstraction more flexible. The proposed business process model abstractions are order-preserving and hierarchical. The authors do not discuss preservation of process non-functional properties. The approach is illustrated by the example in Fig. 7.8. Fig. 7.8(a) shows the initial model PM, where fragments S and R can be aggregated. Fig. 7.8(b) presents the abstraction result with the given fragments being aggregated.

7.1.9 Summary

Table 7.1 summarizes the main properties of the aforementioned abstractions. One can notice that the majority of business process model abstraction approaches puts the focus on the *how* component, leaving *why* and *when* out of scope. However, even within the *how* aspect the focus is on the *structural* side of the model transformation. Typically, the decision which activities should be abstracted is delegated to the user. Furthermore, the business meaning of the coarse-grained activities created during abstraction has to be defined by the user as well. In this setting, it is important to realize if users can link the abstraction methods to their business needs without a detailed study of the *why* and *when*.

As can be seen, activities dominate the “abstraction object” column of Table 7.1. This confirms that in practice the end users perceive activities to be in the center of process model abstraction. The observation is supported by the analysis of the use cases addressed by the existing techniques. The majority focuses on activities. Indeed, most approaches are capable of supporting use cases of the group “Group 1: Preserving Relevant Activities”, as well as the use cases “Get Process Quick View Respecting Ordering Constraints” and “Get Process Quick View Respecting Roles”.

Finally, Table 7.1 bears witness of the fact that the elimination π and aggregation σ are used relatively homogeneously. However, the aggregation prevails the table, being used in each abstraction method, but one.

7.2 Potential Abstraction Enablers

The second tier of related research includes a series of papers that do not target business process model abstraction specifically, yet the research on abstraction can profit from their insights. This category of contributions include formal methods for process modeling, study of model properties and transformations. In particular, we refer to the works on process model transformation and workflow inheritance.

Name	**Why**	**When**	**How**	π	σ	**Supported use cases**	**Abstraction Object**
Cardoso et al.	+	+	+	–	+	12, 13, 15	activity
Liu and Shen	–	–	+	–	+	1–4, 9, 12, 13, 15	activity
Chiu et al.	–	–	+	+	–	9	activity
Pankratius and Stucky	–	–	+	+	+	1–4, 9, 12, 13, 15, 16	model element
Günther and van der Aalst	+	+	+	+	+	1–8, 12, 13, 15, 16	edge, activity
Bobrik et al.	–	–	+	+	+	1–4, 9, 12, 13, 15, 16	activity
Eshuis and Grefen	–	–	+	+	+	1–4, 9, 12, 13, 15, 16	activity
Polyvyanyy et al.	–	–	+	–	+	1–4, 9, 12, 13, 15, 16	activity

Table 7.1. Overview of existing business process model abstraction methods.

Among the papers on process model transformation a vast share of works concentrates on reduction rules and process model decomposition. Within decades the Petri net community studied reduction rule sets facilitating analysis of process models. In [21, 22] Berthelot suggested a rule set capable of reducing live and bounded marked graphs to a single transition. Murata proposed reduction rules preserving the liveness, safeness, and boundedness properties in [108]. Desel and Esparza came up with a complete set of reduction rules for free-choice Petri nets, see [44]. [135] developed a set of graph reduction rules for identification of structural conflicts in process model. Recently, van Dongen and Mendling used reduction rule sets for analysis of process model soundness, see [50, 107]. Such rules have also been defined for workflow graphs [135], EPCs [50, 102] and YAWL [167].

In the context of abstraction, for every set of reduction rules it is essential to show that one of the following statements holds:

– The set of reduction rules is complete to abstract a process model of an arbitrary structure into one node.
– There is a description of the class of models that can be reduced to one node by this set of rules.

As in practice process models have an arbitrary, non-compositional structure, the above requirements are highly relevant to reflect on the applicability of a reduction rules set.

Process model decomposition approaches are free of this limitation: they enable unique decomposition of a process model into a hierarchy of fragments. The generic results for decomposition of graphs have been obtained by Johnson, Pearson, and Pingali in [79] and Tarjan and Valdes in [152]. Later

Vanhatalo et al. adapted these decomposition techniques for business processes [154, 155]. We argue that both reduction techniques and decomposition techniques support elimination and aggregation as the most prominent forms of abstraction.

Preservation of process model behavior during model transformation can be defined in several ways. In [4] van der Aalst and Basten use Petri nets as process formalism and define the notion of process inheritance, which relates to the behavioral aspect of the model. In particular, they identify four types of inheritance: protocol, projection, protocol/projection, and life-cycle inheritance. Further, the authors specify operations on models maintaining inheritance property. The defined operations can be used in different context, for instance, refinement of business process models [6]. The contribution of [4] relates to business process model abstraction, as models PM and PM_a can be seen as those belonging to one of the proposed inheritance relations.

7.3 Farther Afield

We finalize the discussion of the related work looking further afield and considering challenges of model management related to business process model abstraction. In particular, we identify several substreams of the research that relate to abstraction: the research on process model modularization, model refactoring, and relations between already existing models of one business process. The remainder of this section elaborates on these issues one by one.

Business process model abstraction manages the model complexity. Industrial business process modeling notations, like EPCs [80] and BPMN [113], tackle this problem allowing modularization in process models. For instance, BPMN has a *subprocess* concept, while EPCs use the term *process path*. Both languages allow to capture a process part in a model and subsequently reference it in other models. The principles and properties of process model modularization has been studied by Reijers and Mendling in a series of papers, see [105, 128, 129].

Recently, the BPM community investigated methods for process model refactoring. While various refactoring methods vary in goals, it is always the case that one process model is transformed into another describing the same process at the same level of abstraction. For instance, in [117, 119] Polyvyanyy et al. argue if and how unstructured process models can be transformed into well-structured. Similarly, Fahland and van der Aalst proposed a method for simplification of process models mined from logs, see [58]. Leopold, Smirnov, and Mendling discuss the refactoring of activity labels in [91].

While business process model abstraction delivers an abstract process representation given a starting model, multiple models of one process might be already in place. Such models reflect the views of various stakeholders on one business process [83]. Typically, numerous relations exist between these models, which leads to new challenges in model management.

