

# Visualization of Compliance Violation Using Anti-patterns

Ahmed Awad and Mathias Weske  
{ahmed.awad,mathias.weske}@hpi.uni-potsdam.de

Business Process Technology Group  
Hasso-Plattner-Institute, University of Potsdam, Germany

**Abstract.** Checking for compliance is of major importance in nowadays business. Several approaches have been proposed to address different aspects of compliance checking. One of the important aspects of compliance checking is to ensure that business activities will be executed in a certain order. In a previous work, we have presented a formal approach for efficient compliance checking based on model checking technology. A limitation of that approach and of similar approaches is the lack of explanation about how violations could occur. In this paper we resolve this limitation by exploiting the notion of patterns/anti patterns. Execution ordering compliance rules are expressed as BPMN-Q queries. For each query a set of anti pattern queries is automatically derived and checked against process models as well. When a violation (an anti pattern) finds a match, the violating part of the process is shown to the user so that corrective actions can be taken.

## 1 Introduction

Enterprises are using business process models to run their services smoothly. These artifacts are defining how the enterprise works and they are a good means of controlling if the business is committed to follow them. To be in line with their business goals, but also with legal regulations, companies need to make sure that their operations satisfy a set of policies and rules. i.e. they need to design compliance rules and implement compliance checking mechanisms.

Aspects of compliance are divergent. They also are changing by time. Some of them have the force of law e.g. the Sarbanes-Oxley Act of 2002 [1]. Some other rules come as quality standards like ISO 9000. Keeping process models compliant is an expensive operation [18]. It costs both time and money. Automated approaches have emerged to address compliance issue from different points of view. On one hand, some approaches favor deriving process models by compliance rules [19, 15, 14]. On the other hand delaying the checking of the compliance rules to a post design step are discussed in other approaches [9, 11, 20, 3]. With respect to compliance rules regarding execution ordering of activities, deriving the business process model by compliance rules guarantees a compliant by design business process. However, there is still a need to recheck for compliance each time rules change or new rules are added. A limitation of the second approach

is its binary nature of answer i.e. it reports either compliant or non compliant. Both approaches are missing a mechanism that helps modelers focus on parts of models that violate the rules.

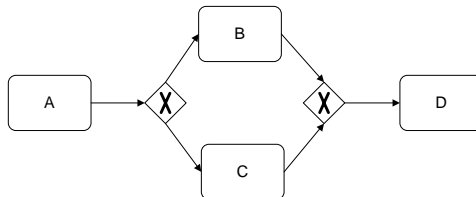
Explaining violation of compliance rules is necessary to help modelers take corrective actions i.e. to correct the models in a way that they are compliant with rules. Formal approaches such as model checking have the capability of providing counter examples when the rule to be checked is not satisfied by the process model [3, 20, 9]. Unfortunately, these counter examples are given in terms of internal state transitions rather than in terms of process models that are too technical to be understood by a non-technical user. To benefit from these counter examples, the output of the model checker must be translated to the notation the user can understand. These translations are usually dependent on both the model checker software and the visual notation the user understands. Also, these translations form a cost in the tool chain added to the cost of first mapping the system to be checked into the input language of a model checker.

In this paper, we build upon our work in [3] by showing how BPMN-Q [2] queries can be used to show violation to compliance rules regarding ordering of activities execution. Our contribution comes in Section 2 where we use BPMN-Q queries to express rules about ordering of activities. Section 3 discusses a case where rules about ordering of execution of activities needs to be validated. Related work is discussed in Section 4. Paper is concluded in Section 5 with a discussion.

## 2 Detection and Visualization of Violation

In [3] we used BPMN-Q queries to express compliance rules regarding execution ordering of activities. A Query was used in a twofold way. As a query, it was used to find the set of process models that are subject to compliance checking in a repository of process models. Therefore, saving the effort of manually identifying such models. Later on, a temporal logic formula was derived from the query. A reduced version of the checked process model was translated into a finite state machine. Finally, a model checker was used to check whether the reduced model satisfies the formula or not. We have identified two types of execution ordering constraints namely, *precedes* and *leads to*. Figure 1 helps clarify the difference between the two types of rules where A *precedes* B but A *does not lead to* B—for more details about how rules are expressed as queries in BPMN-Q please refer to [3].

