Reducing Complexity of Large EPCs

Artem Polyvyanyy, Sergey Smirnov, and Mathias Weske
Business Process Technology Group
Hasso Plattner Institute at the University of Potsdam
D-14482 Potsdam, Germany
(artem.polyvyanyy,sergey.smirnov,mathias.weske)@hpi.uni-potsdam.de

Abstract: Business processes are an important instrument for understanding and improving how companies provide goods and services to customers. Therefore, many companies have documented their business processes well, often in the Event-driven Process Chains (EPC). Unfortunately, in many cases the resulting EPCs are rather complex, so that the overall process logic is hidden in low level process details. This paper proposes abstraction mechanisms for process models that aim to reduce their complexity, while keeping the overall process structure. We assume that functions are marked with efforts and splits are marked with probabilities. This information is used to separate important process parts from less important ones. Real world process models are used to validate the approach.

1 Introduction

Business process modeling plays an important role in the design of how companies provide services and products to their customers. To improve the understanding of processes and to enable their analysis, business processes are represented by models [Dav93, Wes07]. Business process models consist of automated and/or manual activities executed by an employee with a support of an information system. The goal of a process model is to provide a basis for defining and optimizing working procedures. Often achievement of this goal is traded for the cost of complex, "wallpaper-like" models, that tend to capture every small detail and exceptional case. Fine granular process models distract attention of a reader from the overall process logic by exhaustive details.

This paper proposes abstraction mechanisms that transform detailed process models in less detailed ones that still reflect the overall process logic. We do not assume any limitations on the initial process model control flow structure: proposed process model abstraction mechanisms implicitly define a set of addressed control flow patterns. The results are developed for EPC [KNS92, STA05]. However, they can be adapted to any graph-structured process modeling notation, for instance the Business Process Modeling Notation (BPMN) [BPM04].

The basic principle of the abstraction methodology proposed in this paper can be described as follows. Starting with a complex, detailed process model, a number of abstractions are performed. Formally, each abstraction takes a process model as input and generates a process model as output where an abstracted process fragment is replaced by a new one. The

new process fragment gives a generalized view of the substituted process fragment. Each individual abstraction leads to process details become concealed in a resulting process model.

The presented results were obtained in a joint research project with the health insurance company. Operational processes of the company are captured in about $4\,000$ EPCs. Detailed models lead to information overload creating a demand for abstracted process models. The models are enriched with information about the effort required to complete each function of each process and probabilities of connection transitions from source to the target. The project partner uses proprietary tools to calculate the number of employees and their roles to enact all process instances that need to be executed. Since process models are the basis for head count estimations, an overall process effort after abstractions must remain unchanged.

This paper is structured as follows: Section 2 makes a survey on related work. Afterwards, the fundamental concepts are explained in Section 3. Elementary abstraction mechanisms are presented in Section 4. Concluding remarks complete this paper.

2 Related Work

The abstraction approach discussed in this paper bases on the set of elementary abstraction rules. Each rule specifies how a process model fragment can be transformed in order to simplify the process model. Graph transformation rules are well studied in literature [DJVVA07, LS03, MVD$^+$08, SO00, VVL07]. These studies introduce graph reduction rules in order to facilitate analysis of process model soundness by means of state space reduction. An approach proposed in [SO00] presents rules facilitating soundness analysis of process models captured in the notation proposed by Workflow Management Coalition. The given set of rules can not analyze process models containing loops. [DJVVA07] and [MVD$^+$08] specify reduction rules for structural analysis of EPCs. In [BRB07] the authors use graph reduction rules to create customized process views. Two kinds of rules are proposed: reduction rules and aggregation rules. It should be noticed that the named approaches do not define how such properties as process execution effort or execution cost can be preserved during transformations.

Cardoso et al. in [CMSA02] propose an approach for estimation of workflow properties (e.g., execution cost, execution time, and reliability) using the properties of activities constituting the process. The approach enables analysis of block-structured process models containing sequences, XOR blocks, AND blocks, and structured loops. However, the approach does not address processes which contain OR blocks and which are not block-structured.

A statistical approach to simplification of process models mined from execution logs is presented in [GA07]. It exploits various metrics for judging about the significance of process model elements and enables aggregation and reduction of insignificant elements. However, the approach does not address particularities of EPC and properties of a process, such as process effort, are not preserved.

