

Visualization of Compliance Violation in Business Process Models

Ahmed Awad and Mathias Weske

Business Process Technology Group
Hasso Plattner Institute at the University of Potsdam
D-14482 Potsdam, Germany
{ahmed.awad, mathias.weske}@hpi.uni-potsdam.de

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.

1 Introduction

Enterprises are using business process models to run their services smoothly. These artifacts define how the enterprise works and they are a good means of checking control requirements. 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]. Keeping processes compliant is an expensive operation [11]. Automated approaches emerged to address compliance issue from different points of view. On one hand, some approaches favor deriving process models by compliance rules [13, 8]. On the other hand delaying the checking of the compliance rules to a post design step is discussed in [15, 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. 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, 15]. 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 [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 that is checked against the process model. To check the temporal formula against the process model, we used BPMN semantics in [4] to derive the behavioral model of a BPMN process.

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 extend the set of execution ordering compliance rules that can be expressed in BPMN-Q. Also, we show how BPMN-Q is used to visualize possible violation scenarios. 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 Patterns and Anti Patterns

In the next subsection we briefly introduce BPMN-Q and how it was used to was used to express compliance rules. In subsequent subsections we discuss how BPMN-Q queries can be used to express more compliance rules (patterns) and violation scenarios (anti patterns) respectively.

2.1 BPMN-Q

Based on BPMN, BPMN-Q [2] is a visual language that is designed to query business process models by matching a process to a query structurally. In addition to the sequence flow edges of BPMN, BPMN-Q introduces the concept of path edges as illustrated in Fig. 1(b). Such a path might match a sub-graph of a BPMN process — the highlighted part of Fig. 1(a) is the matching part to the path edge of Fig. 1(b).

While such a path considers only the structure of a process, execution semantics have to be considered in the query if BPMN-Q is used for compliance checking. In this case, we type paths between two activities as being either *precedes* (cf. Fig. 1(d)) or *leads to* (cf. Fig. 1(c)) paths [3]. The former requires that before activity B is about to execute, activity A has already been executed. The latter, in turn, states that an execution of the first activity is *eventually* followed by an execution of the second activity. Considering the process in Fig. 1(a), it is obvious that A *precedes* D is satisfied, while A *leads to* D is not. A BPMN-Q query with path edges typed as *leads to* and/or *precedes* is a behavioral query. Otherwise, it is a structural query.

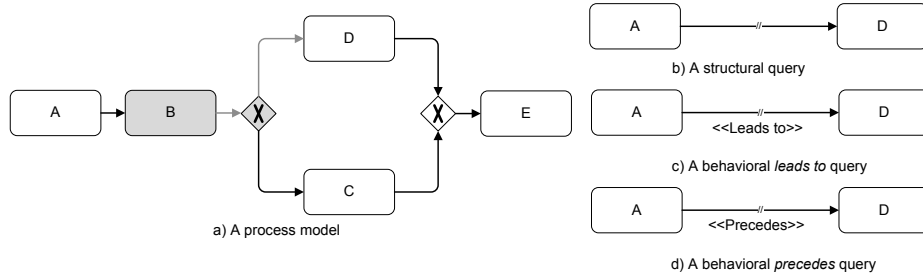


Fig. 1. BPMN-Q Path Edges

The path edge has one more property called the *exclude* property. Imagine a structural query with a path from activity A to activity E where *exclude* is set to D. Matching this query to the process in Fig. 1(a) yields the whole model except activity D.

Moreover, behavioral BPMN-Q queries are wrappers for past linear time temporal logic PLTL [21] expressions, PLTL provides more temporal operators that allows reasoning about the past states of a system. That is, *leads to* paths are transformed into an implication with the *eventually* quantifier, whereas *precedes* paths map to an implication with the *once* operator, see Table 1 for the mapping. Setting the *exclude* property for behavioral paths affects the PLTL formula.

