Structural Abstraction of Process Specifications

Artem Polyvyanyy

Business Process Technology Group
Hasso Plattner Institute at the University of Potsdam
Prof.-Dr.-Helmert-Str. 2–3, D-14482 Potsdam, Germany
`Artem.Polyvyanyy@hpi.uni-potsdam.de`

Abstract. Software engineers constantly deal with problems of designing, analyzing, and improving process specifications, e.g., source code, service compositions, or process models. Process specifications are abstractions of behavior observed or intended to be implemented in reality which result from creative engineering practice. Usually, process specifications are formalized as directed graphs in which edges capture temporal relations between decisions, synchronization points, and work activities. Every process specification is a compromise between two points: On the one hand engineers strive to operate with less modeling constructs which conceal irrelevant details, while on the other hand the details are required to achieve the desired level of customization for envisioned process scenarios. In our research, we approach the problem of varying abstraction levels of process specifications. Formally, developed abstraction mechanisms exploit the structure of a process specification and allow the generalization of low-level details into concepts of a higher abstraction level. The reverse procedure can be addressed as process specialization.

Keywords: Process abstraction, process structure, process connectivity, process modeling

1 Introduction

Process specifications represent exponential amounts of process execution scenarios with linear numbers of modeling constructs, e.g., service compositions and business process models. Nevertheless, real world process specifications cannot be grasped quickly by software engineers due to their size and sophisticated structures, leading to a demand for techniques to deal with this complexity. The research topic of process abstraction emerged from a joint research project with a health insurance company. Operational processes of the company are captured in about 4 000 EPCs. The company faced the problem of information overload in the process specifications when employing the models in use cases other than process execution, e.g., process analysis by management. To reduce the modeling effort, the company requested to develop automated mechanisms to derive abstract, i.e., simplified, process specifications from the existing ones. The research results derived during the project are summarized in [1].

Abstraction is the result of the generalization or elimination of properties in an entity or a phenomenon in order to reduce it to a set of essential characteristics.

Information loss is the fundamental property of abstraction and is its intended outcome. When working with process specifications, engineers operate with abstractions of real world concepts. Process specifications are special types of entities that describe principles of observed or intended behavior. In our research, we develop mechanisms to perform abstractions of formal process specifications. The challenge lies in identifying what the units of process logic suitable for abstraction are, and then performing the abstraction afterwards. Once abstraction artifacts are identified, they can be eliminated or replaced by concepts of higher abstraction levels which conceal, but also represent, abstracted detailed process behavior. Finally, individual abstractions must be controlled in order to achieve an abstraction goal—a process specification that suits the needs of a use case.

Fig. 1 shows an example of two process specifications, given as BPMN process models, which are in the abstraction relation. The example is adapted from [2]. The model at the top of the figure is the abstract version of the model at the bottom. Abstract tasks are highlighted with a grey background; the corresponding concealed fragments are enclosed within the regions with a dashed borderline. The fragments have structures that result in an abstract process which captures the core process behavior of the detailed one. The abstract process has dedicated routing and work activity modeling constructs and conceals detailed behavior descriptions, i.e., each abstracted fragment is composed of several work activities. The research challenge lies in proposing mechanisms which allow examining every process fragment prior to performing abstraction and suggesting mechanisms which coordinate individual abstractions, i.e., assign higher priority to abstracting certain fragments rather than the others.

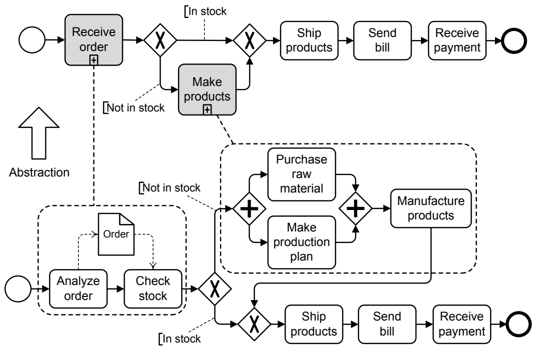

Fig. 1. Process abstraction

The rest of the paper is organized as follows: The next section presents the connectivity-based framework designed to approach the discovery of process fragments suitable for abstraction. Sect. 3 discusses issues relevant to the control of process abstraction. The paper closes with conclusions which summarize our findings and ideas on further research steps.

2 Discovery of Process Fragments

A necessary part of a solution for process abstraction is a mechanism for the discovery of process fragments, i.e., parts of process logic suitable for abstraction. The chances of making a correct decision on which part of a process specification to abstract from can only be maximized if all the potential fragment candidates for conducting an abstraction are considered. To achieve this completeness, we employ the *connectivity* property of process graphs—directed graphs used to capture process specifications.