First, it is crucial to manage the consistency of multiple process specifications. In [49] Dijkman et al. addressed the problem of information system design, where multiple stakeholders contribute to the system creation. The authors developed a framework suggesting 1) the use of basic concepts shared by the stakeholders to be facilitate communication, 2) means to manage consistency between relations of system specifications. The paper illustrates the applicability of the approach, establishing relations between structural and behavioral models. Recently, Weidlich et al. investigated the consistency of models formalizing one business process, see [158, 159]. While [159] focuses on the behavioral aspects of process models, [158] studies methods for identification of correspondence relations between elements of different specifications. The existence of multiple models for one object is inherent not only for BPM, but also for software engineering and requirements engineering [82]. Finkelstein et al. in [60] argued that model inconsistencies are inevitable and suggested a formal approach to deal with them. [110] suggested a framework for managing multiple views and their inconsistencies in the context of requirements engineering. The aforementioned papers study intra- and intermodel relations between model elements. We believe that their results can be useful in the context of abstraction's *when* and *how*.

Second, the multiple models of a process can be seen as its "partial" models: each model presents one perspective on the subject. Integration of such partial models into one facilitates more comprehensive process understanding. Preuner, Conrad, and Schrefl designed a method for integration of models that capture business object life cycles [125]. Two integration types are distinguished: integration of type hierarchies and integration of behavior of object types. The integration makes use of generalization/specialization and extension/refinement operations. In [106] Mendling and Simon propose another approach for process view integration. The approach implies that one business process can have several specifications, each—an EPC model. It assumes that the correspondences between elements of different models are known. Given two views of one business process and the element correspondences, the approach delivers an integrated process model. The relations between the integrated model and the initial models can be traced back to relation between PM and PM_a, respectively.

7.4 Discussion

We conclude the discussion of the related work establishing a connection between the discussed contributions and the findings of Chapter 3—the abstraction framework and the use case catalog. In particular, we position the existing business process model abstraction methods along with the abstraction enablers against the identified use cases and the *why*, *when*, and *how* aspects of abstraction. As an outcome, we observe the "big picture" of the existing methods. Section 7.4.1 describes the abstraction retrospective, while Section 7.4.2 presents its perspective.

7.4.1 Retrospective

Table 7.2 provides correspondences between the abstraction use cases from the catalog, see Section 3.4, and the existing techniques for business process model abstraction. Notice that the table refers to the works directly addressing process model abstraction along with the techniques potentially helpful in the abstraction context. A table row specifies the papers related to one or several use cases from the catalog. The columns distinguish the papers according to the three aspects: *why*, *when*, and *how*. The *how* aspect further refines the classification into papers on process model transformation and papers specifying how to apply algorithms for business process model abstraction.

Table 7.2 allocates several papers on dedicated rows by which we emphasize the role of these works. [55, 66, 121] propose *comprehensive* solutions for particular abstraction use cases. By this, we mean that the papers discuss all the three aspects of process model abstraction. We emphasize [32], as it discusses not only a structural perspective on model transformations, but also the principles for the evaluation of non-functional properties.

Finally, we relate the contributions of this thesis to the abstraction use cases and the three abstraction aspects: *why*, *when*, and *how*. First, we notice that the *when* aspect was barely addressed by this work. The *when* aspect has been taken into account by the threshold values that parameterize abstraction algorithms introduced in Chapters 4–6. The *why* of abstraction was tackled by Chapter 4 and Chapter 5 arguing about activity aggregation criteria. Altogether, the work has a sharp focus on the abstraction *how* discussing it through all the core chapters, Chapters 4–6. With respect to the supported abstraction use cases, the thesis elaborates on the use cases of the group "Group 4: Obtaining a Process Quick View" that are of the great interest for practitioners. In particular, this thesis advances the existing abstraction methods with respect to semantics of process model elements.

7.4.2 Perspective

Another look at the Table 7.2 reveals a disproportion in the related work: as the *how* is thoroughly investigated, the *why* and *when* are hardly touched. The *why* calls for research on how the user can formulate the abstraction goal and what is the frontier of abstraction application. The *when* question is concerned with the definition of the *sign* function, which can be non-trivial, e.g., consider Use Case 9. We conjecture that a more complete coverage of these components will simplify the uptake of the available techniques by industry.

As we argued earlier, the *how* also displays some white spots. For instance, a high user demand for Use Case 12 is a strong motivation to develop techniques delivering aggregations of activities that semantically belong together.

Use case name	**Why**	**When**	**How**	
			Model transformation	**Abstraction algorithm**
Use Case 1–4			[4, 21, 22, 28, 29, 30, 44, 50, 55, 66, 79, 106, 107, 108, 115, 122, 123, 125, 135, 152, 158]	[30, 94, 141]
Use Case 5–8			[21, 22, 44, 50, 66, 79, 107, 108, 135, 152]	
Use Case 9	[55]	[55]	[55] [4, 21, 22, 28, 29, 30, 44, 50, 79, 94, 107, 106, 108, 115, 122, 123, 125, 135, 141, 152, 154, 155, 158]	[55] [30, 94, 141]
Use Case 10–11			[4, 21, 22, 44, 50, 79, 107, 106, 108, 125, 135, 152, 154, 155, 158]	[30]
Use Case 12	 [66, 121]	 [66, 121]	[32] [66, 121] [4, 21, 22, 28, 29, 30, 44, 50, 55, 66, 79, 94, 107, 106, 108, 115, 122, 123, 125, 135, 141, 152, 154, 155, 158]	[32] [66, 121] [28, 29, 30, 94, 122, 123, 141, 142]
Use Case 13,15			[4, 21, 22, 28, 29, 30, 32, 44, 50, 55, 66, 79, 94, 107, 106, 108, 115, 121, 122, 123, 125, 135, 141, 152, 154, 155, 158]	[28, 29, 30, 94, 141, 142]
Use Case 16			[4, 21, 22, 28, 29, 30, 44, 50, 55, 66, 79, 107, 106, 108, 115, 122, 123, 125, 135, 152, 158]	[28, 29, 30]

Table 7.2. Existing techniques related to identified business process model abstraction use cases.

By semantics we mean the domain semantics of the model elements. A related problem is how to label an aggregating activity delivered by the abstraction. Finally, it is interesting to determine, when abstraction methods are preferable over alternative visualization techniques and textual reports.

Furthermore, even the referenced techniques provide only partial support for some of the use cases. For instance, although [55] proposes an abstraction method covering all the aspects of abstraction, the approach is only capable of handling well-structured process models. Similarly, a business process model abstraction developed in [121] is restricted by a set of rules that enable it. Finally, in [66] Günther and Van der Aalst propose an approach that supports Use Case 12, but it is only capable of handling process models in a very simplistic notation. While the contributions of all these papers are acknowledged, it is also apparent that their applicability can be enhanced.

