On Application of Structural Decomposition for Process Model Abstraction

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: Real world business process models may consist of hundreds of elements and have sophisticated structure. Although there are tasks where such models are valuable and appreciated, in general complexity has a negative influence on model comprehension and analysis. Thus, means for managing the complexity of process models are needed. One approach is abstraction of business process models—creation of a process model which preserves the main features of the initial elaborate process model, but leaves out insignificant details. In this paper we study the structural aspects of process model abstraction and introduce an abstraction approach based on process structure trees (PST). The developed approach assures that the abstracted process model preserves the ordering constraints of the initial model. It surpasses pattern-based process model abstraction approaches, allowing to handle graph-structured process models of arbitrary structure. We also provide an evaluation of the proposed approach.

1 Introduction

Business process management is an important approach to represent and improve the way companies work in dynamic and competitive settings [HC94]. In most cases one model for one business process is not enough, since different types of business process analysis require different perspectives of one process. Certain analysis tasks imply exhaustive process description, while others need only a process overview. Therefore, there is a demand for several models of the same process, but with different levels of abstraction. Meanwhile, without a formal relation between these models, consistency between them cannot be guaranteed. Model maintenance becomes a pricey and error-prone task.

To cope with this problem, modeling notations, like Business Process Modeling Notation (BPMN) [BPM08] or Event-driven Process Chains (EPC) [KNS92, STA05], allow hierarchical model structuring. Such structuring gives a user a possibility to organize model details putting them to the appropriate level in the model hierarchy. However, hierarchical structuring requires a user to decide to which abstraction level an element should be related.

Business process model abstraction addresses the outlined problem. Abstraction generalizes the model, leaving out insignificant details. Under the assumption that a company already possesses a repository of detailed process models, abstraction derives a set of

coarse-granular process representations.

There exists a number of approaches which address the abstraction task. However, they have certain limitations. One common drawback is handling of block-structured process models only [EG08, PSW08b]. This limitation becomes crucial in practical tasks where process models have arbitrary structure. Other approaches are generic, but cannot guarantee preservation of the ordering constraints of the initial model [BRB07, GA07].

In this paper we propose an approach for business process model abstraction based on the construction and analysis of process structure trees (PST) [VVL07]. The proposed approach allows to handle arbitrary graph-structured process models. Furthermore, it preserves the ordering constraints of the initial model. The developed process model abstraction focuses on the structural aspects of the abstraction, rather than on the semantics of the model. This means that the method tells how model elements can be correctly abstracted, but does not tell which elements should be abstracted. We evaluate efficiency of the presented technique, i.e., its capability to leave out process details gradually.

This paper has the following structure. Section 2 explains the concept of process model abstraction. In section 3 we identify requirements a business process model abstraction should meet and formulate key assumptions of this work. Section 4 describes a method of PST construction and the abstraction mechanism. An evaluation of the proposed abstraction mechanism concludes the section. In section 5 an outline of the related work is given. Finally, the conclusions and the future work are provided.

2 Business Process Model Abstraction

Process model abstraction is generalization of a model which leaves out insignificant process details in order to reduce complexity of the model and retain information relevant for a particular purpose. Therefore, information loss is the fundamental property of abstraction and is its desirable outcome. When business process model abstraction is discussed, one can imagine various details to be omitted: activities, events, or even whole execution paths. The choice of subjects to be abstracted is usually dictated by user needs. In [PSW08a] we identified several use cases for process model abstraction. For each use case it was shown which elements of the model are subject for abstraction.

When business users talk about process model abstraction, they often imply abstraction of activities, requesting a transition from low level steps to high level tasks. Several research projects in this area (see [BRB07, EG08]), as well as our research experience [PSW08a, PSW08b], prove that activities are often in the focus of abstraction. Therefore, we use activities as the abstraction subject.

In this paper we introduce a simplified process modeling notation, rather than using notations like EPC or BPMN. We aim at choosing the simplest process model formalism that enables our task. In the model we consider activities, which are abstraction subjects, and gateways, which define control flow logic. We do not address events. Obviously, events are the core elements of many modeling notations, like EPC or BPMN. Nevertheless, without loss of generality we neglect events in the developed abstraction approach. This design de-

cision allows us to adapt the approach to various modeling notations, where semantics of events may vary from one notation to another. Accordingly, we define a business process model.

