Structural Detection of Deadlocks in Business Process Models

Ahmed Awad and Frank Puhlmann

Business Process Technology Group
Hasso Plattner Institut
University of Potsdam, Germany
(ahmed.awad,frank.puhlmann)@hpi.uni-potsdam.de

Abstract. A common task in business process modelling is the verification of process models regarding syntactical and structural errors. While the former might be checked with low efforts, the latter usually requires a complex state-space analysis to prove properties like deadlock-freedom of the models. In this paper, we address the issue of deadlock detection with a novel approach based on business process querying. Using queries, we are able to detect a broad range of common structural errors that lead to deadlocks, such as misaligned splits and joins. While not being complete, the proposed approach has the advantages of low computational efforts as well as providing graphical outputs that directly lead to the errors.

1 Introduction

With the maturing of business process management (BPM) as an integrated approach ranging from modeling over execution up to evaluation [3], the verification of process models becomes ever more important. This is obvious, since the direct execution of process models requires the detection of errors before the process models are executed. Typical errors can be classified into two categories, either syntactical or structural. A syntactical error is given if modeling elements are used in an invalid manner. The valid and invalid combinations are usually prescribed by the corresponding standard that is used. The Business Process Modeling Notation (BPMN), for instance, does not allow events with more than one outgoing arc [1]. Syntactical errors can usually be found within reasonable time by simply parsing through the process model. Structural errors, such as given by misaligned splits and joins, cannot be detected easily, since the runtime behavior of the process needs to be investigated. To achieve a complete judgement whether a process model fulfills certain structural criteria—such as deadlock freedom—usually the complete state-space has to be analyzed. This analysis, however, is costly in terms of required memory and computing time. In some cases, the result cannot be computed at all [2].

In this paper, we address the structural analysis of business process models regarding deadlocks. A deadlock in a process model is given if a certain instance of the model (but not necessarily all) cannot continue working, while it has

not yet reached its end. The proposed approach is based on graphical queries given in BPMN-Q as introduced in [7]. A BPMN-Q query is represented a as small business process diagram that might contain additional query elements that will be substituted with BPMN elements during its processing. The result of such a graphical query is given by a sub-graph of the original process model. We discuss the idea of detecting deadlocks in process models by formulating—and evaluating—queries that only result in non-empty sub-graphs of the queried process models if a deadlock is contained. For this purpose, we present a set of so called *deadlock patterns* whose occurrence in process models usually leads to deadlocks.

While the approach is not complete—in a sense of finding all possible deadlock sources—each matching query relates to a structural error in the process model. The approach has two major advantages. First, we assume that is computable in polynomial time, meaning that a majority of structural errors are actually detectable. A formal proof, however, is ongoing research. Second, if an error is found, it provides a direct graphical output leading to the error—the resulting sub-graph of the query. In contrast to state-space analysis that requires high efforts and is sometimes not computable, our approach is actually suited to support business process modelers in finding errors in their process models.

The paper is structured as follows. We first extend the motivation and discuss related work in section 2. In section 3 the preliminaries—that is BPMN-Q, the graphical query language, as well as existing deadlock pattern—are introduced. The contribution is presented in section 4, where we discuss BPMN-Q-based deadlock queries. Section 5 gives a larger example. Finally, the paper is concluded with a discussion of future work in section 6.

2 Motivation and Related Work

As already stated, the major motivation of our work is given by lowering the required efforts for detecting deadlocks in process models. Obviously, a full state-space analysis in a transition system underlying a graphical process model (e.g. Petri nets, π-calculus) is sufficient and complete. The drawbacks—in many cases—are the high computational efforts as well as the typically binary result. The backtracking of transition sequences to actual elements of a graphical process model is a complex process in itself. Our approach, in contrast, focuses on the graphical model, allowing the designer to receive direct feedback on erroneous part(s) of the process model. The technique that we apply is based on graphical queries denoted in BPMN-Q. While BPMN-Q in itself is very helpful for process modelers, e.g. in searching existing process models, we discuss how BPMN-Q can be applied to detect deadlocks in process models. Hence, we also provide an additional application area for graphical process queries.