The presented outlook of the related work witnesses: there is no comprehensive approach which solves the task discussed in this paper. 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. [CMSA02]) supporting process properties estimation, but it is limited to block-structured processes without OR blocks. Therefore, there is a lack of approach capable of handling graph-structured process models, i.e., providing appropriate graph transformation rules and rules for estimating process properties. In this study we target this challenge.

3 Fundamentals

This section introduces fundamentals of the approach—formalization of the extended for our purposes variant of event-driven process chains. There exist several works on formalization of EPC [Aal98, MA07, Wes07]. In this paper we use the formal definition proposed in [Wes07] and extend it by introducing concepts of function efforts and probabilities of connection transitions.

Definition 1 A tuple $(E, F, C, A, t, e_r, p_r)$ is an *extended EPC* if:

- E is a set of events, $E \neq \emptyset$
- F is a set of functions, $F \neq \emptyset$
- C is a set of connectors
- $N = E \cup F \cup C$ is a set of nodes, such that E, F, and C are pairwise disjoint
- $A \subseteq N \times N$ is a set of connections
- $t : C \rightarrow \{and, or, xor\}$ is a mapping assigning connector type to a connector
- $e_r : F \rightarrow \mathbb{R}^+$ is a mapping associating a function with an effort required to complete it (effort is measured in time units, e.g., minutes or hours)
- $p_r : A \rightarrow [0, 1]$ is a mapping assigning transition probability to a connection
- (N, A) is a connected graph
- Each function has exactly one incoming and one outgoing connection.
- There is at least one start event and at least one end event. Each start (end) event has exactly one outgoing (incoming) connection and no incoming (outgoing) connections. All the other events have exactly one incoming and one outgoing connections.
- Each event can only be followed (possibly via a connector) by a function and each function can only be followed (possibly via a connector) by an event.

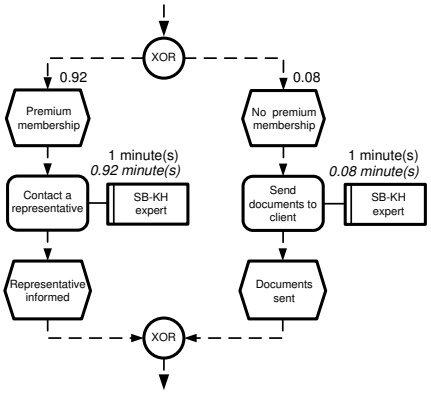

Figure 1: Real world example of the EPC fragment enriched with probabilities and efforts

- There is no cycle that consists of connectors only.
- No event is followed by an OR or a XOR split connector.

In order to address regions of an EPC we define an EPC process fragment as a connected subgraph of the (N, A) graph.

We assume that process models follow proposed formal EPC definition. However, this is not always true, e.g., in the investigated process models, events within a sequence of functions might be omitted. If this is the case, we assume a preprocessing step that modifies EPC to conform to proposed definition, i.e., missing events are automatically inserted.

To continue the discussion we need to define several auxiliary concepts.

Definition 2 *Mean occurrence number of a node* is the mean number that the node will occur in a process instance.

Definition 3 *Absolute effort of a process function* (e_a) is the mean effort contributed to the execution of the function in a process instance: $e_a : F \rightarrow \mathbb{R}^+$. Absolute effort can be obtained as the relative effort multiplied by the mean occurrence number of the function.

Definition 4 *Process absolute effort* (e_a^p) is the mean effort required to execute a process instance: $e_a^p : P \rightarrow \mathbb{R}^+$, where P is a set of process models. Process absolute effort can be obtained as the sum of absolute efforts of process functions.

Figure 1 shows the EPC fragment and illustrates presented concepts. Here, all the outgoing connections of the exclusive or split are supplied with the relative probabilities that sum up to one. All the other connections are assumed to have the relative probability of one. Each function is enriched with the 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. Semantically the effort concept is close to the concept of cost. For instance, if two activities are executed in parallel their total effort is the sum of efforts of both activities. In this study we do not address the waiting time between activities.

4 Elementary Abstractions

In this section elementary abstractions are presented. Elementary abstractions define how certain types of process fragments are generalized. The abstractions can be applied in any order or frequency, provided that a process model contains the structures required for a particular abstraction. This also assumes that any function can be the result of a prior abstraction.