Definition 1 $(N, E, type)$ is a *business process model* where:

- $N = N_A \cup N_G$ is a set of nodes, where $N_A \neq \emptyset$ is a set of activities and N_G is a set of gateways; the sets are disjoint
- $E \subseteq N \times N$ is a set of directed edges between nodes representing control flow
- (N, E) is a connected graph
- every activity has at most one incoming and at most one outgoing edge
- there is at least one activity which has no incoming edges—a start activity, and at least one activity which has no outgoing edges—an end activity
- $type : N_G \rightarrow \{and, xor, or\}$ is a function that assigns to each gateway a control flow construct
- every gateway is either a split or a join; splits have exactly one incoming edge and at least two outgoing; joins have at least two incoming edges and exactly one outgoing.

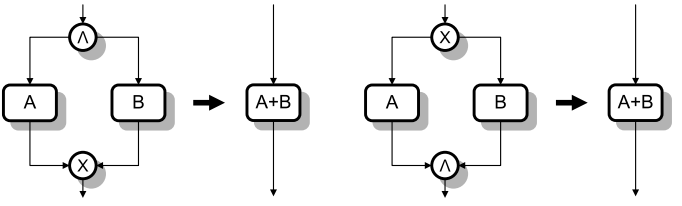
According to this definition an activity has no more than one incoming and one outgoing edge. Some modeling notations impose this restriction (e.g., EPC), while others (e.g., BPMN) allow activities to have multiple incoming/outgoing edges. However, by introducing gateways, it is always possible to transform the model in the way that every activity has exactly one incoming edge and one outgoing edge.

We propose to implement process model abstraction in several steps. In every *abstraction step* one activity from a set of insignificant activities is processed. The output of the current abstraction step is a new process model, which is the input for the next abstraction step. Abstraction evolves until every insignificant activity is handled.

Every abstraction step reduces the number of model elements. We distinguish two methods to abstract an element: *aggregation* or *elimination*. Aggregation replaces several elements with one aggregating element. The properties of an aggregating element are derived from the properties of the elements which are aggregated. Therefore, aggregation preserves information about the abstraction subject in a model. Elimination, on the other hand, does not preserve information. Elimination simply omits the element in the abstracted process model.

3 Assumptions and Requirements

In this section we discuss the underlying assumptions and the requirements of the process model abstraction approach to be proposed. We argue why the formulated assumptions are



(a) Abstraction of a model fragment with lack of synchronization

(b) Abstraction of a model fragment containing a deadlock

Figure 1: Abstraction of process models that are not sound

important. Following in this section we introduce two requirements: *order preservation* and *smoothness*.

3.1 Assumptions on Process Models

Business process modeling is a creative task that allows humans to represent process knowledge in a formal way. However, modeling practices permit people to end up with wrong models, i.e., models that are not sound or not safe. Safeness assures that no activity is enabled more than once at one point in time. The idea of soundness is to make sure that all tasks can participate in a process instance, every process instance eventually terminates, and when it terminates there is no running activities in a process [Wes07]. An abstraction step performs generalization of a *process model fragment*—a connected subgraph of a graph representing a process model. As a result, the problems existing in the model might be unintentionally concealed. Thus, the abstracted process model might become correct, while the original model was not. In Figure 1(a) an example of a process model fragment with multi-merge control flow pattern [RHAM06] is shown (i.e., model is not safe). After fragment generalization the problem is hidden. Similarly, an abstraction step might generalize a process model with a deadlock to a sound one (see Figure 1(b)). The given examples clearly illustrate that abstraction can substantially change the process logic, not only because insignificant fragments are generalized, but also because modeling errors can get hidden. To avoid confusing situations we allow abstraction of only correct, i.e., sound process models.

The explicit assumption on process soundness further imply that every process model can have only one distinguished start node and only one distinguished end node. This observation is important for the future discussion of the abstraction approach, since it is based on the decomposition algorithm that assumes models to have exactly one start node and one end node.

