Reducing the Complexity of Large EPCs

Artem Polyvyanyy, Sergey Smirnov, Mathias Weske

Technische Berichte Nr. 22

des Hasso-Plattner-Instituts für Softwaresystemtechnik
an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik
an der Universität Potsdam

Nr. 22

Reducing the Complexity of Large EPCs

Artem Polyvyanyy, Sergey Smirnov, Mathias Weske

Potsdam 2008

Bibliografische Information Der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar

Die Reihe *Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam* erscheint aperiodisch.

Herausgeber: Professoren des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam

Redaktion: Artem Polyvyanyy, Sergey Smirnov, Mathias Weske

E-Mail: {artem.polyvyanyy; sergey.smirnov; mathias.weske}@.hpi.uni-potsdam.de

Verlag: Universitätsverlag Potsdam
Am Neuen Palais 10
14469 Potsdam
Fon +49 (0) 331 977 4623
Fax +49 (0) 331 977 4625
E-Mail: ubpub@uni-potsdamde
http://info.ub.uni-potsdam.de/verlag.htm

Druck: allprintmedia gmbH
Blomberger Weg 6a
13437 Berlin
email: info@allprint-media.de

Heft Nr 22 (2008)
ISBN 978-3-940793-29-4
ISSN 1613-5652

Reducing the Complexity of Large EPCs

Artem Polyvyanyy, Sergey Smirnov, Mathias Weske

Business Process Technology Group
Hasso Plattner Institute at the University of Potsdam
D-14482 Potsdam, Germany
{polyvyanyy,smirnov,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 [6]. To improve the understanding of processes and to enable their analysis, business processes are represented by business process models [4, 15]. In typical projects, business processes are described in a detailed manner. The resulting complex process models are hard to comprehend, because every event and every function is represented.

This paper proposes abstraction mechanisms that transform detailed process models in less detailed ones that still reflect the overall process logic. The results are developed for the event-driven process chains (EPC) [7, 13]. However, they can be adapted to any graph-based process modeling notation, for instance the Business Process Modeling Notation (BPMN) [3].

Recently a number of research projects has focused on similar tasks [9, 12, 14]. In [2] the authors discuss a method for creating customized views for an existing process model. An approach for reducing the complexity of process models mined from process logs is presented in [5].

This paper is structured as follows: Section 2 motivates the approach. Afterwards, the fundamental concepts are explained in Section 3. Elementary abstraction mechanisms are presented in Section 4, subsequently it is shown how elementary abstractions can be composed together. An example illustrates the approach in Section 5. Concluding remarks complete this paper.

2 Motivation and Goal

The work was conducted in a joint research project with the health insurance company AOK Brandenburg in Teltow, Germany. The operational processes of the company are captured in no less than 4 000 EPCs, enriched with information about the effort required to complete each function of each process. AOK uses the process models for estimation of workforce required for running the processes; in addition, working procedures are developed from the process models. Detailed models lead to information overload creating a demand for abstracted process models.

Process models consist of automated or/and 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 that are used by companies in their daily work. 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. On the other hand, models are useful for process improvements. This task does not always require detailed specifications of business processes. More so, fine granular process models distract attention of a reader from the overall process logic by exhaustive details.

Agile business environment, where regulations and markets keep changing, makes companies to modify their processes regularly. Since processes are modeled in a very detailed manner, keeping process models up to date requires considerable effort.

Given this background, the goal of this paper is to provide an approach enabling abstraction from process model details. A resulting process model should be easier to comprehend and to adapt to changing requirements.

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.

A complex event-driven process chain from the project partner describing a business process in the medical insurance sector is shown in Figure 1. The process model consists of 333 nodes, 130 of which are functions. Each function of the EPC has an effort associated—a time period required to execute this function. Each connection carries information about a probability of a transition from its source to the target. Based on this information and the number of process instances per year, the annual effort of this process can be determined. The average processing time of a business process instance is 5.57 minutes. With roughly 240 000 instances per year, the overall process effort adds up to over 22 000 working hours per year. A method for estimation of a business process cost (which is similar to effort concept) based on the extended BPMN diagram is proposed in [10]. However, it does not address the problem of process abstraction.

