Electronic Communications of the EASST
Volume 25 (2010)

Proceedings of the Workshop
Visual Formalisms for Patterns
at VL/HCC 2009

Visualization of Business Process Modeling Anti Patterns

Ralf Laue and Ahmed Awad

12 pages

Guest Editors: Paolo Bottoni, Esther Guerra, Juan de Lara
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Visualization of Business Process Modeling Anti Patterns

Ralf Laue[1] **and Ahmed Awad**[2]

[1]Chair of Applied Telematics / e-Business
Computer Science Faculty, University of Leipzig, Germany
laue@ebus.informatik.uni-leipzig.de

[2]Business Process Technology Group
Hasso Plattner Institute, University of Potsdam, Germany
ahmed.awad@hpi.uni-potsdam.de

Abstract:

The most common way to model business processes is to use a graphical modeling language. The most widespread notation are business process diagrams modeled in the language BPMN. In this paper, we formalize structural patterns that can lead to control flow errors in such graphical models. For expressing such error patterns, we use the visual query language BPMN-Q . By using a query processor, a business process modeler is able to identify possible errors in business process diagrams. Moreover, the erroneous parts of the business process diagram can be highlighted when an instance of an error pattern is found. This way, the modeler gets an easy-to-understand feedback in the visual modeling language he or she is familiar with.

Keywords: business process model, business process diagram, BPMN-Q, visualization

1 Introduction

Patterns are used in software engineering to describe reusable solutions for common problems. A prominent example are design patterns [GHJV95], reusable solutions in the field of software design. Patterns have also been used to describe commonly occurring *bad* practices. These patterns are also known as *anti-patterns*. In this article, we will formalize structural anti-patterns in business process models (BPM).

BPM are advanced variants of flow charts. Different business process modeling languages share a core set of modeling constructs. Within a BPM, activities can be arranged in sequential order, and routing constructs can be used for modeling alternative and parallel threads.

In the last years, several approaches for detecting errors in the control flow of BPM (for example deadlocks) have been published. Those approaches varied from structural analysis of BPM to the examination of behavioral state space.

There are already several tools that detect problems in BPM. Examples can be found in [Esh02, CK04, Wyn06] – this list is far from being complete. While the problem of detecting errors can be regarded as being solved, many of the tools still fail to give a readable feedback on how to correct the error.

In this paper, we address the presentation of errors a visual manner. We formalize several error patterns as BPMN-Q queries [Awa07]. When a query is structurally matched by a BPM, the matching part of the model is the part containing a problem.

2 Preliminaries

2.1 Business Process Modelling Notation

There are several visual languages for modeling business processes. The Business Process Modeling Notation (BPMN) is the most widespread language, for this reason we use it for the examples in this paper. However, the techniques described in this paper can also be applied for other languages, because most business process modeling languages share certain basic constructs. In this section, we will shortly describe those basic constructs of the BPMN language. For more details about BPMN, the reader is referred to `www.bpmi.org`.

Events (something that happens during the lifetime of a business process) are represented by a circle. Although not formally required by the standard, every BPMN model should have at least a start event (depicting the fact that the process is instantiated) and an end event (depicting the fact that the process has been completed). **Activities** (tasks that have to be performed) are represented by a rectangle with rounded corners. The flow of control (called sequence flow in BPMN terminology) between the activities is depicted by arcs. The direction of such an arc shows in which order the activities have to be performed. **Gateways** can be used for forking and joining paths that have to be performed in parallel or (based on certain conditions) alternatively. There are two kinds of gateways: Splits have more than one outgoing arc, and joins have more than one incoming arc. Gateways are represented by a diamond shape.

When used as a split, an *exclusive gateway* splits the sequence flow to exactly one of its outgoing branches. When used as a join, it awaits one incoming branch being completed before triggering the outgoing flow. This kind of gateways (which we call *XOR-gateways*) is depicted by a ⊗ symbol.

When splitting, a *parallel gateway* activates all outgoing branches; the activities on these branches are executed in parallel. When used as a join, a parallel gateway waits for all incoming branches to complete before triggering the outgoing flow. Parallel gateways are depicted by a ⊕ symbol. We will refer to this kind of gateways as *AND-gateways*.