Table 7.2 illustrates that abstraction has been studied by a number of researchers and that various techniques have become available in the past years. Yet, there is still considerable room for improvement and extension, on the one hand by tackling the almost unexplored *why* and *when* aspects and one the other hand by extending the range of advanced techniques addressing the *how*.

7.5 Summary

This chapter outlined the research related to business process model abstraction. Structuring this research in three tiers, the chapter discussed the existing abstraction methods, formal techniques that might enable abstraction, and model management challenges similar to abstraction. The contributions of the first group were explained in detail using the vocabulary of the framework introduced in Section 3.2 and related to the abstraction use cases identified in Section 3.4. The papers and articles of the first two tiers are compared. The comparison reveals a “big picture” view on business process model abstraction. Finally, the third tier puts business process model abstraction in the context of other process model management tasks.

8 Conclusion

The current chapter concludes this thesis. The chapter summarizes the core contributions, briefly discusses the relation of abstraction to other research areas, and gives an outline of the future work.

8.1 Summary of the Results

This doctoral thesis investigated the problem of business process model abstraction and developed methods addressing the actual user demand in abstraction techniques. The core contributions of this thesis are as follows.

Business Process Model Abstraction Framework

We have systematically investigated business process model abstraction and proposed a new framework that organizes this domain. Our study includes the analysis of the related work, as well as the opinions of the BPM practitioners. As an outcome, the framework brings together various issues of business process model abstraction and provides a coherent view on process model abstraction. Using the framework we have compared the existing abstraction methods, identified the well researched areas of abstraction, and discovered the research gaps.

Catalog of Business Process Model Abstraction Use Cases

To the best of our knowledge the existing research papers are limited to the investigation of isolated abstraction use cases, providing no reflection on the “big picture” of abstraction. To fill this research gap we conducted an exploratory study of the user demand for business process model abstraction and drew up a catalog of abstraction use cases. The catalog describes each use case, organizes them into groups, and ranks the use cases according to the user demand. We notice that the catalog is an outcome of the study that analyzed the related research contributions and the interviews with BPM experts from industry: consultants, software vendors, and end users.

Abstraction of a Process Model According to Model Structure

This thesis developed two abstraction methods based on process model structure analysis. While the first method builds on the well known idea of model transformation by means of structural patterns, the second approach relies on the process model decomposition. In this way we extend the body of knowledge with the two novel abstraction algorithms. The introduction of the two approaches is complemented by the discussion of their properties and the application context.

Discovery of Semantically Related Activities within a Process Model

Since several business process model abstraction use cases demand obtaining coarse-grained activities, we designed two novel methods for activity aggregation. Both methods aim at delivering activity groups with the self-contained business semantics. In this vein the methods procure such activity groups, where a group corresponds to a coarse-grained activity of the abstract process model. As both methods are agnostic to the process model control flow relation, they leverage alternative information sources. The first method analyzes the activity meronymy relation provided by an ontology, while the second method considers non-control flow model elements. The thesis provided an empirical argument witnessing that the aggregation methods perform well in practice. Against this background, the thesis extends the research on business process model abstraction in the new direction considering the semantics of process model elements.

Control Flow Discovery within a Non-Hierarchical Business Process Model Abstraction

Motivated by the demand for non-hierarchical refinement and generalization, see [49, 60, 82, 157], the current thesis developed an approach for non-hierarchical abstraction. Assuming that groups of related activities in the initial process model are in place (see the previous contribution), the approach focuses on the synthesis of the abstract model control flow relation. The use of process model behavioral abstraction, behavioral profiles, allows to relax assumptions about groups of related activities: one activity may belong to multiple groups and an activity group can be freely spread over the model. Against this background, the engineered abstraction approach is novel and can be seen as the generalization of the existing structure-based abstraction methods. We complement the conceptual discussion of the abstraction approach by the presentation of its software implementation.

8.2 Discussion

Business process model abstraction enjoys the close vicinity of several research directions that mutually enrich each other. Briefly positioning the abstraction against such topics, this section exposes the “big picture” of abstraction.

Business process model abstraction is motivated by the user demand for process models with an appropriate level of model element granularity. At the same time, the models should fulfill other requirements. A broad spectrum of research studies the quality aspects of process models. Vivid quality aspect examples are the quality of model element labels [91, 104], structural complexity of models [101], and behavioral correctness of process models [2]. Among various process model quality aspects we distinguish process model modularization that studies the principles of model modularization and its impact on the model understandability. Modularization directly deals with the challenges of model complexity management. In this context, business process model abstraction can be seen as the tool delivering appropriate model complexity and assuring model quality.

As it follows from this thesis, model transformation methods are the prominent tool of business process model abstraction. The Petri net community, along with the BPM researchers, has thoroughly investigated model transformation methods, e.g., see [4, 6, 21, 22, 50, 108, 135] to name but a few. Obviously, business process model abstraction profits from this knowledge: various abstraction use cases can be supported through the adaption of the existing model transformations.

As soon as the user requests an abstract process model, one can treat business process model abstraction as the model synthesis problem. Since our notion of a process model explicitly supports concurrency, we consider the research on Petri net synthesis to be related. While this thesis adapted and extended methods for the synthesis of models from behavioral relations, see Section 6.5.2, abstraction may synthesize models from states as well. Against this background, model synthesis methods can procure the formal foundation for business process model abstraction.

While business process model abstraction investigates model synthesis, it also studies the relations between the initial detailed model and its abstract counterpart. Thereby, abstraction explores the relations between various models of one system, a business process in our case. Several software engineering contributions discuss the relations between different models of one system, e.g., see [60]. The specific aspects of process model consistency have been addressed as well [49, 160]. As the two areas are rather close, we conclude that business process model abstraction and the research on model consistency mutually enrich each other.

8.3 Future Work

Putting business process model abstraction at the core of the thesis, we make several scoping decisions. As an outcome, there are research directions that remain out of scope of this thesis. The remainder of this chapter presents the emerging challenges of business process model abstraction to be addressed by the future work.

First, we scope our study to the analysis of the control flow model elements. We studied the impact of abstraction on the activities and the ordering constraints between them. Meanwhile, process models are often enriched with non-control flow elements capturing, for instance, data or organizational roles. While we have studied how non-control flow model elements facilitate activity aggregation, the impact of abstraction on such elements calls for investigation. Second, this thesis argues about the process models that can be mapped to sound free choice workflow nets. Sometimes we impose further restrictions requiring models to be well-structured. In this setting the natural next step is to widen the class of abstracted models. Third, this thesis explored the features of abstraction as a model transformation. In particular, we argued which properties of the abstract process model can be guaranteed given the initial model, e.g., we discussed abstractions that preserve soundness or are order-preserving. At the same time, we leave open the question which statements about the initial process model can be made, once we possess an abstract model.

