1

Business Process Model Abstraction

Artem Polyvyanyy, Sergey Smirnov, and Mathias Weske

Business Process Technology Group
Hasso Plattner Institute at the University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3, D-14482 Potsdam, Germany
(Artem.Polyvyanyy,Sergey.Smirnov,Mathias.Weske)@hpi.uni-potsdam.de

Summary. In order to execute, study, or improve operating procedures companies document them as business process models. Often business process analysts capture every single exception handling or alternative task handling scenario within a model. Such a tendency results in large process specifications. The core process logic becomes hidden in numerous modeling constructs. To fulfill different tasks companies develop several model variants of the same business process at different abstraction levels. Afterwards, maintenance of such model groups involves a lot of synchronization effort and is erroneous.

We propose an abstraction technique that allows generalization of process models. Business process model abstraction assumes a detailed model of a process to be available and derives coarse grained models from it. The task of abstraction is to tell significant model elements from insignificant ones and to reduce the latter. We propose to learn insignificant process elements from supplementary model information, e.g., task execution time or frequency of task occurrence. Finally, we discuss a mechanism for user control of the model abstraction level—an abstraction slider.

1.1 Introduction

Business process modeling is crucial when it comes to design of how companies provide services and products to customers or how they organize internal operational processes. To improve the understanding of processes and to enable their analysis, business processes are represented by models [5, 9, 18]. Process models are used for different purposes: to communicate a message, to share knowledge or vision, as a starting point for re-designing or optimizing processes, or as precise instructions for executing business tasks. In such conditions, the goal of a process model is to capture working procedures at a level of detail appropriate to fulfill its envisioned tasks. Often, achievement of such a goal results in complex, "wallpaper-like" models, that tend to capture every minor detail and exceptional case that might occur during process execution.

The desired level of model granularity also depends on a stakeholder working with a model and a current task. Top level company management

appreciates coarse grained process descriptions that allow fast and correct business decisions. At the same time, employees who directly execute processes value fine granular specifications of their daily job. Thus, it might be often the case that a company ends up with maintaining several models of one business process.

Abstraction is generalization that reduces undesired details in order to retain only essential information about an entity or a phenomenon. Business process model abstraction goal is to produce a model containing significant information based on the detailed model specification. Significant information is the information required by a certain stakeholder to fulfill his/her tasks.

We propose a business process model abstraction methodology that can be summarized as follows. As input we assume to possess a complex process model (a detailed process specification). Afterwards, a number of abstractions are performed on the initial model. Conceptually, each abstraction is a function that takes a process model as input and produces a process model as output. In the resulting model initial process fragment gets replaced with its generalized version. Thus, each individual abstraction hides process details and brings a model to a higher abstraction level.

When applied separately, process model abstractions do not provide much value to an end user. Rather, it is of interest to study how individual abstractions can be combined together and afterwards controlled in order to deliver the desired abstraction level. As a solution we propose an abstraction slider—a mechanism providing a user control over process model abstraction.

The rest of the chapter is organized as follows. In the next section we discuss several application scenarios of process model abstraction. Section 1.3 introduces a slider and explains how it is employed for the control of process model abstraction. Transformation rules and their composition aimed to allow process model graph generalization are discussed in section 1.4. Section 1.5 presents results of a case study on abstraction efficiency and usefulness conducted together with an industry partner. The chapter concludes with a survey on related work and summarizing remarks.

1.2 Process Model Abstraction Scenarios

Abstraction generalizes insignificant model elements. Abstraction scenarios have direct implication on the identification of insignificant elements. In this section we clarify the concept of process model abstraction and discuss its common use cases. We then extract abstraction criteria from the proposed use cases. Abstraction criteria are properties of process model elements that enable their partial ordering. Afterwards, obtained partial ordering is used when telling significant model elements from insignificant ones. It is not claimed for the proposed list of scenarios to be complete. It should be extended once there is a demand for new abstraction scenarios.

Essentially, business process model abstraction deals with finding answers to two questions of *what* and *how*:

- What parts of a process model are of low significance?
- How to transform a process model so that insignificant parts are removed?

Answers to both questions should address the current abstraction use case. The choice of an abstraction criterion helps in answering the *what* question. Whereas, an answer to the *how* question allows deriving models where insignificant elements get generalized.

Considering aforesaid, business process model abstraction is a function for which holds:

- a detailed process model and an *abstraction criterion* are the input of this function; an abstraction criterion helps to tell significant model elements from insignificant
- the function output is an abstracted process model
- from the structural perspective abstraction reduces the number of model elements
- from the semantic perspective abstraction generalizes initial model.

When studying a business process model analysts might be interested in tasks which are executed frequently in a process. One can presume that frequent tasks capture main process logic while non-frequent ones constitute seldom alternative scenarios or exceptional flow. Preservation of only frequent process tasks might allow faster understanding of the core process logic by an end user. In order to fulfill the described use case, one might classify significant process elements as those that have a high occurrence number. Thus, the abstraction criterion is the mean occurrence number of a process task.