The inclusive gateway is something in-between the exclusive and the parallel gateway. When used as a split, some of the outgoing branches (but at least one) are activated. When merging, an inclusive gateway waits until all active incoming branches have been completed before triggering the outgoing flow. Inclusive gateways (or *OR-gateways*) are depicted by a ◈ symbol.

Fig. 1 contains all mentioned kinds of gateways. The model shows a simplified business process in a bank. When a customer applies for a real-estate credit, the customer's credit rating, the real estate construction documents and the land register record are checked. All these activities are done in parallel, therefore an AND-gateway is used in the model. As the result of those assessments, the application either will be rejected or the contract is to be prepared. The XOR-gateway means that only one of the activities ”Reject Application” and ”Prepare Contract” can take place. After the contract has been prepared, the process either can end or the bank might offer additional products: a loan protection insurance and a residence insurance. Whether a loan protection insurance, a residence insurance or both are offered, has to be decided case-by-case. The OR-gateway shows that only one of the activities or both of them (in parallel) can take place.

2.2 BPMN-Q: A Visual Language for Querying Business Processes

BPMN-Q [Awa07, ADW08] is a visual language based on BPMN. It is used to query BPM by matching a process model graph to a query graph.

A BPMN-Q query is represented as a business process diagram that might contain the following additional elements (whose graphical representation is shown in shown in Fig. 2):

(a) **Variable Node**: refers to (unknown) activities in a query.
(b) **Generic Node**: refers to an unknown node in a process. It could evaluate to any node type.
(c) **Generic Split / Generic Join**: refers to any type of split / join gateways.
(d) **Negative Sequence Flow**: states that there is no arc from a node A to a node B.
(e) **Path**: states that there must be a path from a node A to a node B.
(f) **Negative Path**: states that there is no path from a node A to a node B.

The result of a graphical query is given by a sub-graph of the original BPM. An exemplary query and its match are shown in Fig. 3. When matching the process graph in Fig. 3(a) to the query in Fig. 3(b), the result of the query is the sub-graph that contains the nodes B and D and all nodes on the path from B to D. (see Fig. 3(c)).

The query shown in Fig. 3 looks for *all* paths between B and D. It is also possible to exclude some graph elements from a path search by assigning names to elements and adding an *exclude* property to a *Path* edge: For instance, in the query in Fig. 5(a), the XOR-split is named *?s*. By adding the *exclude* property to the path search from *?nd1* to *?j*, the search will be limited to paths from *?nd1* to *?j* which do not pass the XOR-split *?s*.

To process a BPMN-Q query, a query graph is matched to the structure of the business process. A BPMN business process diagram can be defined as a directed typed graph as follows:

Definition 1 A business process diagram (or process graph) is a tuple $P_G = (N, A, E, G, F)$ where

- N is a finite set of nodes that is partitioned into the set of activities A, the set of events E, and the set of gateways G. An event $e \in E$ is called a start event if it does not have incoming edges. It is called an end event if it has no outgoing edges. All nodes in a process graph are attributed by unique IDs.
- $F \subseteq N \times N$ is the sequence flow relation between nodes.

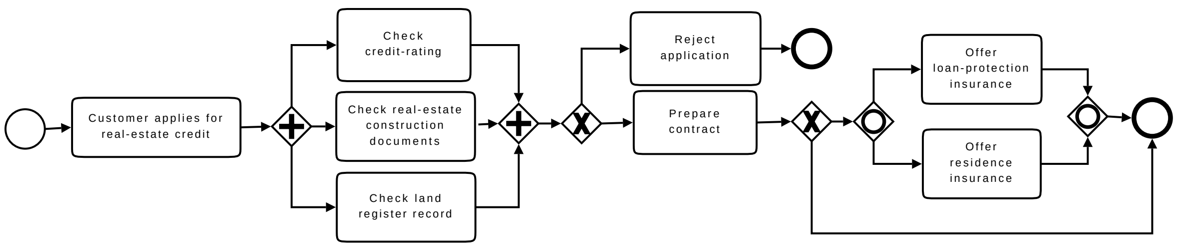

Customer applies for real-estate credit
Check credit-rating
Check real-estate construction documents
Check land register record
Reject application
Prepare contract
Offer loan-protection insurance
Offer residence insurance

Figure 1: BPMN Example Model

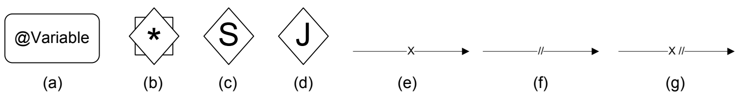


Figure 2: BPMN-Q Elements.

