Specification, Verification and Explanation of Violation for Data Aware Compliance Rules

Ahmed Awad, Matthias Weidlich, and Mathias Weske

Hasso-Plattner-Institute, University of Potsdam, Germany
{ahmed.awad,matthias.weidlich,mathias.weske}@hpi.uni-potsdam.de

Abstract. Compliance checking is becoming an inevitable step in the business processes management life cycle. Languages for expressing compliance requirements should address the fundamental aspects of process modeling, i.e. control flow, data handling, and resources. Most of compliance checking approaches focus on verifying aspects related to control flow. Moreover, giving useful feedback in case of violation is almost neglected. In this paper, we demonstrate how data can be incorporated into the specification of compliance rules. We call these rules data aware. Building upon our previous work, we extend BPMN-Q, a query language we developed, to express these rules as queries and formalize these rules by mapping them into PLTL. In addition, whenever a compliance rule is violated, execution paths causing violations are visualized to the user. To achieve this, temporal logic querying is used.

Keywords: Compliance Checking, Business Process Querying, Violation Explanation, Temporal Logic Querying.

1 Introduction

Business process models are the means to formalize the way services are composed in order to provide an added value [1]. Evidently, the notion of a service in this context depends on the purpose of the process model. High-level models capture the way business goals laid by top management are achieved, whereas low-level models describe technical service orchestrations. When process models define how the day to day business is enacted in a certain organizational and technical environment, they are the best place to check for and enforce compliance to organization policies and external regulations.

Compliance rules originate from different sources and keep changing over time. Also, these rules address different aspects of business processes, for example a certain order of execution between activities is required. Other rules force the presence of activities under certain conditions, e.g. reporting banking transactions to the central bank, when large deposits are made. Violation to compliance requirements originating from regulations, e.g., the Sarbanes-Oxley Act of 2002 [2] could lead to penalties, scandals, and loss of business reputation. Therefore, compliance checking is crucial for business success.

As both compliance requirements and processes evolve over time, it becomes necessary to have automated approaches to reason about the adherence of process models to these requirements. In this context, there is a number of challenges. First, the question how to express the compliance requirements has to be addressed. Second, process models that are subject to checking within large repositories containing hundreds to thousands of process models have to be identified. Third, there has to be an appropriate formalism for automatic checking of compliance rules against a process model. Fourth, users should be provided with useful feedback in case of violations.

While there are different notations available in order to express compliance rules, most of the approaches consider solely control flow aspects [3]. Moreover, the second challenge, that is automatic identification of processes that are subject to checking, was almost neglected in existing work. Further on, different formalism have been used to check for compliance. Here, model checking [4] is the most popular. The fourth challenge for compliance checking was neglected either. In case of violation, there is almost no feedback conveyed to the user in common approaches.

In a previous work [5], we demonstrated an approach that partially addresses these challenges. We employed BPMN-Q [6], a visual language we developed for querying business process models, to express compliance requirements (compliance rules) regarding execution ordering of activities (services) in process models. BPMN-Q was capable of expressing control flow aspects. Rules, expressed as BPMN-Q queries, were mapped into past linear time logic PLTL formulae [7]. Next, the resulting PLTL formulae were model checked [4] against the process models to decide about compliance.

Our contribution in this paper is twofold. First, we build upon work in [5] by incorporating data aspects. With data coming into play, the user can express so called data flow rules and conditional rules. Also, the mapping to PLTL is not straightforward. We achieve this by extending BPMN-Q with data aspects. Second, we introduce an approach to explain violations to compliance requirements. Whenever a rule is violated by the process model, we use temporal logic querying [8] techniques along with BPMN-Q queries to visually explain violations in the process model.

The use of BPMN-Q queries is manifold. First of all, BPMN-Q allows users to express compliance requirements in a visual way very similar to the way processes are modeled. That, in turn, simplifies application of our approach, as the business expert abstracts from the technical details. Second, a compliance rule that is defined as a query automatically determines the set of process models that are subject to checking in a repository. That is of particular importance, as such a repository might contain hundreds to thousands of process models. Finally, due to the nature of BPMN-Q query processing, the matching part(s) of the processes under investigation to the query are used to show execution scenarios causing violations directly on the process model level.

While we use BPMN for illustrating our contributions, results are applicable to other process modeling languages. The rest of the paper is organized as follows. Section 2 introduces an exemplary business process that needs to satisfy certain compliance rules. Section 3 is devoted to preliminaries on the applied techniques and Section 4 shows how BPMN-Q was extended to express data-aware compliance rules. Discussion of violation explanation is given in Section 5 and Section 6 gives details on our implementation. Related work is reviewed in Section 7, before we conclude in Section 8.

2 Motivating Example

A process model, expressed in BPMN notation, to open a correspondent bank account is shown in Fig. 1. The process starts with activity "Receive correspondent Account open request". Afterwards, the bank identity is looked up ("Identify Respondent Bank"). If