Matching a behavioral BPMN-Q query to a process model is a two-step approach. Firstly, the implied structural query is matched to the process model. Secondly, depending on the result the behavior of the matching part is checked against the PLTL formula of the behavioral query. Matching a structural query is in turn a two-step approach. Firstly, *all* activities mentioned in the query have to be present in the process model. Secondly, *all* path edges in the query have to evaluate to a non empty subgraph of the process model.

2.2 Patterns for Execution Ordering Compliance Rules

Based on [5], we can describe the presence, absence, and/or the ordering of activities within a scope. A scope is either global, i.e., the whole process model, before some other activity, after some other activity, or between two activities.

With regard to a single activity, it might be required execute it in all process instances, e.g., in a shipment process the received packets must be inspected in every case. Thus, we call such pattern a global presence as shown in Fig. 2(a). On the other hand, it might be the case that certain activity must not execute at all i.e. such an activity is absent from the process model. This case is called the global absence as shown in Fig. 2(b). For a *before* scope, an activity A might be required to be absent before the execution of another activity B as shown in Fig. 2(c), e.g., it is not allowed to send goods to the customer before receiving payment. Similarly is the *after* scope as shown in Fig. 2(d). Response pattern [5] is the typical case of leads to compliance rule that was presented in [3]. It is shown in Fig. 2(e). This pattern also is the case of *after* scope presence. The case of absence of an activity in a scope *between* two other activities is shown in Fig. 2(f). Also, the precedence pattern, which is similar to the precedes compliance rule in [3] is shown in Fig. 2(g). This pattern can be used also to express *before*

scope presence. A different way to represent the *absence* in *between* scope is as shown in Fig. 2(h). This is a core set of patterns from which one can build more complex ones.

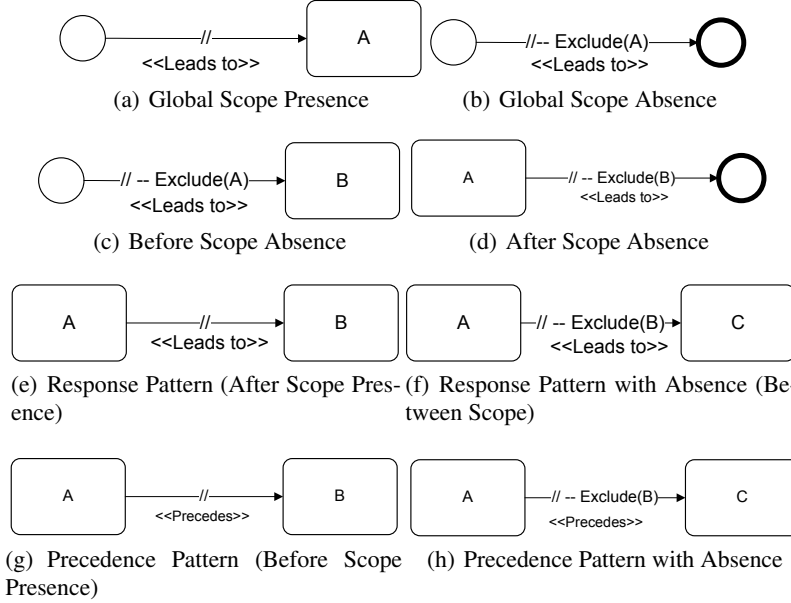


Fig. 2. Patterns for Execution Ordering Compliance Rules

For instance, a *before* scope only *presence* pattern of Activity A and B i.e. activity A must execute before activity B but no after it. This rule is shown in Fig. 3.