Our contribution in this paper addresses the issue of showing violations of ordering of activities compliance rules to the user. In order to tackle this, we had to choose between two options. The first is to develop a mechanism that translates the counter examples generated by model checkers or other verification tools back to the visual notation the user can understand(cf. [8]). The other solution was to develop our own mechanism to show these violations. The drawback of the first solution is manifold. In first place, the generated counter examples depend on the input finite state machine. Since we fed reduced models to the model checker [3], the resulted counter examples are useless. Another drawback



**Fig. 1.** A fragment of a process model to show difference between leads to and precedes concepts. Activity C leads to activity D since in all execution scenarios execution of C will be followed by an execution of D before the process instance terminates. In the mean time activity C does not precede activity D since in some instances execution of D was reachable through execution of activity B rather than C

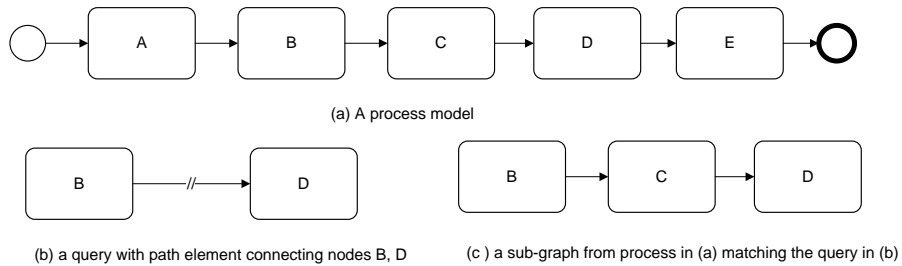
is that such approach depends on the output format of the specific model checker used. Each time model checking software is changed, a re-implementation of the translation software is required (cf. [8]). One more issue is that the generated counter example are given as a disproof and they are not comprehensive. In other words, the process could still have other violations (counter examples) that were not reported by the model checker<sup>1</sup>. All this leads to complex tool chains that might not perform well in all cases.

Taking the specific case of execution ordering rules, a counter example would be the existence of an execution scenario where this order is broken. So taking for example a *leads to* rule e.g. *A leads to B* means that in all execution scenarios If activity A is executed then activity B must be executed some point in future before the process instance terminates. A corresponding temporal expression will be  $G(A \rightarrow F(B))$ , where G is the temporal operator to say that a predicate holds in all states of the system, while F states that eventually the predicate will hold. So, a counter example would be finding at least one execution scenario where activity A executes but activity B does not execute until the end of the process. This counter example can be formalized as  $F(A \wedge ((\neg B) \cup \text{End}))$  where U is a binary temporal operator stating that the left predicate holds in all states until the state where the right predicate is true, and “End” indicates the termination of the process. If we look carefully at which situations the violation can occur it would be: 1) A process model containing both activities A and B but no execution path at all available between them. 2) A process model containing both activities A and B and in some execution scenarios A can execute but not B. 3) A process model containing only activity A without B. Considering the rule  $G(A \rightarrow F(B))$  as the pattern we want to prove satisfied by the process model, we call the second rule  $F(A \wedge ((\neg B) \cup \text{End}))$  the anti pattern which if satisfied by the process model represents a violation to the rule. Such an anti pattern can be represented by a BPMN-Q query, as well as the pattern was represented by a query [3]. Before we go further with the details of using BPMN-Q to express such anti patterns, we briefly introduce notations of BPMN-Q with necessary formalization.

<sup>1</sup> Interested reader can refer to how LoLA [23] model checker works as an example

## 2.1 BPMN-Q

BPMN-Q [2] is a visual language based on BPMN. It is used to query business process models by matching a process graph to a query graph. For more details about how queries are processed please refer to [2, 3]. In addition to the sequence flow edges in BPMN, BPMN-Q introduces the concept of path (see edge in Figure 2 (b)). When matching a process graph (like the one in Figure 2 (a)) to the query in (b), the result of the path edge is the sub-graph of the matching process in which the two nodes along with nodes in between (Figure 2 (c)).



**Fig. 2.** Example of a BPMN-Q query

We now define process graphs, where the set of nodes  $\mathcal{N}$  can be either activities, events or gateways. Set  $\mathcal{F}$  represents the sequence flow edges that can connect nodes. The definition is adapted from [4].

**Definition 1.** A process graph is a tuple  $PG = (\mathcal{N}, \mathcal{F})$  where

- $\mathcal{N}$  is a finite set of nodes that is partitioned into activities, events, and gateways
- $\mathcal{F} \subseteq \mathcal{N} \times \mathcal{N}$  is the sequence flow relation between nodes.

BPMN-Q provides more types of edges to connect nodes. It is possible to express in the query what we call paths as discussed earlier.