Our approach, however, does not resemble existing soundness properties [2]. We are not able to cover even weaker variants—such as lazy or weak soundness [8, 6]—since we do not have the possibilities to query all quantifications based on structural properties only.

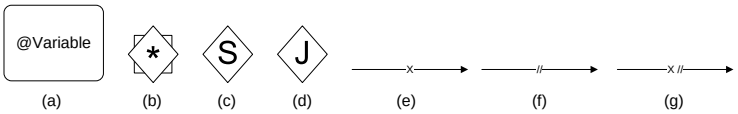

Fig. 1. Extended Elements of BPMN-Q.

Analyzing the structure of a business process to detect deadlocks has already been addressed in literature. In [9], the authors introduced a set of reduction rules for the process graph as means to indicate the correctness. When a process graph can be reduced to an empty graph, the process is said to be conflict free. On the other hand, if the reduction algorithm fails to reduce the process graph to an empty graph, this means the presence of a conflict. One limitation of this approach is that it works on acyclic process graphs only. Another limitation is, that when an identification of a conflict occurs, the reduced graph possibly looks different from the original graph (due to elimination of nodes and edges) which might make the visual identification of the problem difficult. Another approach called causal footprints was introduced recently in [10]. It is able to detect deadlock, trap, and multiple termination patterns by mapping the structure of Event-driven Process Chains [5] into the notion of these causal footprints. Afterwards, reasoning about properties of the resulting causality graphs is possible. Both mentioned approaches, however, have the drawbacks that all detection rules are hard-coded into the supporting tools. This means that any attempt or discovery of further patterns necessitates the modification of source code in these tools.

3 Preliminaries

This section introduces the preliminaries required for detecting deadlocks using BPMN-Q. In particular, BPMN-Q is introduced, existing deadlock patterns are discussed, and limitations and assumptions are given.

3.1 BPMN-Q

BPMN-Q [4] is a visual language that is based on BPMN [1]. It is used to query business process models based on their structure. Beside the set of notations defined in BPMN, BPMN-Q extends them with seven new elements. Some of these elements are flow objects, the others are for connectivity. These elements are shown in figure 1 and are described as follows:

(a) *Variable Node*: it resembles an activity but is distinguished by the @ sign in the beginning of the label. It is used to indicate unknown activities in a query.
(b) *Generic*: this indicates an unknown node in a process. It could evaluate to anything—even null—except for start events.

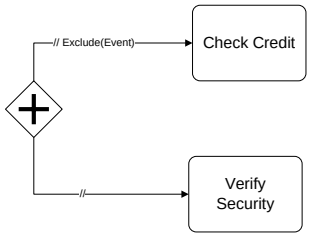

Fig. 2. Sample Query.

(c) *Generic Split*: a generalization of any type of split gateway.
(d) *Generic Join*: a generalization of any type of join gateway.
(e) *Negative Sequence Flow*: states that two nodes A and B are not directly related by sequence flow.
(f) *Path*: states that there must be a path from A to B. A query usually returns all paths.
(g) *Negative Path*: states that there is not any path between two nodes A and B.

An example is shown in figure 2. It shows a simple query that detects if a given process model contains the tasks *Check Credit* and *Verify Security* in a parallel order. Since arbitrary nodes might be in-between the AND-gateway and the required tasks, path query elements are used. A path might have an additional path expression, written in brackets on the path. The only path expression that we use in this paper is the exclusion of certain nodes from a path, denoted as *exclude(NodeType)*. The upper path of the example excludes intermediate events from the path for illustration purposes.