Each of these patterns (Compliance Rule) is mapped into a PLTL formula. The mapping is shown in Table 1. In PLTL, classical logical operators are extended with new ones that allow to evaluate the truth value of predicates in

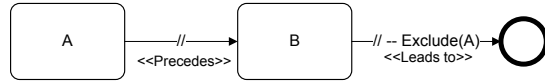


Fig. 3. A Before Only Presence Pattern

past states of a system, process models in this case. The temporal operators used by this paper are **G** the global operator where its argument has to hold in *all future* states. The eventually operator **F** describes that its argument must hold in *some future* state. **O** is its past counter part. The binary operator **U** is called the until operator where pUq describes that p has to hold *until* the point in time q holds. The past operator since **S** is its counter part.

According to Table 1, each compliance query (pattern) has a mapping into PLTL. Thus, each of these queries can be verified against a process model using model checking as we described earlier. Since our objective in this paper is beyond model checking, i.e., we need to know *how* the rule was violated. We describe in the next subsection the derivation of so-called *anti* pattern queries. For each pattern query, there is a set of anti pattern queries. Each anti pattern declaratively describes a violation scenario.

Pattern	PLTL Formula
Global Scope Presence	$F(A)$
Global Scope Absence	$G\neg A$
Before Scope Absence	$\neg AUB$
After Scope Absence	$\neg BSA$
Response (After Scope Presence)	$G(A \rightarrow F(B))$
Response with Absence (Between Scope Absence)	$G(A \rightarrow \neg BUC)$
Precedence (Before Scope Presence)	$G(B \rightarrow O(A))$
Precedence with Absence	$G(C \rightarrow \neg BSA)$

Table 1. Mapping of Patterns into PLTL

2.3 Derivation of Anti Pattern Queries

In order to visualize the possible violations to a compliance rule, 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 understands (cf. [6]). The other solution was to develop our own mechanism to show violations. The drawback of the first solution is manifold. Firstly, the generated counter example is given as a dis-proof and not all possible violations are reported. In other words, the process could still have other violations (counter examples) that were not reported by the model checker. Secondly, the generated counter example depends on the input state transition system of the process model. In case the transition system is generated after using reducing the original process model (cf. [3, 16], the resulting counter example would not be usable on the original process model. Finally, the usability of 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. [6]).

The rationale behind deriving anti-patterns is 1) to analyse the PLTL formula corresponding to each of the patterns shown in Fig. 2. By analysis, we study and enumerate the cases in which the formula can be violated by a process model. 2) For each possible violation opportunity, we develop a BPMN-Q query that captures execution scenario(s) in which violation occurs.

Global Scope Anti Patterns The global scope *presence* requires that certain activity must be executed in *all* instances of a process. This is also similar to the response pattern. Within process models, the violation of such requirement occurs when there are execution paths that lack the required activity. This is captured by the anti pattern query in Fig. 4(a). The opposite case of global *absence* is violated when there is at least one execution path in which activity A is executed. This is represented in Fig. 4(b).

Before Scope Anti Patterns The *presence* case requires that an Activity A is always executed before another activity B. So, the violation occurs when there is in the business process a chance to execute activity B without executing A at all before. This violation is expressed as the BPMN-Q query in Fig. 4(e) where there is an execution path from the start of the process to activity B without doing A at all. The other case of absence necessitates that A must *never* execute before B and B must always execute. The interpretation of this anti pattern is a bit complicated. The rule is violated in one of two cases. Either the activity A occurs and then B occurs thereafter. This case is shown in

Fig. 4(c). The other possibility of violation is that activity B in some instances is not executed at all. This violation can be captured by the query in Fig. 4(b) for activity B.

After Scope Anti Patterns In a similar way, the *presence* case is similar to the response pattern where after activity A is executed; activity B must be executed in some point in the future. The violation for this pattern is that in some instance A executed but never B after that. This meaning is captured by the query in Fig. 4(d). The absence pattern is on the other hand violated when after A executes B also executes. So the query in Fig. 4(c) would also capture this case of violation.

Between Scope Anti Patterns Finally the between scope with presence could be interpreted similarly to the before scope. The only difference that we replace the start event of a process with an activity that determines the beginning of the scope. The absence case violation is captured in Fig. 4(f).