**Definition 2.** A query graph is a tuple  $QG = (\mathcal{N}Q, \mathcal{S}, \mathcal{PA})$  where

- $\mathcal{N}Q$  is a finite set of nodes that is partitioned into activities, events, and gateways
- $\mathcal{S} \subseteq \mathcal{N}Q \times \mathcal{N}Q$  is the set of sequence flow edges between nodes.
- $\mathcal{PA} \subseteq \mathcal{N}Q \times \mathcal{N}Q$  is the set of path edges between nodes.

A process graph is relevant to a query graph only if the set of activity nodes in a process graph is a superset of the activity nodes in a query graph.

A path edge is a placeholder for all nodes that reside on any common execution path between its source and destination. In addition, a path edge has an *exclude* property. For instance, a BPMN-Q containing a path edge from activity A to activity F excluding activity D is meant to find a subgraph of the process graph having nodes between A and F without executing D.

A relevant process graph to a query graph is said to match it if it satisfies all sequence flow and path edges, as in Definition 3.

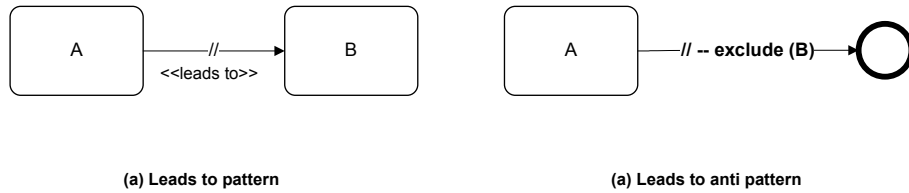
**Definition 3.** A process graph  $PG = (\mathcal{N}, \mathcal{F})$  matches a query graph  $QG = (\mathcal{NQ}, \mathcal{S}, \mathcal{PA})$  iff:

- $\mathcal{NQ} \subseteq \mathcal{N}$ .
- $\mathcal{S} \subseteq \mathcal{F}$ .
- $\forall (n, m) \in \mathcal{PA}$  there is a sequence of nodes in  $PG$  that form a path from  $n$  to  $m$ .

## 2.2 Derivation of Anti Pattern Queries

In [3] we showed how can BPMN-Q be used to express two types of ordering constraints between activities, namely, *leads to* and *precedes*. In this section, we show how the set of anti patterns can be derived automatically from a BPMN-Q query containing *precedes* and *leads to* paths.

**Leads to Pattern/Anti Pattern** Recalling the discussion about finding counterexamples in the beginning of this section, an anti pattern would be informally as find an execution path where the activity A is executed but activity B not i.e.  $F(A \text{ and } (\neg B \cup \text{End}))$ . This is expressed as a BPMN-Q query as shown in Figure 3.



**Fig. 3.** Leads to Pattern and Anti Pattern

**Precedes Pattern/Anti Pattern** In the same fashion, given an A *precedes* B pattern, which can be expressed formally as the temporal expression  $G(B \rightarrow O(A))$  that is described informally as, when activity B is executed then activity A must have been executed some point in the past. An anti pattern would be formally described as  $F(\text{Start} \wedge ((\neg A) \cup B))$  Where “Start” indicates the beginning of execution. Informally, find at least one execution path where activity B executes without executing activity A before at all. So, the anti pattern is captured as a BPMN-Q query as in Figure 4 where we seek an execution path from the beginning of the process that reaches the activity B excluding the activity A.

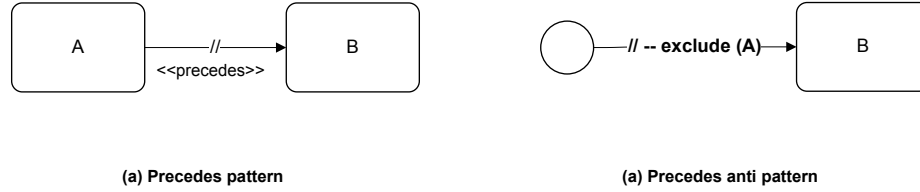


Fig. 4. Precedes Pattern and Anti Pattern

### 2.3 The Validation Process

In this section we discuss how can a pattern (representing the compliance rule) and anti pattern (representing possible violations) queries can be used to prove or disprove (with a visualization of violation) the compliance of process models with rules.

When BPMN-Q finds a match to a query in a process model, this means that in some execution scenarios the query is satisfied. BPMN-Q by itself lacks the ability to express the *for all* quantification. That is why we used model checkers in [3] to check compliance.