Connectivity is a property of a graph. A graph is k*-connected* if there exists no set of $k-1$ elements, each a vertex or an edge, whose removal makes the graph disconnected, i.e., there is no path between some pair of elements in a graph. Such a set is called a separating $(k-1)$-set. 1-, 2-, and 3-connected graphs are referred to as connected, biconnected, and triconnected, respectively. Each separating set of a process graph can be addressed as a set of boundary elements of a process fragment, where a boundary element is incident with elements inside and outside the fragment and connects the fragment to the main flow of the process. Let m be a parameter, the discovery of all separating m-sets (graph decomposition) of the process graph leads to the discovery of all process fragments with m boundary elements—potential abstraction candidates.

The *vertex (edge) connectivity* of a graph is the size of the smallest separating set of the graph that is composed only of vertices (edges). For an arbitrary graph it holds that its vertex connectivity is less than or equal to its edge connectivity [3]. Intuitively, removal of an edge, when testing the edge connectivity, can be substituted with removal of an incident vertex, which implies removal of all incident edges. In general, one can speak about (n,e)*-connectivity* of a graph. A graph is (n,e)*-connected* if there exists no set of n nodes and there exists no set of e edges whose removal makes the graph disconnected. Observe, an (n,e)-connected graph is $(n+e+1)$-connected. A lot of research was carried out by the compiler theory community to gain value from the triconnected decomposition of process specifications, i.e., the discovery of triconnected fragments in process graphs. The decompositions which proved useful were $(2,0)$-decomposition, or the tree of the triconnected components, cf., [4], and $(0,2)$-decomposition, cf., [5]. Triconnected process graph fragments form hierarchies of single-entry-single-exit (SESE) fragments and are used for process analysis, process comparison, process comprehension, etc. For these decompositions, linear-time complexity algorithms exist [6,7]. Recently, these techniques were introduced to the business process management community [8,9]. We employed triconnected process graph fragments to propose mechanisms of process abstraction [10,11]; we discover and generalize the triconnected process fragments to tasks of a higher abstraction level.

Fig. 2 shows an example of a process graph. Routing decisions and synchronization points can be distinguished by the degree of the corresponding vertex, i.e., the number of incident edges, and orientation of the incident edges, i.e., incoming or outgoing. Process starts (ends) have no incoming (outgoing) edges. More-

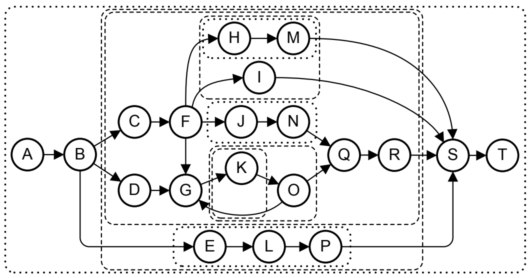

Fig. 2. Process graph, its SESE fragments

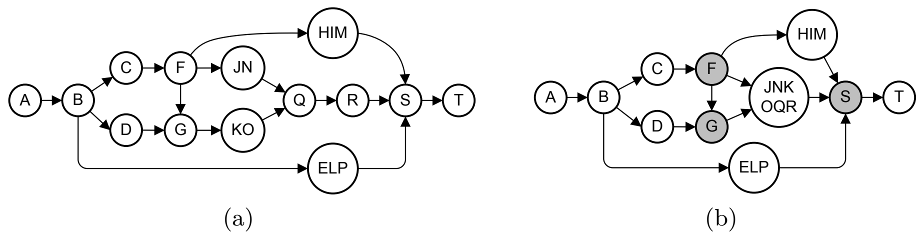

Fig. 3. (a) Abstract process graph, (b) $(3,0)$-connected fragment abstraction

over, Fig. 2 visualizes the triconnected fragments of the graph (SESE fragments). Each triconnected fragment is enclosed in the region and is formed by edges inside or intersecting the region. Fragments enclosed by regions with dotted borderlines can be discovered after performing $(0,2)$-decomposition of the process graph, whereas regions with dashed borderlines define fragments additionally discovered by $(2,0)$-decomposition. Observe that trivial fragments, the fragments that represent sequences composed of a single vertex, are not visualized.

The abstract process graph, obtained from the graph shown in Fig. 2, is given in Fig. 3(a). The abstraction is performed following the principles from [11], i.e., by aggregating the triconnected fragments into concepts of a higher abstraction level. The graph from Fig. 3(a) has a single separating pair B, S. Next abstractions of the triconnected fragments of the graph will result in aggregation of either the unstructured fragment composed of nodes $\{C, ..., R\} \setminus \{E, L, P\}$, or the fragment with entry node B and exit node S. To increase the granularity of process fragments used to perform abstractions, one can start looking for multiple-entries-multiple-exits (MEME) fragments within the triconnected fragments.