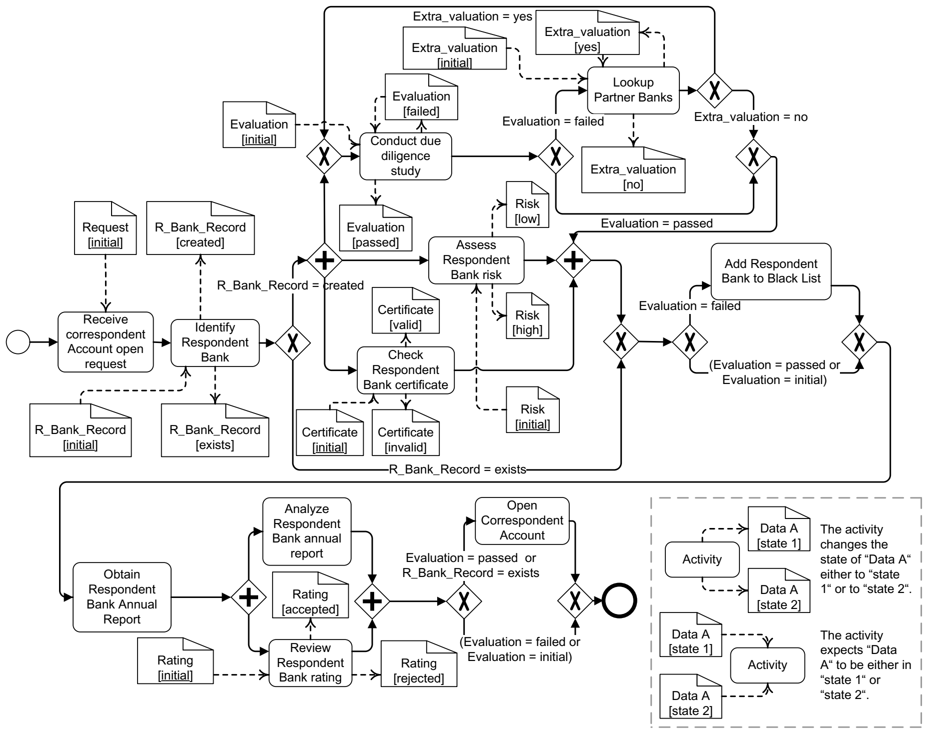

Extra_valuation = yes
Extra_valuation [yes]
Extra_valuation [initial]
Lookup Partner Banks
Evaluation [failed]
Evaluation = failed
Extra_valuation = no
Evaluation [initial]
Conduct due diligence study
Extra_valuation [no]
Risk [low]
Evaluation = passed
Request [initial]
R_Bank_Record [created]
Evaluation [passed]
Assess Respondent Bank risk
Add Respondent Bank to Black List
R_Bank_Record = created
Certificate [valid]
Risk [high]
Evaluation = failed
Receive correspondent Account open request
Identify Respondent Bank
Check Respondent Bank certificate
(Evaluation = passed or Evaluation = initial)
R_Bank_Record [initial]
R_Bank_Record [exists]
Certificate [initial]
Certificate [invalid]
Risk [initial]
R_Bank_Record = exists
Analyze Respondent Bank annual report
Open Correspondent Account
Evaluation = passed or R_Bank_Record = exists
Obtain Respondent Bank Annual Report
Rating [accepted]
Rating [initial]
Review Respondent Bank rating
Rating [rejected]
(Evaluation = failed or Evaluation = initial)
Activity
Data A [state 1]
Data A [state 2]
The activity changes the state of "Data A" either to "state 1" or to "state 2".
The activity expects "Data A" to be either in "state 1" or "state 2".

Fig. 1. A process model to open a bank account

this is the first time this bank requests to open an account, a new record is *created* and some checks must take place. The bank to open the account needs to conduct a study about the respondent bank due diligence, where the respondent bank may *pass* or *fail* this study. In case of failure, the bank inquires one of its partner banks about the respondent bank ("Lookup Partner Banks"). Then, it is decided, whether to make an extra study. It is also required to assess the risk of opening an account ("Assess Respondent Bank risk") resulting in either *high* or *low* risk. In the mean time, the respondent bank certificate is checked for *validity*. If the evaluation fails, the respondent bank is added to a black list. Subsequently, the bank obtains a report about the performance of the respondent bank ("Obtain Respondent Bank Annual Report"). This report is analyzed, and the respondent bank rate is reviewed. If the respondent bank passes the due diligence evaluation or it has already a record at the bank, an account is finally opened.

To prevent money laundering, various compliance rules are in place for the banking sector. We assume that the following rules must be checked for the process in Fig. 1.

R1: An account is opened only in case that risk is low.

R2: The respondent bank must always be added to the black list in case its due diligence evaluation fails.

R3: Before opening an account, the respondent bank rating must be accepted.

R4: In case the respondent bank rating review is rejected, an account must never be opened.

3 Preliminaries

3.1 Linear Temporal Logic with Past Operators (PLTL)

Linear Temporal Logic (LTL) allows expressing formulae about the future states of systems. In addition to logical connectors $(\neg, \wedge, \vee, \rightarrow, \Leftrightarrow)$ and atomic propositions, LTL introduces temporal operators, such as *eventually* (F), *always* (G), *next* (X), and *until* (U). PLTL [7] extends LTL by operators that enable statements over the past states. That is, it introduces the *previous* (P), *once* (O), *always been* (H), and *since* (S) operators.

3.2 Data Access Semantics

Formalization of data access in process models is needed to be able to reason about. We formalized the semantics of accessing data objects by activities in a BPMN model in [9]. The semantics is inspired by the notion of business object lifecyles (cf. [10]), in which execution of activities might update the state of a data object. For instance, activity "Assess Respondent Bank risk" requires the data object "Risk" to be in state *initial* in order to execute. Since object lifecyles are merely state transition systems, at any point of execution a data object can assume only one state. Thus, an activity that has two or more associations with the same data object but with different states, e.g. activity "Lookup Partner Banks" with the data object "Extra_valuation" in Fig. 1, is interpreted as a disjunction of such states. This data processing semantics along with control flow execution semantics of BPMN given in [11] are used to generate the behavioral model of the process for model checking grounded on the following atomic propositions:

- The predicate $\mathbf{state}(dataObject, stateValue)$ describes the fact that a data object assumes a certain state.
- The predicates $\mathbf{ready}(activity)$ and $\mathbf{executed}(activity)$ state that a certain activity is ready to be executed or has already been executed, respectively.