4.1 Dead End Abstraction

Modeling of exceptional and alternative control flows in EPCs usually results in "spaghetti-like" process models with lots of control flow branches leading to multiple end events. As the primary goal of abstraction is to reduce excessive process details, it is of high importance to be capable of eliminating such flows, leaving only the essential information. To address this problem an elementary abstraction called *dead end abstraction* is introduced. Further discussion requires a precise definition of the term *dead end*.

Definition 5 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 2 illustrates the mechanism of the dead end abstraction. On the left side the initial process fragment containing a dead end is provided. Functions f_0 and f_k, events e_k and e_{k+1} and the XOR split connector constitute the dead end. The XOR split has k outgoing branches and after the abstraction the k-th branch is removed. On the right side of Figure 2 the abstracted process is presented.

As a result of abstraction, a XOR split branch which belongs to a dead end is completely removed from a process model. Function f_0 is replaced by an aggregating function f_D. 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. The probability that function f_k occurs is the probability of reaching function f_k from f_0 in the initial process. If function f_k is executed the branch is terminated and f_D is not left. Otherwise, the execution of the branch continues.

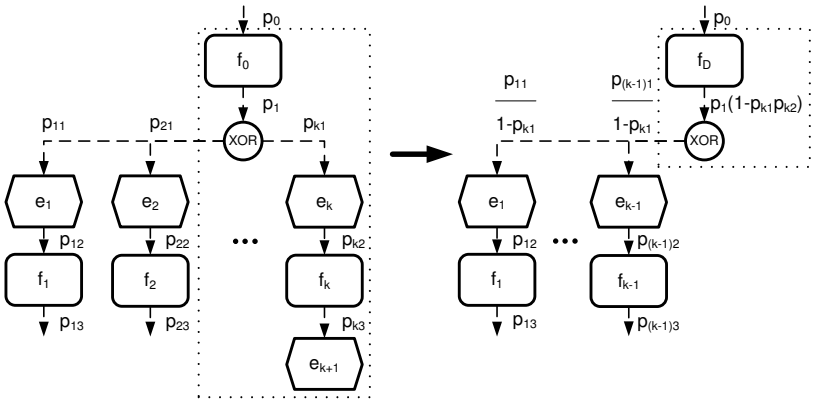

Figure 2: Dead end abstraction

The relative effort of an aggregating function takes into account the relative efforts of functions f_0 and f_k and the probability of f_k occurrence in f_D:

$$e_r(f_D) = e_r(f_0) + e_r(f_k) \cdot p_r((f_0, xor)) \cdot p_r((xor, e_k)) \cdot p_r((e_k, f_k)).$$

The relative probability of reaching a XOR split connector from function f_D is the probability of reaching the XOR connector from function f_0 and not reaching function f_k in the initial process:

$$p_r((f_D, xor)) = p_r((f_0, xor)) \cdot (1 - p_r((xor, e_k)) \cdot p_r((e_k, f_k))).$$

As a result of a dead end abstraction, the relative probability of entering the aggregating function is greater than the relative probability of leaving it: once function f_k is executed, the branch is terminated. Therefore, to find a probability of reaching one node from another, it is always required to take into account probabilities of all intermediate transitions.

Finally, we normalize the probabilities of the XOR split outgoing connections so that the following statements hold:

- the probabilities of reaching events e_i ($i = 1, 2, \ldots, k-1$) from function f_D equal to the probabilities of reaching e_i from f_0 in the initial process
- the sum of the probabilities of the XOR outgoing connections is one.

The normalized relative probabilities are obtained in the following way:

$$p'_r((xor, e_i)) = \frac{p_r((xor, e_i))}{1 - p_r((xor, e_k))}.$$

If a XOR split has only two outgoing connections in the initial process it is possible to omit the XOR split after dead end abstraction.

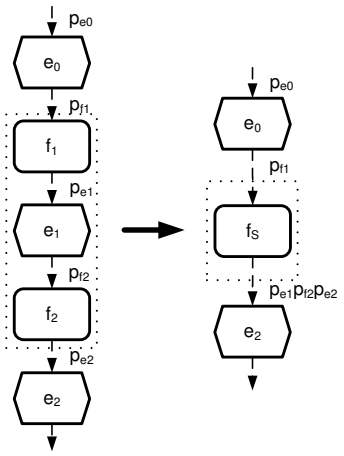

Figure 3: Sequential abstraction

4.2 Sequential Abstraction

Real world business process models of high fidelity often contain sequences of activities, which are captured in EPCs as sequences of functions. Within a *sequential abstraction* a sequence of functions and events is replaced by one function—an aggregating function. An aggregating function has a coarse granularity and brings a process model to a higher level of abstraction.

