A First Step towards a Compiler for Business Processes

Thomas M. Prinz, Norbert Spieß, and Wolfram Amme

Friedrich Schiller University Jena
07743 Jena, Germany
{Thomas.Prinz,Norbert.Spiess,Wolfram.Amme}@uni-jena.de

Abstract. The verification of business processes is crucial since an erroneous execution causes high costs and damages the reputation of the providing company. The first step towards correct business processes is the verification of structural correctness, i.e., the absence of deadlocks and lack of synchronization.

In this demonstration paper, we present a system which was integrated into the Activiti BPMN 2.0 designer for Eclipse, allowing an immediate user support during the development of business processes. Therefore, an entire business process is transformed into semantically equivalent workflow graphs on which a new structural correctness verification is performed directly. This is done for each modification and the determined failures are visualized directly in the business process. The system can be seen as first step towards a compiler for business processes.

1 Introduction

Business processes are well-established in business management, in the context of service-oriented architectures, and cloud computing. Since business processes are described by graphical specification languages like BPMN 2.0 [1], there is a need for transformations into more technical representations to allow and outperform analyses and verifications, i.e., a compiler. The verification of business processes becomes crucial as business processes are frequently used and could have runtimes over months, whereby an erroneous execution causes high costs and could lasting damage the reputation of the providing company. Therefore, support for the development of correct business processes is essential for all business process development tools.

Structural correctness, which focuses only on the structure of business processes without consideration of data aspects, builds the first step towards correct business processes. Business processes can have two kinds of structural errors: *deadlocks* and *lack of synchronization* [2]. Deadlocks are situations in which the execution within business processes blocks partly or completely, and lack of synchronization are situations in which parts of business processes are executed twice unintentionally because of unsuccessfully joined parallel control flows. The absence of deadlocks and lack of synchronization in business processes is called *soundness* in the literature [3,4]. However, we prefer to call it structural correctness like Sadiq and Orlowska [2], since soundness describes the overall correctness.

In our previous work [5,6], we have introduced new compiler-based techniques to find structural errors within workflow graphs. Workflow graphs are a more formal representation of business processes containing exactly one *start* and one *end* node, *activities*, *forks*, *joins*, *splits*, and *merges*. Since the algorithm works only on simple workflow graphs, i.e., on workflow graphs in which each edge contains an activity, and business processes can have more than one start and end node, the transformation of the business process model into the workflow graph representation is not a trivial one-to-one transformation.

In this demonstration paper, we present our analysis tool *mojo* which was integrated into the Activiti BPMN 2.0 designer (`http://activiti.org`), a tool for creating BPMN 2.0 business processes. The resulting system allows immediate and serious support during the development of business processes by visualizing structural errors directly in the graphical model and providing a failure analysis mode, which highlights a selected error. The rest of the paper is structured as follows: Section 2 introduces structural correctness to the reader, Section 3 gives an overview of the implemented system and transformations, whereas Section 4 evaluates their robustness. Eventually, the paper is concluded in Section 5.

2 Informal Description

A formal representation of business processes are workflow graphs. A *workflow graph* is a directed graph $WFG = (N, E)$ such that N consists of activities, forks, joins, splits, merges, and one start as well as one end node. Each activity, split, fork, and the end node have exactly one incoming edge; whereas each activity, merge, join, and the start node have exactly one outgoing edge. Merges and joins have at least two incoming edges, and splits and forks have at least two outgoing edges. Concluding, each node lies on a path from the start to the end node. A workflow graph is called *simple* if for each edge $e = (n_1, n_2) \in E$ the source n_1 or the target n_2 is an activity.

Figure 1 shows a workflow graph with annotated node types. After the instantiation of a workflow graph, a control flow starts at the start node and follows the flow given by the graph. Each node, except a join, i.e., activities, splits, merges, forks, and the end node, can be executed if a control flow reaches one of its incoming edges. However, joins can only fire if all incoming edges are reached by a control flow. Since data aspects are out of the scope of structural correctness, a

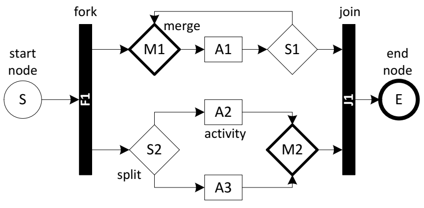


Fig. 1. A workflow graph

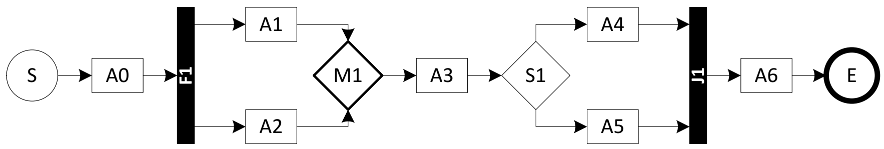


Fig. 2. A workflow graph containing two structural errors

split decides *nondeterministically* which of its outgoing edges will be followed by the control flow. A fork produces a control flow for each of its outgoing edges, i.e., parallelism.