In this paper, we depend on BPMN-Q only not just to prove compliance but also to give explanation of violations if any. The validation process starts by a pattern expressed by the user. A set of anti patterns are generated automatically as discussed in Section 2.2. Both the pattern along with the set of anti patterns are called the validation rule.

**Definition 4.** A validation rule is a pair  $R=(q_p,A)$ , where  $q_p$  is the pattern query given by the user.  $A$  is the set of anti patterns that are automatically generated for each leads to and precedes edge in  $q_p$

When the pattern query is processed by BPMN-Q, according to Definition 3, the set of process models in the repository can be divided into two disjoint sets roughly matching  $M$  and non matching process models  $NM$ . If the pattern query finds a match in a process model, we need to check for a match for any of the anti patterns. If none of the anti patterns finds a match process in  $M$ , the process is guaranteed to be *compliant* with the rule. On the other hand, if any of the anti patterns finds a match to a process in  $M$ , this process is *non compliant* and its subgraph matching the anti pattern query is the scenario that violates the rule. With respect to the pattern query, the set  $NM$  can be further subdivided into the set relevant but not matching processes  $NR$ , the set of partially relevant processes  $PR$ , and the set of non relevant processes.

**Definition 5 (Non-matching Relevant Process).** A process graph  $PG=(\mathcal{N},\mathcal{F})$  is said to be non-matching relevant to query graph  $QG=(\mathcal{N}Q, \mathcal{S}, \mathcal{PA})$  iff  $\mathcal{N}Q \subseteq \mathcal{N}$  and  $\exists(n,m) \in \mathcal{PA}$  where the process graph does not provide an execution path between  $n$  and  $m$ .

$NR$  is the set of all non-matching relevant processes of a query graph.

The set NR holds process models containing violations where activities mentioned in the pattern exist in the process but with no common execution paths at all. The set PR contains violating process models where a proper subset of the pattern activities exist in a process model.

**Definition 6 (Partially Relevant Process).** *A process graph  $PG=(\mathcal{N},\mathcal{F})$  is said to be partially relevant to query graph  $QG=(\mathcal{NQ},\mathcal{S},\mathcal{PA})$  iff  $\mathcal{NQ}\setminus\mathcal{N}\neq\emptyset$   $PR$  is the set of all partially relevant processes of a query graph.*

Figure 5 represents the relationship between these sets on one hand, and the set  $AP = AP' \cup AP''$  represents process models matching anti pattern queries on the other hand. From Figure 5, we can see that it intersects with sets M, NR, PR. All elements contained in AP represent a way to violate the compliance rule (pattern). The three possible intersections correspond to the three possible cases of violation mentioned in Section 2.

- NR is the set of process models containing activities mentioned in the rule but without execution paths at all.
- $AP \cap M$  is the set of process models that contain all activities in the rule but the order can be violated in some execution scenarios.
- $AP \cap PR$  contains process models where some of the activities in the rule exist.

In each of the preceding cases the violating execution scenario can be shown to the user.