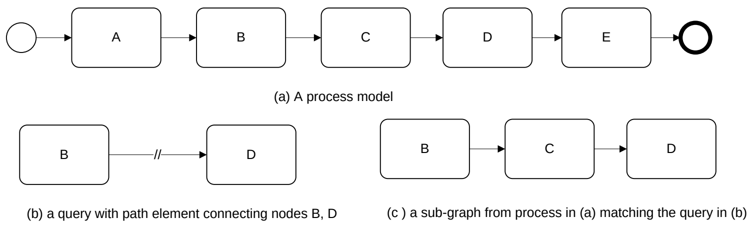

A
B
C
D
E
(a) A process model
B
//
D
(b) a query with path element connecting nodes B, D
B
C
D
(c) a sub-graph from process in (a) matching the query in (b)

Figure 3: Example of a BPMN-Q query

The query language BPMN-Q provides additional types of edges between nodes:

Definition 2 A query graph is a tuple Q_G= (N_Q, A_Q, E_Q,G_Q, S_Q, P_Q,X_Q) where

- N_Q is a finite set of nodes that is partitioned into the set of activities A_Q, the set of events E_Q, and the set of gateways G_Q.
- $S_Q \subseteq N_Q \times N_Q$ is the set of sequence flow edges.
- $P_Q \subseteq N_Q \times N_Q$ is the set of path edges.
- $X_Q \subseteq N_Q \times N_Q$ is the set of negative path edges.

Nodes of the query graph can be identified by assigning labels. Labels can be activity names or special names either starting with '@' for variable activities or with '?' for other nodes and path edges. Thus, a function $l : N_Q \cup P_Q \rightarrow \Sigma^*$ is a labeling function assigning labels (identifiers) to nodes and path edges in the query, where Σ is an alphabet.

The labels of nodes can be used within the query in the *exclude* property of path edges. Thus, $exc : P_Q \rightarrow 2^{\{l(n):n \in N_Q \cup P_Q \ where\ l(n)\ is\ defined\}}$ is a function to evaluate the exclude property of a given path edge.

With the start of query processing, the query processor tries to *bind* the nodes in the query graph to the nodes in the process graph. For each node in the query graph, the set of nodes in the process graph having the same type are identified. A bind is considered *matching* if it satisfies all sequence flow edges, path edges, and negative path edges in the query graph. Otherwise, the binding is dropped. For the query to find a match, each node in the query graph must have at least one *matching binding*. More details about the processing of queries can be found in [Awa07].

3 BPMN Soundness Patterns

3.1 Soundness

The most important correctness criterion for BPM is the soundness property, originally introduced by van der Aalst for workflow nets [van97].

For a business process model to be sound, three properties are required:

1. In every state that is reachable from a start state, there must be the possibility to reach a final state *(option to complete)*.
2. If a state has no subsequent state (according to the transition relation that defines the precise semantics), then only events without outgoing arcs (end events) must be marked as being ”active” in this state *(proper completion)*.
3. There is no element of the model that is never processed in any execution of the model *(no needless elements)*.

Violations of the soundness criterion usually indicate an error in the model.

3.2 Pattern Catalogs

To our knowledge, the first categorization of error patterns based on the structure of a BPM has been compiled at the University of Osaka. In [OIKK99], five so called deadlock-patterns are discussed. The basic concepts used in [OIKK99] are reachability (in a graph) and transferability (the fact that the control flow will always reach some node in a model if another node has been reached before). The authors of [OIKK99] claim that a model always has a deadlock if one of the patterns can be detected. Unfortunately, this claim is wrong: Fig. 4 shows a sound model for which [OIKK99] would report a deadlock between split node x AND-join node a. The reason behind the wrong error report is that by using the concept of transferability, it is not possible to realize that both incoming control flows at join a_1 will always synchronize.