Further, this thesis has identified a catalog of 15 abstraction use cases. Subsequently, we have scoped our research to the methods supporting the use cases of the group "Group 4: Obtaining a Process Quick View". The use cases with no developed abstraction methods are the obvious future research candidates.

The dominating share of our discussion excelled the role of process models in the abstraction context. In this way, the process instances remain out of scope. However, the relations between abstract process models and process instances can be quite complex [28, 160, 144]. One can suggest several scenarios where the relation between abstract model and the instance has to be defined. Consider, for instance, the capabilities of monitoring process instances by means of abstract process models. This setting opens a new unexplored research field.

References

1. W. M. P. van der Aalst. A Class of Petri Net for Modeling and Analyzing Business Processes. Technical Report 95/26, Eindhoven University of Technology, Eindhoven, 1995.
2. W. M. P. van der Aalst. Verification of Workflow Nets. In *Application and Theory of Petri Nets 1997*, pages 407–426, Berlin, Germany, 1997. Springer.
3. W. M. P. van der Aalst. The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.
4. W. M. P. van der Aalst and T. Basten. Life-Cycle Inheritance: A Petri-Net-Based Approach. In *ICATPN 1997*, LNCS, pages 62–81, London, UK, 1997. Springer.
5. W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske. Business Process Management: A Survey. In *BPM 2003*, volume 2678 of *LNCS*, pages 1–12. Springer, 2003.
6. W. M. P. van der Aalst, N. Lohmann, P. Massuthe, Ch. Stahl, and K. Wolf. From public views to private views—correctness-by-design for services. In *WS-FM 2007*, volume 4937 of *LNCS*, pages 139–153. Springer, 2007.
7. W. M. P. van der Aalst and A. H. M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.
8. W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14:5–51, July 2003.
9. W. M. P. van der Aalst and K. van Hee. *Workflow Management: Models, Methods, and Systems.* MIT Press, 2002.
10. W. M. P. van der Aalst, A. J. M. M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.
11. W. M. P. van der Aalst, M. Weske, and D. Grünbauer. Case Handling: a New Paradigm for Business Process Support. *Data and Knowledge Engineering*, 53(2):129–162, 2005.
12. A. V. Aho and J. D. Ullman. *Foundations of Computer Science.* W. Freeman and Company, New York, USA, 1995.
13. A. Awad, G. Decker, and M. Weske. Efficient Compliance Checking Using BPMN-Q and Temporal Logic. In *BPM 2008*, volume 5240 of *LNCS*, pages 326–341. Springer, 2008.

14. A. Awad, S. Smirnov, and M. Weske. Resolution of Compliance Violation in Business Process Models: A Planning-Based Approach. In *OTM Conferences 2009*, volume 5870 of *LNCS*, pages 6–23. Springer, 2009.
15. A. Awad, M. Weidlich, and M. Weske. Visually Specifying Compliance Rules and Explaining their Violations for Business Processes. *Journal of Visual Languages and Computing*, 22(1):30–55, 2011.
16. R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.
17. F. Barbier, B. Henderson-Sellers, A. Le Parc-Lacayrelle, and J.-M. Bruel. Formalization of the Whole-Part Relationship in the Unified Modeling Language. *IEEE Transactions on Software Engineering*, 29(5):459–470, 2003.
18. J. Becker, M. Kugeler, and M. Rosemann. *Process Management: A Guide for the Design of Business Processes*. Springer, Berlin, Germany, 2003.
19. J. Becker and D. Kuropka. Topic-based Vector Space Model. In *BIS 2003*, pages 7–12. GI, 2003.
20. J. Becker, M. Rosemann, and Ch. von Uthmann. Guidelines of Business Process Modeling. In *BPM 2000*, volume 1806 of *LNCS*, pages 30–49. Springer, 2000.
21. G. Berthelot. Checking Properties of Nets using Transformation. In *Advances in Petri Nets 1985*, volume 222 of *LNCS*, pages 19–40, London, UK, 1986. Springer.
22. G. Berthelot. Transformations and Decompositions of Nets. In *Advances in Petri nets 1986*, volume 254 of *LNCS*, pages 359–376, London, UK, 1987. Springer.
23. E. Best. Structure Theory of Petri Nets: the Free Choice Hiatus. In *Advances in Petri Nets 1986*, volume 254 of *LNCS*, pages 168–205. Springer, 1986.
24. E. Best, R. R. Devillers, A. Kiehn, and L. Pomello. Concurrent Bisimulations in Petri Nets. *Acta Informatica*, 28(3):231–264, 1991.
25. J. C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 1981.
26. K. Bhattacharya, C. E. Gerede, R. Hull, R. Liu, and J. Su. Towards Formal Analysis of Artifact-Centric Business Process Models. In *BPM 2007*, volume 4714 of *LNCS*, pages 288–304. Springer, 2007.
27. K. Bhattacharya, R. Guttman, K. Lyman, F. F. Heath III, S. Kumaran, P. Nandi, F. Y. Wu, P. Athma, Ch. Freiberg, L. Johannsen, and A. Staudt. A Model-driven Approach to Industrializing Discovery Processes in Pharmaceutical Research. *IBM Systems Journal*, 44(1):145–162, 2005.
28. R. Bobrik, Th. Bauer, and M. Reichert. Proviado—Personalized and Configurable Visualizations of Business Processes. In *EC-Web*, pages 61–71, 2006.
29. R. Bobrik, M. Reichert, and T. Bauer. Parameterizable Views for Process Visualization. Technical Report TR-CTIT-07-37, Centre for Telematics and Information Technology, University of Twente, Enschede, April 2007.
30. R. Bobrik, M. Reichert, and T. Bauer. View-Based Process Visualization. In *BPM 2007*, volume 4714 of *LNCS*, pages 88–95, Berlin, 2007. Springer.
31. K. E. Brassel and R. Weibel. A Review and Conceptual Framework of Automated Map Generalization. *International Journal of Geographical Information Science*, 2(3):229–244, 1988.
32. J. Cardoso, J. Miller, A. Sheth, and J. Arnold. Modeling Quality of Service for Workflows and Web Service Processes. Technical report, University of Georgia, 2002. Web Services.