Mean occurrence number of a process task (m_i) is the mean number that the process task i occurs in a process instance.

Alternatively, analysts might be interested in process tasks that consume most of the process execution time (execution *effort*). These tasks are natural candidates for being studied during the task of process improvement. Once such tasks are optimized, the overall process execution time might drop considerably. Also, in many cases, cost required to execute process tasks is proportional to the execution time. Process task effort is another process model abstraction criterion.

Relative effort of a process task (e_r) is time required to execute the task.

Absolute effort of a process task (e_a) is the mean effort contributed to the execution of the process task in a process instance. Absolute effort can be obtained as the product of the relative effort and the mean occurrence number of the process task.

As proposed, the effort of a process task is measured in time units (e.g., minutes or hours) and quantitatively coincides with the duration. However, semantically the effort concept resembles the concept of cost. For instance, if

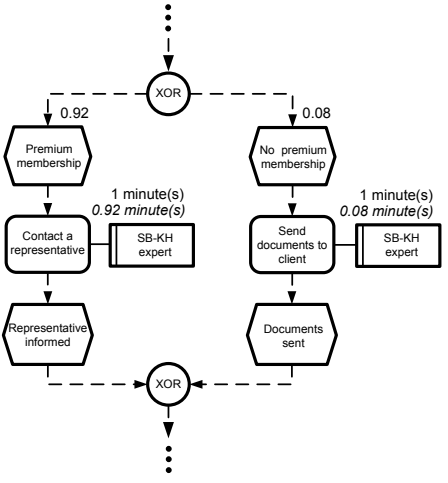


Fig. 1.1. Example of the EPC fragment enriched with probabilities and efforts

two process tasks run in parallel their total effort is the sum of efforts of each task.

The cost of process tasks and the overall process execution cost are important properties of business processes. Similar to *process task effort* one might define a process model abstraction criterion of *process task cost*.

Process model abstraction criteria can be also defined on process fragments. For example, one might be interested in "typical" executions of a business process model. A typical business process execution means that among all possible ways of a process completion it is the one that is executed most often. Applying such an abstraction to a process model should result in a new model which reflects only most common process scenarios. A process scenario is a minimal part of a process model that covers certain instance execution.

Probability of a process scenario (P_i) is the probability of a process scenario i to happen when executing the process.

Similarly, process scenarios with the highest duration or cost may be in the focus of process abstraction. As a result of the abstraction one should obtain a model representing either the most time consuming or the most "expensive" process execution paths.

Effort of a process scenario (E_i) is the effort to be invested in the execution of a process scenario i and can be found as the sum of efforts of all the tasks executed within this scenario.

Figure 1.1 shows the process model fragment, modeled using EPC notation [10, 16], and illustrates presented concepts. Here, all the outgoing connections of the exclusive or split are supplied with transition probabilities that sum up to one. All the other connections are assumed to have the transition probability of one. Each function is enriched with relative and absolute (visualized in italic type) efforts given by the time interval in minutes that a

worker needs to perform a function. For instance, the function "Contact a representative" has the relative effort of one minute, meaning that it is expected to take one minute of worker's time once reached in a process instance. On average, this function requires $1 \cdot 0.92 = 0.92$ minutes in every process instance which constitutes the absolute effort of the function. The absolute effort is obtained under the assumption that the process fragment is reached only once in a process instance with the probability of one.

Often, abstraction criteria require models to be annotated with additional information like statistical data on average time required in order to perform process tasks, probabilities of reaching tasks in a process, etc. In many cases incorporation of such information requires extension of modeling notation.

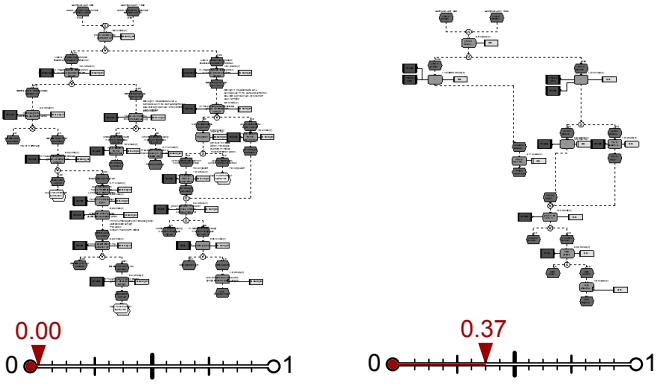
1.3 Abstraction Slider

In this section we focus on the *what* question of process abstraction. We propose a *slider metaphor* [13] as a tool for enabling flexible control over the process model abstraction level. We explain how the slider can be employed for distinguishing significant process model elements from insignificant ones. We provide an example of applying the abstraction slider.