3.2 Requirements

The developed technique of process model abstraction should be order preserving and smooth. We realize process model abstraction as a sequential application of abstraction steps. Both of the stated requirements can be discussed within a scope of a single abstraction step.

Essentially, process model abstraction should preserve the ordering constraints of an initial model. 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 two activities appear in sequence. Assume that activity A should be abstracted in the current abstraction step. Let f_A be a process fragment affected by this abstraction step (f_A contains A). As a result of abstraction, f_A gets replaced by activity F. If activity B also belongs to f_A, information about the ordering constraints between activities A and B is lost. However, the order preserving abstraction should assure that for any pair of activities not in f_A, e.g., activities C and D, the ordering constraints between them are preserved. Furthermore, the order preserving abstraction must guarantee that the ordering constraints between any activity not in f_A, e.g., activity E, and any activity in f_A, activities A or B in our example, are the same as between activities E and F. In the end, the order preserving abstraction secures the overall process logic to be reflected in abstracted model.

The fundamental characteristic of abstraction is that it leads to information loss. If a sequence of activities is abstracted to one activity, loss of information about generalized activities and their relations is intended. Abstraction technique should provide effective mechanisms to achieve (and not to under- or overachieve) the desired level of information loss and available abstraction steps might allow or not allow this. In general, the "smaller" the abstraction step (the less process information it generalizes), the better it suits for achievement of a precise model information level. In order to quantitatively measure the precision of the abstraction technique we introduce a notion of *abstraction smoothness*. Abstraction smoothness quantitatively estimates the information loss produced by one abstraction step. In case of an abstraction which is based on activity generalization, the abstraction smoothness reflects the difference between the number of activities in the process model before and after one abstraction step. The less activities are generalized in a single step, the smoother is the abstraction. From the user perspective it is important to have a smooth abstraction which allows reaching required model information level and forbids undesired side effects. The smoothness of application of an abstraction technique can be obtained as the mean of smoothness for every abstraction step.

4 Abstraction Approach

In this section we present the algorithm of PST decomposition of a process model. Afterwards, we show how PST can be used for the purpose of process abstraction. Finally, we evaluate the approach.

4.1 Process Model Decomposition

As we have argued, the abstraction algorithm transforms a process model stepwise, affecting one model fragment at a time. We propose to use a well established algorithm to derive fragments from a process model. The algorithm enables decomposition of a process model into special kind of process fragments called *canonical single entry single exit (SESE) fragments*. Informally, a SESE fragment is a fragment which has exactly one incoming edge and exactly one outgoing edge. From the perspective of the abstraction task SESE fragments are very handy: structurally every SESE fragment can be replaced with one aggregating activity. The semantics of this new aggregating activity corresponds to the semantics of the replaced process fragment.

To formalize the concept of canonical SESE fragments, auxiliary concepts have to be introduced. We assume a process model to have one start activity and one end activity. This assumption aligns with the discussion of the assumptions and requirements in section 3. We say that node x *dominates* node y in a process model graph if every path from start activity to y includes x. Node x *postdominates* node y in a process model graph if every path from y to end activity includes x. The concepts of dominance and postdominance can be transferred to edges. Thus, SESE fragment and canonical SESE fragment can be defined in the following way.

Definition 2 *A SESE fragment in graph G is a process model fragment defined by an ordered edge pair (a, b) of distinct control flow edges a and b, where:*

1. *a dominates b,*
2. *b postdominates a,*
3. *every cycle containing a also contains b and vice versa.*

Edge c belongs to the SESE fragment defined by (a, b), if c postdominates a and c dominates b. We say that node n belongs to the node set of a SESE fragment if all the incident edges of this node belong to the fragment.

SESE fragment defined by (a, b) is canonical if b dominates b' for any SESE fragment defined by (a, b') and a postdominates a' for any SESE fragment defined by (a', b).

Definition 1 distinguishes two node types: activities and gateways. In the decomposition algorithm we do not make use of it, since activities and gateways are treated simply as nodes of a graph. In Figure 2 an example of a process model is shown. Canonical SESE fragments are marked with a dashed line; those which contain more than one activity are named X, Y, and Z.