33. F. Casati and M.-Ch. Shan. Semantic Analysis of Business Process Executions. In *EDBT 2002*, volume 2287 of *LNCS*, pages 287–296, London, UK, 2002. Springer.
34. D. K. W. Chiu, S. C. Cheung, S. Till, K. Karlapalem, Q. Li, and E. Kafeza. Workflow View Driven Cross-Organizational Interoperability in a Web Service Environment. *Information Technology and Management*, 5(3–4):221–250, 2004.
35. D. K. W. Chiu, K. Karlapalem, Q. Li, and E. Kafeza. Workflow View Based E-Contracts in a Cross-Organizational E-Services Environment. *Distributed and Parallel Databases*, 12(2–3):193–216, 2002.
36. C. M. Christensen. *The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail.* Harvard Business School Press, Boston, 1997.
37. C. Combi and R. Posenato. Controllability in Temporal Conceptual Workflow Schemata. In *BPM 2009*, volume 5701 of *LNCS*, pages 64–79. Springer, 2009.
38. J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Deriving Petri Nets from Finite Transition Systems. *IEEE TC*, 47(8):859–882, August 1998.
39. P. Dadam, K. Kuhn, M. Reichert, Th. Beuter, and M. Nathe. ADEPT: Ein integrierender Ansatz zur Entwicklung flexibler, zuverlässiger, kooperierender Assistenzsysteme in klinischen Anwendungsumgebungen. In *GI Jahrestagung*, pages 677–686, 1995.
40. R. Dapoigny and P. Barlatier. Towards an Ontological Modeling with Dependent Types: Application to Part-Whole Relations. In *ER 2009*, volume 5829 of *LNCS*, pages 145–158, Gramado, Brazil, 2009. Springer.
41. T. Davenport. *Process Innovation: Reengineering Work through Information Technology.* Harvard Business School Press, Boston, MA, USA, 1993.
42. G. Decker, H. Overdick, and M. Weske. Oryx - Sharing Conceptual Models on the Web. In *ER 2008*, volume 5231 of *LNCS*, pages 536–537. Springer, 2008.
43. J. Dehnert and W. M. P. van der Aalst. Bridging The Gap Between Business Models And Workflow Specifications. *International Journal of Cooperative Information Systems*, 13(3):289–332, 2004.
44. J. Desel and J. Esparza. *Free Choice Petri Nets.* Cambridge University Press, New York, NY, USA, 1995.
45. C. Di Francescomarino, A. Marchetto, and P. Tonella. Cluster-based Modularization of Processes Recovered from Web Applications. *Journal of Software Maintenance and Evolution: Research and Practice*, 2010.
46. R. M. Dijkman, M. Dumas, and L. García-Bañuelos. Graph Matching Algorithms for Business Process Model Similarity Search. In *BPM 2009*, LNCS, pages 48–63, Berlin, 2009. Springer.
47. R. M. Dijkman, M. Dumas, and Ch. Ouyang. Semantics and Analysis of Business Process Models in BPMN. *Information and Software Technology*, 50(12):1281–1294, 2008.
48. R. M. Dijkman, M. M. Dumas, B. F. van Dongen, R. Käärik, and J. Mendling. Similarity of Business Process Models: Metrics and Evaluation. *Information Systems*, 36(2):498–516, 2011.
49. R. M. Dijkman, D. A. C. Quartel, and M. J. van Sinderen. Consistency in Multi-Viewpoint Design of Enterprise Information Systems. *Information and Software Technology*, 50(7–8):737–752, 2008.
50. B. F. van Dongen, M. H. Jansen-Vullers, H. M. W. Verbeek, and W. M. P. van der Aalst. Verification of the SAP Reference Models using EPC Reduction, State-Space Analysis, and Invariants. *Computers in Industry*, 58(6):578–601, 2007.

51. B. F. van Dongen, J. Mendling, and W. M. P. van der Aalst. Structural Patterns for Soundness of Business Process Models. In *EDOC 2006*, pages 116–128. IEEE Computer Society, 2006.
52. M. Dumas, L. García-Bañuelos, and R. M. Dijkman. Similarity Search of Business Process Models. *IEEE Data Engineering Bulletin*, 32(3):23–28, 2009.
53. M. Dumas, L. García-Bañuelos, A. Polyvyanyy, Y. Yang, and L. Zhang. Aggregate Quality of Service Computation for Composite Services. In *ICSOC 2010*, volume 6470 of *LNCS*, pages 213–227. Springer, 2010.
54. A. Ehrenfeucht and G. Rozenberg. Partial (Set) 2-structures. Part II: State Spaces of Concurrent Systems. *Acta Informatica*, 27:343–368, January 1990.
55. R. Eshuis and P. Grefen. Constructing Customized Process Views. *Data and Knowledge Engineering*, 64(2):419–438, 2008.
56. J. Esparza. Reduction and Synthesis of Live and Bounded Free Choice Petri Nets. *Information and Computation*, 114(1):50–87, 1994.
57. J. Esparza, S. Römer, and W. Vogler. An Improvement of McMillan's Unfolding Algorithm. *Formal Methods in System Design*, 20(3):285–310, 2002.
58. D. Fahland and W. M. P. van der Aalst. Simplifying Mined Process Models: An Approach Based on Unfoldings. In *BPM 2011*, volume 6896 of *LNCS*, pages 362–378. Springer, 2011.
59. D. Fahland, C. Favre, J. Koehler, N. Lohmann, H. Völzer, and K. Wolf. Analysis on Demand: Instantaneous Soundness Checking of Industrial Business Process Models. *Data and Knowledge Engineering*, 70(5):448–466, 2011.
60. A. C. W. Finkelstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency Handling in Multiperspective Specifications. *IEEE Transactions on Software Engineering*, 20(8):569–578, 1994.
61. D. S. Fussell, V. Ramachandran, and R. Thurimella. Finding Triconnected Components by Local Replacement. *SIAM Journal on Computing*, 22(3):587–616, 1993.
62. D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management: from Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3:119–153, April 1995.
63. Th. Gschwind, J. Koehler, and J. Wong. Applying Patterns during Business Process Modeling. In *BPM 2008*, volume 5240 of *LNCS*, pages 4–19. Springer, 2008.
64. G. Guizzardi. Modal Aspects of Object Types and Part-Whole Relations and the *de re/de dicto* Distinction. In *CAiSE 2007*, volume 4495 of *LNCS*, pages 5–20. Springer, 2007.
65. C. W. Günther and W. M. P. van der Aalst. Mining Activity Clusters from Low-Level Event Logs. Technical Report WP 165, Eindhoven University of Technology, Eindhoven, 2006.
66. C. W. Günther and W. M. P. van der Aalst. Fuzzy Mining-Adaptive Process Simplification Based on Multi-perspective Metrics. In *BPM 2007*, volume 4714 of *LNCS*, pages 328–343, Berlin, 2007. Springer.
67. C. Gutwenger and P. Mutzel. A Linear Time Implementation of SPQR-Trees. In *Graph Drawing 2001*, volume 1984 of *LNC*, pages 77–90. Springer, 2001.
68. A. Hallerbach, T. Bauer, and M. Reichert. Capturing Variability in Business Process Models: The Provop Approach. *Journal of Software Maintenance*, 22(6-7):519–546, 2010.
69. M. Hammer and J. Champy. *Reengineering the Corporation: A Manifesto for Business Revolution.* HarperBusiness, April 1994.