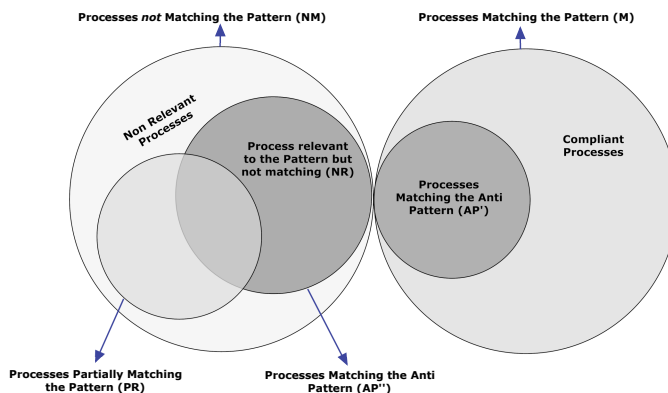


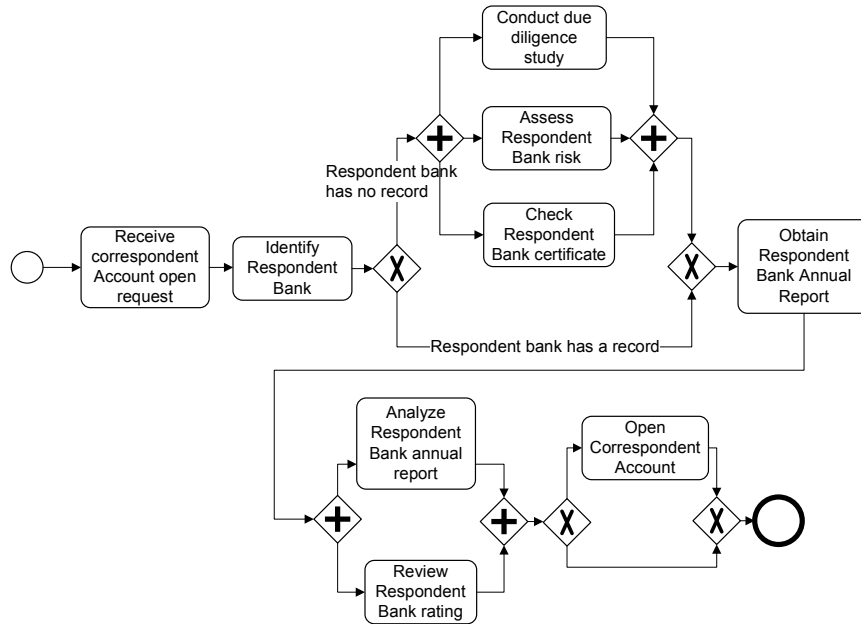
Fig. 5. Relation between sets of process models in the repository

### 3 Compliance Example

In this section, we apply our approach on a business process from the banking sector. In the financial sector, banks are obliged to conform to several compliance

rules. Consider the process model in Figure 6 (expressed in BPMN notation) for opening a correspondent bank account. This type of accounts is opened by a bank (respondent bank) in some country in a foreign bank in another country to ease and speed the operation of money transfer.

The process starts with “Receive correspondent Account open request” to open an account. Bank Identity is determined in order to go on with the procedure of opening the account. If this is the first time such respondent bank requests to open an account, some checks must take place. The bank to open the account needs to conduct a study about the respondent bank due diligence “Conduct due diligence study”, it also needs to assess the risk of opening an account for that respondent bank “Assess Respondent Bank risk”, and to check respondent bank certificate in order to proceed with opening the account. On the other hand, if such respondent bank has a record with the bank, these checks are skipped. In any of the cases, the bank has to obtain a report about the performance of the respondent bank “Obtain Respondent Bank Annual Report”. This report is analyzed by the bank “Analyze Respondent Bank annual report”, and the respondent bank rate is reviewed “Review Respondent Bank rating”. If the respondent bank passes the checks, an account is opened “Open Correspondent Account”.

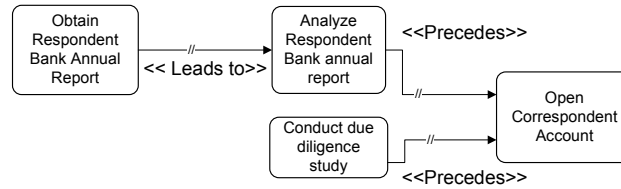


**Fig. 6.** Opening a correspondent account business process

To avoid unwanted money transfer for funding criminal actions, new rules to prevent money laundering have been developed by a central bank. The compli-

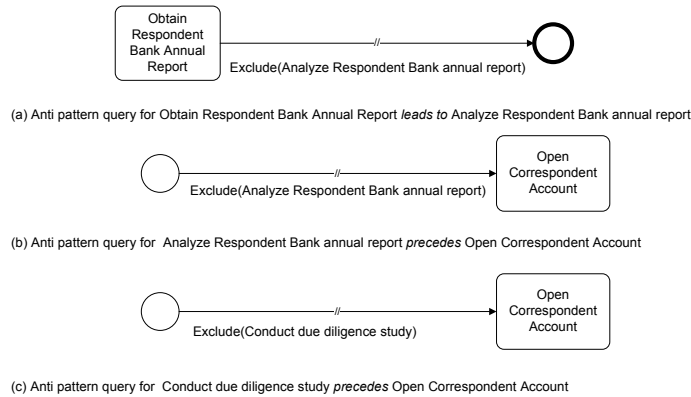
ance officer of the bank wants to check the compliance of the process in Figure 6 with the following rule.

*Before opening a correspondent account, a due diligence study must be conducted. Respondent annual report is analyzed when it is obtained before opening the correspondent account.* Based on the above rule, the officer formulated a compliance rule (pattern) as the BPMN-Q query shown in Figure 7.