Without loss of generality, each workflow graph is *simple* for the remainder of this paper, since there is a fast transformation from common to simple workflow graphs, e.g., by placing a new activitiy on each edge. It simplifies analyses and algorithms.

Structural correctness describes the absence of deadlocks and lack of synchronization. A *deadlock* appears for a join, if the join was not executed as often as each of its direct predecessor nodes and cannot be executed in future. An executable fork causes a *lack of synchronization* if it may result in simultaneous executions of the same node.

Structural correctness is a standard problem of business processes and has been solved efficiently and with detailed failure information in our previous work [5,6]. The basic idea is to start the analysis for structural correctness at different points (nodes) of a workflow graph, which we call *entrypoints*. It is comparable to a compiler trying to find the next safe program point in order to find further errors after a previous failure. Take the workflow graph of Fig. 2 as example. It contains a deadlock in join $J1$ as well as a lack of synchronization caused by fork $F1$. Starting an analysis in the start node would detect only the lack of synchronization, whereas restarting the analysis at split $S1$ identifies also the deadlock.

We have found out, that *activation points* are good entrypoints for the detection of deadlocks. An activation point *pnt* belongs always to another node n whereas the execution of *pnt* guarantees the future execution of n. Each activation point of a join has to be also an activation point of all of its predecessor nodes, and each closest activation point of a join, i.e., there is at least one path to the join without another activation point, has to be a fork. As a result, a join in a workflow graph without lack of synchronization has a deadlock if on at least one path, from the start node to itself or from itself to itself, lies none of its activation points. In other words, before any control flow ever reaches a join within a workflow graph, being free of lack of synchronization, one of its activation point must be executed.

The identification of lack of synchronization starts in forks, since only forks build more than one control flow causing simultaneous executions of the same node. Generally, two control flows have to be joined before the end node. These joining nodes are called *intersection points*. It is valid, that there are two disjoint paths from the fork to each of its intersection points. If a lack of synchronization occurs at runtime, there is an intersection point of a fork which is not a join, or there is a path from a fork to itself and a path from this fork to the end node, such that both paths are disjoint.

The conditions of deadlocks as well as of lack of synchronization describe supersets of them, since parts of it never occur at runtime, because forgoing deadlocks prevent their execution. Such failures within these supersets are called *potential*. We have proven, that the absence of deadlocks and lack of synchronization corresponds to the absence of potential deadlocks and potential lack of synchronization.

The detection of potential deadlocks can be efficiently implemented via dataflow analyses, being realized in the presented system, whereas the detection of potential lack of synchronization can be done with the help of dominators, postdominators, loop detection, and decomposition. In summary, compiler-based techniques can be successfully applied to the analysis of workflow graphs.

3 System Overview

The Activiti BPMN 2.0 Designer is an Eclipse plugin which allows for the development of business processes with a subset of BPMN 2.0 model elements. Developed business processes are held as graphical models which can be accessed over an extension point for adding business process verifications. We have created own extensions of this extension point which will be executed for every modification of the business process. These extensions are called *mojo* - our open source business process analysis tool (`http://www.bpmn-compiler.org`, `https://sourceforge.net/projects/bpmojo`). Figure 3 visualizes our system. The transformation and structural correctness verification can be used as an extension of Activiti. The following steps are performed by our tool.

At first, the entire business process is transformed into at least one semantically equivalent workflow graph. Therefore, all end events of a BPMN business process are mapped to a single one using the algorithm of Kiepuszewski et al. [7], which was extended for working on workflow graphs instead of Petri nets. The start events of a BPMN business process are merged into a single one using

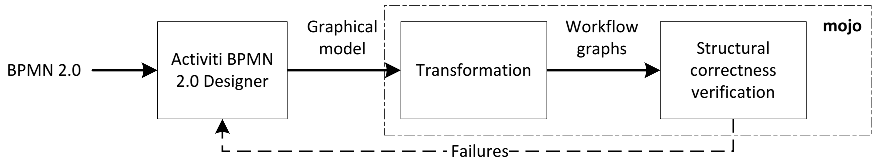


Fig. 3. The verification system

the rules of the BPMN 2.0 specification [1], e.g., tasks without an incoming edge are combined by a fork and start events are combined by a split.

If the business process is not connected, it is translated into different workflow graphs, whereas each workflow graph will be verified for structural correctness in isolation. Gateways with multiple incoming and multiple outgoing edges are disassembled into two gateways, so that the first gateway has all incoming edges and the second all outgoing edges. XOR gateways are transformed to splits and merges, and AND gateways to forks and joins in conclusion. Finally, each task becomes an activity.