70. F. Harary. *Graph Theory.* Addison-Wesley, 1969.
71. P. Harmon. *Business Process Change, Second Edition: A Guide for Business Managers and BPM and Six Sigma Professionals.* Morgan Kaufmann, 2007.
72. J. Hartigan. *Clustering Algorithms.* John Wiley and Sons, New York, USA, 1975.
73. R. Hauser, M. Friess, J. M. Küster, and J. Vanhatalo. Combining Analysis of Unstructured Workflows with Transformation to Structured Workflows. In *EDOC 2006*, pages 129–140. IEEE Computer Society, 2006.
74. M. Hepp, F. Leymann, J. Domingue, A. Wahler, and D. Fensel. Semantic Business Process Management: A Vision Towards Using Semantic Web Services for Business Process Management. In *ICEBE 2005*, pages 535–540. IEEE CS, 2005.
75. J. E. Hopcroft and R. E. Tarjan. Dividing a Graph into Triconnected Components. *SIAM Journal on Computing*, 2(3):135–158, 1973.
76. R. Hull, N. Narendra, and A. Nigam. Facilitating Workflow Interoperation Using Artifact-Centric Hubs. *Service-Oriented Computing*, pages 1–18, 2009.
77. J. Hündling. *Modellierung von Qualittsmerkmalen fr Services.* PhD thesis, Hasso Plattner Institut, Potsdam, Germany, 2008.
78. T. Jin, J. Wang, N. Wu, M. La Rosa, and A. H. M. ter Hofstede. Efficient and Accurate Retrieval of Business Process Models through Indexing. In *OTM 2010*, volume 6426 of *LNCS*, pages 402–409. Springer, 2010.
79. R. Johnson, D. Pearson, and K. Pingali. The Program Structure Tree: Computing Control Regions in Linear Time. *SIGPLAN Not.*, 29:171–185, June 1994.
80. G. Keller, M. Nüttgens, and A. Scheer. Semantische Prozessmodellierung auf der Grundlage "Ereignisgesteuerter Prozessketten (EPK)". Technical Report Heft 89, Veröffentlichungen des Instituts für Wirtschaftsinformatik University of Saarland, 1992.
81. B. Kiepuszewski, A. H. M. ter Hofstede, and Ch. Bussler. On Structured Workflow Modelling. In *CAiSE 2000*, volume 1789 of *LNCS*, pages 431–445. Springer, 2000.
82. A. Knoepfel, B. Groene, and P. Tabeling. *Fundamental Modeling Concepts: Effective Communication of IT Systems.* John Wiley and Sons, Ltd., 2005.
83. J. Koehler, R. Hauser, J. M. Küster, K. Ryndina, J. Vanhatalo, and M. Wahler. The Role of Visual Modeling and Model Transformations in Business-driven Development. *Electronic Notes in Theoretical Computer Science*, 211:5–15, 2008.
84. M. Kunze, M. Weidlich, and M. Weske. Behavioral Similarity—A Proper Metric. In *BPM 2011*, volume 6896 of *LNCS*, pages 166–181. Springer, 2011.
85. D. Kuropka, P. Tröger, S. Staab, and M. Weske. *Semantic Service Provisioning.* Springer, 2008.
86. M. La Rosa. *Managing Variability in Process-Aware Information Systems.* PhD thesis, Queensland University of Technology, Brisbane, Australia, 2009.
87. C. Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.
88. J. M. Lau, C. Iochpe, L. Thom, and M. Reichert. Discovery and Analysis of Activity Pattern Co-occurrences in Business Process Models. In *ICEIS 2009*, pages 83–88, 2009.

89. R. Laue and J. Mendling. Structuredness and its Significance for Correctness of Process Models. *Information Systems and E-Business Management*, 8(3):287–307, 2010.
90. H. Leopold, J. Mendling, and Hajo A. Reijers. On the Automatic Labeling of Process Models. In *CAiSE 2011*, volume 6741 of *LNCS*, pages 512–520. Springer, 2011.
91. H. Leopold, S. Smirnov, and J. Mendling. Refactoring of Process Model Activity Labels. In *NLDB 2010*, volume 6177 of *LNCS*, pages 268–276. Springer, 2010.
92. F. Leymann and D. Roller. *Production Workflow—Concepts and Techniques*. Prentice Hall, 2000.
93. C. Li, M. Reichert, and A. Wombacher. Mining Business Process Variants: Challenges, Scenarios, Algorithms. *Data and Knowledge Engineering*, 70(5):409–434, 2011.
94. D. Liu and M. Shen. Workflow Modeling for Virtual Processes: an Order-Preserving Process-View Approach. *Information Systems*, 28(6):505–532, 2003.
95. Th. W. Malone, K. Crowston, and G. A. Herman. *Organizing Business Knowledge: The MIT Process Handbook*. The MIT Press, Cambridge, MA, USA, 1st edition, 2003.
96. P. Massuthe, A. Serebrenik, N. Sidorova, and K. Wolf. Can I Find a Partner? Undecidability of Partner Existence for Open Nets. *Information Processing Letters*, 108(6):374–378, 2008.
97. R.M. McConnell and F. de Montgolfier. Linear-time Modular Decomposition of Directed Graphs. *Discrete Applied Mathematics*, 145(2):198–209, 2005.
98. R. B. McMaster and S. K. Shea. Generalization in Digital Cartography. In *Resource Publication of the Association of American Geographers*, Washington D.C., USA, 1992.
99. K. L. McMillan. A Technique of State Space Search Based on Unfolding. *Formal Methods in System Design*, 6(1):45–65, 1995.
100. A. K. A. De Medeiros, W. M. P. van der Aalst, and C. Pedrinaci. Semantic Process Mining Tools: Core Building Blocks. In *ECIS 2008*, pages 1953–1964, Galway, Ireland, 2008.
101. J. Mendling. *Detection and Prediction of Errors in EPC Business Process Models*. PhD thesis, Institute of Information Systems and New Media Vienna University of Economics and Business Administration, Vienna, Austria, 2007.
102. J. Mendling and W. M. P. van der Aalst. Formalization and Verification of EPCs with OR-Joins Based on State and Context. In *CAiSE 2007*, volume 4495 of *LNCS*, pages 439–453, Trondheim, Norway, 2007. Springer.
103. J. Mendling, H. A. Reijers, and W. M. P. van der Aalst. Seven Process Modeling Guidelines (7pmg). *Information and Software Technology*, 52(2):127–136, 2010.
104. J. Mendling, H. A. Reijers, and J. Recker. Activity Labeling in Process Modeling: Empirical Insights and Recommendations. *Information Systems*, 35(4):467–482, 2010.
105. J. Mendling, H.A. Reijers, and J. Cardoso. What Makes Process Models Understandable? In *BPM 2007*, volume 4714 of *LNCS*, pages 48–63. Springer, 2007.
106. J. Mendling and C. Simon. Business Process Design by View Integration. In *Business Process Management Workshops 2006*, volume 4103 of *LNCS*, pages 55–64. Springer, 2006.