**Fig. 7.** A BPMN-Q query capturing the compliance rule

By starting to process this query, anti pattern queries are generated automatically for each type of path edge in the compliance rule. The generated anti pattern queries are shown in Figure 8

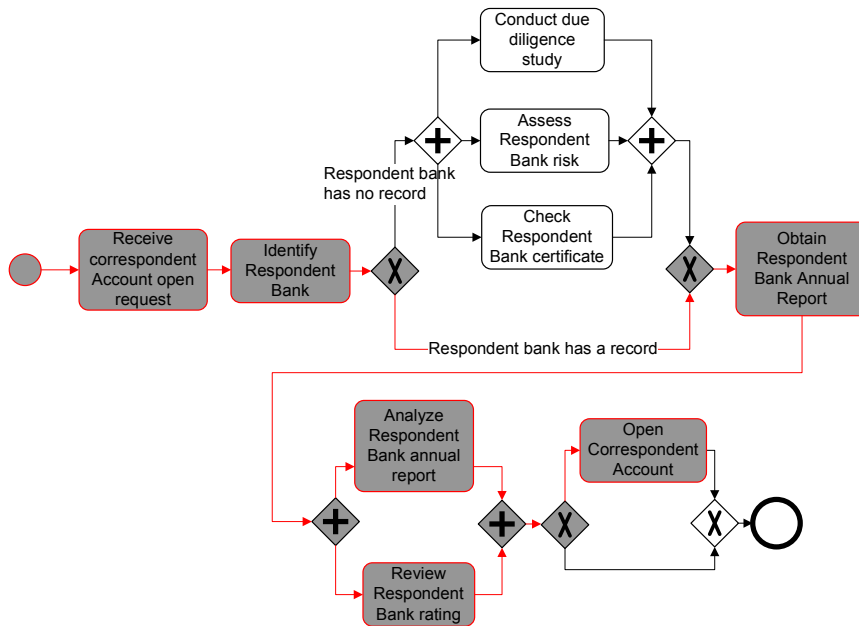


**Fig. 8.** Anti pattern queries

The pattern query found a match in the process of Figure 6. This means that there are execution scenarios that satisfy the rule. In order to declare full compliance, the process must be free from a match to any of the anti patterns.

By examining each of the anti patterns in Figure 8, anti pattern in (a), which tries to find an execution path where activity “Obtain Respondent Bank Annual

Report” executes and the activity “Analyze Respondent Bank Annual Report” does not till the process terminates, will not find any matches. Please note that the sequence <Obtain Respondent Bank Annual Report, AND Split, Review Respondent Bank rating, AND Join, ..., end event > cannot be considered as a match, because AND Split node after Obtain Respondent Bank Annual Report activity will activate both activities Review Respondent Bank rating and Analyze Respondent Bank Annual report. This is a feature of BPMN-Q query processor, whenever a node is excluded, all parallel nodes to it are excluded as well in order to guarantee correct results. With the same semantic, anti pattern query in (b) will fail to find a match.



**Fig. 9.** A violation to the compliance rule

Anti pattern in (c) finds a match. The match is shown in Figure 9 Where there is an execution scenario that starts from the beginning of the process and selects an alternative path to that where the parallel block to which the activity Conduct due diligence study belongs. It is clear that this scenario represents the way the compliance rule can be violated, and this case needs correction by experts.

## 4 Related Work

We can divide work related to that presented in this paper into three areas : the usage of patterns / anti-patterns, compliance checking of business process models, and finally visualization of verification tools output.

The term *pattern* in software engineering describes common and frequently occurring practice that is proved useful. In the same fashion, anti patterns refer to practices in software engineering that are far from being useful [12]. In [25] authors have applied the concept of patterns to workflow modeling. A description of bug patterns was introduced in [26] where authors have derived a set of bug (anti) pattern for each of the control flow workflow patterns. The purpose behind the derivation of these bug patterns was to enhance the performance of model checkers by guiding them to possible violations. In comparison to our work, our anti patterns are derived automatically from user inputs and we do not require model checking to discover bugs.

According to [22] checking for compliance can occur either *after-the-fact* or *before-the-fact*. Manually auditing processes is one way to check compliance in an *after-the-fact* fashion. An automated approach to detect violations from workflow logs using LTL checkers was introduced in [24].

*Before-the-fact* approaches can be further categorized as either (a) compliance-aware design or (b) post design verification. [22] is an example for compliance-aware design. Here, control objectives are modeled independently, that way addressing conflicting requirements between processes and regulations. The authors build their approach on a requirements modeling framework that later on propagates (forces) these requirements onto business processes. Another approach to guarantee compliance by design is given in [15], introducing PENELOPE as a declarative language to capture obligations and permissions imposed by business policies (sequencing and timing constraints between activities) and later on automatically generate business processes that are, by design, compliant with these policies.