3.3 BPMN-Q

Based on BPMN, BPMN-Q [6] 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. 2(b). Such a path might match a sub-graph of a BPMN process — the highlighted part of the process in Fig. 2(a) is the matching part to the path edge of Fig. 2(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,

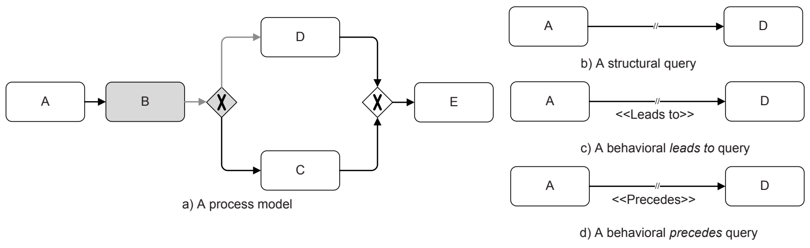


Fig. 2. BPMN-Q Path Edges

we type paths between two activities as being either `precedes` (cf. Fig. 2(d)) or `leads to` (cf. Fig. 2(c)) paths [5]. 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 Fig. 2(a), it is obvious that A `precedes` D is satisfied, while A `leads to` D is not.

Moreover, behavioral BPMN-Q queries are wrappers for PLTL expressions. 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. Thus, the mappings of the queries in Fig. 2(c) and Fig. 2(d) into PLTL are $G(\mathbf{executed}(A) \rightarrow F(\mathbf{ready}(D)))$ and $G(\mathbf{ready}(D) \rightarrow O(\mathbf{executed}(A)))$, respectively. The resulting expressions might then be checked against the process model's execution state space.

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. Then matching this query to the process in Fig. 2(a) would yield the whole model except activity *D*. Setting the $exclude$ property for behavioral paths affects the PLTL formula.

4 BPMN-Q for Data Aware Compliance Rules

In this section, we demonstrate how to express data aspects in compliance rules by extending the BPMN-Q language. Section 4.1 illustrates the extensions and introduces different kinds of data aware queries by using the aforementioned compliance rules as examples. We define the syntax for BPMN-Q in Section 4.2 and specify query semantics by mapping the queries into PLTL expressions in Section 4.3.

4.1 Examples for Data Aware Compliance Rules

R1: An account is opened only in case that risk is low. Data objects and data associations are used in BPMN-Q in the same way as in BPMN. BPMN-Q additionally introduces a new type of association edges called behavioral associations. This association represents an implicit association between a data object and an activity. This captures R1, which requires that the data object "Risk" must be in state low when the activity "Open Correspondent Account" is about to execute. Fig. 3 depicts the query for R1. A behavioral association edge is visualized with a double arrow head. Rules specifying solely data dependencies for a single activity are called *data rules*.

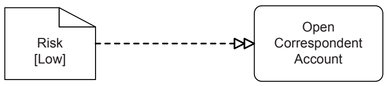
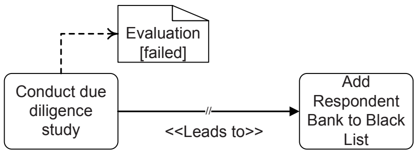
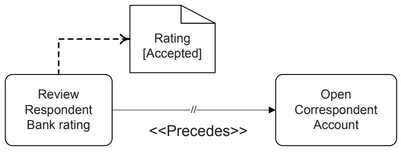


Fig. 3. Query for R1

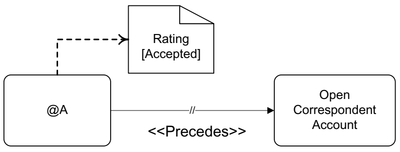
R2: The respondent bank must always be added to the black list in case its due diligence evaluation fails. R2 requires that once the due diligence evaluation fails as a result of executing activity "Conduct due diligence study", the process must proceed in a way that the respective bank is added to a black list. The BPMN-Q query representing this rule is shown in Fig. 4. We call this rule a `conditional leads to` rule. It is a refinement of the `leads to` introduced above.


Evaluation
[failed]
Conduct due diligence study
//
<<Leads to>>
Add Respondent Bank to Black List
Fig. 4. Query for R2

R3: Before opening an account, the respondent bank rating must be accepted. This rule might be modeled similarly to R1. In this case, we want to be sure that the state of the "Rating" data object will always be *accepted* when the activity "Open Correspondent Account" is about to execute. Another way to model R3 is shown in Fig. 5(a). This query requires that when the activity "Open Correspondent Account" is ready to execute, the "Rating" was *accepted* as a result of the execution of activity "Review Respondent Bank rating". Unlike the first case, the state of the data object *may* change in between. We call the latter query a `conditional precedes` rule.



(a) A conditional precedes relation



(b) A less strict form of 5(a)

Fig. 5. Different BPMN-Q queries to capture R3

Fig. 5(b) shows an even less strict variant of the query in Fig. 5(a). Here, focus is only on the data condition that must have held once *before* the execution of activity "Open Correspondent Account". Using the BPMN-Q variable activity, denoted as an activity with the label "@A", relieves the modeler from explicitly mentioning the activity that sets the "Rating" to *accepted*. Rule R2 could have been modeled in the same way.

R4: In case the respondent bank rating review is rejected, an account must never be opened. This rule is another way of stating a requirement similar to this of R3. When a certain condition holds, i.e. the "Rating" is *rejected*, it has to be ensured that the activity "Open Correspondent Account" will *never* be executed. The query in Fig. 6 captures this requirement. The variable activity "@A" with an association to the data object "Rating" with state *rejected* represents the data condition. Moreover, there is a `leads to` path from this activity to an end event with the *exclude* property set to the activity "Open Correspondent Account". That is interpreted as: activity "Open Correspondent Account" must never be executed from the point the data condition holds to the end of the process.