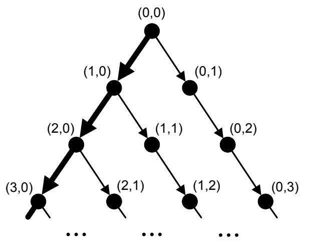

Fig. 4. Connectivity-based process graph decomposition framework

Fig. 4 visualizes a connectivity-based process graph decomposition framework, i.e., the scheme for process fragment discovery. In the figure, each dot represents a connectivity property of the process graph (process fragment) subject to decomposition, e.g., $(0,0)$ means that the graph is connected if no nodes and no edges are removed. Edges in the figure suggest which decompositions can be performed for a graph with a certain connectivity level. For example, one can decompose a $(0,0)$-connected graph by looking for a single node or an edge which renders the graph disconnected. Afterwards, obtained fragments can be treated as subjects for $(2,0)$-, $(1,1)$-, or $(0,2)$-decomposition. By proceeding in this way, highly connected fragments get gradually decomposed.

A careful reader might notice that Fig. 4 does not suggest all the possibilities for accomplishing decomposition. For instance, $(0,2)$-connected fragments can be subjects for $(2,0)$-decomposition, cf., Fig. 2. As the vertex connectivity of a graph is less than or equal to its edge connectivity, we propose to give the preference to the vertex-based decomposition over the edge-based one (refer to the left-most path in Fig. 4). Such a strategy stimulates the fragments to be discovered “faster”, i.e., by performing less decomposition steps, and “finer”, i.e., by discovering more fragments. However, in the cases when it is important to achieve the granularity on the level of edges, we suggest to deviate from the main strategy only once by switching from the vertex-based to the edge-based decomposition strategy.

An (n, e)-decomposition $(n+e \geq 3)$ allows to decompose unstructured process graphs into MEME fragments with $n + e$ entries and exits. A process graph in Fig. 3(b) is obtained by abstracting a $(3, 0)$-connected fragment defined by a separating set $\{F, G, S\}$ (F and G are the entries and S the exit of the fragment, highlighted with grey background). For reasonable $n + e$ combinations, it is possible to perform decomposition in low polynomial-time complexity. For example, the $(3, 0)$-decomposition of a $(2, 0)$-connected graph can be accomplished by removing a vertex from the graph and afterwards running the triconnected decomposition, for which the linear time complexity algorithm exists [7]. Each discovered separation pair together with the removed vertex form a separating triple of the original graph. The procedure should be repeated for each vertex of the graph. Hence, a square-time complexity decomposition procedure is obtained. Following the described rationale, one can accomplish $(k, 0)$-decomposition in $O(n^{k-1})$ time, where n is proportional to the size of the graph.

When performing the $(n + 1, e)$- or $(n, e + 1)$-decomposition of an (n, e)-connected graph, one can remove a vertex or, respectively, an edge from the graph and run the same algorithm that was used when obtaining the (n, e)-connected graph; the procedure should be repeated for each vertex or, respectively, edge. The $(2, 0)$-decomposition results in the hierarchy of $(2, 0)$-connected fragments organized in a tree structure [4]. In the case of the $(3, 0)$-decomposition, one obtains a forest, where each tree is the result of the $(2, 0)$-decomposition of the graph after the removal of a single vertex. The amount of trees in the forest is equal to the number of vertices in the graph. Such a forest can be treated as the structural characterization of the $(2, 0)$-connected graph.

By following the principles of the connectivity-based decomposition framework, we not only discover process fragments used to perform process abstraction, but also learn their structural characteristics. Structural information is useful at other stages of abstraction, e.g., when introducing control over abstraction. Initial work on classifying and checking the correctness of process specifications based on discovered process fragments was accomplished in [8,12].

3 Abstraction Control

The task of adjusting the abstraction level of process specifications requires intensive intellectual work and in most cases can only be accomplished by process analysts manually. However, for certain use cases it is possible to derive automated or to support semi-automated abstraction control mechanisms. The task of abstraction control lies in telling significant process elements from insignificant ones and to abstract the latter. In [1], work activities are classified as insignificant if they are rarely observed during process execution. We were able to establish the abstraction control when investigated processes were annotated with information on the average time required to execute activities. Process fragments which contain insignificant activities get abstracted. Hence, we actually deal with the significance of fragments which represent detailed work specifications.

Significant and insignificant process fragments can be distinguished once a technique for fragment comparison is in place, i.e., a partial order relation is defined for process fragments. The average time required to execute activities in the process provides an opportunity to derive a partial order relation, i.e., fragments which require less time are considered insignificant. Other examples of criteria and the corresponding abstraction use cases are discussed in [13].