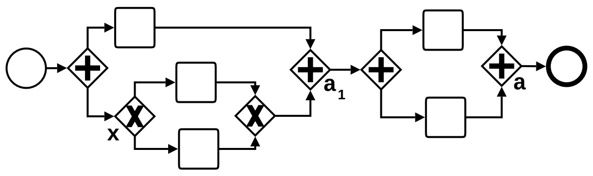


Figure 4: [OIKK99] would wrongly report a deadlock for this model

In [LK05], Liu und Kumar analyzed how unstructured BPM can be mapped into structured ones with the same behavior. For this purpose, they categorized entries into and exits from a control structure between a split and a corresponding join. In particular, they named the combinations that will lead to control-flow errors.

Koehler and Vanhatalo discuss ”typical modeling errors extracted from hundreds of actual process models created in different tools” [KV07]. The anti-patterns discussed in [KV07] are well-known cases for an incongruity between the type of a split and the type of a join.

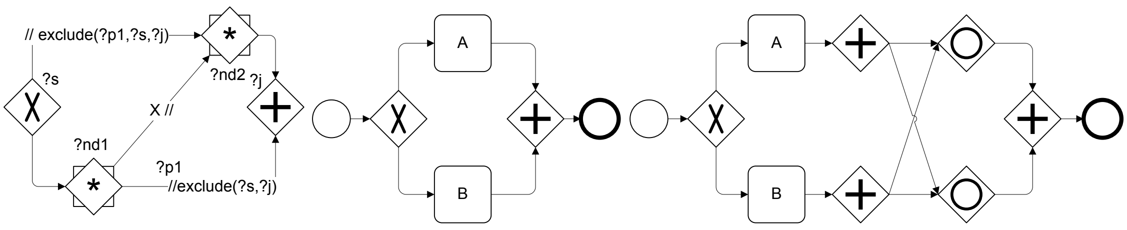
Mendling [Men07] uses reduction rules for finding errors in Event Driven Process Chains. These reduction rules include information about possible error cases in a model.

All mentioned pattern systems have in common that they include the typical errors that result from a mismatch between the type of a split and the type of a join and from choosing a non-XOR gateway as a loop entry or loop exit. In the next chapter, we will use BPMN-Q for expressing these patterns.

4 Anti Patterns Expressed in BPMN-Q

In this section, we express the patterns mentioned in the previous section as BPMN-Q queries. When a query finds a match, the result is the matching sub-graph of the process. This way, the localization of the erroneous part of the BPM is given for free without a need to translate between verification tools and the visual representation of the model.

4.1 (X)OR-split/AND-join Combination

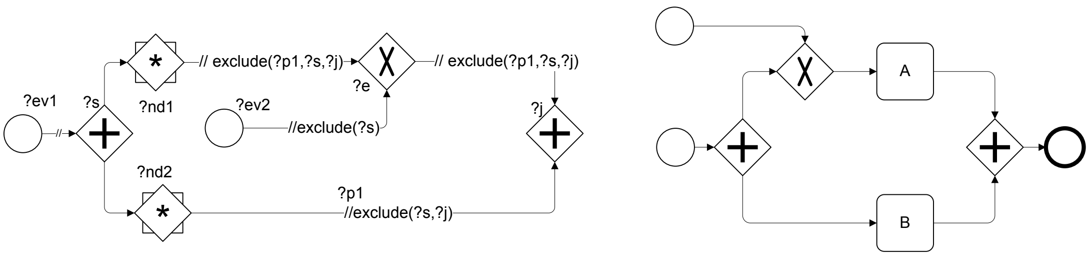


(a) The Query (b) A Process Matching the Query (c) A Process not Matching the Query

Figure 5: Query for an (X)OR-split/AND-join Combination

A deadlock can occur when an (X)OR-split opens alternative paths that are later joined by an AND-join. The query in Fig. 5(a) captures the this situation between an XOR-split $?s$ and the AND-join $?j$. In order to find two paths from $?s$ to $?j$ whose only common nodes are $?s$ and $?j$, we try to find a path from $?nd1$ (the successor of $?s$) to $?j$. This path is given the name $?p1$. Similarly, we try to find another path from s to $?nd2$. The *exclude* property of the latter path is set to $?s, ?j, ?p1$. Exclusion of $?p1$ instructs the BPMN-Q query processor to evaluate path $?p1$ first and then to search for other paths which do not share any node with $p1$. The exclusion of $?s, ?j$ on both paths is necessary to prevent false alarms that can result from loops. Finally, the negative path from $?nd1$ to $?nd2$ prevents false alarms in cases like the one shown in Fig. 9(c).