3.2 Deadlock Patterns

Deadlock patterns have already been identified by Onada et al. in [7]. Two concepts were behind these patterns. The first is *reachability*. Reachability between two nodes A and B in a process graph simply means that there is at least one path from A to B. The second is *absolute transferability*. This is a much stronger concept because it states that a token (work item) can always be transferred from node A to *all* input points of node B. What makes absolute transferability reduce reachability between two nodes is the existence of *routing* control nodes in between. From the semantics of AND-join nodes, we know that a deadlock occurs if not all its input points are activated. If we analyze the type of connectivity between an AND-join node and its sources with respect to the absolute transferability and reachability concepts, we can come out with three combinations:

– *Reachability with absolute transferability*. The execution path from a node A to the input points of an AND-join is free of XOR and OR splits. According to the definition of absolute transferability, there is no chance for deadlocks.

– *Reachability without absolute transferability.* Here the execution path from some node A to the input points of an AND-join node includes XOR (OR) splits. Here, there is a possibility for deadlocks.
– *No Reachability.* This means that there is no path from a certain node to the inputs of an AND-join, so no chance for deadlocks to occur.

It is now clear, that whenever there is a reachability without absolute transferability, there is a chance for a deadlock.

The authors of [7] also identified the possibility for a reachability without absolute transferability from the output of an AND-join back to its input. They called this behavior the *loop deadlock type* (see Table 3 row 4 in [7]). Another pattern they identified is the *multiple source deadlock type*. It occurs when an AND-join has input that stems from different sources (see Table 3 row 5 in [7]). According to the above discussion, five deadlock patterns were identified. We represent them next (in a compact form as three of them are variants of each other) along with a discussion when applying them to BPMN models.

– *Loop*: occurs when there is an execution path from the output of an AND-join back to a subset of its input points. If this path contains an XOR-split, deadlock occurs only when the branch leading to the loop is chosen. In case there is a path that does not contain XOR-splits deadlock occurrence is certain.
– *Multiple Source*: occurs when an AND-join has input points which are at some point in the process up-stream originate from two different sources. Assuming that none of the source nodes is the AND-Join itself, we can see that the multiple source pattern can occur (distinctly from other pattern) only when the process structure is one of the following:
  – One of the two sources is an XOR-split. This specification intersects with the third type of patterns shown below.
  – The process has multiple start points that are later on synchronized. In case of models specified in BPMN, multiple starts are permissible. Actually, multiple start points resemble an AND-split between the start events, hence we can deduce that there is reachability between two or more sources (start events) to the AND-join node.
– *Improper structuring*: an AND-join receives input that early started from an XOR-split.

These types of patterns are shown in figure 3, where sequence flow edges labeled with 1 show a trivial representation of the multiple source pattern. Edges labeled with 2 show both patterns of loop and improper structuring, indicating that these patterns are not disjoint.

3.3 Limitations and Assumptions

To highlight the key concepts, we limit our work on process models containing only AND- and XOR- gateways. We leave the—more complex— discussion related to OR-joins for future work. We also assume that the queried process models do not contain implicit splits or merges. This means, that the pattern shown

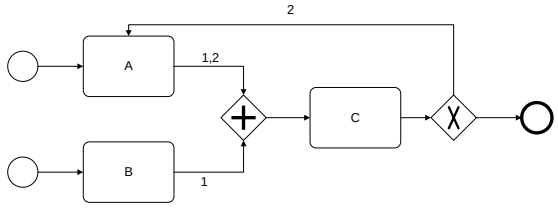

Fig. 3. Faulty Process Model indicating the Types of Deadlock Patterns.

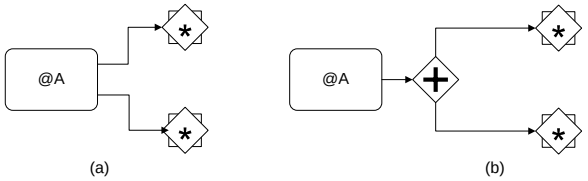

Fig. 4. Implicit versus explicit AND-split.