Once an abstraction criterion, e.g., the average execution time of activities, is accepted for abstraction, one can identify a minimal and a maximal values of the criterion for a given process specification. In our example, the minimal value corresponds to the most rarely observed activity of the process and the maximal value corresponds to the average execution time of the whole process. By specifying a criterion value from the interval, one identifies insignificant process fragments—those that contain activities for which the criterion value is lower than the specified value. Afterwards, insignificant fragments get abstracted. In [13], we proposed an abstraction slider as a mechanism to control abstraction. An abstraction slider is an object that can be described by a slider interval defined by a minimal and a maximal allowed values for an abstraction criterion, has a state—a value that defines the desired abstraction level of a process specification, and exposes behavior—an operation that changes its state.

4 Conclusion

In our research, we develop methods which allow the derivation of high abstraction level process specifications from detailed ones. In order to discover fragments suitable for abstraction, we employ the structure of process specifications, which are usually formalized as directed graphs. As an outcome, developed techniques can be generalized to any process modeling notation which uses directed graphs as the underlying formalism.

It is a highly intellectual task to bring a process specification to a level of abstraction that fulfills emergent engineering needs, and one without a single perfect solution. By employing the technique for the discovery of abstraction fragments, one can approach the problem as a manual engineering effort. Besides, when it is sufficient to fulfill certain use case, cf., [1], one can define principles for a semi-automated or fully automated composition of individual abstractions.

As future steps aimed at strengthening the achieved results, we plan to validate the applicability of the connectivity-based process graph decomposition framework for the purpose of process abstraction with industry partners and to look for process abstraction use cases for which automated control mechanisms can be proposed. Finally, studies regarding the methodology of abstractions need to complement the technical results.

References

1. Polyvyanyy, A., Smirnov, S., Weske, M.: Reducing complexity of large EPCs. In: Modellierung betrieblicher Informationssysteme. Volume 141 of Lecture Notes in Informatics., Saarbruecken, Germany, GI (November 2008) 195–207
2. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer Verlag (2007)
3. Diestel, R.: Graph Theory. Springer Verlag (2005)
4. Tarjan, R.E., Valdes, J.: Prime subprogram parsing of a program. In: Proceedings of the 7th Annual ACM Symposium on Principles of Programming Languages, New York, NY, USA, ACM Press (1980) 95–105
5. Johnson, R.: Efficient Program Analysis using Dependence Flow Graphs. PhD thesis, Cornell University, Ithaca, NY, USA (1995)
6. Johnson, R., Pearson, D., Pingali, K.: The program structure tree: Computing control regions in linear time. In: Proceedings of the ACM Conference on Programming Language Design and Implementation, ACM Press (1994) 171–185
7. Gutwenger, C., Mutzel, P.: A linear time implementation of SPQR-trees. In: Proceedings of the 8th International Symposium on Graph Drawing, London, UK, Springer Verlag (2001) 77–90
8. Vanhatalo, J., Völzer, H., Leymann, F.: Faster and more focused control-flow analysis for business process models through SESE decomposition. In: Proceedings of the 5th International Conference Service-Oriented Computing. Volume 4749 of Lecture Notes in Computer Science., Springer Verlag (September 2007) 43–55
9. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. In: Proceedings of the 6th International Conference on Business Process Management. Volume 5240 of Lecture Notes in Computer Science., Milan, Italy, Springer Verlag (September 2008) 100–115
10. Polyvyanyy, A., Smirnov, S., Weske, M.: On application of structural decomposition for process model abstraction. In: Proceedings of the 2nd International Conference on Business Process and Service Computing. Volume 147 of Lecture Notes in Informatics., Leipzig, Germany, GI (2009) 110–122
11. Polyvyanyy, A., Smirnov, S., Weske, M.: The triconnected abstraction of process models. In: Proceedings of the 7th International Conference on Business Process Management. Volume 5701 of Lecture Notes in Computer Science., Ulm, Germany, Springer Verlag (September 2009) 229–244
12. Polyvyanyy, A., García-Bañuelos, L., Weske, M.: Unveiling hidden unstructured regions in process models. In: Proceedings of the 17th International Conference on Cooperative Information Systems. Volume 5870 of Lecture Notes in Computer Science., Vilamoura, Algarve-Portugal, Springer Verlag (November 2009) 340–356
13. Polyvyanyy, A., Smirnov, S., Weske, M.: Process model abstraction: A slider approach. In: Proceedings of the 12th International IEEE Conference on Enterprise Distributed Object Computing, Munich, Germany, IEEE Computer Society (September 2008) 325–331