When a user selects suitable abstraction criterion, the desired level of abstraction should be specified. Abstraction level cannot be predicted without a priori knowledge about the abstraction context. In the best case, the user should be able to change abstraction level smoothly from an initial detailed process model to a process model containing only one task. In this example, the single abstracted process task semantically corresponds to the abstraction of the whole original process model.

A slider is an object that operates on a slider interval $[S_{min}, S_{max}]$. The interval is constrained by the minimum and maximum values of the abstraction criterion. The slider specifies criterion value as a slider state $s \in [S_{min}, S_{max}]$ and allows operation of a state change within this interval.

All of the discussed abstraction criteria (see section 1.2) have quantitative measurement. Therefore, criterion values for a particular criterion type are in a partial order relation. Correspondingly, the partial order relation can be transferred on process model elements by arranging them according to values of some particular criterion. For example, if a criterion is *task relative effort* then a two minutes task precedes a four minutes task. The partial order relation enables element classification. It is possible to split model elements into two classes: those with criterion value less than and those with criterion value greater than some designed separation point. Elements which are the members of the first class are assumed to be insignificant and have to be omitted in the abstracted model. Members of the other class are significant and should be preserved in the abstracted model. We refer to the separation point according to which the elements classes are constructed as *abstraction threshold*. Assuming an abstraction threshold of three minutes in the example



(a) Initial process model (b) Slider set to 0.37

Fig. 1.2. Process model abstraction slider (function names unreadability intended)

discussed above, the two minutes task is insignificant and has to be reduced. On the opposite, the four minutes task is significant and should be preserved in the abstracted model.

Thus, a *process model abstraction slider* is a slider which for a given process model fragment and a specified abstraction threshold classifies the fragment as significant or not. The abstraction slider interval is defined on an interval of abstraction criterion values and the slider state is associated with the abstraction threshold.

A slider control regulates the amount of elements preserved in an abstracted process model. In the simplest case a user specifies an arbitrary value used as a threshold (which means that the slider interval is $[-\infty, +\infty]$). The challenge for a user in this approach is to inspect a process model in order to choose a meaningful threshold value. A threshold value which is too low makes all the process model elements to be treated as significant, i.e., no nodes or edges are reduced. On the other hand, a threshold which is too high may result in a one task process model. To avoid such confusing situations, the user should be supported by suggesting an interval in which all the "useful" values of abstraction criterion lie. Alternatively, the abstraction slider can control a share of nodes to be preserved in a model. In this case abstraction mechanism has to estimate the threshold value which results in the reduction of the specified share of the process model.

Figure 1.2 exemplifies the work of process model abstraction slider. It provides a comparison of the initial process model (a) and its two abstracted models. The business process is captured in EPC notation. In the example we have used the abstraction criterion of absolute effort of a process function. Functions with a higher absolute effort are considered to be more significant.

(a) shows the business process model that corresponds to the abstraction slider state of 0.00—the original process model. The model visualized in (b) is obtained by changing the abstraction threshold to 0.37. In the proposed example more than 50% of the model nodes get reduced. The process model shrinks to one function when the slider state is set to 1.00.

1.4 Process Model Transformation

In this section we address the *how* question of the process model abstraction task. We base our solution on process model transformation rules. In this section two classes of abstraction rules are introduced: elimination and aggregation. Afterwards, requirements for abstraction and their influence on the transformation rules are discussed. We argue when each of the techniques is appropriate. Finally, an example of transformation rules is presented.

1.4.1 Elimination vs. Aggregation

When the insignificant process model elements are identified, they have to be abstracted. Several techniques can be proposed for reduction, we distinguish two: elimination and aggregation.

Elimination means that a process model element is omitted in the abstracted process model. The main feature of elimination is that the resulting model does not contain any information about the eliminated element. Elimination has to assure that the resulting process model is well-formed and that the ordering constraints of the initial model are preserved.

Aggregation implies that insignificant elements of a process model are grouped with other elements. Aggregation preserves information about the abstracted element in the resulting model. When two sequential tasks are aggregated into one, properties of the aggregating task are derived from the properties of the aggregated tasks, e.g., the execution cost of an aggregating task is the sum of execution costs of aggregated tasks.

In general case the rules of elimination are simpler than the aggregation rules. Aggregation requires more sophisticated specification of how the properties of the aggregated elements influence properties of aggregating elements. In many cases elimination is insufficient, since it leads to the loss of important information. If an abstraction cannot tolerate information loss aggregation should be used.

1.4.2 Transformation Requirements

Preservation of the process execution logic is an essential abstraction requirement. This means that process model abstraction should neither introduce new ordering constraints, nor change the existing ones. For instance, if an

original process model specifies to execute either activity A or B, it should not be the case that in the abstracted model these activities appear in a sequence. Another essential abstraction requirement is that well-formed process models should be produced. Thus, used transformation rules should take into account features of modeling notations. Consequently, we can expect different rules to be used, e.g., for EPC and for BPMN.