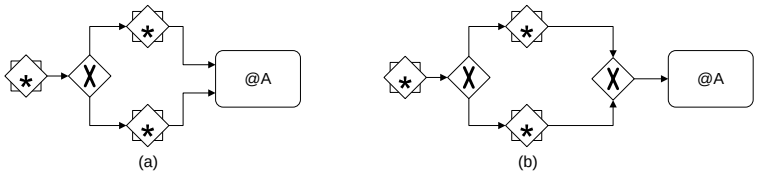

Fig. 5. Implicit versus explicit XOR-join.

in figure 4 (a) must not appear; instead the explicit split shown in figure 4 (b) must be used. The similarity is stated in the BPMN specification document [1, p.111]. Unfortunately, implicit merges may not be mapped directly to explicit representations. The equivalent explicit representation depends on the type of the preceding split. Figure 5 shows the specific case, where an implicit merge is interpreted as an XOR-split (for more details about other scenarios please refer to the specification document). Another issue regarding multiple start events is given when all outgoing flows from these start events are leading to the same activity. In this case, there is an implicit AND-join node in front of the activity and all start events are necessary to start execution. To remove this ambiguity, we enforce an explicit AND-join.

4 Deadlock Queries

In this section we present a set of queries that detect the different deadlock patterns. Some of these queries are direct representation of the patterns, whereas others need more details to ensure correct capturing of the patterns.

4.1 Loop Deadlock Pattern

Figure 6 shows the corresponding query in BPMN-Q that declaratively describes the loop pattern. We have numbered the nodes to ease the explanation. Nodes 1 and 2 are called generic nodes that at query evaluation can match to any type of nodes in the process model. Both nodes 1, 2 represent input points to the AND-join (node 3). The path edge starting from node 3 back to node 1 represents an execution path from the AND-join back to node 1 (loop). To be sure that the resolved paths only reach a subset of the input points to node 3, we have put an extra constraint that is represented in the negative path from node 3 to node 2. This means that for some input point(s) of node 3 we fail to find that loop; i.e. the loop covers only a subset of the input points of the AND-join and hence a deadlock occurs.

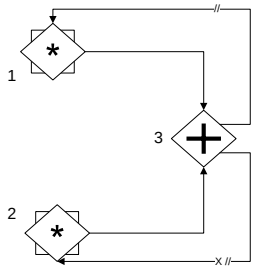

Fig. 6. Loop Deadlock Query.

4.2 Multiple Source Deadlock Pattern

With this pattern we address the case when a process has multiple start points (start event) which are later on synchronized. The query in figure 7 indicates two sources (nodes 1, 2), which are independent of each other. Each of them provides input to a subset of input points (nodes 3,4) to the AND-join (node 5). If we assume that only one of the start events is required to instantiate the process—such as in BPMN—a deadlock occurs at the AND-join.

4.3 Improper Structuring

The mapping of the first two deadlock pattern to BPMN-Q was almost straightforward. When we consider the third case of deadlocks, the mapping is not straightforward. If we consider a direct mapping from the pattern description to a query, we would come out with a query like the one shown in figure 8. If we apply this query to the process model shown in figure 9(a), however, it would result in figure 9(b), showing a match. Although the query found a match in the

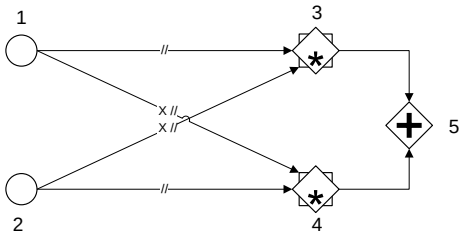

Fig. 7. Multiple Source Query.

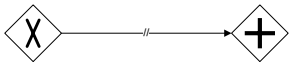

Fig. 8. Direct Description of the Improper Structuring Pattern as a Query.