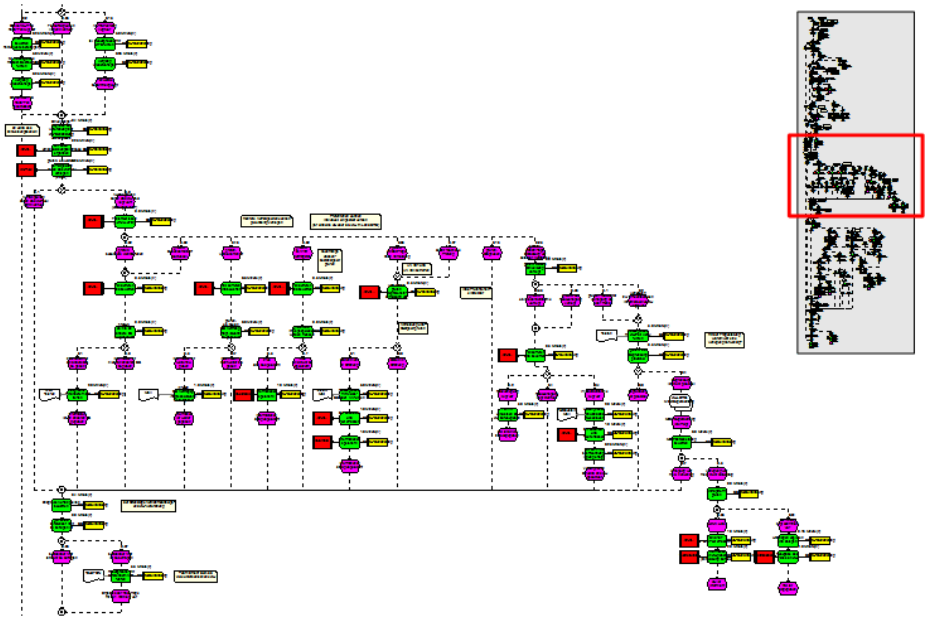

Fig. 1. Complex event-driven process chain (unreadability intended)

The investigated process models are used to estimate the work force required to run business processes. This means that function efforts, connection probabilities and the number of cases per year are used as basis for an estimation of a head count of departments. 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.

3 Fundamentals

In this section the fundamentals of the approach are introduced, i.e. formalization of the event-driven process chains, requirements and assumptions of the approach, concepts of efforts and probabilities.

3.1 Formalizing EPC

The event-driven process chains play an important role in the business process engineering research community. There exist several works on formalization of EPC [1, 8, 11, 15]. In this paper we base our approach on the formal definition proposed in [15].

Definition 1. A tuple (E, F, C, A, t) is an *event-driven process chain* 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
- (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 event has exactly one outgoing connection and no incoming connections. Each end event has exactly one incoming connection and no 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.
- 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 (see Figure 2). 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.

3.2 Assumptions, Stipulations, Requirements

As a requirement, process model abstraction methodology must be mean process effort preserving, i.e. the mean process effort before and after abstraction should be the same. Mean process effort can be obtained from the process model where each function is enriched with mean effort and mean number of occurrences. Such information can be obtained from process execution statistics, e.g. collected by a workflow execution engine, or estimated by a process modeler. As an alternative, a process model can be supplied with connection transition probabilities, which unambiguously define mean number of process node occurrences and vice versa.

Further we assume that probabilities are associated with process connection transitions and function mean execution efforts are provided. All the subsequently presented abstraction rules will exploit this knowledge to derive probabilities and efforts for the modified parts of the output process model. Proposed modifications will have to ensure a mean process effort to stay unchanged.

We do not assume any limitations on the initial process model control flow structure. However, proposed process model abstraction mechanisms implicitly define a set of addressed control flow patterns.

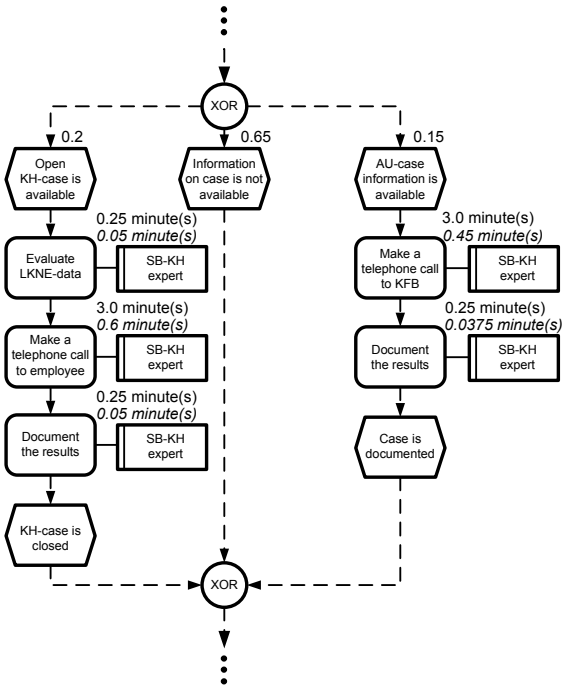

Fig. 2. Real world example of the EPC fragment enriched with probabilities and efforts (Definition 1 is not satisfied because events are missing between successive functions)