Further, extra requirements on abstraction rules can be imposed. For instance, a company may use process models for estimation of the workforce required to execute business processes. In this case information about the overall effort of process execution should be preserved. Process model abstractions preserving process properties are called *property preserving abstractions*. Elimination can be used in a property preserving abstraction with restrictions, since once a model element is omitted all the information about its properties is lost. Therefore, elimination can be applied only to those elements which do not influence the property being preserved.

Every new requirement imposed on an abstraction restricts transformation rules and makes the design of these rules more complex. It is important to learn which class of process models can be abstracted to one task by a given set of rules and abstraction requirements. An abstraction which is not capable of reducing a process model to one function is called *best effort abstraction*. Such an abstraction *tries* to assure that a given process model is abstracted to the requested level using the given set of rules.

1.4.3 Transformation Rules

In [14] a process model abstraction approach is presented. Its cornerstone is a set of abstraction rules. We would like to use these rules as an illustration of the concepts discussed earlier and demonstrate how these rules can function together with the abstraction slider and task absolute effort abstraction criterion.

The approach presented in [14] is capable of abstracting process models captured in EPC notation. Two requirements are imposed on abstraction:

1. ordering constraints of a process model should be preserved,
2. absolute process effort should be preserved.

The approach is based on the set of transformation rules called *elementary abstractions*. Four elementary abstractions are proposed: sequential, block, loop, and dead end abstraction. Every elementary abstraction defines how a certain type of a process fragment is generalized. The order of elementary abstractions can vary. Application of an elementary abstraction may succeed once there is a suitable process fragment in a process model. This also means that any function can be the result of a prior abstraction.

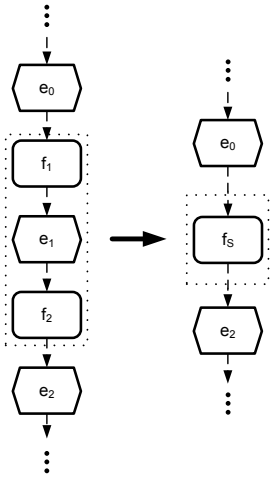


Fig. 1.3. Sequential abstraction

Sequential Abstraction

Business process models of high fidelity often contain sequences of tasks. In EPCs such sequences turn into sequences of functions. *Sequential abstraction* replaces a sequence of functions and events by one aggregating function. This function is more coarse-grained and brings a process model to a higher abstraction level.

Definition 1. An EPC process fragment is a *sequence* if it is formed by a function, followed by an event, followed by a function.

The mechanism of sequential abstraction is sketched in Figure 1.3. Functions f_1, f_2, and event e_1 constitute a sequence. Aggregating function f_S replaces this sequence. Semantically the aggregating function corresponds to execution of functions f_1 and f_2.

Block Abstraction

To model parallelism or a decision point in a process, modelers use split connectors with outgoing branches. Depending on the desired semantics, an appropriate connector type is selected: AND, OR, or XOR. In the subsequent parts of a process model these branches are synchronized with the corresponding join connectors. A process fragment enclosed between connectors usually has a self-contained business semantics. Therefore, the fragment can be replaced by one function of coarse granularity. *Block abstraction* enables this generalization. To define block abstraction we use a notion of a path in EPC—a sequence of nodes such that for each node there exists a connection to the next node in the sequence.

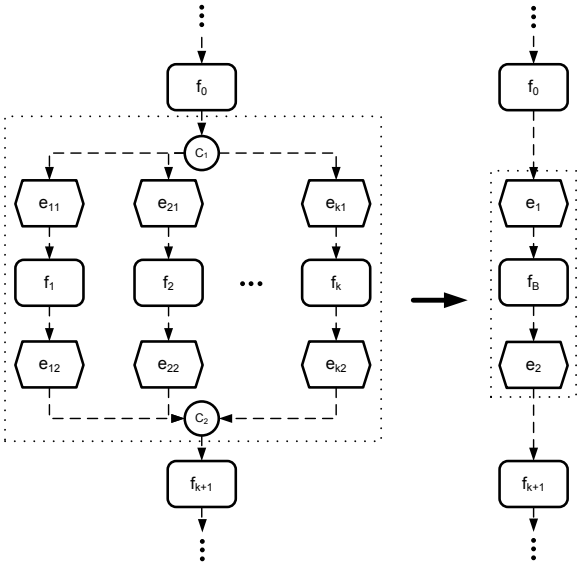


Fig. 1.4. Block abstraction

Definition 2. An EPC process fragment is a *block* if:

- it starts with a split and ends with a join connector of the same type
- all paths from the split connector lead to the join connector
- there is at most one function on each path
- each path between the split and the join contains only events and functions
- the number of the outgoing connections of the split connector equals the number of the incoming connections of the join connector
- the split connector has one incoming connection and the join connector—one outgoing.