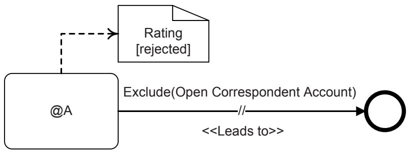


Fig. 6. Query for R4

4.2 Syntax of Data Aware BPMN-Q Queries

After we have introduced data aware BPMN-Q queries by exemplary compliance rules, we define their syntax formally. Therefore, the notions of a *process graph* and a *query graph* as introduced in [5] have to be extended with data related concepts. We introduce these extensions formally solely for the query graph, as they subsume the extensions needed for the process graph. We begin by postulating infinite sets of activities $\mathfrak{A}$, data objects $\mathfrak{D}$, and labels of data object states $\mathfrak{L}$.

Definition 1 (Query Graph). *A BPMN-Q query graph is a tuple* $QG = (\mathcal{A}_Q, \mathcal{E}_Q, \mathcal{D}_Q, \mathcal{P}, \mathcal{X}, \mathcal{C}, \mathcal{T}, \mathcal{L})$ *where:*

- $\mathcal{A}_Q \subset (\mathfrak{A} \cup \{@A\})$ *is the set of activities with* $@A$ *as the distinguished variable activity,* $\mathcal{E}_Q \subseteq \{e_S, e_E\}$ *is the set of events that might contain a dedicated start and a dedicated end event, and* $\mathcal{D}_Q \subset \mathfrak{D}$ *is the set of data objects.* $\mathcal{A}_Q, \mathcal{E}_Q$*, and* $\mathcal{D}_Q$ *are finite and disjoint sets.*
- $\mathcal{P} \subseteq (\{e_S\} \cup \mathcal{A}_Q) \times (\{e_E\} \cup \mathcal{A}_Q)$ *is the path relation.*
- $\mathcal{X} : \mathcal{P} \rightarrow \wp(\mathcal{A}_Q)$ *defines the exclude property for paths.*
- $\mathcal{C} \subseteq (\mathcal{D}_Q \times \mathcal{A}_Q) \cup (\mathcal{A}_Q \times \mathcal{D}_Q)$ *is the data access relation.*
- $\mathcal{T} : (\mathcal{P} \rightarrow \{leadsto, precedes, none\}) \cup (\mathcal{C} \rightarrow \{behavioral, none\})$ *assigns stereotypes to paths and data associations.*
- $\mathcal{L} : \mathcal{D}_Q \rightarrow \wp(\mathfrak{L})$ *assigns status labels to data objects.*

We see that a query graph might contain data objects that are accessed by data associations. The latter, in turn, might be of type *behavioral*, which captures an indirect data dependency as explained above. Moreover, a set of status labels is assigned to each data object. The labeling function $\mathcal{L}$, the set of data objects $\mathcal{D}_Q$, and the data access relation $\mathcal{C}$ are data related extensions that are applied for process graphs as well.

Definition 1 allows to define query graphs that are unconnected or show anomalies as, for instance, variable activities that are targets of a path. Therefore, we restrict the definition to *well-formed query graphs*. As a short-hand notation, we use $\mathcal{N}_Q = \mathcal{A}_Q \cup \mathcal{E}_Q$ for all nodes, $\mathcal{S}_Q = \{n_2 \in \mathcal{N}_Q |\ \nexists\, n_1 \in \mathcal{N}_Q\ [\ (n_1, n_2) \in \mathcal{P}\ ]\}$ for start nodes, and $\mathcal{T}_Q = \{n_1 \in \mathcal{N}_Q |\ \nexists\, n_2 \in \mathcal{N}_Q\ [\ (n_1, n_2) \in \mathcal{P}\ ]\}$ for end nodes.

Definition 2 (Well-Formed Query Graph). *A query graph* $QG = (\mathcal{A}_Q, \mathcal{E}_Q, \mathcal{D}_Q, \mathcal{P}, \mathcal{X}, \mathcal{C}, \mathcal{T}, \mathcal{L})$ *is well-formed, iff*

- $\forall\, n \in \mathcal{N}_Q\ [\exists\, s \in \mathcal{S}_Q, e \in \mathcal{T}_Q\ [s\mathcal{P}^* n\ \wedge\ n\mathcal{P}^* e\,]\,]$ *with* $\mathcal{P}^*$ *as the transitive reflexive closure of* $\mathcal{P}$*, i.e. activities and events are connected,*
- $\forall\, d \in \mathcal{D}_Q\ [\ \exists\, a \in \mathcal{A}_Q\ [\ (d, a) \in \mathcal{C} \vee (a, d) \in \mathcal{C}\ ]\ ]$*, i.e. data objects are accessed,*
- $\forall\, (n_1, n_2) \in \mathcal{P}\ [\ (n_2 \neq @A)\ \wedge\ (n_1 = @A \Rightarrow \exists\, d \in \mathcal{D}_Q\ [\ (n_1, d) \in \mathcal{C}\ ])\ ]$*, i.e. the variable activity is never target of a path and must have data access.*

We restrict our discussion to well-formed query graphs and use the term *query* as a short form for *query graph*. A query is called *compliance query*, if every path is of type `precedes` or `leads to`. Note that we do not consider paths of type `none` at this point. These paths are not applicable to specify compliance rules as BPMN-Q queries, as they specify structural requirements for the process model rather than behavioral requirements. Nevertheless, these queries are well-formed queries, which might be generated in order to explain violations of compliance rules. Depending on how data aspects are considered in the query, we distinguish *data queries*, *control flow queries*, and *conditional queries*. *Data queries* specify data constraints for solely one activity. In contrast, *control flow queries* are all BPMN-Q queries that do not consider any data dependencies. A *conditional query* combines data and control flow dependencies, such that a control flow dependency is required to hold under certain data conditions.