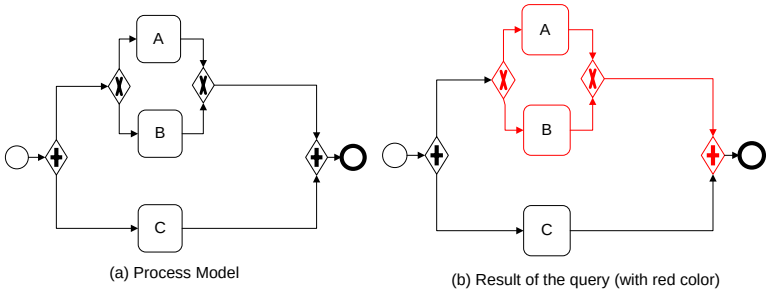

Fig. 9. Example: Deadlock-free Process Model and Query Result.

process model, the model actually is not suffering from deadlocks. The mistake lies in the fact that there is *no* improper structuring here because the AND-join gate is not the match of the XOR-split, since there is an XOR-join in-between.

To solve the issue, we need to modify this query to address patterns where an AND-join node is considered as the match to an XOR-split and whatever lies in-between is properly structured. We look for patterns that start with an XOR-split and end with an AND-join and what lies in-between is properly structured. The different combinations of proper structures in-between an XOR-split and an AND-join are given by:

- Balanced XOR-splits/joins and AND-splits/joins.
- Balanced XOR-splits/joins.
- Balanced AND-splits/joins.
- Sequence of nodes that does not contain gateways.

In the following subsections we will discuss each case along with the query that expresses it.

Balanced XOR-splits/joins and AND-splits/joins. The query shown in figure 10 looks for a pattern where there is some XOR-split (node 1) and an AND-join (node 6) and in-between there are balanced structures of XOR-splits/joins

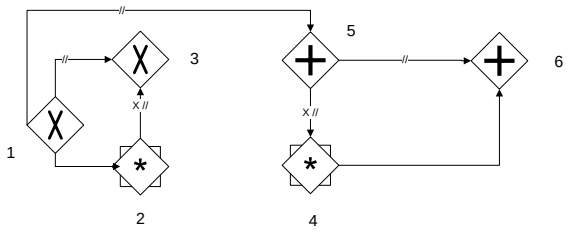

Fig. 10. Improper Structure with in-between Balanced XOR/AND-splits/joins.

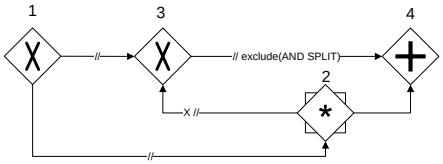

Fig. 11. Improper Structure with in-between Balanced XOR-splits/joins.

and AND-splits/joins. To illustrate that the XOR-split (node 1) is not matched by an XOR-join (node 3), we stated that there is some node in the process graph (node 2) that follows the XOR-split (sequence flow edge between nodes 1, 2) but this node (node 2) is not enclosed in an execution path to the XOR-join (negative path edge between nodes 2, 3). Also we account for the existence of balanced AND-split/AND-join. In the same way we need to prove that there is *no* AND-split (node 5) that is considered as a match to the AND-join (node 6), by showing that there is some input node (node 4) to the AND-join (sequence flow edge between nodes 4, 6) that is not included in an execution path starting at the AND-split (negative path edge between nodes 5, 4).

Balanced XOR-splits/joins. In this case, we focus on only detecting balanced XOR-gateways that lie between a source XOR-split node and a target AND-join node. Figure 11 is the query that detects this type of deadlock. The detection of the unmatched XOR-split is similar to figure 10. The difference is, that an execution path from the XOR-join (node 3) to the AND-join (node 4) excludes any AND-split nodes (path edge between nodes 3, 4).

Balanced AND-splits/joins. Here, only balanced AND-splits/joins are in-between a source XOR-split and a destination AND-join. Figure 12 shows the corresponding query. We follow the same approach as with balanced XOR-split/joins, but this time we need to show that we fail to find and AND-split (node 2) that acts as a match to the AND-join (node 4). We express this no match by finding a node (node 3) which is an input to the AND-join that is not enclosed by the AND-split (negative path edge between nodes 2, 3). We stress that the path from the XOR-split (node 1) to the AND-split (node 2) contains no XOR-joins (path edge between nodes 1, 2).