Figure 1.4 describes the mechanism of block abstraction. Block abstraction replaces an initial process fragment by a sequence of event, aggregating function, and another event. Events assure that a new EPC is well-formed. Semantics of the aggregating function corresponds to the semantics of the abstracted block and conforms to the block type. For instance, if a XOR block is considered the aggregating function states that only one function of the abstracted fragment is executed.

Loop Abstraction

Often tasks (or sets of tasks) are iterated for successful process completion. In a process model the fragment to be repeated is enclosed into a loop construct. In EPC notation control flow enables loop modeling. Wide application of loops by modelers makes support of loop abstraction an essential part of the abstraction

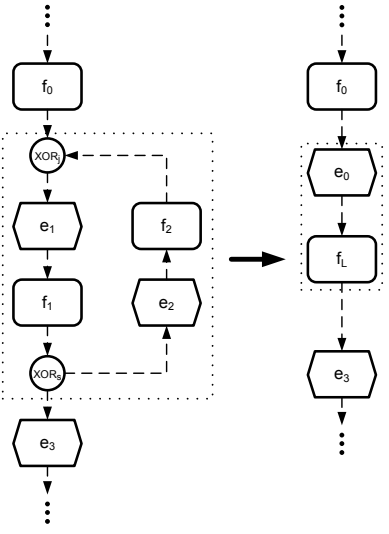


Fig. 1.5. Loop abstraction

approach. Therefore, one more elementary abstraction—*loop abstraction*— is introduced. Following we define the process fragment considered to be a loop.

Definition 3. An EPC process fragment is a *loop* if:

- it starts with a XOR join connector and ends with a XOR split connector
- the process fragment does not contain any other connectors
- the XOR join has exactly one outgoing and two incoming connections
- the XOR split has exactly one incoming and two outgoing connections
- there is exactly one path from the split to the join and exactly one path from the join to the split
- there is at least one function in the process fragment.

As shown in Figure 1.5 aggregating function f_L replaces the whole process fragment corresponding to a loop. Event e_0 is inserted between functions f_0 and f_L in order to obtain a well-formed EPC model. An aggregating function states that functions f_1 and f_2 are executed iteratively.

Dead End Abstraction

Exceptional and alternative control flows result in "spaghetti-like" process models with lots of control flow branches leading to multiple end events. Abstraction aims to reduce excessive process details. Thus, abstraction mechanism should be capable of eliminating these flows. *Dead end abstraction* addresses this problem. First, the term *dead end* should be specified.

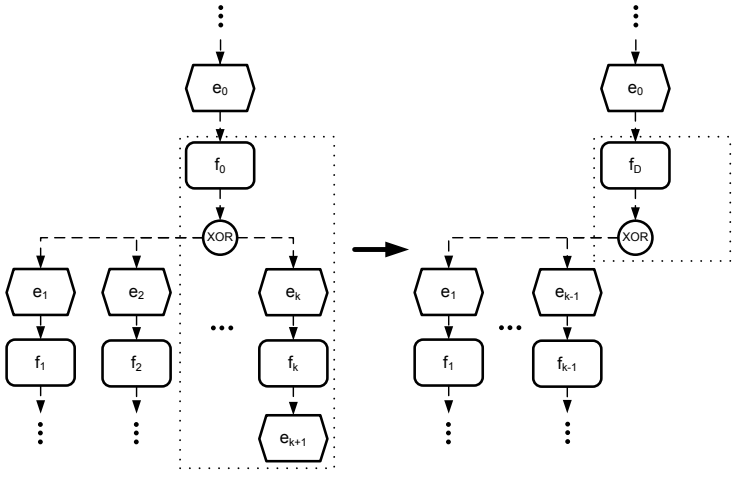


Fig. 1.6. Dead end abstraction

Definition 4. An EPC process fragment is a *dead end* if it consists of a function, followed by a XOR split connector, followed by an event, followed by a function, followed by an end event. The XOR split connector has only one incoming connection.

Figure 1.6 visualizes the dead end abstraction mechanism. The initial process fragment is provided on the left side of the figure. The dead end is formed by functions f_0 and f_k, events e_k and e_{k+1} and the XOR split connector. The XOR split has k outgoing branches and abstraction removes the k-th branch. The abstracted process is presented on the right side of Figure 1.6. Rectangles with dotted borders enclose the dead end fragment and its replacement.

Dead end abstraction completely removes a XOR split branch which belongs to a dead end. Aggregating function f_D replaces function f_0. An aggregating function in dead end abstraction has the following semantics: upon an occurrence of function f_D in a process, function f_0 is executed. Afterwards, function f_k may be executed. Upon execution of function f_k the branch is terminated and f_D is not left. Otherwise, the execution of the branch is continued. When a XOR split has two outgoing connections in the initial process model, the XOR split in the abstracted process model can be omitted. A new connection from the aggregating function to the event, following the omitted XOR split, should be added to the EPC.