Definition 3 (Data Query). *A query* $Q = (\mathcal{A}_Q, \mathcal{E}_Q, \mathcal{D}_Q, \mathcal{P}, \mathcal{X}, \mathcal{C}, \mathcal{T}, \mathcal{L})$ *is called* data query*, iff* $|\mathcal{A}_Q| = 1$, $\mathcal{A}_Q \neq \{@A\}$*, and* $\mathcal{E}_Q = \emptyset$*, the query contains exactly one activity (not the variable activity), but no events.*

Definition 4 (Control Flow Query). *A query* $Q = (\mathcal{A}_Q, \mathcal{E}_Q, \mathcal{D}_Q, \mathcal{P}, \mathcal{X}, \mathcal{C}, \mathcal{T}, \mathcal{L})$ *is called* control flow query, *iff* $\mathcal{D}_Q = \emptyset$.

Definition 5 (Conditional Leads to / Precedes Query). *A query* $Q = (\mathcal{A}_Q, \mathcal{E}_Q, \mathcal{D}_Q, \mathcal{P}, \mathcal{X}, \mathcal{C}, \mathcal{T}, \mathcal{L})$ *is called* conditional query, *iff*

- $(|\mathcal{A}_Q| = 2) \vee ((|\mathcal{A}_Q| = 1) \wedge (\mathcal{E}_Q = \{e_E\}))$, *the query contains two activities or events, but no start event,*
- $(|\mathcal{P}| = 1) \wedge \forall (p_1, p_2) \in \mathcal{P} [\, p_1 \neq p_2 \,]$, *that are connected by a path,*
- $\forall\, d \in \mathcal{D}_Q [\, \exists\, (a_1, n) \in \mathcal{P} [\, (a_1, d) \in \mathcal{C} \,]\,]$, *all data objects are written by the node that is the origin of the path.*

A conditional query is called conditional leads to query, *iff* $\forall\, p \in \mathcal{P} [\, \mathcal{T}(p) = leadsto \,]$, *or* conditional precedes query, *iff* $\forall\, p \in \mathcal{P} [\, \mathcal{T}(p) = precedes \,]$.

4.3 Mapping Queries into PLTL

After we specified the syntax for BPMN-Q queries, this section introduces the mapping of a query into a PLTL formula in order to model check them against process models. This mapping is based on the aforementioned classification of BPMN-Q queries. We focus on the mapping of *data queries* and *conditional queries*, and refer to [5] for a mapping of *control flow queries*.

Mapping Data Queries. The mapping into PLTL is straightforward. A certain data condition must always hold at the time an activity is about to execute.

Definition 6 (PLTL for Data Query). *For a data query* $Q = (\mathcal{A}_Q, \mathcal{E}_Q, \mathcal{D}_Q, \mathcal{P}, \mathcal{X}, \mathcal{C}, \mathcal{T}, \mathcal{L})$*, the corresponding PLTL formula* P_Q *is defined as:* $P_Q = G(\mathbf{ready}(a) \rightarrow \bigwedge_{d \in \mathcal{D}_Q} P_d)$ *with* $a \in \mathcal{A}_Q,\ P_d = \bigvee_{s \in \mathcal{L}(d)} \mathbf{state}(d, s)$.

According to this definition, the mapping of the query in Fig. 3 into PLTL is $G(\mathbf{ready}(Open\ Correspondent\ Account) \rightarrow \mathbf{state}(Risk, low))$.

Mapping `Conditional Leads to` Queries. These queries can be distinguished into *presence* and *absence* queries, depending on whether the execution of an activity has to be ensured (presence) or prevented (absence). The query in Fig. 4 is an example for a *presence* query, whereas the query in Fig. 6 is an *absence* query.

A mapping of these conditional queries to PLTL is not straightforward. Considering the query in Fig. 4, a first attempt to map this query might result in $G(\mathbf{executed}(Conduct\ due\ diligence\ study) \wedge \mathbf{state}(Evaluation, failed) \rightarrow F(\mathbf{ready}(Add\ Respondent\ Bank\ to\ Black\ List)))$.

At the first glance, the formula captures the requirement. Whenever the activity "Conduct due diligence study" is executed and the bank evaluation failed, the respondent bank must be added to a black list. Referring to the process in Fig. 1, we see that this requirement is satisfied. However, model checking this formula against the process model tells that the model does *not* satisfy the formula. The reason is that the formula has not been specified properly. Imagine the execution scenario where "Conduct due diligence study" is executed for the first time and as a result the evaluation fails, i.e., the condition of the above mentioned formula holds. Next, "Lookup Partner Bank" is executed with the result to make an extra diligence study. In the second execution of the diligence study, the “Evaluation” is *passed*. From that point the process continues without adding the respondent bank to the black list. Thus, the rule is violated.

As a result, the aforementioned mapping cannot be applied. Instead, for this specific example, we need the model checker to record that the evaluation failed only when there is no chance to pass the evaluation in the future. We say that the data object state, and consequently the predicates, $\mathbf{state}(Evaluation, failed)$ and $\mathbf{state}(Evaluation, passed)$ are contradicting. We assume that we have the knowledge about these contradicting states before we start the process of rule mapping. The corrected PLTL formula is $G(\mathbf{executed}(Conduct\ due\ diligence\ study) \wedge \mathbf{state}(Evaluation, failed) \wedge G(\neg\ \mathbf{state}(Evaluation, passed)) \rightarrow F(\mathbf{ready}(Add\ Respondent\ Bank\ to\ Black\ List)))$.