107. J. Mendling, H. Verbeek, B. F. van Dongen, W. M. P. van der Aalst, and G. Neumann. Detection and Prediction of Errors in EPCs of the SAP Reference Model. *Data and Knowledge Engineering*, 64(1):312–329, 2008.
108. T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.
109. B. G. Nickerson and H. R. Freeman. Development of a Rule-based System for Automatic Map Generalization. In *ISSDH*, pages 537–556, Seattle, Washington, USA, January 1986.
110. B. Nuseibeh, J. Kramer, and A. Finkelstein. A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification. *IEEE Transactions on Software Engineering*, 20(10):760–773, 1994.
111. OASIS. Web Services Business Process Execution Language Version 2.0, April 2007.
112. OMG. *Meta Object Facility (MOF) Core Specification*, 2.0 edition, January 2006.
113. OMG. Business Process Modeling Notation (BPMN) Version 1.2, January 2009.
114. OMG. Omg unified modeling language (omg uml) 2.3, May 2010.
115. V. Pankratius and W. Stucky. A Formal Foundation for Workflow Composition, Workflow View Definition, and Workflow Normalization based on Petri Nets. In *APCCM 2005*, pages 79–88, Darlinghurst, Australia, 2005. Australian Computer Society, Inc.
116. C. A. Petri. *Kommunikation mit Automaten.* PhD thesis, Institut für instrumentelle Mathematik, Bonn, Germany, 1962.
117. A. Polyvyanyy, L. García-Bañuelos, and M. Dumas. Structuring Acyclic Process Models. In *BPM 2010*, volume 6336 of *LNCS*, pages 276–293. Springer, 2010.
118. A. Polyvyanyy, L. García-Bañuelos, and M. Dumas. Structuring Acyclic Process Models. *Information Systems*, 2011.
119. A. Polyvyanyy, L. García-Bañuelos, D. Fahland, and M. Weske. Maximal structuring of acyclic process models. *CoRR*, abs/1108.2384, 2011.
120. A. Polyvyanyy, S. Smirnov, and M. Weske. Process Model Abstraction: A Slider Approach. In *EDOC 2008*, pages 325–331, 2008.
121. A. Polyvyanyy, S. Smirnov, and M. Weske. Reducing Complexity of Large EPCs. In *EPK 2008 GI-Workshop*, Saarbrücken, Germany, 11 2008.
122. A. Polyvyanyy, S. Smirnov, and M. Weske. On Application of Structural Decomposition for Process Model Abstraction. In *BPSC 2009*, pages 110–122, Leipzig, 2009.
123. A. Polyvyanyy, S. Smirnov, and M. Weske. The Triconnected Abstraction of Process Models. In *BPM 2009*, volume 5701 of *LNCS*, pages 229–244, Ulm, Germany, 2009. Springer.
124. A. Polyvyanyy, J. Vanhatalo, and H. Völzer. Simplified Computation and Generalization of the Refined Process Structure Tree. In *WS-FM 2010*, volume 6551 of *LNCS*, pages 25–41. Springer, 2011.
125. G. Preuner, S. Conrad, and M. Schrefl. View Integration of Behavior in Object-Oriented Databases. *Data and Knowledge Engineering*, 36(2):153–183, 2001.
126. M. Rastrepkina. Managing Variability in Process Models by Structural Decomposition. In *BPMN*, volume 67 of *LNBIP*, pages 106–113. Springer, 2010.