4.2 Entry Into a Parallel Control Block

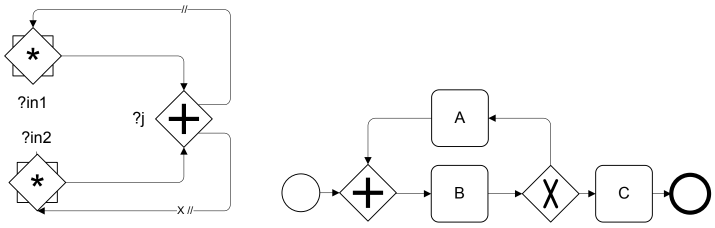


(a) The Query (b) A Process Matching the Query

Figure 6: Query for an entry into a parallel control block

Usually, in a block that starts and ends with an AND-gateway, there is no chance for a deadlock, because all incoming branches of the AND-join have been activated before. However, if on a path from the split to the join there is an (X)OR-join that can receive activations not originating from the AND-split, a deadlock can occur. We call this (X)OR-join an entry into the AND block. This situation is captured declaratively in Fig. 6. The use of two different start events, $?ev1$,$?ev2$, forces the query processor to find matches in BPMs having more than one start event.

4.3 AND-Join as an Entry Into a Loop

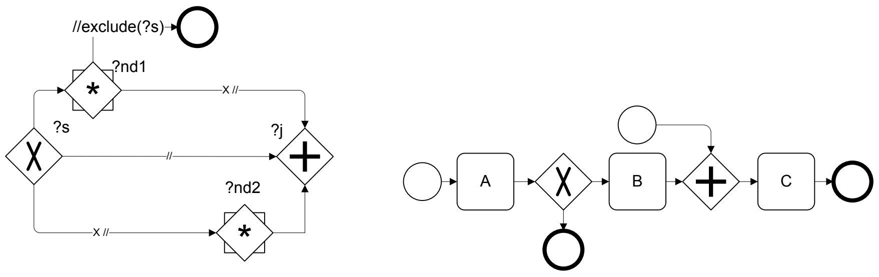


(a) The Query (b) A Process Matching the Query

Figure 7: Query for an AND-Join as an Entry Into a Loop

The query in Fig. 7 describes another situation where a BPM could suffer from a deadlock. Whenever an AND-join is part of a loop where only a subset of its input points are activated, a deadlock occurs. This is declaratively represented by a path edge from $?j$ to $?in1$ and a negative path edge from $?j$ to $?in2$.

4.4 AND-Join After (X)OR-Split Does Not Synchronize



(a) The Query (b) A Process Matching the Query

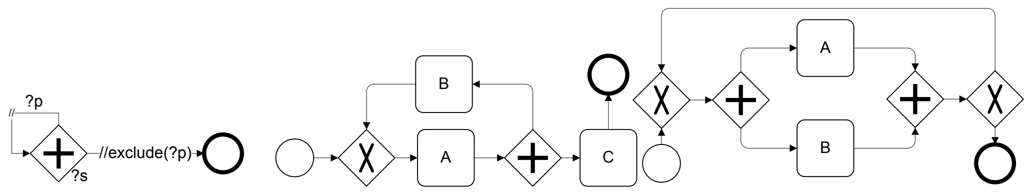
Figure 8: Query for an AND-join after (X)OR-split

A deadlock can occur if an AND-join awaits to be activated on all its incoming arcs, but an (X)OR-split before this AND-join can lead the flow of control away in another direction. The query in Fig. 8 describes this situation.

4.5 AND-split/XOR-join Combination

Lack of synchronization is a modeling error that occurs whenever an AND-split is combined with an XOR-join. All outgoing branches from the AND-split will be activated. However, the semantics of the XOR-join is to wait for the completion of *exactly one* of its incoming branches. Due to space limitations, we do not provide a separate query for this type of errors. Rather, we can reuse the query in Fig. 5(a) with modifications: We switch the roles of the XOR and the AND in that query. Also, we drop the negative path between $?nd1$ and $?nd2$. The resulting query captures the "lack of synchronization errors".