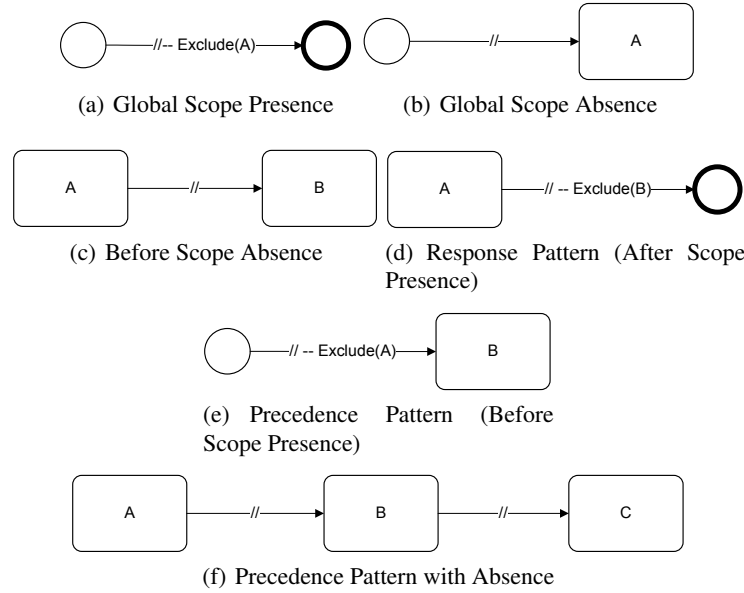


Fig. 4. Anti Patterns for Execution Ordering Compliance Rules

2.4 The Validation Process

The validation process starts by a pattern expressed by the user. A set of anti patterns are generated automatically as discussed in Section 2.3. When the pattern query is processed by BPMN-Q, the set of process models in the repository can be divided into two disjoint sets. The set of matching process models M , and the set of 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 NP . The set NR holds process models containing violations where activities mentioned in the pattern exist in the process but (some) path edges in the query are not satisfied. The set PR contains violating process models where a proper subset of the pattern activities exist in a process model.

Fig. 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 Fig. 5, we can see that AP intersects with sets M , NR , PR . All elements

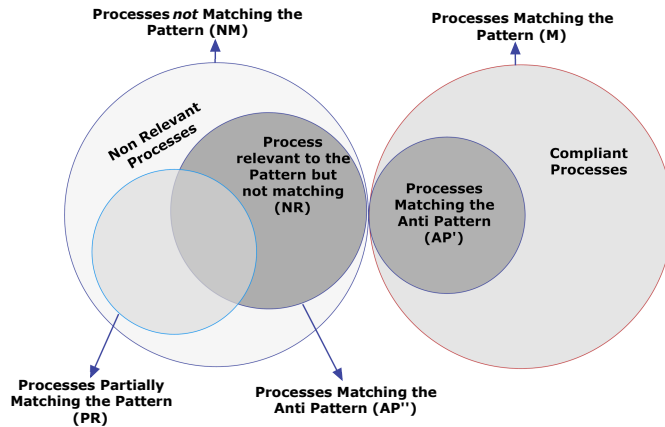


Fig. 5. Relation between sets of process models in the repository

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.

This approach to detect and visualize violation is fully implemented in an extension of BPMN-Q query processor. Once the pattern query is received, the query processor generates anti pattern queries for each *leads to* or *precedes* edge in the query. If any matches to any of the anti pattern queries is found, the matching part of the process to the anti pattern query is highlighted and returned to the user.

3 Compliance Example

In this section, we apply our approach on a business process from the banking sector. Consider the process model in Fig. 6 (expressed in BPMN notation) for opening a correspondent bank account.

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”.