Definition 6 An EPC process fragment is a *sequence* if it is formed by a function, followed by an event, followed by a function.

Figure 3 exemplifies the concept of sequential abstraction. Functions f_1, f_2, and event e_1 form a sequence. As a result of sequential abstraction, a sequence is replaced by an aggregating function f_S. Semantics of the aggregating function is the following: function f_1 is executed and afterwards function f_2 occurs with the probability equal to the probability of reaching function f_2 from f_1 in the initial process.

The relative effort of an aggregating function depends on the relative efforts of functions f_1 and f_2 and the probability that f_2 occurs in f_S: $e_r(f_S) = e_r(f_1) + e_r(f_2) \cdot p_r((f_1, e_1)) \cdot p_r((e_1, f_2))$.

The relative probability of an aggregating function incoming connection is $p_r(e_0, f_1)$. The relative probability of an aggregating function outgoing connection is defined as $p_r((f_S, e_2)) = p_r(f_1, e_1) \cdot p_r(e_1, f_2) \cdot p_r(f_2, e_2)$

4.3 Block 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 connectors. Depending on the desired semantics, an appropriate connector type is selected: AND, OR, or XOR. A process fragment enclosed between connectors has a precise and self-contained business semantics. Therefore, the fragment can be replaced by one function of coarse granularity. *Block abstraction* enables this operation. 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.

Definition 7 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 4 shows an example of a block. After block abstraction, an original process fragment is replaced by an event, followed by an aggregating function, followed by another event (events are added to assure that a new EPC is well-formed). The approach introduced in this paper supports AND, OR, and XOR connectors. Semantics of the aggregating function conforms to the semantics of the abstracted block and depends on the block type. In case of a XOR block the aggregating function (named f_B) means that only one function of the abstracted fragment is executed.

The relative effort of an aggregating function is independent of a block type and considers the relative efforts of functions f_i and probabilities of reaching these functions from a split connector: $e_r(f_B) = \sum_{i=1}^{k} e_r(f_i) \cdot p_r((c_1, e_{i1})) \cdot p_r((e_{i1}, f_i))$, where k is the number of split outgoing connections.

The relative probability of reaching event e_1 from f_0 equals to the relative probability of reaching node c_1 from its predecessor. The relative probabilities of connections (e_1, f_B) and (e_2, f_{k+1}) are one.

A method for p_x (cf. Figure 4) estimation is block type specific. Let us introduce probability p_i—the probability that a control flow reaches the join connector from the split connector on the i-th branch. Then the probability of reaching e_2 from f_B in an AND block is the probability that control flow on every branch reaches the join connector $p_r((f_B, e_2)) = \prod_{i=1}^{k} p_i$.

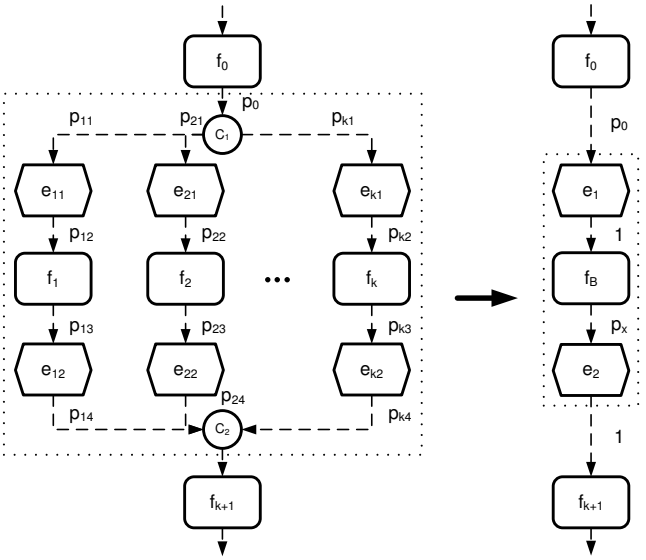

Figure 4: Block abstraction

For a XOR block this probability equals to the probability that the control flow on any branch reaches the join connector $p_r((f_B, e_2)) = \sum_{i=1}^{k} p_i$.