Before we introduce the mapping of `conditional leads to` BPMN-Q queries into PLTL formulae, we introduce two auxiliary predicates that will be used in the mapping of all conditional queries.

Definition 7 (Full Data Condition Predicate). *For a set of data objects $\mathcal{D}_Q$ and a labelling function $\mathcal{L}$, the* full data condition *is a PLTL predicate defined as:* $PD_{(\mathcal{D}_Q,\mathcal{L})} = \bigwedge_{d\in\mathcal{D}_Q}(\bigvee_{s\in\mathcal{L}(d)} \mathbf{state}(d,s)) \wedge G(\bigwedge_{s_c\in\mathcal{L}_\mathcal{C}(d,s)} \neg\ \mathbf{state}(d,s_c))$.

Definition 8 (Variable Activity Condition Predicate). *For a node n, the* variable activity condition *is a PLTL predicate defined as:*

$$PV_{(n)} = \begin{cases} \mathbf{true} & \textit{iff} \quad n = @A \\ \mathbf{executed}(n) & \textit{else} \end{cases}.$$

The full data condition requires all data objects to be in one state out of a set of states. In addition, it prohibits contradicting data object states. As mentioned above, we assume the knowledge about contradicting states to be part of the business context. This is formalized as a function $\mathcal{L}_\mathcal{C} : \mathfrak{D} \times \mathfrak{L} \rightarrow \wp(\mathfrak{L})$ that returns all contradicting states for a pair of a data object and a state. The second auxiliary predicate, namely the variable activity condition, requires the execution of an activity. In case of the variable activity “@A” this predicate is simply true.

Definition 9 (PLTL for Conditional Leads To Query). *For a conditional leads to query $Q = (\mathcal{A}_Q, \mathcal{E}_Q, \mathcal{D}_Q, \mathcal{P}, \mathcal{X}, \mathcal{C}, \mathcal{T}, \mathcal{L})$, the corresponding PLTL formula P_Q is defined as: $P_Q = G((PV_{(src)} \wedge PD_{(\mathcal{D}_Q,\mathcal{L})}) \rightarrow P_{tar})$ with $(src, tar) = p \in \mathcal{P}$,*

$$P_{tar} = \begin{cases} \bigwedge_{a\in\mathcal{X}(p)}(\neg\ \mathbf{executed}(a))U(\mathbf{ready}(tar)) & \textit{iff} \quad p \in dom(\mathcal{X}) \\ F(\mathbf{ready}(tar)) & \textit{else} \end{cases}.$$

We distinguish *presence* and *absence* queries by the definition of the predicate P_{tar}, which is defined based on whether the exclude property is set for the path.

Mapping `Conditional Precedes` Queries. Similarly, we can derive the PLTL formula for a `conditional precedes` query. For instance, consider the rule in Fig. 5(a). Informally the rule states that at the point activity “Open Correspondent Account” is ready to execute, i.e. $\mathbf{ready}(Open\ Correspondent\ Account)$

holds, there was a previous state in which activity “Review Respondent Bank Rating” was executed and the “Rating” was *accepted.* In other words, the predicates $\mathbf{executed}(Review\ Respondent\ Bank\ Rating)$ and $\mathbf{state}(Rating, accepted)$ were true before. Following the argumentation on the change of data states given above, we need to be sure that the state of the data object “Rating” did not change to a contradicting state. Therefore, the PLTL formula to capture the query in Fig. 5(a) is defined as $G(\mathbf{ready}(Open\ Correspondent\ Account) \rightarrow O(\mathbf{state}(Rating, accepted) \wedge \mathbf{executed}(Review\ Respondent\ Bank\ Rating) \wedge G(\neg\, \mathbf{state}(Rating, rejected))))$. For the rule in Fig. 5(b), the mapping is quite similar except the treatment of the variable activity (according to Definition 8).

While the former queries require the *presence* of an execution of a certain activity, *absence* queries can be mapped similarly. They require the absence of an execution of certain activities between two activities taking the data conditions into account. The `conditional precedes` query is mapped to a PLTL formula as follows.

Definition 10 (PLTL for Conditional Precedes Query). *For a conditional precedes query* $Q = (\mathcal{A}_Q, \mathcal{E}_Q, \mathcal{D}_Q, \mathcal{P}, \mathcal{X}, \mathcal{C}, \mathcal{T}, \mathcal{L})$*, the corresponding PLTL formula* P_Q *is defined as:* $P_Q = G(\mathbf{ready}(tar) \rightarrow P_{src})$ *with* $(src, tar) = p \in \mathcal{P}$,

$$P_{src} = \begin{cases} \bigwedge_{a \in \mathcal{X}(p)}(\neg\, \mathbf{executed}(a)) S (PV_{(src)} \ \wedge \ PD_{(\mathcal{D}_Q, \mathcal{L})}) & \textit{iff} \quad p \in dom(\mathcal{X}) \\ O(PV_{(src)} \ \wedge \ PD_{(\mathcal{D}_Q, \mathcal{L})}) & \textit{else} \end{cases}.$$

Predicate P_{src} reflects the difference between presence/absence queries.

5 Explanation of Violation

When the rules R1 to R4 introduced in Section 4.1 are checked against the process model in Fig. 1, we get the following result. R2 is satisfied by the model, whereas R1, R3, and R4 are violated. That, in turn, leads to the question *why* a certain rule is violated.