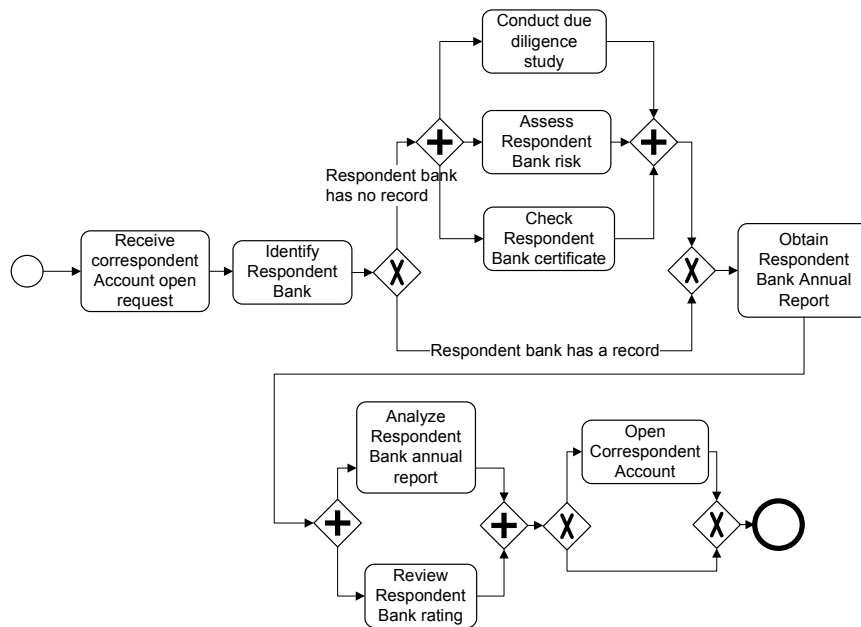


Fig. 6. Opening a correspondent account business process

New rules to prevent money laundering have been developed by a central bank. The compliance officer of the bank wants to check the compliance of the process in Fig. 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 Fig. 7. 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 Fig. 8.

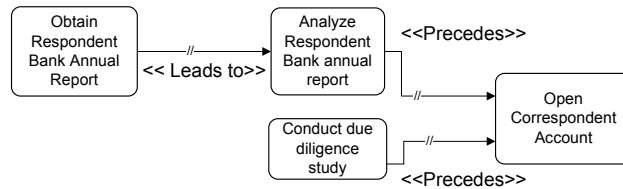


Fig. 7. A BPMN-Q query capturing the compliance rule

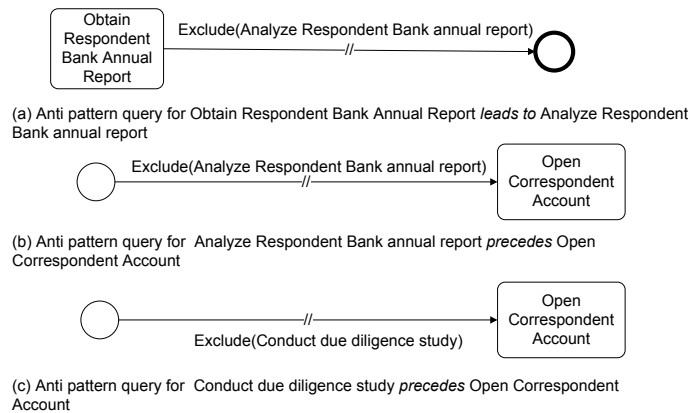


Fig. 8. Anti pattern queries

The pattern query found a match in the process of Fig. 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 the anti patterns in Fig. 8, the one in Fig. 8(a), looking for 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. 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. Similarly, anti pattern query in Fig. 8(b) will not find a match. Anti pattern in Fig. 8(c) finds a match. The match is shown in Fig. 9 where