In general, the resulting workflow graphs are not simple regarding to its definition in Section 2. Thus, each edge having no activity as source or sink is replaced by two edges, the first connecting the source with a new activity, whereas the second connects the new activity with the sink. The dependencies between the workflow graph and the graphical model of the business process are built up in each step of the transformation algorithm.

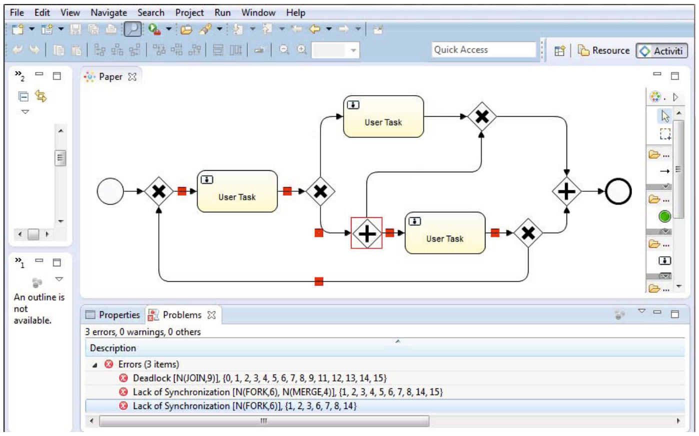
In the second step, our algorithm for structural correctness verification [5,6] is performed directly on each workflow graph. The algorithm finds all potential deadlocks and lack of synchronization, and localizes them precisely. The results of the structural correctness verification are visualized in the graphical model of Activiti: (1) directly in the business process with marks on failure producing nodes, (2) as an error list in the error view of Eclipse, and (3) as a detailed failure highlighting for the failure analysis mode. The failure analysis mode is one of the features of our system (see Fig. 4). It allows for the selection of an error within the error view of Eclipse, which then will be highlighted in the business process in isolation. Therefore, the developer of the business process can find the reasons of these errors.

4 Evaluation

The system was evaluated, using benchmarks of real world business processes, with regard to robustness and performance. On the one hand, the transformation process was tested with the BPMN 2.0 benchmark of IBM `http://www.zurich.ibm.com/csc/bit/downloads.html`, and, on the other hand, the structural correctness verification was validated and evaluated with the business process benchmark of `http://www.service-technology.org/soundness`. Both algorithms run stable and take less than one millisecond in the average case (see our previous work [5] for more details). Therefore, the analysis can be done for every modification of the business process, instead only by saving or on demand.

5 Conclusion

In this demonstration paper, we presented a system which allows direct user support during the development of business processes. It is based on the Activiti BPMN 2.0 Designer and our analysis tool *mojo*, transforming an entire



3 errors, 0 warnings, 0 others

Description

Errors (3 items)
- Deadlock [N(JOIN,9)], {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15}
- Lack of Synchronization [N(FORK,6), N(MERGE,4)], {1, 2, 3, 4, 5, 6, 7, 8, 14, 15}
- Lack of Synchronization [N(FORK,6)], {1, 2, 3, 6, 7, 8, 14}

Fig. 4. Visualizing an error in failure analysis mode

business process into semantically equivalent workflow graphs, on which a structural correctness verification is performed. The determined structural errors are directly visualized in the business process, which supports an immediate correction. Furthermore, the system is fast enough to perform the verification for each modification of the business process.

In the future, the system will be extended by analyses considering data aspects, extensive failure explanations, and an automatically correction of structural errors. Furthermore, a coding into a mobile format with a virtual machine is in the scope of our work.

References

1. OMG: Business process model and notation. Specification (2.0) (March 2011)
2. Sadiq, W., Orlowska, M.E.: Analyzing process models using graph reduction techniques. Inf. Syst. 25(2), 117–134 (2000)
3. van der Aalst, W.M.P., Hirnschall, A., Verbeek, H.M.W.: An alternative way to analyze workflow graphs. In: Pidduck, A.B., Mylopoulos, J., Woo, C.C., Ozsu, M.T. (eds.) CAiSE 2002. LNCS, vol. 2348, pp. 535–552. Springer, Heidelberg (2002)
4. Fahland, D., Favre, C., Koehler, J., Lohmann, N., Völzer, H., Wolf, K.: Analysis on demand: Instantaneous soundness checking of industrial business process models. Data Knowl. Eng. 70(5), 448–466 (2011)
5. Prinz, T.M., Amme, W.: Practical compiler-based user support during the development of business processes. In: Service-Oriented Computing - ICSOC 2013 Workshops. Springer, December 2013 (to be published)
6. Prinz, T.M., Amme, W.: Practical compiler-based user support during the development of business processes. Technical Report Math/Inf/02/13, Friedrich Schiller University Jena, 07743 Jena, Thuringia, Germany (June 2013)
7. Kiepuszewski, B., Hofstede, A.H.M.T., van der Aalst, W.: Fundamentals of control flow in workflows. Acta Informatica 39, 143–209 (2002)