3.3 Probabilities and Efforts

As input for an abstraction we obtain an EPC with additional data on connection transition probabilities and time required to execute process functions. Following definitions are based on this information.

Definition 2. *Relative probability* of reaching a process node n from one of its predecessors n_p is the probability of a connection transition from n_p to n: $p_r : \{(n_p, n) \in A | n_p \in N, n \in N\} \rightarrow [0, 1]$.

Definition 3. *Mean occurrence number of a node* is the mean number that the node will occur in a process instance.

Definition 4. *Relative effort of a process function* (e_r) is the time required to execute the function: $e_r : F \rightarrow \mathbb{R}^+$.

Definition 5. *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 6. *Process absolute effort* (e_a^p) is the mean effort required to execute a process instance: $e_a^p : P \to \mathbb{R}^+$, where P is a set of process models. Process absolute effort can be obtained as the sum of absolute efforts of process functions.

Figure 2 shows the fragment of the EPC from Figure 1 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 "Make a telephone call to employee" has the relative effort of 3 minutes meaning that it is expected to take 3 minutes of worker's time once reached in a process instance. On average, this function requires $3 \cdot 0.2 = 0.6$ 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.

4 Elementary Abstractions

In this section elementary abstractions are presented. Elementary abstractions define how certain types of process fragments are generalized. Elementary abstractions satisfy the requirements defined in the previous section, e.g. they are effort preserving. This section discusses dead end, sequential, block and loop abstraction.

The abstractions can be applied in any order or frequency, provided 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 7. 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 3 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

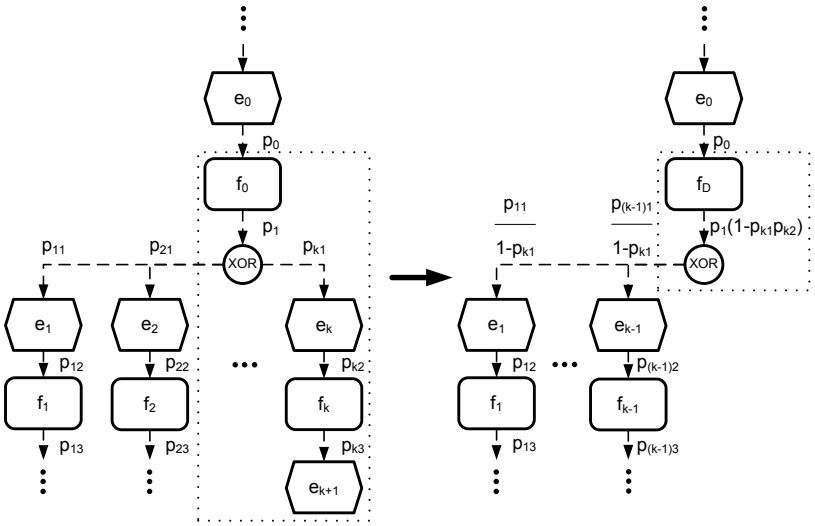

Fig. 3. Dead end abstraction

end. The XOR split has k outgoing branches and after the abstraction the k-th branch is removed. On the right side of Figure 3 the abstracted process is presented. The dead end fragment and its replacement are enclosed within the rectangles with dotted borders.

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 is continued.

From a practical point of view, the naming of an aggregating function is very important. Rather than providing a generic name automatically, we combine the names of the aggregated functions and allow a modeler to edit the name of the aggregating function.

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 an aggregating function from event e_0 equals the relative probability of reaching the replaced function f_0 from event e_0 in the initial process:

$$p_r((e_0, f_D)) = p_r((e_0, f_0)).$$

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, the dead end abstraction leaves only one XOR split outgoing connection. Therefore, it is possible to omit the XOR split in the abstracted process. A new connection from the aggregating function to the event, following the omitted XOR split, should be added to the EPC. The connection relative probability equals the probability of reaching this event from function f_0 in the initial process.