4.6 Infinite Loop



(a) The Query (b) A Process Matching the Query (c) A Process not Matching the Query

Figure 9: Query for an Infinite Loop

Another modeling error occurs when a part of the process is activated infinitely often. This can happen when a modeler mistakenly represent a loop exit with an AND-split rather than an XOR. The query in Fig. 9(a) describes this situation: Path edge $?p$ from the AND Split $?s$ back to itself represents the looping case. To prevent reporting false alarms, e.g., the case of a parallel block nested in a loop, the query requires to have a totally distinct path from $?s$ to an end event.

5 Using the Patterns

As common BPM languages share the most basic modeling elements, the application of our patterns is not restricted to the language BPMN. Previously, the patterns have been successfully used for the language Event-Driven Process Chains [van99]. For this purpose, they have been implemented into the modeling tool *bflow*[1], using the openArchitectureWare Check language[2] [KKGL08]. Another implementation, using a larger repository of 23 patterns, has been made using logical reasoning with Prolog. In [GL09], it has been shown that a pattern-based heuristic reasoning detects control-flow errors almost as accurate as model checkers which explore the whole state space of a model. However, other than approaches using model checkers, it does not suffer from the state-space explosion problem.

To test our approach based on graphical BPMN-Q queries, we searched for the patterns discussed in the previous section in 109 models taken from the public repository of the modeling

[1] http://www.bflow.org
[2] http://www.eclipse.org/gmt/oaw/

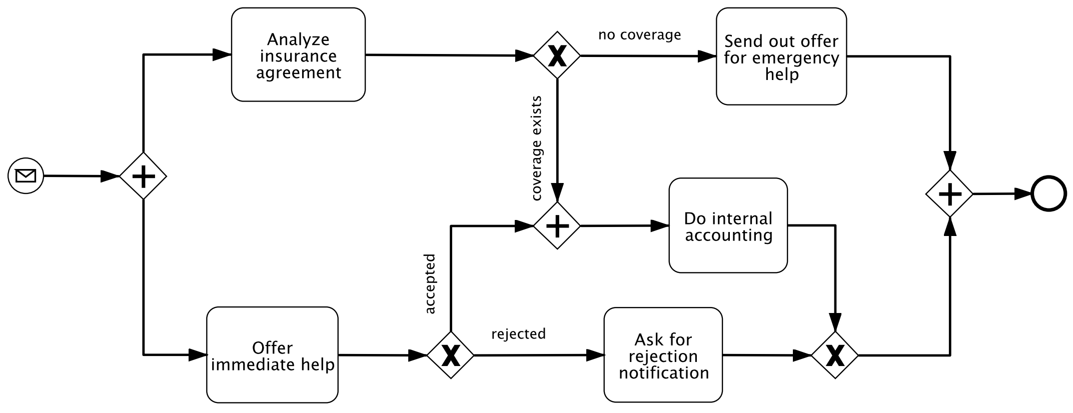

Analyze insurance agreement
no coverage
Send out offer for emergency help
coverage exists
Do internal accounting
accepted
rejected
Offer immediate help
Ask for rejection notification

Figure 10: A process model with a false alarm

tool *Oryx* (`www.oryx-editor.org`). Table 1 shows how many models were detected that contained instances of the patterns. It also shows the false alarms, i.e. the cases for which the model does not suffer from the discussed problem despite matching the query. The query processor ran on a PC with 2GB RAM and an Intel dual core processor at 1.83 GHz under the operating sytem Windows XP with Service Pack3 .

An observation that is worth mentioning is that all false alarms occurred because of other errors that are located elsewhere in a model. An example is shown in Fig. 10. It matches the BPMN-Q query *AND-split/XOR-join Combination*, because there are two distinct paths from the AND-split to the XOR-join. However, at execution, this error would never occur because the process would either terminate without problems or deadlock before reaching the XOR-join. Of course, this possible deadlock will be detected by the pattern *AND-Join After (X)OR-Split Does Not Synchronize*. In situations like this, our pattern-based approach delivers too many warnings (which is better than missing a legitimate warning).