The same authors have discussed in [14] the importance of explicitly modeling business rules as an enabler of flexibility and generation of less complex business processes. A more recent approach to compliance by design is introduced in [21] which can be seen as an extension of [22] with a special focus on assisting the process designer to create compliant business processes.

We categorize our work as *before-the-fact* and *post design*. Other work in this category is briefly discussed. The Formal Contract Language (FCL) was introduced in [16] to formally measure the compliance between a business contract and a business process. In [13] an approach to check compliance of business processes and the resolution of violations was introduced. The paper defines Semantic Process Networks (SPN), where each activity is further annotated with effect predicates. Rules are then verified against this network. In [20], a formal approach based on model checking was given to check for compliance of processes defined in BPEL against constraints defined in the Business Property Specification Language (BPSL) that are translated to LTL. The approach is close to ours. However, we are able to express constraints in PLTL rather than LTL only, which gives our approach more expressiveness over the other. Similar work that

verifies BPEL processes is in [27] where authors propose their own language PROPOLS to capture patterns to be checked against a business process. The approach depends on transforming PROPOLS expressions into FSAs and BPEL into LTS/FSA and check the language inclusion between the two FSAs. Work in [9–11] defines a set of visual patterns using the Process Pattern Specification Language (PPSL). These patterns are used to express constraints against UML Activity Diagrams. PPSL patterns are then translated into PLTL formulae. The approach is the closest to ours from the point of expressiveness i.e. it supports reasoning with PLTL, yet we offer a small set of constructs to express the same set of concepts. Similar work on verification of properties against UML Activity Diagrams has been accomplished earlier by Erik et al in [5, 7, 6], where they offered their own formalization of ADs and used model checking to verify properties against them.

Visualization of possible violations to compliance rules has been addressed in [8, 20]. In [8], the purpose of visualization was to show parts of process models (workflow nets) that make the model unsound. Their work depends on translating violation messages from Woflan back to the input workflow net. Work in [20] visualized counter examples generated by model checker on the level of the finite state machine. A framework for guiding service compositions based on PROPOLS [17] proactively suggested next step activities in a composition in order not to violate temporal business rules. The framework can also suggest remedy actions when violations are detected. All approaches require a state space exploration, a cost that is avoided in our approach.

## 5 Conclusion

In this paper we discussed an approach to visualize violation of control flow ordering compliance rules. The compliance rules are expressed in BPMN-Q and are called patterns. The anti pattern queries are derived automatically. The merits of the approach are 1) Expressing rules visually in BPMN-Q in a way similar to modeling 2) The querying nature of BPMN-Q allows to discover the process models in a repository that are subject for compliance checking 3) Automatic generation of anti pattern queries 4) Anti patterns help discover non-compliant process models that would not be discovered by patterns. The approach works under the assumption that activity names are aligned to a common ontology respected by all business modelers. The approach is limited to control flow only. This limitation is subject for future work, where it will be possible to combine rules containing conditions about data objects in a process model.

A source of efficiency in the approach is avoiding expensive model checking approaches. Thus, saving costs of transformation between the modeling language and the input formalism of the model checker, the model checking, and the reverse mapping from the output of the model checker back to the visual representation of the modeling language.

One of the interesting future enhancements is to study the possibility to automatically suggest modifications to process models in order to resolve violations.