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

Figure 4 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.

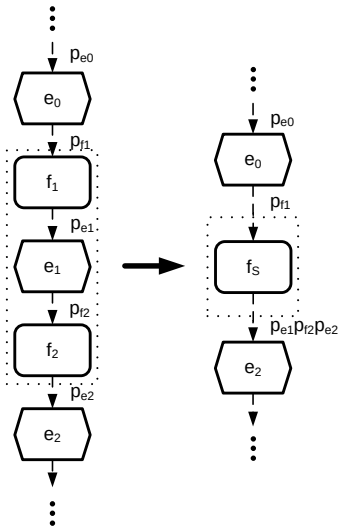

Fig. 4. Sequential abstraction

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 connector. 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 9. 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

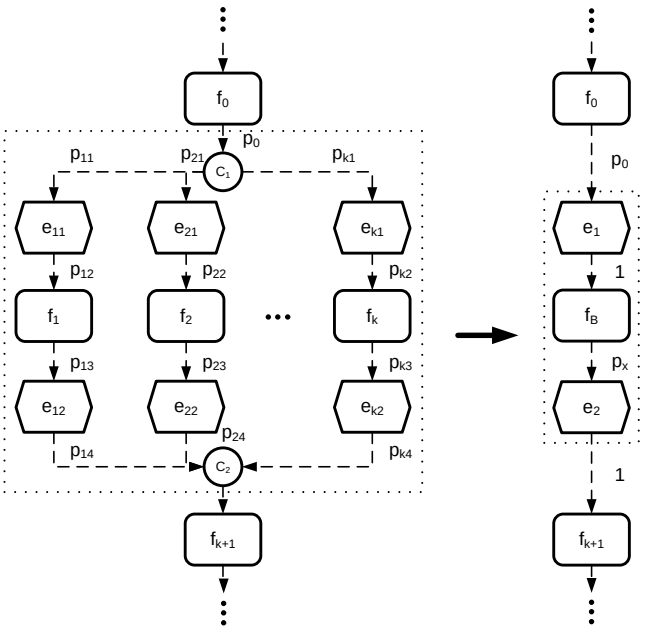

Fig. 5. Block abstraction

- 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.

It should be noticed that the definition does not allow a join connector to have incoming branches from outside the block. Figure 5 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. For instance, in case of a XOR block the aggregating function means that only one function of the abstracted fragment is executed. In general we name an aggregating function f_B.

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 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.

Hitherto, we have specified every block abstraction rule, except a method for finding the relative probability of an aggregating function outgoing connection (denoted with p_x in Figure 5). If in the initial process no branch of a block contained a dead end, the probability of leaving the block equals the probability of reaching it and p_x is one. However, if a branch of a block contained a dead end, a dead end abstraction has to be performed to enable the block abstraction. As a result of dead end abstraction, a function on a block branch is left with probability less than one (see subsection 4.1). In this case a method for p_x 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:

$$p_r((f_B, e_2)) = \prod_{i=1}^{k} p_i.$$

For a XOR block this probability equals the probability that the control flow on any branch reaches the join connector:

$$p_r((f_B, e_2)) = \sum_{i=1}^{k} p_i.$$

For an OR block where at least one probability of leaving a function is less than one further research is required.

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 10. 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.

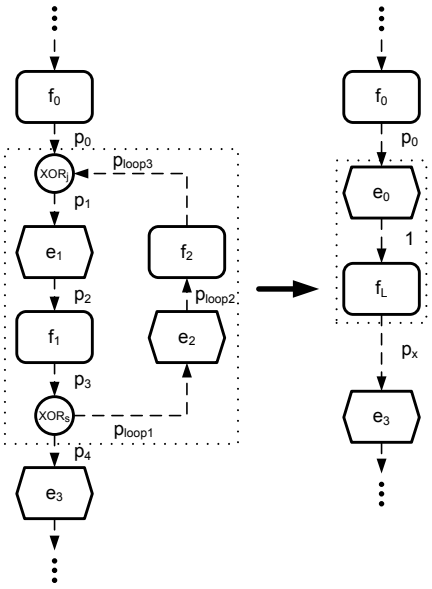

Fig. 6. Loop abstraction

The whole process fragment corresponding to a loop is replaced after the abstraction by one aggregating function f_L (see Figure 6). An extra event e_0 is inserted between the functions f_0 and f_L in order to obtain correct EPC. An aggregating function states that functions f_1 and f_2 are executed iteratively. 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:

$$e_r(f_L) = p_r((xor_j, e_1)) \cdot p_r((e_1, f_1)) \cdot \\ \cdot \sum_{i=0}^{\infty} p^i \cdot (e_r(f_1) + e_r(f_2) \cdot p_r((e_1, xor_s)) \cdot p_l \cdot p_r((e_2, f_2))),$$