Anti-pattern	Erroneous Models	False Alarms	Processing Time
(X)OR-split/AND-join Combination	4	0	208.735 sec
Entry Into a Parallel Control Block	0	0	237.515 sec
AND-Join as an Entry Into a Loop	1	0	219.453 sec
AND-Join After (X)OR-Split does not Synchronize	7	0	221.250 sec
AND-split/XOR-join Combination	2	5	212.391 sec
Infinite Loop	0	0	229.578 sec

Table 1: Results of applying anti-pattern queries

6 Related Work

While soundness is necessary for the correctness of a BPM, it does not yet guarantee that the model really conforms to the business rules. For this reason, several researchers applied model-checking in order to verify statements like ”Each delivery must always be preceded by a payment”. These approaches make it necessary to specify the properties to validate as temporal formulas. Usually, this is too difficult for the business people who work with the models.

As an alternative, several authors developed notations based on a process modeling language to express the allowed executions of a BPM. Such approaches have been presented for Event-Driven Process Chains [Rum99, SM06, FF08], BPML [Bra05], UML Activity Diagrams [FESS07] and for BPEL specifications [WKH08, LMX07]. Quartel et al. [QDS05] use the Interaction Systems Design Language for expressing dependencies among (distributed) business processes. Van der Aalst and Pesic [vP06] suggested a new language DecSerFlow for specifying the properties of a single service or service compositions. As a different stream of research, Barros et al. [BDG07] and Rommelspacher [Rom08] suggested graphical languages for expressing complex events in business processes.

The main difference between those graphical languages and BPMN-Q is that BPMN-Q is used to formulate queries about the business process model itself (i.e. its graphical structure), not about the state space of its executions. This makes it possible to use BPMN-Q for searching for modeling problems without having to compute the state space of all possible executions.

Another query language that is working on the graphical structure of a model is BPMN VQL [FT08]. Its main purpose is to find crosscutting concerns in BPM. Our language BPMN-Q is more expressive than BPMN VQL. For instance, generic nodes, negative paths, and variable names in BPMN-Q have no equivalent constructs in BPMN VQL.

7 Conclusion and Directions for Future Research

The approach presented in this paper can be used for detecting control flow errors in business process models. By using BPMN-Q queries, it is possible to give a business process modeler a feedback not only about the presence of errors but also about the part of the model that causes the error. This information is given in the visual formalism the modeler is familiar with.

So far, more advanced BPMN constructs like exception handling have not yet been considered in the graphical language. Inclusion of such constructs would enable us to express even more complex patterns like the ones published in [GL07] and [RPH08].

Another direction of future research can be to reduce false warnings in cases like the one shown in Fig. 10 where a model contains more than one instance of one of our error patterns.

Bibliography

[ADW08] A. Awad, G. Decker, M. Weske. Efficient Compliance Checking Using BPMN-Q and Temporal Logic. In *BPM ’08: Proceedings of the 6th International Conference on Business Process Management*. Pp. 326–341. 2008.

[Awa07] A. Awad. BPMN-Q: A Language to Query Business Processes. In Reichert et al. (eds.), *Proceedings of the 2nd International Workshop on Enterprise Modelling and Information Systems Architectures (EMISA'07)*. LNI P-119, pp. 115–128. GI, 2007.

[BDG07] A. P. Barros, G. Decker, A. Großkopf. Complex Events in Business Processes. In Abramowicz (ed.), *Proceedings of the 10th International Conference on Business Information Systems*. LNCS 4439, pp. 29–40. Springer, 2007.

[Bra05] M. Brambilla. LTL formalization of BPML semantics and visual notation for linear temporal logic. Technical report, Politecnico di Milano, 2005.

[CK04] N. Cuntz, E. Kindler. On the semantics of EPCs: Efficient calculation and simulation. In *EPK 2004: Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten, Proceedings*. Pp. 7–26. 2004.

[Esh02] R. Eshuis. *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*. PhD thesis, University of Twente, Enschede, 2002.

[FESS07] A. Forster, G. Engels, T. Schattkowsky, R. V. D. Straeten. Verification of Business Process Quality Constraints Based on Visual Process Patterns. In *Symposium on Theoretical Aspects of Software Engineering*. Pp. 197–208. 2007.