Abstraction Strategy

A single application of an elementary abstraction is not of great value for the task of process abstraction. Therefore, elementary abstractions can be invoked

according to an *abstraction strategy*—a rule of composition of elementary abstractions. An abstraction strategy is a sequence of elementary abstraction steps. Every step aims to simplify a process model. At each abstraction step one elementary abstraction is applied. Since elementary abstractions are atomic, i.e., they do not depend on the previous ones, one might come up with various abstraction strategies. In general case different strategies lead to different resulting process models.

We propose to organize the abstraction strategy in compliance with the slider concept. Hence, first we aim to abstract from functions of low significance. Once the function with the lowest significance is identified, it is tested to which type of process fragment it belongs. If a process fragment is recognized, appropriate abstraction transformation rules are applied. Otherwise, another elementary abstraction is tested. The next elementary abstraction to test is selected according to the predefined priority. Abstraction is continued until either no more elementary abstraction process fragments are recognized, or the lowest element significance in the process has reached the preset threshold.

An abstraction strategy using only one type of elementary abstraction can be seen as a basic abstraction strategy. Basic abstraction strategy result in process models where only sequential, dead end, block, or loop process fragments are reduced. For instance, in case of the basic sequential abstraction strategy sequences of an arbitrary length are reduced.

Advanced abstraction strategies combine several elementary abstractions and define their priority. The priority dictates the application order of elementary abstractions. One possible strategy is the precedence of sequential, dead end, block, and then loop abstraction. Application of one elementary abstraction might enable further application of another one.

1.5 Case Study

In this section we conduct an in-depth analysis of the proposed mechanisms. We evaluate the results of process model abstractions conducted in a joint project with an industry partner. The project objective was to derive process model abstraction mechanisms and to apply them on a process model repository composed of around 4 000 models captured in EPC notation. The additional requirement for abstraction was to preserve overall process effort, i.e., the overall process effort before and after abstraction should stay unchanged. We evaluate the developed abstraction mechanisms in terms of efficiency and usefulness. An estimation of abstraction efficiency is based on the analysis of the number of model nodes reduced by abstractions. Obviously, this measure does not witness the usefulness of the abstraction. In order to learn the usefulness of abstractions we appeal to the project partner's expertise.

Following, we provide the results of performing abstraction on a subset of models from the repository composed of 1 195 models. Each model consists of 10 or more nodes. Models with less than 10 nodes are not considered. Three

abstraction strategies take part in the case study. Each strategy uses one or several elementary abstractions and applies them iteratively (see section 1.4.3). The following abstraction strategies are used:

1. basic sequential abstraction (strategy 1)
2. sequential then block abstraction (strategy 2)
3. sequential, dead end, block, then loop abstraction (strategy 3).

Abstraction strategies are applied with a threshold level equal to the overall process effort. This guarantees that an abstraction tries to reduce all the nodes in a model to the point when no more abstractions are applicable.

Table 1.1 presents results of applying abstraction strategies, i.e., correspondence between intervals of number of nodes in a model and the number of models that fall into the interval, provided for original as well as abstracted models. The table illustrates how different abstraction strategies reduce the amount of nodes in models.

Table 1.1. Comparison of node reduction caused by various abstraction strategies

Number of nodes	Original	Strategy 1	Strategy 2	Strategy 3
1 to 10	0	274	511	871
11 to 20	464	359	306	156
21 to 30	225	182	137	82
31 to 40	130	150	81	54
41 to 50	118	69	56	20
51 to 60	65	36	38	2
61 to 70	47	33	29	4
71 to 80	31	29	18	4
81 to 90	22	15	5	0
91 to 100	22	14	2	0
> 100	71	34	12	2

Additionally, we use the notion of abstraction compression coefficient—a ratio between the number of nodes in abstracted and original models. Each line in Figure 1.7 corresponds to the probability density function of the compression coefficient for a certain abstraction strategy. The line for strategy 1 hints on the fact that most of the models were reduced by 40% or less. Whereas in the case of strategy 3 the number of nodes in most models were reduced by 70% or more. This clearly witnesses that strategy 3 excels its evaluated competitors.

In order to evaluate the usefulness of the abstraction approach we refer to project partner's experts. Abstractions capable of aggregating more model elements are considered as most valuable. Thus in general case, strategy 3 can be seen as more useful strategy. The project partners argued that the choice of an abstraction method depends on the structure of a particular process model.