127. M. Reichert and P. Dadam. ADEPT$_{\text{flex}}$-Supporting Dynamic Changes of Workflows Without Losing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.
128. H. A. Reijers and J. Mendling. Modularity in Process Models: Review and Effects. In *BPM 2008*, volume 5240 of *LNCS*, pages 20–35, Milan, Italy, 2008. Springer.
129. H. A. Reijers, J. Mendling, and R. M. Dijkman. On the Usefulness of Subprocesses in Business Process Models. BPM Center Report BPM-10-03, BPMcenter.org, 2010.
130. H. A. Reijers, J. H. M. Rigter, and W. M. P. van der Aalst. The Case Handling Case. *International Journal Cooperative Information Systems*, 12(3):365–391, 2003.
131. H. A. Reijers and I. T. P. Vanderfeesten. Cohesion and Coupling Metrics for Workflow Process Design. In *BPM 2004*, volume 3080 of *LNCS*, pages 290–305. Springer, 2004.
132. W. Reisig. *Petri Nets: An Introduction*, volume 4 of *Monographs in Theoretical Computer Science. An EATCS Series*. Springer, 1985.
133. W. Reisig. *Elements of Distributed Algorithms: Modeling and Analysis with Petri Nets*. Springer, 1998.
134. M. Rosemann and W. M. P. van der Aalst. A Configurable Reference Modelling Language. *Information Systems*, 32(1):1–23, 2007.
135. W. Sadiq and M. E. Orlowska. Analyzing Process Models Using Graph Reduction Techniques. *Information Systems*, 25(2):117–134, 2000.
136. G. Salton, A. Wong, and C. S. Yang. A Vector Space Model for Automatic Indexing. *Communications of the ACM*, 18(11):613–620, 1975.
137. S. E. Schaeffer. Graph Clustering. *Computer Science Review*, 1(1):27–64, 2007.
138. A.-W. Scheer. *ARIS – Modellierungsmethoden, Metamodelle, Anwendungen*. Springer, 4 edition, 2001.
139. K. Schmidt. Controllability of Open Workflow Nets. In *EMISA*, volume 75 of *LNI*, pages 236–249. GI, 2005.
140. A. Sharp and P. McDermott. *Workflow Modeling: Tools for Process Improvement and Applications Ddevelopment*. Artech House Publishers, 2008.
141. M. Shen and D. Liu. Discovering Role-Relevant Process-Views for Recommending Workflow Information. In *DEXA 2003*, pages 836–845, 2003.
142. S. Smirnov. Structural Aspects of Business Process Diagram Abstraction. In *International Workshop on BPMN 2009*, pages 375–382, Vienna, Austria, July 2009.
143. S. Smirnov, R. M. Dijkman, J. Mendling, and M. Weske. Meronymy-based Aggregation of Activities in Business Process Models. In *ER 2010*, volume 6412 of *LNCS*, pages 1–14. Springer, 2010.
144. S. Smirnov, A. Z. Farahani, and M. Weske. State Propagation in Abstracted Business Processes. In *ICSOC 2011*, LNCS. Springer, 2011. to appear.
145. S. Smirnov, H. A. Reijers, and M. Weske. A Semantic Approach for Business Process Model Abstraction. In *CAiSE 2011*, volume 6741 of *LNCS*, pages 497–511. Springer, 2011.
146. S. Smirnov, M. Weidlich, J. Mendling, and M. Weske. Action Patterns in Business Process Models. In *ICSOC/ServiceWave 2009*, volume 5900 of *LNCS*, pages 115–129. Springer, 2009.

147. S. Smirnov, M. Weidlich, J. Mendling, and M. Weske. Object-Sensitive Action Patterns in Process Model Repositories. In *Business Process Management Workshops*, volume 66 of *LNBIP*, pages 251–263. Springer, 2010.
148. A. Smith. *An Inquiry into the Nature and Causes of the Wealth of Nations*. W. Strahan and T. Cadell, 1776.
149. H. Smith and P. Fingar. *Business Process Management (BPM): The Third Wave*. Meghan-Kiffer Press, 2003.
150. H. Stachowiak. *Allgemeine Modelltheorie*. Springer, 1973.
151. A. Streit, B. Pham, and R. Brown. Visualization Support for Managing Large Business Process Specifications. In *BPM 2005*, volume 3649 of *LNCS*, pages 205–219. Springer, 2005.
152. R. E. Tarjan and J. Valdes. Prime Subprogram Parsing of a Program. In *POPL 1980*, pages 95–105, New York, NY, USA, 1980. ACM.
153. R. Uba, M. Dumas, L. García-Bañuelos, and M. La Rosa. Clone Detection in Repositories of Business Process Models. In *BPM 2011*, volume 6896 of *LNCS*, pages 248–264. Springer, 2011.
154. J. Vanhatalo, H. Völzer, and J. Koehler. The Refined Process Structure Tree. In *BPM 2008*, volume 5240 of *LNCS*, pages 100–115, Milan, Italy, September 2008. Springer.
155. J. Vanhatalo, H. Völzer, and F. Leymann. Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition. In *ICSOC 2007*, volume 4749 of *LNCS*, pages 43–55. Springer, 2007.
156. I. Weber, J. Hoffmann, and J. Mendling. Beyond Soundness: on the Verification of Semantic Business Process Models. *Distributed and Parallel Databases*, 27:271–343, June 2010.
157. M. Weidlich, A. Barros, J. Mendling, and M. Weske. Vertical Alignment of Process Models - How Can We Get There? In *BPMDS 2009*, volume 29 of *LNBIP*, pages 71–84. Springer, 2009.
158. M. Weidlich, R. M. Dijkman, and J. Mendling. The ICoP Framework: Identification of Correspondences between Process Models. In *CAiSE 2010*, volume 6051 of *LNCS*, pages 483–498. Springer, 2010.
159. M. Weidlich, R. M. Dijkman, and M. Weske. Deciding Behaviour Compatibility of Complex Correspondences between Process Models. In *BPM 2010*, volume 6336 of *LNCS*, pages 78–94. Springer, 2010.
160. M. Weidlich, R. M. Dijkman, and M. Weske. Deciding Behaviour Compatibility of Complex Correspondences between Process Models. In *BPM 2010*, volume 6336 of *LNCS*, pages 78–94. Springer, 2010.
161. M. Weidlich, F. Elliger, and M. Weske. Generalised Computation of Behavioural Profiles based on Petri-Net Unfoldings. In *WS-FM 2010*, volume 6551 of *LNCS*, pages 101–115. Springer, 2010.
162. M. Weidlich, J. Mendling, and M. Weske. Efficient Consistency Measurement based on Behavioural Profiles of Process Models. *IEEE Transactions on Software Engineering*, 37(3):410–429, 2011.
163. M. Weidlich, A. Polyvyanyy, J. Mendling, and M. Weske. Efficient Computation of Causal Behavioral Profiles Using Structural Decomposition. In *Petri Nets 2010*, volume 6128 of *LNCS*, pages 63–83. Springer, 2010.
164. M. Weidlich, S. Smirnov, Ch. Wiggert, and M. Weske. Flexab—Flexible Business Process Model Abstraction. In *CAiSE Forum 2011*, volume 734 of *CEUR Workshop Proceedings*, pages 17–24. CEUR-WS.org, 2011.

165. M. Weske. *Business Process Management: Concepts, Languages, Architectures.* Springer Verlag, 2007.
166. S. K. M. Wong, W. Ziarko, and P. C. N. Wong. Generalized Vector Spaces Model in Information Retrieval. In *SIGIR 1985*, pages 18–25, New York, NY, USA, 1985. ACM.
167. M. Th. Wynn, H. M. W. Verbeek, W. M. P. van der Aalst, A. H. M. ter Hofstede, and D. Edmond. Reduction Rules for YAWL Workflows with Cancellation Regions and OR-joins. *Information and Software Technology*, 51(6):1010–1020, 2009.