We define two types of relations between canonical SESE fragments: parent-child and predecessor-successor. From Definition 2 it follows that the node sets of two canonical SESE fragments are either disjoint or one contains the other. That is why a parent-child relation can be introduced for canonical SESE fragments. If the node set of SESE fragment s_1 is the subset of the node set of SESE fragment s_2, then s_1 is the *child* of s_2 and s_2 is

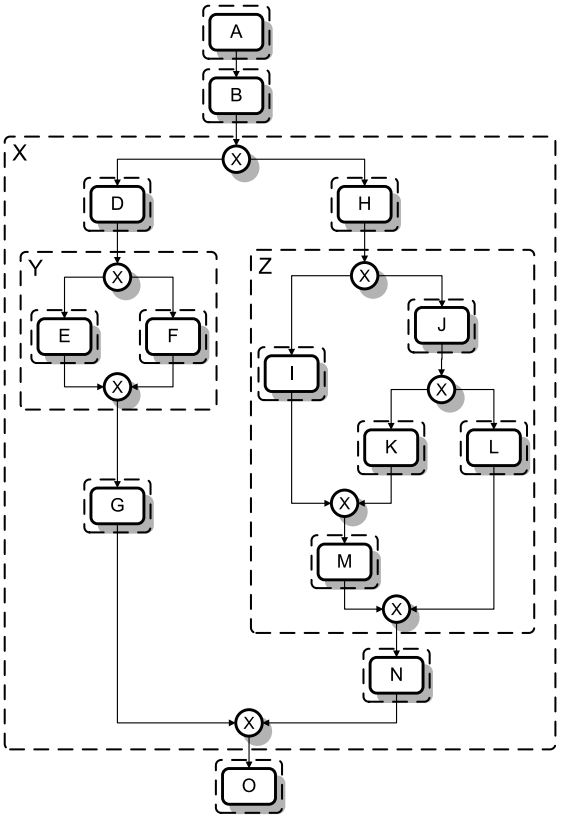


Figure 2: Process model decomposed into canonical SESE fragments

the *parent* of s_1. If s_1 is the child of s_2 and there is no s_3, such that s_3 is the child of s_2 and s_3 is the parent of s_1, s_1 is the *direct child* of s_2. Canonical SESE fragments can be organized into a hierarchy according to the parent-child relation. The hierarchy is represented with a directed tree called *process structure tree*. The tree nodes represent canonical SESE fragments. Let tree nodes n_1 and n_2 correspond to SESE fragments s_1 and s_2 respectively. An edge leads from tree node n_1 to n_2 if SESE fragment s_1 is the direct parent of s_2. Figure 3 presents the PST for the process model from Figure 2. Node R is the root and corresponds to the whole process model. Canonical SESE fragment K is the direct child of Z, therefore, there is a directed edge between the corresponding nodes in the tree.

Two canonical SESE fragment can be in the predecessor-successor relation. We say that s_1 precedes s_2 (and s_2 succeeds s_1) if the outgoing edge of s_1 is the incoming edge of s_2. One can observe that only the sibling nodes can be in the predecessor-successor relation. In the PST we visualize sequences of nodes which are in predecessor-successor relation using dotted border rectangles. For instance, canonical SESE fragments H, Z, and N are put in the rectangle.