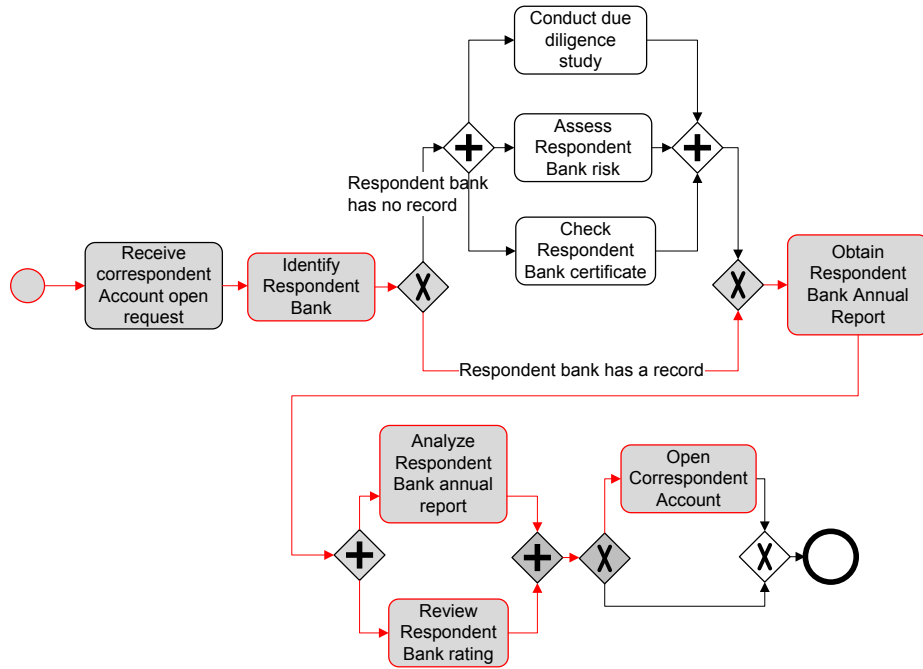


Fig. 9. A violation to the compliance rule

there is an execution scenario that starts from the beginning of the process and selects the lower choice branch (*Respondent bank has a record*). 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

Compliance Checking approaches can be categorized as either (a) Compliance by Design, where compliance rules are taken as input in the design process of *new* process models. The other approach depends on checking for compliance in a post design step. Thus, separating the modeling phase of a process model from the checking phase [19]. Our approach belongs to the second category.

Work in [13, 8, 17] deal with the enforcement of compliance requirements in the design process of new business process models. By definition, there is no chance for

violations to occur. However, once a new compliance requirement is introduced or the process model is modified, the checking for compliance is needed.

On the other hand, approaches like [20, 15] employ model checking to verify that process models satisfy the compliance rules. Comparing to our work, the notion of explaining violations in an intuitive way to the user was not addressed in that work.

Deontic logic was employed as a formalism to express these requirements in [9, 19]. It is possible to express *alternative* actions to be taken when a primary one is not done. Thus, it is possible to express how to handle *exceptions* but still there is no notion of tracking violation. Work in [12] has addressed the consistency between business process models and lifecycle of business objects processed in these models. One merit of that approach is to consider both control and data flow aspects of process models for compliance checking. Yet, explanation of deviations and their representation was not addressed. In [7] an approach to check compliance of business processes and the resolution of violations was introduced. Although automated resolution is important, the paper discussed it from a high level point of view. We believe that this point needs further investigation and we remark it as a future work.

A recent approach to measure the compliance distance between a process model and a rule was introduced in [14]. This could be seen as an intermediary step towards capturing deviations (violations) from the ideal scenarios.

Visualization of possible violations to rules has been addressed in [6, 15]. In [6], the purpose of visualization was to show parts of process models (workflow nets) that make the model unsound. Work in [15] visualized counter examples generated by model checker on the level of the finite state machine. A framework for guiding service compositions based on PROPOLS [10] proactively suggests next step activities in a composition in order not to violate temporal business rules. All approaches require a state space exploration, a cost that is avoided in our approach.