We would like to answer this question by showing execution scenarios that violate the rule directly in the process model. One could think of using the counterexample returned by the model checker when the rule is not satisfied. However, there are two problems with that approach. Firstly, the counterexample is given as a trace of states that violate the rule. Therefore, we need to translate it back to the level of the model structure. Secondly, counterexamples given by a model checker are not exhaustive. That is, they do not show every possible violation to the rule, rather, they show the first met violation.

In order to tackle this problem we use a two-step approach. First, we extract the data conditions under which the violation occurred. Second, this violation is visualized on the process model level. For the first step we use Temporal Logic Querying (TLQ) [8]. For the purpose of visualizing the violations based on the results of the first step, we use BPMN-Q to formulate the so-called anti-pattern queries.

We briefly introduce TLQ in Section 5.1. Subsequently, Sections 5.2 to 5.4 demonstrate the application of this two-step approach for each category of queries.

5.1 Temporal Logic Querying

Temporal Logic Querying (TLQ) was first introduced by Chen in [8] in order to find software model invariants and gain understanding about the behavior of the model. So, model checking can be seen as a subproblem of temporal logic querying. In model checking, we issue Boolean queries only. In the general case of TLQ, we ask a TLQ solver (e.g. [12]) to find a propositional formula that would make our query hold true when seen as a temporal logic formula. The question mark '?' is used in a temporal logic query as a placeholder for such a propositional formula, which might also be limited to certain predicates. For instance, the query $G(?\{p, q\})$ looks for invariants that are based on the predicates p and q.

5.2 Explanation of Data Rules Violations

A data compliance query (cf. Definition 6) is violated if there is a state in which the respective activity (a) is ready to execute ($\mathbf{ready}(a)$ holds), but the data condition is not fulfilled. This occurs in case the data objects that are relevant to the compliance rule, assume states other than specified in the rule. We issue the temporal logic (TL) query $G(\mathbf{ready}(a) \rightarrow \mathbf{state}(?_{dob}, ?_{st}))$ to discover the violation. Thus, we are asking about the data states that are set at the point $\mathbf{ready}(a)$ holds. Here, the symbol $?_{dob}$ is a placeholder for the data objects that were mentioned in the compliance query; while $?_{st}$ is the placeholder for their respective states. In general, such a query delivers the different assignments of data object states that make the statement hold. The general form of the query result is $\bigwedge_{d \in \mathcal{D}_Q}(\bigvee_{s \in \mathcal{L}(d)} \mathbf{state}(d, s))$.

For the case of rule R1, the result of the TL query $G(\mathbf{ready}(\mathit{Open\ Correspondent\ Account}) \rightarrow \mathbf{state}(\mathit{Risk}, ?_{st}))$ is $\mathbf{state}(\mathit{Risk}, \mathit{low}) \vee \mathbf{state}(\mathit{Risk}, \mathit{high})$. Thus, there is a possible execution trace where the state of data object "Risk" is set to high and remains in this state until activity "Open Correspondent Account" is ready to execute. In order to visualize this execution trace on the process model level, we need to find a path from some activity that sets the state of "Risk" to high and another path from this activity to the activity "Open Correspondent Account". That is captured by the *anti-pattern*, which is illustrated in Fig. 7. Such anti-pattern matches the process part that causes the violation of the original compliance rule.

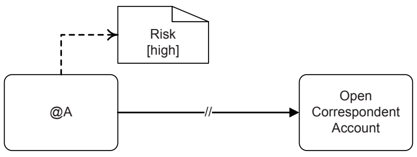


Fig. 7. Anti-pattern for R1

5.3 Explanation of `Conditional Leads to` Violations

Derivation of anti-patterns for `conditional leads to` compliance rules is straight-forward. Such a rule is violated when there is at least one execution trace in which the source activity is executed and the data condition holds, and the execution continues to the end of the process without executing the target activity. On the other hand, a `conditional absence leads to` rule is violated, if the activity required to be absent is executed in at least one possible execution trace. Rule R4 is an example for the latter kind of compliance

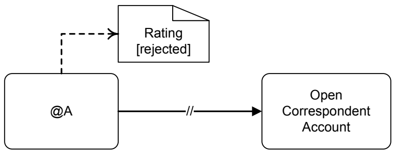


Fig. 8. Anti-pattern for R4

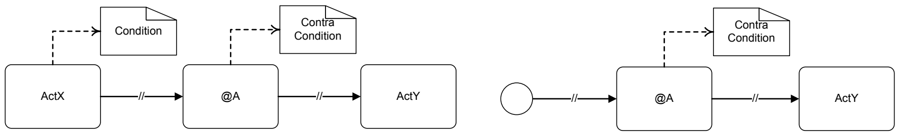
rule. The corresponding anti-pattern query is shown in Fig. 8. Here, the path edge connects a variable activity at which the data condition holds to the activity "Open Correspondent Account". The type of path is $none$. Thus, we look for a structural match.

5.4 Explanation of `Conditional Precedes` Rules Violations

Explanation of violations of this type of rules is more complex than for the case of `conditional leads to` rules. According to Definition 10, a violation might be traced back to the following reasons.

1. $PV_{src} \wedge \bigwedge_{d \in \mathcal{D}_Q} (\bigvee_{s \in \mathcal{L}(d)} \mathbf{state}(d, s))$ did not occur before activity *tar* is reached. That, in turn, might be traced back to one of the following reasons:
   (a) Either activity *src* was not executed at all, or
   (b) the data condition $\bigwedge_{d \in \mathcal{D}_Q} (\bigvee_{s \in \mathcal{L}(d)} \mathbf{state}(d, s))$ was not fulfilled.
2. $G(\bigwedge_{s_c \in \mathcal{L}_\mathcal{C}(d,s)} \neg\, \mathbf{state}(d, s_c))$ was not fulfilled, i.e., the state of the data object had been altered to a contradicting data state before activity tar was ready to execute.