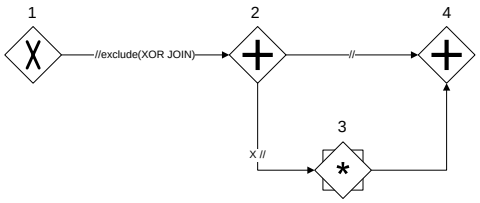

Fig. 12. Improper Structure with in-between Balanced AND-splits/joins.

Sequence of nodes. The fourth case occurs when an XOR-split is matched by an AND-join with only a sequence of activities or intermediate events in-between (without any gateways). To express this as a query, we use the *exclude* property of path operator in BPMN-Q. The visualization of the query is the same as the one in figure 8, but the path exclude condition is *Type(Gateway)*.

Finally, we need to state that the different queries have to be applied in a certain order. First, the query shown in figure 10 needs to be applied. If it does not find a match, either the query shown in figure 11 or 12 are applied. If both do not find a match, finally the query from figure 8 with the additional exclude condition of gateways should be applied. If all these queries fail to find a match, the process model is free of the kinds of deadlocks we discussed.

5 Example

Figure 13 shows a sample process model that should be examined. When we check queries in the order specified in the end of section 4, the query in figure 10 is the one that finds a match in the process as follows: the XOR-Split (node 1) in the query graph is bound to XOR-Split 1 in the process graph. Node 1 did not bind to XOR-Split 2 because BPMN-Q failed to find a node (in process graph) that is a successor of XOR-Split 2 and in mean time does not have a path to XOR-Join. Generic node (node 2) is bound to activity *Obtain Customer Info* as it is a direct successor to the XOR-Split (by the sequence flow edge) and it has no path to an XOR-join node (negative path edge between node 2 and 3 in the query graph). In turn, node 3 in the query is bound to the XOR-Join in the process graph. Generic Node 4 is bound to the XOR-join as this node satisfies the constraint of being a predecessor of an AND-Join node (AND-Join 2) and there is no execution path from an AND-Split to it. Node 5 is bound to the AND-Split node and the AND-Join node (node 6) is bound to AND-Join 2 as we pointed shortly before.

To visualize the query, we reproduced the process model from figure 13 in figure 14. It contains all nodes in the match of the query where the source of the mismatch is XOR-Split 1 and the destination is the AND-Join 2.