Declarative business process modeling is a way to allow flexibility in processes. Processes are modeled by specifying a set of execution ordering constraints on a set of activities [18]. In this way, compliance rules discussed in this paper could be expressed to guide the execution of process instances. Thus, there is no chance for violation. Comparing to our approach, we are concerned with detecting and visualizing possible violations on *imperative* process models.

5 Discussion

In this paper we discussed an approach to visualize violation of control flow ordering compliance rules. This step provides useful feedback to the user in order to correct violations. The compliance rules are expressed as behavioral BPMN-Q queries and are called patterns. The anti pattern queries are derived automatically as structural BPMN-Q queries.

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.

A limitation of the approach is the assumption that activity names are aligned to a common ontology respected by all business modelers. The presented patterns/anti

patterns are core ones. However, extensibility to express more complex situations is possible as was shown in Section 2.2.

In future, we investigate the inclusion of data aspects for both verification and visualization of violation. However, the challenge is to find/develop appropriate formalism that helps explain 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 *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 *BPM*, volume 5240 of *LNCS*, pages 326–341. Springer, 2008.
4. R. M. Dijkman, M. Dumas, and C. Ouyang. Semantics and analysis of business process models in BPMN. *Inf. Softw. Technol.*, 50(12):1281–1294, 2008.
5. M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *ICSE*, pages 411–420, New York, NY, USA, 1999. ACM.
6. C. Flender and T. Freytag. Visualizing the soundness of workflow nets. In *AWPN 2006*. Department Informatics Report 267, University of Hamburg, Germany.
7. A. Ghose and G. Koliadis. Auditing business process compliance. In *ICSOC*, volume 4749 of *LNCS*, pages 169–180. Springer, 2007.
8. S. Goedertier and J. Vanthienen. Designing Compliant Business Processes from Obligations and Permissions. In *BPM Workshops*, volume 4103 of *LNCS*, pages 5–14. Springer, 2006.
9. G. Governatori, Z. Milosevic, and S. Sadiq. Compliance checking between business processes and business contracts. In *EDOC*, pages 221–232. IEEE Computer Society, 2006.
10. 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.
11. T. E. Hartman. *The Cost of Being Public in the Era of Sarbanes-Oxley*. [Chicago, Ill.] : Foley & Lardner, June 2006.
12. J. M. Küster, K. Ryndina, and H. Gall. Generation of Business Process Models for Object Life Cycle Compliance. In *BPM*, volume 4714 of *LNCS*, pages 165–181. Springer, 2007.
13. R. Lu, S. Sadiq, and G. Governatori. Compliance aware business process design. In *Business Process Management Workshops*, volume 4928 of *LNCS*, pages 120–131, 2007.
14. R. Lu, S. Sadiq, and G. Governatori. Measurement of Compliance Distance in Business Processes. *Inf. Sys. Manag.*, 25(4):344–355, 2008.
15. Y. Lui, S. Müller, and K. Xu. A static compliance-checking framework for business process models. *IBM SYSTEMS JOURNAL*, 46(2):335–362, 2007.
16. J. Mendling. *Detection and Prediction of Errors in EPC Business Process Models*. PhD thesis, Institute of Information Systems and New Media Vienna University of Economics and Business Administration (WU Wien) Austria, May 2007.
17. Z. Milosevic, S. W. Sadiq, and M. E. Orłowska. Translating business contract into compliant business processes. In *EDOC*, pages 211–220. IEEE Computer Society, 2006.
18. M. Pesic, H. Schonenberg, and W. M. P. van der Aalst. DECLARE: Full Support for Loosely-Structured Processes. In *EDOC*, pages 287–300. IEEE Computer Society, 2007.
19. S. W. Sadiq, G. Governatori, and K. Namiri. Modeling control objectives for business process compliance. In *BPM*, volume 4714 of *LNCS*, pages 149–164. Springer, 2007.
20. 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.
21. L. Zuck. *Past Temporal Logic*. PhD thesis, Weizmann Intitute, Rehovet, Israel, August 1986.