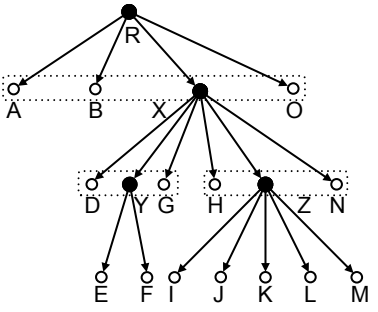


Figure 3: PST corresponding to the process model from Figure 2

4.2 Abstraction Mechanism

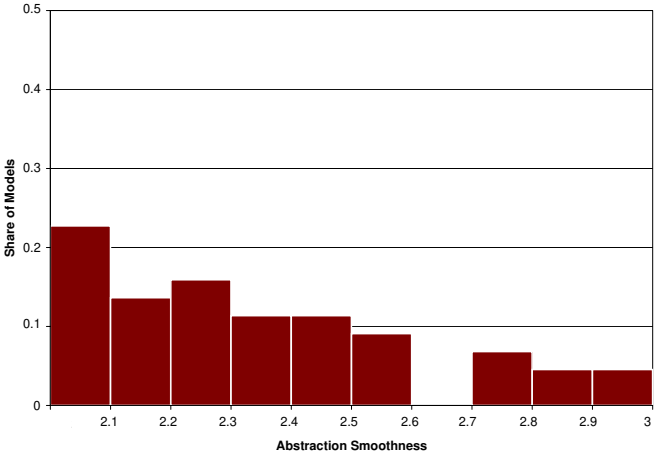
Once a process model is decomposed into canonical SESE fragments and the corresponding PST is built, abstraction can be applied. In the abstraction approach we rely solely on aggregation of activities. This means that in every abstraction step two or more activities are aggregated. Let A be an activity to be abstracted in the current step. We aim to find the minimal canonical SESE fragment $sese_{min}$, containing A and at least one more activity in order to perform generalization. Every activity has one incoming edge and one outgoing edge. Thus, it constitutes a canonical SESE fragment, represented by a leaf in the PST. Hence, we traverse all the leaves in the PST and find the one containing A. Let us call it $sese_A$. The discovered fragment contains only A and is of no use for the abstraction; $sese_A$ cannot be used as $sese_{min}$. There are two options for the selection of $sese_{min}$:

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

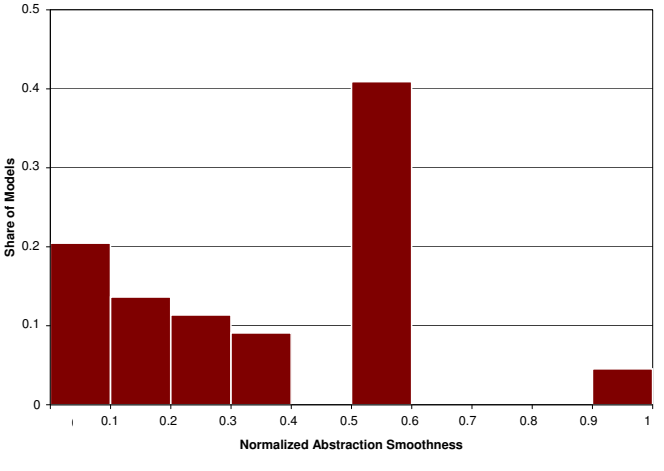
Once $sese_{min}$ is identified, it is replaced with one aggregating activity in the process model. The incoming edge of the aggregating activity is the incoming edge of $sese_{min}$, while its outgoing edge is the outgoing edge of $sese_{min}$.

After the abstraction mechanism was discussed, we would like to summarize its properties. The developed process model abstraction preserves the ordering constraints in the sense described in section 3. This can be shown with the following reasoning. Assume we abstract activity E contained in canonical SESE fragment Y (see Figure 2). According to the algorithm, SESE fragment Y is replaced with one activity EF. Information about the control flow within Y is lost. Ordering constraints between any two activities, which do not belong to Y (e.g., D and H), are preserved. This holds, since Y has exactly one incoming and one outgoing edge and all the transformations are localized within fragment Y. Control flow relations between any activity, which does not belong to Y, (e.g., D)

and the aggregating activity EF are the same as between D and every activity which is contained in Y. Again, this is true, since Y has exactly one incoming and one outgoing edge.



(a) Optimistic case



(b) Pessimistic case

Figure 4: Evaluation of abstraction smoothness

4.3 Smoothness Evaluation

The abstraction smoothness of the presented approach can be measured as the average number of nodes aggregated in one abstraction step. Theoretically the approach demonstrates the best smoothness if in every abstraction step only two activities are aggregated. However, this optimal condition may not hold: process model structure may lead to ag-

gregation of more than two nodes at once. For instance, if activity K has to be abstracted (see Figure 2), activities I, J, L, and M are affected as well, since they are contained in canonical SESE fragment Z, and Z is $sese_{min}$ for K. To evaluate the smoothness of the developed abstraction approach we conduct an experiment and statistically analyze the results. Initially, we select a collection of process models to be abstracted. Afterwards, each model is abstracted to one activity. While models are being abstracted, information about the smoothness is collected.