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 6) is the probability of leaving a loop in the initial 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:

$$p_{stop}^{loop} = 1 - p_r((e_2, f_2)) \cdot p_r((f_2, xor_j)).$$

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

$$p_r((f_L, e_3)) = 1 - \sum_{i=0}^{\infty} p^i \cdot (p_{stop}^{path} + (1 - p_{stop}^{path}) \cdot p_l \cdot p_{stop}^{loop}).$$

4.5 Applying Elementary Abstractions

So far elementary abstractions that aim to get rid of process details were presented. Each abstraction is characterized by a process fragment it abstracts from and defined transformation rules. A single application of an elementary abstraction is not of great value for the task of process abstraction. Therefore, strategies prescribing rules of elementary abstraction compositions are introduced. Conceptually, an abstraction strategy is a sequence of elementary abstraction steps. At each abstraction step one of the proposed elementary abstractions is applied. An abstraction step is atomic, i.e. does not depend on the previous ones. Thus, one might come up with various project specific abstraction strategies, leading to different resulting abstracted process models.

Each abstraction step is aimed to reduce the overall number of functions in a process model. We propose to derive an order of abstraction steps based on the function absolute effort priority. Hence, at first place we aim to abstract from functions of low weight, i.e. concepts that require less effort. Once the function with the lowest absolute effort is identified, it can be questioned to be a part of the elementary abstraction process fragment. Upon success, abstraction transformation rules are applied. Otherwise, an attempt to apply another elementary abstraction is made. The choice of an elementary abstraction can be carried out based on the predefined priority. Abstraction process is continued until there is no more elementary abstraction process fragment recognized, or the lowest absolute effort of a function in the process has reached the preset threshold.

As basic abstraction strategies one can identify pure strategies of iteratively applying only one kind of elementary abstraction. As a result, one will obtain an abstracted process model where only sequential, dead end, block or loop abstraction is performed. For instance, in case of pure sequential abstraction strategy, sequences of an arbitrary length can be reduced to one function.

An advanced abstraction strategy can be obtained by specifying the priority order on elementary abstractions in which they are attempted to be applied on a function with the lowest absolute effort. The precedence of sequential, dead end, block and then loop abstraction is one possible strategy. This strategy ensures that all elementary abstractions are performed; application of one elementary abstraction might enable further application of another that will be triggered at one of the subsequent abstraction steps.

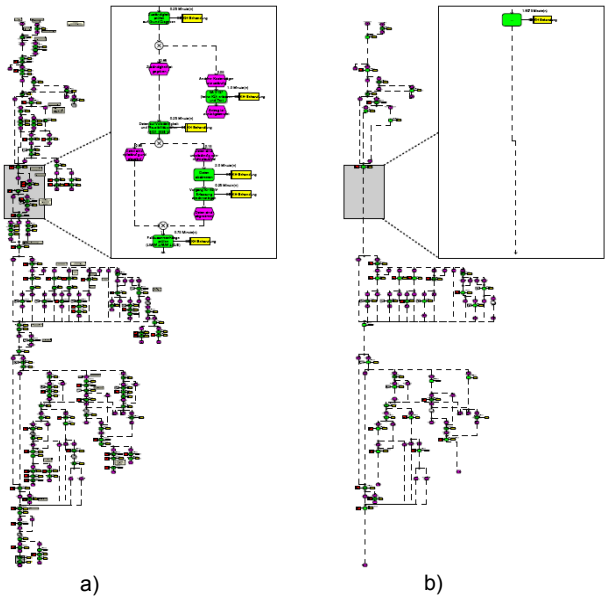

Fig. 7. Original and abstracted process models

5 Real World Example

In this section the developed abstraction approach is illustrated by a real world example. The process model presented in Figure 1 is chosen as the subject for abstraction. The following abstraction strategy is used: sequential, dead end, block and then loop elementary abstraction.