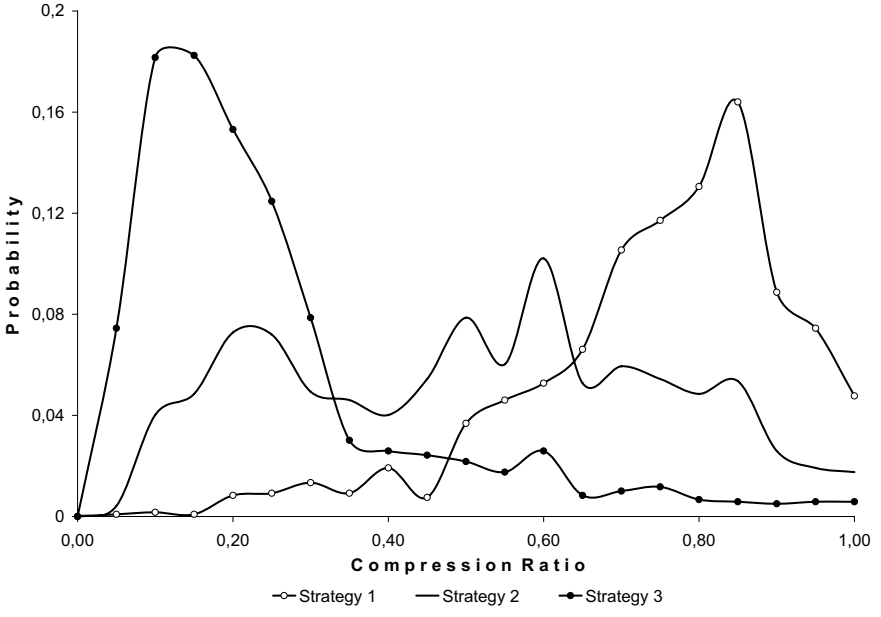


Fig. 1.7. Comparison of compression ratios for 3 discussed abstraction strategies

For instance, strategy 1 can be seen as useful for some particular process model if it allows same generalization as in the case of strategy 3.

1.6 Related Work

The problem of managing large complex process models emerges as BPM technologies penetrate modern enterprises. This challenging situation is addressed by various approaches. The authors of several process modeling notations, like Business Process Modeling Notation (BPMN) [3] or Yet Another Workflow Language (YAWL) [1] envisioned this problem. These notations allow hierarchical structuring of models. The goal of the hierarchical model organization is to distribute information describing a process among several levels with the general process flow on the highest level of hierarchy and the process details on the lowest one. Unfortunately, such a mechanism is not sufficient to cope with the problem, since it assumes that the hierarchy is designed and maintained manually. In [19] the authors propose an algorithm for identifying special kind of regions called *reducible subflows* in workflow nets. Once such regions are found, a process model can be decomposed into their hierarchy.

A number of studies focused on creation of process views from available process models. The purpose of a process view is to hide certain fragments of a process model. For instance, one can imagine an actor specific process

view or a process view reflecting parts of a process instance to be executed (the last case corresponds to a process view on an instance level). Therefore, the goal of a process view creation differs from the goal of process model abstraction and can be seen as a more generic task. On the other hand, process view creation focuses on the *how* question, but does not discuss the *what* of abstraction, i.e. it does not say how to identify significant model elements. Bobrik et al. in [2] propose an approach capable of creating customized process views on model level and on instance level. The approach relies on graph reduction rules. In [7] the authors propose a method for constructing views aiming to ease communication between partners by adapting internal process descriptions into ones suitable for external usage. As an input the approach takes a process model captured in UML activity diagram notation and a user requirement to hide certain process elements. In [11] the authors propose an order preserving approach for creation of process views. An important issue is that the mentioned approaches do not incorporate the notion of non-functional properties of a process and, thus, do not define how non-functional properties of a process (e.g., execution effort and execution cost) can be preserved during transformations.

In [8], Günther and van der Aalst proposed a framework allowing to judge about significance of model elements basing on their non-functional properties. The framework bases on various metrics evaluating significance of process model nodes and edges. The proposed technique can be employed to answer the *what* question of abstraction, i.e. to derive reasonable significance values for process model elements.

The abstraction mechanism proposed in this chapter makes use of the set of elementary abstraction rules. Each rule has the goal of model simplification and defines how a process model fragment is transformed. In [14] it is shown how these rules can be extended for evaluation of non-functional properties of model elements. In particular, it is described how properties of aggregating elements are derived from the properties of aggregated. Graph transformation rules are widely used for analysis of process model soundness and are well studied in literature [6, 11, 12, 15, 17]. An approach proposed in [15] presents rules facilitating soundness analysis of process models captured in the notation proposed by Workflow Management Coalition. In [6] and [12] the authors focus on the rules facilitating analysis of EPC models soundness. Cardoso et al. in [4] propose a method for the evaluation of workflow properties (e.g., execution cost, execution time, and reliability) based on the properties of workflow tasks. However, the approach is restricted to block-structured process models free of OR blocks.