In the experiment we use a set of 50 real world process models, capturing business processes of a large German health insurance company. The models vary in size from 50 to 204 nodes. The experiment goal is to design strategies representing the "optimistic" and "pessimistic" abstraction scenarios and evaluate their smoothness. For both scenarios we have employed greedy algorithms. In the optimistic scenario the algorithm abstracts a process model in the way that the minimal number of activities is reduced in every abstraction step. In the pessimistic scenario we use the algorithm that abstracts the maximal number of activities per step. Then, the smoothness of model abstraction is found as the mean value of activities reduced at every abstraction step.

Figure 4 presents the distribution of abstraction smoothness obtained in the experiment. Results for the optimistic scenario are shown in Figure 4(a) and for pessimistic—in Figure 4(b). In the optimistic scenario all the models were abstracted with the smoothness between 2.0 and 3.0; more than a half—with the smoothness under 2.5. This means that very often only two activities were aggregated, which is close to the best theoretically possible result. Pessimistic strategy aims to abstract the maximal number of activities in every step. Since models vary in size, in this scenario we use normalized smoothness, dividing abstraction smoothness by the number of nodes in a model. According to the diagram, around 40% of the models were abstracted in huge steps—about half of the model per step, while a few were abstracted even in one step. This statistics proves that the smoothness of the approach relying only on activity aggregation can be quite poor.

Introduction of activity elimination improves smoothness of the abstraction. If the size of a SESE fragment to be aggregated is too large, aggregation of all the activities constituting the fragment leads to high information loss. Not to loose valuable process details, abstraction can be realized through elimination. Instead of replacing the canonical SESE fragment with one aggregating activity, the abstracted activity is eliminated. The choice between aggregation and elimination depends on what operation leads to smaller information loss: elimination of one activity or aggregation of the whole SESE fragment. If the abstraction is performed in semiautomatic manner the user can make this decision. In case of fully automatic abstraction the decision should rely on a criterion. Identification of such criteria is a very interesting problem. However, it is out of scope of this paper and is the subject for the future work.

5 Related Work

The abstraction technique proposed in this paper is based on the PST construction proposed in [VVL07]. In that work the authors showed how control flow graph analysis techniques developed in [JPP94] can be applied for the analysis of business process mod-

els. The prototype of the PST for business process models was discussed in [HFKV06], where fragment with multiple entry and exit nodes were addressed. Finally, in [VVK08] the authors elaborate on the idea of using fragments having single entry single exit nodes. This results in more fine granular tree—refined process structure tree.

Alternatively, process model abstraction can be realized by means of patterns. In this case the abstraction approach defines a set of patterns to be recognized in the process model and the rules how these patterns should be handled. Usually the transformation aims to simplify the model through minimization of the nodes number. In literature there is a number of works which employed transformation rules for analysis of process models. The examples are [SO00] and [MVD+08, DJVVA07]. Abstraction methods can be based on such rules: in [LS03] the authors show how process views can be constructed using techniques from [SO00]. In [PSW08b] four abstraction patterns were introduced: sequential, block, loop and dead end. The patterns describe not only structural transformations, but define how to derive properties of new process elements from the original ones. In [BRB07] the authors propose a comprehensive approach to construction of process views. The approach relies upon the wide set of elementary operations, enabling stepwise construction of a view. The defined operations can be used on model and instance levels. The given operations are very generic and include, for instance, those which are not order preserving. The main limitation of the approaches based on patterns is that they cannot handle arbitrary models. In the real life, however, users tend to capture processes in sophisticated models.

The authors of [EG08] introduce a two step approach for creation of process views, which targets cross-organizational collaboration. In the first step the process owner can hide private or irrelevant details, while in the second step elements which are not in the focus of the process consumer are omitted. The views are constructed basing on the sets of nodes to be eliminated and does not rely on patterns. On the other hand, only block structured process models are handled.