Figure 7 gives a good comparison of the initial process model (a) and the result of its abstraction (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 composed of: 44 functions, 82 events and 41 connector. The overall reduction of process nodes is near 50%.

Application of the abstraction steps is illustrated in Figure 8. Here, Figure 8.a shows the initial EPC process fragment (enlarged in Figure 7.a). All of the subsequent fragments are derived by applying one or several elementary abstractions to the initial process fragment. As a result (see Figure 8.e), we obtain the process fragment enlarged in the abstracted process model in Figure 7.b. The first step is application of the sequential abstraction inside of the block structure: two sequential functions are replaced by one (see Figure 8.b). As the next step, the dead end abstraction is applied (see Figure 8.c). At the third step, we perform the block abstraction; the result is visualized in Figure 8.d. Finally, in order to obtain the process fragment given in Figure 8.e. three successive sequential abstractions need to be performed. Consequently, the initial process fragment is represented by one aggregating function that incorporates information about

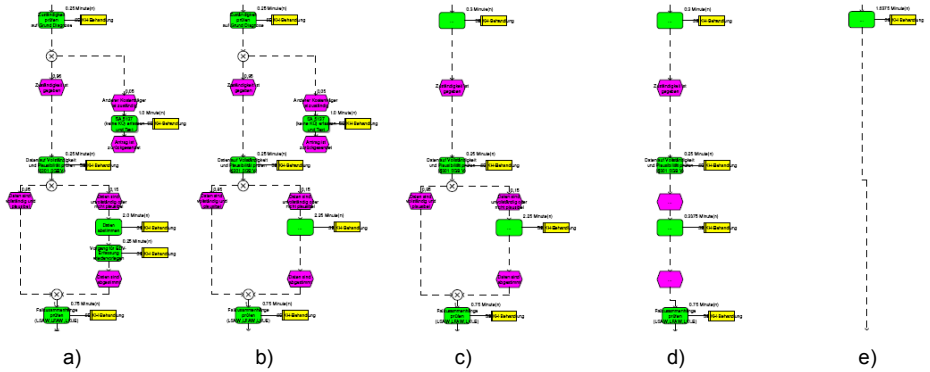

Fig. 8. Applying a sequence of abstraction steps

the initial control flow structure and has the absolute effort equal to the absolute effort of the aggregated fragment.

6 Conclusions

In this paper the approach to effort preserving abstraction of business process models was presented. In the beginning we have described the challenges of our partner, AOK Brandenburg, which they came across managing their process models and which motivated this work. Afterwards, the main concepts employed in the approach were introduced. Further, elementary abstractions: dead end, sequential, block and loop abstraction were proposed. For every elementary abstraction it was defined to which type of process fragment it can be applied and in which model transformation it results. Abstraction strategies, the approach to combining elementary abstractions, were introduced. Finally, the abstraction of the EPC from the project partner was discussed.

Theoretical results of this work were used in the implemented tool prototype. The task of the tool is to provide automatic abstraction of process models captured in EPC. The results of the tool work are presented in Figure 8 and are discussed in detail in Section 5. The tool supports all types of elementary abstractions proposed in this paper. The prototype implements the abstraction strategy based on the function effort priority.

As the future steps we identify the task of developing additional elementary abstractions. This implies the theoretical foundations of abstraction mechanisms as well as their prototypical implementation. 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 or to prove that such a set does not exist.

Acknowledgments: The authors acknowledge the support of the project partner AOK Brandenburg in Teltow, Germany, in particular Dr. Anke-Britt Möhr, Norbert Sandau and Anja Niedersätz.