In case of an OR block where all probabilities of leaving functions are equal to one, probability of leaving the block is also one. In general case computation of the probability of leaving an OR block requires prior derivation of probabilities for selecting each branch combination of an OR block. This information can not be obtained solely based on probabilities of executing separate branches.

4.4 Loop Abstraction

It is a common situation when a task (or a set of tasks) in a business process is iterated to complete the process. In a model, capturing such a process, a task (or a set of tasks) is put into a loop construct. EPC enables loop modeling 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 abstraction*. Let us define what kind of process fragment is considered to be a loop.

Definition 8 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

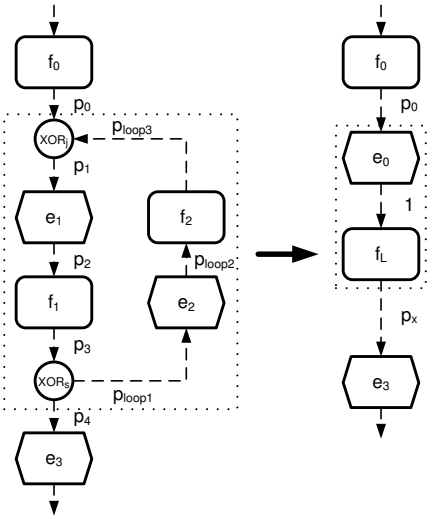

Figure 5: Loop abstraction

- 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.

The whole process fragment corresponding to a loop is replaced after the abstraction by one aggregating function f_L (see Figure 5). An extra event e_0 is inserted between the functions f_0 and f_L in order to obtain well-formed EPC. An aggregating function states that functions f_1 and f_2 are executed iteratively. The definition allows either f_1 or f_2 to be missing. Information about the number of loop iterations is hidden inside the aggregating function and is reflected in its relative effort and connections relative probabilities in the abstracted process model.

The relative effort of an aggregating function can be found as:

$$\begin{aligned} e_r(f_L) &= p_r((xor_j, e_1)) \cdot p_r((e_1, f_1)) \cdot \\ &\cdot \frac{1}{1-p} \cdot (e_r(f_1) + e_r(f_2) \cdot p_r((e_1, xor_s)) \cdot p_l \cdot p_r((e_2, f_2))), \end{aligned}$$

where $p = p_r((xor_j, e_1)) \cdot p_r((e_1, f_1)) \cdot p_r((e_1, xor_s)) \cdot p_r((e_2, f_2)) \cdot p_r((f_2, xor_j)) \cdot p_l$ and $p_l = p_r((xor_s, e_2))$.

After loop abstraction, the probability of reaching e_0 from f_0 equals the probability of reaching the XOR join from function f_0 in the initial process. The probability of reaching aggregating function f_L from event e_0 is one. Probability of leaving the aggregating function (denoted with p_x in Figure 5) is the probability of leaving a loop in the initial

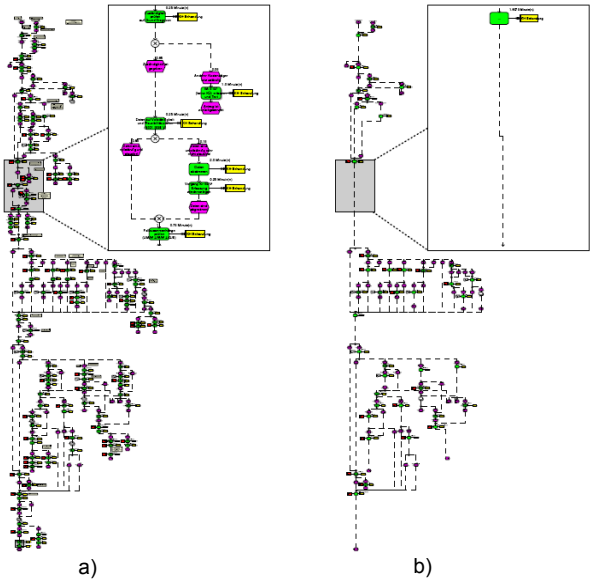

Figure 6: Original (a) and abstracted (b) process models (unreadability intended)

process. Since we assume that probabilities of leaving functions are not always one, it is possible that the control flow does not leave a loop in a process instance. The probability that a loop is stopped between xor_j and xor_s is $p_{stop}^{path} = 1 - p_r((xor_j, e_1)) \cdot p_r((e_1, f_1)) \cdot p_r((f_1, xor_s))$. The control flow stops on the path between xor_s and xor_j with probability $p_{stop}^{loop} = 1 - p_r((e_2, f_2)) \cdot p_r((f_2, xor_j))$.