## References

1. *Sarbanes-Oxley Act of 2002*. Public Law 107-204, (116 Statute 745), United States Senate and House of Representatives in Congress, 2002.
2. A. Awad. BPMN-Q: A Language to Query Business Processes. In M. Reichert, S. Strecker, and K. Turowski, editors, *EMISA*, volume P-119 of *LNI*, pages 115–128. GI, 2007.
3. A. Awad, G. Decker, and M. Weske. Efficient compliance checking using bpmn-q and temporal logic. In M. Dumas, M. Reichert, and M.-C. Shan, editors, *BPM*, volume 5240 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2008.
4. R. M. Dijkman, M. Dumas, and C. Ouyang. Formal Semantics and Analysis of BPMN Process Models using Petri Nets. Technical report, Faculty of Information Technology, Queensland University of Technology, 2007.
5. H. Eshuis. *Semantics and Verification of UML Activity Diagrams for Workflow Modeling*. PhD thesis, Centre for Telematics and Information Technology (CTIT) University of Twente, 2002.
6. R. Eshuis. Symbolic model checking of uml activity diagrams. *ACM Trans. Softw. Eng. Methodol.*, 15(1):1–38, 2006.
7. R. Eshuis and R. Wieringa. Tool support for verifying uml activity diagrams. *IEEE Transactions on Software Engineering*, 30(7):437–447, 2004.
8. C. Flender and T. Freytag. Visualizing the soundness of workflow nets. In *13th Workshop Algorithms and Tools for Petri Nets, AWPN 2006*, pages 47–52, 2006.
9. A. Förster, G. Engels, and T. Schattkowsky. Activity diagram patterns for modeling quality constraints in business processes. In *MoDELS*, pages 2–16, 2005.
10. A. Förster, G. Engels, T. Schattkowsky, and R. V. D. Straeten. A pattern-driven development process for quality standard-conform business process models. In *IEEE Symposium on Visual Languages and Human-Centric Computing VL*, 2006.
11. A. Förster, G. Engels, T. Schattkowsky, and R. V. D. Straeten. Verification of business process quality constraints based on visual process patterns. In *TASE*, pages 197–208. IEEE Computer Society, 2007.
12. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.
13. A. Ghose and G. Koliadis. Auditing business process compliance. In B. J. Krämer, K.-J. Lin, and P. Narasimhan, editors, *ICSOC*, volume 4749 of *Lecture Notes in Computer Science*, pages 169–180. Springer, 2007.
14. S. Goedertier and J. Vanthienen. Compliant and flexible business processes with business rules. In G. Regev, P. Soffer, and R. Schmidt, editors, *BPMDs*, volume 236 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.
15. S. Goedertier and J. Vanthienen. Designing Compliant Business Processes from Obligations and Permissions, 2nd Workshop on Business Processes Design (BPD’06), Proceedings. In J. Eder and S. Dustdar, editors, *Business Process Management Workshops*, volume 4103 of *Lecture Notes in Computer Science*, pages 5–14. Springer Verlag, 2006. Springer-Verlag Berlin Heidelberg 2006.
16. G. Governatori, Z. Milosevic, and S. Sadiq. Compliance checking between business processes and business contracts. In *EDOC ’06: Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC’06)*, pages 221–232, Washington, DC, USA, 2006. IEEE Computer Society.
17. J. Han, Y. Jin, Z. Li, T. Phan, and J. Yu. Guiding the service composition process with temporal business rules. In *ICWS*, pages 735–742. IEEE Computer Society, 2007.

18. T. E. Hartman. *The Cost of Being Public in the Era of Sarbanes-Oxley*. [Chicago, Ill.] : Foley & Lardner, June 2006.
19. R. Lu, S. Sadiq, and G. Governatori. Compliance aware business process design. In *3rd International Workshop on Business Process Design*, 2007.
20. Y. Lui, S. Müller, and K. Xu. A static compliance-checking framework for business process models. *IBM SYSTEMS JOURNAL*, 46(2):335–362, 2007.
21. S. S. R. Lu and G. Governatori. Compliance aware business process design. In *3rd International Workshop on Business Process Design (BPD07), in Conjunction with 5th International Conference on Business Process Management*, 2007.
22. S. W. Sadiq, G. Governatori, and K. Namiri. Modeling control objectives for business process compliance. In G. Alonso, P. Dadam, and M. Rosemann, editors, *BPM*, volume 4714 of *Lecture Notes in Computer Science*, pages 149–164. Springer, 2007.
23. K. Schmidt. Lola a low level analyser. In *Application and Theory of Petri Nets 2000: 21st International Conference, ICATPN 2000, Aarhus, Denmark, June 2000. Proceedings*, volume 1825, page 465. Springer Berlin / Heidelberg, 2000.
24. W. M. P. van der Aalst, H. T. de Beer, and B. F. van Dongen. Process mining and verification of properties: An approach based on temporal logic. In *OTM Conferences (1)*, pages 130–147, 2005.
25. W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
26. K. Xu, Y. Liu, and C. Wu. Guided reasoning of complex e-business process with business bug patterns. In *ICEBE '06: Proceedings of the IEEE International Conference on e-Business Engineering*, pages 195–202, Washington, DC, USA, 2006. IEEE Computer Society.
27. J. Yu, T. P. Manh, J. Han, Y. Jin, Y. Han, and J. Wang. Pattern based property specification and verification for service composition. In *WISE*, pages 156–168, 2006.