The presented outlook of the related work witnesses: there is no comprehensive approach which addresses all the aspects of the business process model abstraction task. Several approaches provide a solid basis of reduction rules, capable of handling sophisticated graph-structured processes. However, these approaches do not allow estimating process properties, such as effort or cost. On the other hand, there is an approach (cf. [4]) supporting process properties

estimation, but it is limited to block-structured processes excluding OR block constructs. Finally, to the best of our knowledge, there is no means for controlling process abstraction. Therefore, in this chapter we have shown how process model abstraction can be conceptually realized. We have introduced the slider concept—a mean for the user to control the abstraction. The approach uses transformation rules proposed in [14]. The rules prescribe how the process non-functional properties can be estimated.

1.7 Conclusions

In this chapter we presented a business process model abstraction technique—an approach to derive process models of high abstraction level from the detailed ones. We argued that the abstraction task can be decomposed into two independent subtasks: learning process model elements which are insignificant (abstraction *what*) and abstracting from those elements (abstraction *how*). The proposed technique can be applied for abstraction of an arbitrary graph-structured process model.

Several abstraction scenarios were provided to motivate the task of business process model abstraction. These scenarios were used to extract abstraction criteria. Afterwards, we proposed to adopt a slider concept in order to achieve control over abstraction process. Finally, we discussed process model transformation rules which can be employed together with the slider for abstraction of insignificant model elements.

We proposed a concrete scenario of applying graph transformation rules for the purpose of model abstraction. Elementary abstractions: sequential, block, loop, and dead end abstraction were presented. For every elementary abstraction it was defined to which type of process fragment it can be applied and in which model transformation it results. It was explained how these individual abstractions can be combined into abstraction strategies. Derived abstraction methodology preserves function ordering constraints of the initial model. To the limitation of the approach one can count the fact that not an arbitrary model can be abstracted to one function, if such a behavior is desired. We conducted a case study on abstraction efficiency and usefulness with the industry project partner and presented obtained statistical results. The technique of process model abstraction can be extended by other transformation rules that assume process graph generalization, e.g., rules proposed in [11, 15].

References

1. W. M. P. van der Aalst and A. H. M. ter Hofstede. YAWL: Yet Another Workflow Language (Revised version). Technical Report FIT-TR-2003-04, Queensland University of Technology, Brisbane, 2003.
2. R. Bobrik, M. Reichert, and Th. Bauer. View-Based Process Visualization. In *BPM*, volume 4714 of *LNCS*, pages 88–95. Springer, 2007.

3. BPMI. *Business Process Modeling Notation*, 1.1 edition, February 2008.
4. 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.
5. T. Davenport. *Process Innovation: Reengineering Work through Information Technology.* Harvard Business School Press, Boston, MA, USA, 1993.
6. B. van Dongen, M. Jansen-Vullers, H. Verbeek, and W. M. P. van der Aalst. Verification of the SAP Reference Models Using EPC Reduction, State-space Analysis, and Invariants. *Comput. Ind.*, 58(6):578–601, 2007.
7. R. Eshuis and P. Grefen. Constructing Customized Process Views. *Data Knowl. Eng.*, 64(2):419–438, 2008.
8. C. 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 Verlag.
9. M. Hammer and J. Champy. *Reengineering the Corporation: A Manifesto for Business Revolution.* HarperBusiness, April 1994.
10. 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.
11. D. Liu and M. Shen. Workflow Modeling for Virtual Processes: an Order-preserving Process-view Approach. *Information Systems*, 28(6):505–532, 2003.
12. J. Mendling, H. Verbeek, B. van Dongen, W. M. P. van der Aalst, and G. Neumann. Detection and Prediction of Errors in EPCs of the SAP Reference Model. *Data Knowl. Eng.*, 64(1):312–329, 2008.
13. A. Polyvyanyy, S. Smirnov, and M. Weske. Process Model Abstraction: A Slider Approach. In *EDOC '08: Proceedings of the 12th IEEE International Enterprise Distributed Object Computing Conference*, München, Germany, 9 2008. IEEE Computer Society.
14. A. Polyvyanyy, S. Smirnov, and M. Weske. Reducing Complexity of Large EPCs. In *EPK'08 GI-Workshop*, Saarbrücken, Germany, 11 2008.
15. W. Sadiq and M. Orlowska. Analyzing Process Models Using Graph Reduction Techniques. *Information Systems*, 25(2):117–134, 2000.
16. A. Scheer, O. Thomas, and O. Adam. *Process Aware Information Systems: Bridging People and Software through Process Technology*, chapter Process Modeling Using Event-Driven Process Chains, pages 119–145. John Wiley & Sons, 2005.
17. 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, pages 43–55. Springer, 2007.
18. M. Weske. *Business Process Management: Concepts, Languages, Architectures.* Springer Verlag, 2007.
19. L. Zerguini. A Novel Hierarchical Method For Decomposition And Design Of Workflow Models. volume 8, pages 65–74, Amsterdam, The Netherlands, 2004. IOS Press.