Thus, the relative probability of leaving an aggregating function equals to:

$$p_r((f_L, e_3)) \quad = \quad 1 - \frac{1}{1-p} \cdot (p_{stop}^{path} + (1 - p_{stop}^{path}) \cdot p_l \cdot p_{stop}^{loop}).$$

4.5 Real World Example

In Figure 6 we present an example of a real world process model from our project partner (cf. Figure 6.a) and the result of its abstraction using presented elementary abstractions (cf. Figure 6.b). The initial process model is composed of 333 nodes: 130 functions, 137 events, and 66 connectors. After abstraction, the number of process model nodes was reduced to 167: 44 functions, 82 events, and 41 connectors. The overall reduction of process nodes is near 50%.

The proposed abstractions allow a company to deal with coarse grained functions in business processes, while keeping the overall process logic intact. In terms of organization and management, these coarse grained functions (with effort of minutes rather than of seconds)

facilitate process improvement on a higher level. Tedious discussions on low granularity functions are no longer required. Instead, process participants can apply improvements within the functions, keeping the overall process logic in sync with the process model.

5 Conclusions

In this paper the elementary abstractions: dead end, sequential, block, and loop abstraction were proposed. In the beginning we have described the challenges of our partner, which they came across managing their process models and which motivated this work. Proposed abstractions can be applied to an arbitrary graph-structured process model. Application of each elementary abstraction aggregates process fragment and brings model to a higher abstraction level. 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. Also, proposed elementary abstractions only address preserving of process effort property (as in Definition 3), as it is the primary requirement of the project partner. However, the approach can be easily extended for recalculation of other properties as in [CMSA02].

Theoretical results of this work are used in the implementation of a tool prototype. The task of the tool is to provide automatic abstraction of process models captured in EPC. The tool supports all types of elementary abstractions proposed in this paper.

As the future steps we identify the task of developing additional elementary abstractions. This implies theoretical foundations of abstraction mechanisms as well as their prototypical implementation. Also, rules for composition of elementary abstractions (application order strategies) need to be studied. An important finding will be to show which class of EPCs can be abstracted to one function by a given set of elementary abstractions. It is also of great interest to learn which set of elementary abstractions is capable of reducing an EPC to one function.

References

[Aal98] W. Aalst. Formalization and Verification of Event-driven Process Chains. Computing Science Reports, Eindhoven University of Technology, Eindhoven, 1998.

[BPM04] BPMI.org. *Business Process Modeling Notation*, 1.0 edition, May 2004.

[BRB07] R. Bobrik, M. Reichert, and Th. Bauer. View-Based Process Visualization. In *BPM*, volume 4714 of *LNCS*, pages 88–95. Springer, 2007.

[CMSA02] 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.

[Dav93] T. Davenport. *Process Innovation: Reengineering Work through Information Technology*. Harvard Business School Press, Boston, MA, USA, 1993.

[DJVVA07] B. Dongen, M. Jansen-Vullers, H. Verbeek, and W. Aalst. Verification of the SAP Reference Models Using EPC Reduction, State-space Analysis, and Invariants. *Comput. Ind.*, 58(6):578–601, 2007.

[GA07] C. Günther and W. 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.

[KNS92] 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.

[LS03] D. Liu and M. Shen. Workflow Modeling for Virtual Processes: an Order-preserving Process-view Approach. *Information Systems*, 28(6):505–532, 2003.

[MA07] J. Mendling and W. Aalst. Formalization and Verification of EPCs with OR-Joins Based on State and Context. In *CAiSE*, pages 439–453, 2007.

[MVD+08] J. Mendling, H. Verbeek, B. Dongen, W. Aalst, and G. Neumann. Detection and Prediction of Errors in EPCs of the SAP Reference Model. *Data Knowl. Eng.*, 64(1):312–329, 2008.

[SO00] W. Sadiq and M. Orlowska. Analyzing Process Models Using Graph Reduction Techniques. *Information Systems*, 25(2):117–134, 2000.

[STA05] 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.

[VVL07] 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.

[Wes07] M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer Verlag, 2007.