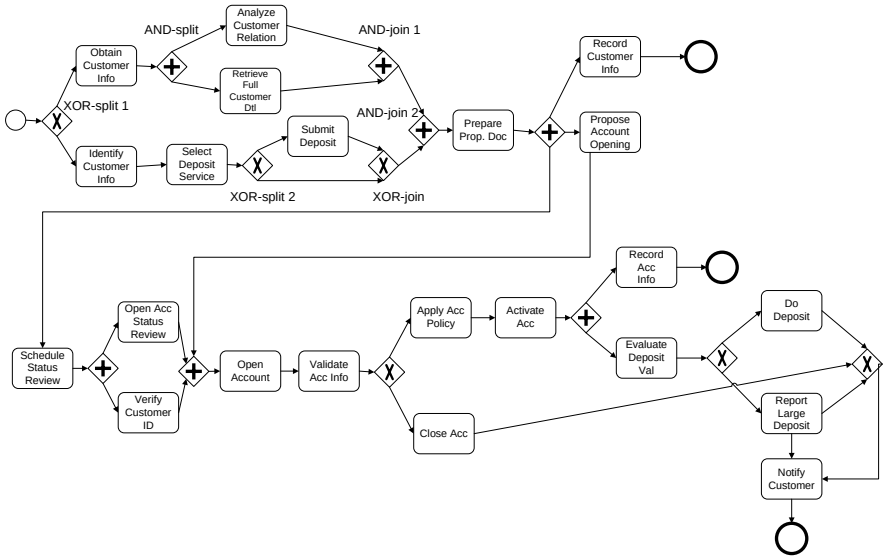

Fig. 13. Process Model suffering from a deadlock

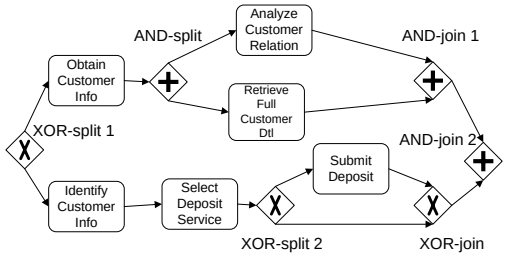

Fig. 14. Query of figure 10 matched to process model in figure 13

6 Conclusion

The paper discussed a novel approach to detect deadlocks in process models via querying them with certain deadlock patterns. In particular, we discussed the translation from deadlock patterns into BPMN-Q queries. This translation required a deeper understanding of the BPMN as well as the BPMN-Q semantics. The results, however, may be applied by process designers without any deeper understanding of BPMN-Q. If a BPMN-Q deadlock query produces a non-empty subset of a process model's graph, this sub-graph directly highlights the problematic areas of the process model. Hence, the process designer is able to directly fix the problems that occur. Regard practical feasibility, our approach has the advantage of requiring low computational effort. The drawback, however, is the

incompleteness of the results. If a query matches, a deadlock is found. If no deadlock query matches a given process model at all, this doesn't guarantee that the model is deadlock free. What has been proven instead, is that certain types of deadlocks do not occur in the process model.

Future work will focus on expanding the deadlock detection patterns. We will add support for additional types of unwanted behavior in process models, like for instance lifelocks. We also plan to investigate the required efforts for a larger set of process models, where we additionally plan to compare the results with traditional soundness investigations [2].

References

1. Business Process Modeling Notation (BPMN) Specification, Final Adopted Specification. Technical report, OMG, 2006.
2. W. Aalst. Verification of Workflow Nets. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997, volume 1248 of LNCS*, pages 407–426, Berlin, 1997. Springer Verlag.
3. W. Aalst, A. ter Hofstede, and M. Weske. Business Process Management: A Survey. In W. Aalst, A. Hofstede, and M. Weske, editors, *Business Process Management, volume 2678 of LNCS*, pages 1–12, Berlin, 2003. Springer Verlag.
4. A. Awad. BPMN-Q a Language to Query Business Processes. In *Proceedings of the 2nd International Workshop on Enterprise Modelling and Information Systems Architectures(EMISA 2007)*, Oct. 2007.
5. G. Keller, M. Nüttgens, and A. Scheer. Semantische Prozessmodellierung auf der Grundlage "Ereignisgesteuerter Prozessketten (EPK)". Technical Report 89, Institut für Wirtschaftsinformatik, Saarbrücken, 1992.
6. A. Martens. Analyzing Web Service based Business Processes. In M. Cerioli, editor, *Fundamental Approaches to Software Engineering (FASE'05), volume 3442 of LNCS*, pages 19–33. Springer Verlag, April 2005.
7. S. Onoda, Y. Ikkai, T. Kobayashi, and N. Komoda. Definition of deadlock patterns for business processes workflow models. In *HICSS '99: Proceedings of the Thirty-second Annual Hawaii International Conference on System Sciences-Volume 5*, pages 50–65, Washington, DC, USA, 1999. IEEE Computer Society.
8. F. Puhlmann and M. Weske. Investigations on Soundness Regarding Lazy Activities. In S. Dustdar, J. Fiadeiro, and A. Sheth, editors, *Business Process Management, volume 4102 of LNCS*, pages 145–160, Berlin, 2006. Springer Verlag.
9. W. Sadiq and M. E. Orlowska. Applying graph reduction techniques for identifying structural conflicts in process models. In *CAiSE '99: Proceedings of the 11th International Conference on Advanced Information Systems Engineering*, pages 195–209, London, UK, 1999. Springer-Verlag.
10. B. F. van Dongen, J. Mendling, and W. M. P. van der Aalst. Structural patterns for soundness of business process models. In *EDOC '06: Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06)*, pages 116–128, Washington, DC, USA, 2006. IEEE Computer Society.