6 Conclusions and Future Work

In this paper we have proposed the new approach to process model abstraction. The approach exploits decomposition of a process model into a hierarchy of SESE fragments called process structure tree. Since SESE fragments have arbitrary inner structure, the approach can successfully abstract graph-structured process models. This is one of the main advantages of the solution. This approach allows us to handle fragments with higher flexibility than the technique based on patterns [PSW08b]. However, there are algorithms enabling more fine-granular decomposition of process models, for instance, SPQR [DBT89, DBT96] and RPST [VVK08], which are based on triconnected graph decomposition technique [TV80]. Therefore, the fundamental value of the approach proposed in this paper is the idea of using process structure tree for process model abstraction. With the example of SESE decomposition we have illustrated and evaluated how PST can be employed for the abstraction task. The direct continuation of this paper is the study of methods of more fine-granular model decomposition.

It should be noticed that PST addresses only the structural aspect of abstraction. This means that it splits a model into elements to be abstracted, but does not tell which elements should be abstracted. Therefore, another direction of future work is the search for criteria to judge about significance of model elements. Here, analysis of semantics and non-functional properties of a process model should be taken into account.

References

[BPM08] BPMI. *Business Process Modeling Notation*, 1.1 edition, February 2008.

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

[DBT89] G. Di Battista and R. Tamassia. Incremental Planarity Testing. In *FOCS*, pages 436–441, 1989.

[DBT96] G. Di Battista and R. Tamassia. On-Line Maintenance of Triconnected Components with SPQR-Trees. *Algorithmica*, 15(4):302–318, 1996.

[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.

[EG08] R. Eshuis and P. Grefen. Constructing Customized Process Views. *Data Knowl. Eng.*, 64(2):419–438, 2008.

[GA07] C. Günther and W. Aalst. Fuzzy Mining–Adaptive Process Simplification Based on Multi-perspectiveMetrics. In Gustavo Alonso, Peter Dadam, and Michael Rosemann, editors, *Business Process Management, 5th International Conference, BPM 2007, Brisbane, Australia, September, 2007, Proceedings*, volume 4714 of *Lecture Notes in Computer Science*, pages 328–343. Springer, 2007.

[HC94] M. Hammer and J. Champy. *Reengineering the Corporation: A Manifesto for Business Revolution*. HarperBusiness, April 1994.

[HFKV06] R. Hauser, M. Friess, J. M. Kuster, and J. Vanhatalo. Combining Analysis of Unstructured Workflows with Transformation to Structured Workflows. pages 129–140, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[JPP94] R. Johnson, D. Pearson, and K. Pingali. The Program Structure Tree: Computing Control Regions in Linear Time. pages 171–185. ACM Press, 1994.

[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.

[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.

[PSW08a] A. Polyvyanyy, S. Smirnov, and M. Weske. Process Model Abstraction: A Slider Approach. In *Proceedings of the 12th International Conference on Enterprise Distributed Object Computing (EDOC)*, 2008.

[PSW08b] A. Polyvyanyy, S. Smirnov, and M. Weske. Reducing Complexity of Large EPCs. In *EPK'08 GI-Workshop*, Saarbruecken, Germany, 11 2008.

[RHAM06] Nick Russell, A.H.M. ter Hofstede, W.M.P. van der Aalst, and Natalya Mulyar. Workflow Control-Flow Patterns: A Revised View. Technical report, BPMcenter.org, 2006.

[SO00] W. Sadiq and M. E. 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.

[TV80] R.E. Tarjan and J. Valdes. Prime Subprogram Parsing of a Program. In *POPL '80: Proceedings of the 7th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 95–105, New York, NY, USA, 1980. ACM.

[VVK08] J. Vanhatalo, H. Völzer, and J. Koehler. The Refined Process Structure Tree. In *Business Process Management, 6th International Conference, BPM 2008 Milan, Italy, September, 2008, Proceedings*, volume 5240 of *Lecture Notes in Computer Science*. Springer, 2008.

[VVL07] J. Vanhatalo, H. Völzer, and F. Leymann. Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition. In Bernd J. Krämer, Kwei-Jay Lin, and Priya Narasimhan, editors, *Service-Oriented Computing - ICSOC 2007, Fifth International Conference, Vienna, Austria, September 17-20, 2007, Proceedings*, volume 4749 of *Lecture Notes in Computer Science*, pages 43–55. Springer, 2007.

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