[FF08] S. Feja, D. Fötsch. Model Checking with Graphical Validation Rules. *Engineering of Computer-Based Systems, IEEE International Conference on the* 0:117–125, 2008.

[FT08] C. D. Francescomarino, P. Tonella. Crosscutting Concern Documentation by Visual Query of Business Processes. In *Proceedings of the International Workshop on Business Process Design*. 2008.

[GHJV95] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[GL07] V. Gruhn, R. Laue. Good and Bad Excuses for Unstructured Business Process Models. In *Proceedings of 12th European Conference on Pattern Languages of Programs (EuroPLoP 2007)*. 2007.

[GL09] V. Gruhn, R. Laue. A Heuristic Method for Business Process Model Evaluation. In *5th International Workshop on Enterprise and Organizational Modeling and Simulation (EOMAS 2009)*. 2009.

[KKGL08] S. Kühne, H. Kern, V. Gruhn, R. Laue. Business Process Modelling with Continuous Validation. In Pautasso and Koehler (eds.), *MDE4BPM 2008 1st International Workshop on Model-Driven Engineering for Business Process Management*. 2008.

[KV07] J. Koehler, J. Vanhatalo. Process anti-patterns: How to avoid the common traps of business process modeling, Part 1 - Modelling control flow. *IBM WebSphere Developer Technical Journal* 10.4., April 2007.

[LK05] R. Liu, A. Kumar. An Analysis and Taxonomy of Unstructured Workflows. In Aalst et al. (eds.), *Business Process Management*. Volume 3649, pp. 268–284. 2005.

[LMX07] Y. Liu, S. Müller, K. Xu. A static compliance-checking framework for business process models. *IBM Systems Journal* 46(2):335–361, 2007.

[Men07] J. Mendling. *Detection and Prediction of Errors in EPC Business Process Models*. PhD thesis, Vienna University of Economics and Business Administration, 2007.

[OIKK99] S. Onoda, Y. Ikkai, T. Kobayashi, N. Komoda. Definition of Deadlock Patterns for Business Processes Workflow Models. In *Proceedings of the 32nd Annual Hawaii International Conference on System Sciences*. P. 5065. IEEE Computer Society, 1999.

[QDS05] D. Quartel, R. Dijkman, M. van Sinderen. An approach to relate business and application services using ISDL. In *EDOC '05: Proceedings of the Ninth IEEE International EDOC Enterprise Computing Conference*. Pp. 157–168. 2005.

[Rom08] J. Rommelspacher. Modelling Complex Events with Event-Driven Process Chains. In Hesse and Oberweis (eds.), *SIGSAND-EUROPE*. LNI 129, pp. 79–82. GI, 2008.

[RPH08] T. Rozman, G. Polancic, R. V. Horvat. Analysis of Most Common Process Modeling Mistakes in BPMN Process Models. In *2008 BPM and Workflow Handbook*. 2008.

[Rum99] F. J. Rump. *Geschäftsprozeßmanagement auf der Basis ereignisgesteuerter Prozeßketten*. B. G. Teubner Verlag Stuttgart Leipzig, 1999.

[SM06] C. Simon, J. Mendling. Verification of Forbidden Behavior in EPCs. In Mayr and Breu (eds.), *Modellierung*. LNI 82, pp. 233–242. GI, 2006.

[van97] W. M. van der Aalst. Verification of Workflow Nets. In Azéma and Balbo (eds.), *Application and Theory of Petri Nets 1997, 18th International Conference, ICATPN '97, Toulouse, France, June 23-27, 1997, Proceedings*. Pp. 407–426. 1997.

[van99] W. M. van der Aalst. Formalization and verification of event-driven process chains. *Information & Software Technology* 41(10):639–650, 1999.

[vP06] W. M. van der Aalst, M. Pesic. Specifying, discovering, and monitoring service flows: Making web services process-aware. Technical report BPM-06-09, BPM Center Report, BPMcenter.org, 2006.

[WKH08] R. Wörzberger, T. Kurpick, T. Heer. Checking Correctness and Compliance of Integrated Process Models. In *Proceedings of the 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2008)*. 2008.

[Wyn06] M. T. Wynn. *Semantics, Verification, and Implementation of Workflows with Cancellation Regions and OR-joins*. PhD thesis, Queensland University of Technology Brisbane, Australia, 2006.