References

1. W. Aalst. Formalization and Verification of Event-driven Process Chains. Computing Science Reports, Eindhoven University of Technology, Eindhoven, 1998.
2. R. Bobrik, M. Reichert, and T. Bauer. View-Based Process Visualization. In *BPM 2007, volume 4714 of LNCS*, pages 88–95, Berlin, 2007. Springer Verlag.
3. BPMI.org. *Business Process Modeling Notation*, 1.0 edition, May 2004.
4. T.H. Davenport. *Process Innovation: Reengineering Work through Information Technology.* Harvard Business School Press, Boston, MA, USA, 1993.
5. C. W. 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.
6. M. Hammer and J. Champy. *Reengineering the Corporation: A Manifesto for Business Revolution.* HarperBusiness, April 1994.
7. G. Keller, M. Nüttgens, and A.W. Scheer. Semantische Prozessmodellierung auf der Grundlage "Ereignisgesteuerter Prozessketten (EPK)". Technical Report Heft 89 (in German), Veröffentlichungen des Instituts für Wirtschaftsinformatik University of Saarland, Saarbrücken, 1992.
8. P. Langner, C. Schneider, and J. Wehler. Petri Net Based Certification of Event-Driven Process Chains. In *Proceedings of the 19th International Conference on Application and Theory of Petri Nets*, pages 286–305, London, UK, 1998. Springer Verlag.
9. D. Liu and M. Shen. Workflow Modeling for Virtual Processes: an Order-preserving Process-view Approach. *Information Systems*, 28(6):505–532, 2003.
10. M. Magnani and D. Montesi. BPMN: How Much Does It Cost? An Incremental Approach. In *BPM 2007, volume 4714 of LNCS*, pages 80–87, Berlin, 2007. Springer Verlag.
11. J. Mendling and W. Aalst. Formalization and Verification of EPCs with OR-Joins Based on State and Context. In *CAiSE*, pages 439–453, 2007.
12. W. Sadiq and M. E. Orlowska. Analyzing Process Models Using Graph Reduction Techniques. *Information Systems*, 25(2):117–134, 2000.
13. A.W. 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.
14. 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, Berlin, 2005. Springer Verlag.
15. M. Weske. *Business Process Management: Concepts, Languages, Architectures.* Springer Verlag, 2007.

Aktuelle Technische Berichte des Hasso-Plattner-Instituts

Band	ISBN	Titel	Autoren / Redaktion
21	978-3-940793-17-1	**"Proceedings of the 2nd International Workshop on e-learning and Virtual and Remote Laboratories"**	Bernhard Rabe, Andreas Rasche
20	978-3-940793-02-7	**STG Decomposition: Avoiding Irreducible CSC Conflicts by Internal Communication**	Dominic Wist, Ralf Wollowski
19	978-3-939469-95-7	**A quantitative evaluation of the enhanced Topic-based Vector Space Model**	Artem Polyvyanyy, Dominik Kuropka
18	978-939-469-58-2	**Proceedings of the Fall 2006 Workshop of the HPI Research School on Service-Oriented Systems Engineering**	Benjamin Hagedorn, Michael Schöbel, Matthias Uflacker, Flavius Copaciu, Nikola Milanovic
17	3-939469-52-1 / 978-939469-52-0	**Visualizing Movement Dynamics in Virtual Urban Environments**	Marc Nienhaus, Bruce Gooch, Jürgen Döllner
16	3-939469-35-1 / 978-3-939469-35-3	**Fundamentals of Service-Oriented Engineering**	Andreas Polze, Stefan Hüttenrauch, Uwe Kylau, Martin Grund, Tobias Queck, Anna Ploskonos, Torben Schreiter, Martin Breest, Sören Haubrock, Paul Bouché
15	3-939469-34-3 / 978-3-939469-34-6	**Concepts and Technology of SAP Web Application Server and Service Oriented Architecture Products**	Bernhard Gröne, Peter Tabeling, Konrad Hübner
14	3-939469-23-8 / 978-3-939469-23-0	**Aspektorientierte Programmierung – Überblick über Techniken und Werkzeuge**	Janin Jeske, Bastian Brehmer, Falko Menge, Stefan Hüttenrauch, Christian Adam, Benjamin Schüler, Wolfgang Schult, Andreas Rasche, Andreas Polze
13	3-939469-13-0 / 978-3-939469-13-1	**A Virtual Machine Architecture for Creating IT-Security Labs**	Ji Hu, Dirk Cordel, Christoph Meinel
12	3-937786-89-9 / 978-3-937786-89-6	**An e-Librarian Service - Natural Language Interface for an Efficient Semantic Search within Multimedia Resources**	Serge Linckels, Christoph Meinel
11	3-937786-81-3	**Requirements for Service Composition**	Prof. Dr. M. Weske, Dominik Kuropka Harald Meyer
10	3-937786-78-3	**Survey on Service Composition**	Prof. Dr. M. Weske, Dominik Kuropka Harald Meyer
9	3-937786-73-2	**Sichere Ausführung nicht vertrauenswürdiger Programme**	Andreas Polze Johannes Nicolai, Andreas Rasche

ISBN 978-3-940793-29-4
ISSN 1613-5652