In order to identify the exact reason for the violation, we have to issue a sequence of TL queries. Depending on the results, anti-pattern queries are derived. First, we check whether the predicates for the execution of the source activity and the data condition hold when the target activity is ready to execute, i.e. $G(\mathbf{ready}(tar) \rightarrow O(PV_{src} \wedge \bigwedge_{d \in \mathcal{D}_Q} (\bigvee_{s \in \mathcal{L}(d)} \mathbf{state}(d, s))))$. Note that, again, we use the variable activity predicate PV_{src} (Definition 8) resolving to $\mathbf{executed}(src)$ for ordinary activities and to $true$ for the variable activity "@A". If this query returns a positive result, we know that violation occurred owing to the second of the aforementioned reasons (2). That is, the states of the data objects are altered, such that $G(\bigwedge_{s_c \in \mathcal{L}_\mathcal{C}(d,s)} \neg\, \mathbf{state}(d, s_c))$ is not satisfied. The corresponding anti-pattern query is sketched in Fig. 9(a).



(a) Occurrence of contradicting data states (b) Non occurrence of data condition

Fig. 9. Anti-pattern queries for conditional precedes

On the other hand, if the result is negative; either the source activity has not been executed at all before the target activity (reason 1a) or the data condition did not hold (reason 1b). In order to decide on a reason, we issue a TL query that checks, whether the source activity (*src*) is always executed before the target activity (*tar*), i.e. $G(\mathbf{ready}(tar) \rightarrow O(PV_{(src)}))$. If this query is not satisfied, then we know that the target activity (*tar*) might be executed without executing the source activity (*src*) before. Thus, the violation can be identified by finding paths from the start of the process to the target activity without executing the source activity (which is captured by the exclude property). On the other hand, if this query is satisfied; we know that in *some* cases the data condition does not hold. To identify the data states that violate the data condition, we query the states of data objects that result from an execution of the source activity (*src*), as a TL query $G(\mathbf{ready}(tar) \rightarrow O(PV_{(src)} \wedge \mathbf{state}(?_{dob}, ?_{states})))$. For each resulting data state, a query as in Fig. 9(b) shows the violation in the process model.

Finally, the case of `conditional absence` *`precedes`* compliance rules adds one more potential reason for violation. That is, the excluded activities might have been executed. Again, the violation can be captured by issuing a query where there is a path from the start of the process to the activity that should be absent, and another path from this activity to the target activity. With respect to our examples, the anti-pattern query for rule R3 is illustrated in Fig. 10. This query matches the whole process model, such that activity "Review Respondent Bank Rating" is executed, the "Rating" is $rejected$, whereas activity "Open Correspondent Account" might be executed.

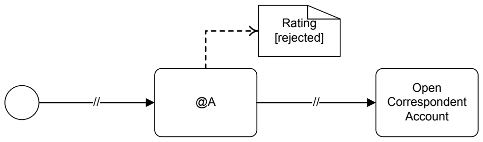


Fig. 10. Anti-pattern for R3

6 Implementation

Our approach has been implemented within the BPMN-Q query processor engine. The implementation covers mapping of the discussed rules into corresponding PLTL formulas as shown earlier. To prepare the investigated process models for model checking, the mapping proposed in [11,9] is used to generate the behavioral model.

For our work, we were not able to use existing temporal logic query solvers [12,13] as they support only CTL based queries. However, according to [14] it is possible to implement a temporal logic query solver by using a model checker and issuing all possible 2^{2^n} combinations, where n is a finite set of predicates, and tabulating the result of each combination. In our case, we adopted this approach in an even simplified form as we know for data states that they are mutually exclusive, i.e., a data object can have only one state at a time [9]. The implementation of this special case TL query solver is an integral part of the BPMN-Q query processor.

7 Related Work

There has been a large body of research interested in compliance checking on business process models. We focus on work done regarding execution ordering between activities on a business process. In this regard, we can divide work done on compliance into two areas, namely *compliance by design* and *post-design compliance checking*.

Compliance by design takes compliance rules as input for the design of new process models. Work in [15,16,17,18] shows how compliance requirements are enforced in the design process of new business processes. 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.

Post-design compliance checking, in turn, targets checking for compliance for existing process models. Thus, it separates the modeling phase of a process model from the checking phase. Our approach belongs to this category. Similar approaches [19,20] also employ model checking to verify that process models satisfy the compliance rules. Although some of these approaches are able to express what we call conditional rules, it remains open how these approaches can be applied to express so-called data flow rules. Taking business contracts as a source for compliance rules, deontic logic was employed as a formalism to express these requirements in [21,22,23]. It is possible to express the notion of obligation and permission and prohibition. Thus, it is possible to express $alternative$ actions to be taken when a primary one is not done. However, the data perspective is largely neglected. Further on, work in [24,10] addressed the consistency between business process models and lifecycles of business objects processed in these models. Yet, explanation of the points of deviation and their representation has not been addressed. A recent approach to measure the compliance distance between a process model and a rule was introduced in [25]. This approach enables measuring the $degree$ of compliance on the scale from 0 to 1. Again, data aspects are not considered.

Another field of related work deals with resolution of compliance violations. In [26] 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 has to be tackled in future work.

Explanation of violations was also addressed in the area of workflow verification [27] as well as service orchestration [28]. In both approaches, the explanation was a translation of the output of the verification tools. Thus, it might be the case that some violation scenarios were not discovered.

The unique features of our approach are 1) the possibility of identifying process models subject to checking by means of queries and 2) giving explanations of possible violations on the process model level.

8 Conclusion

In this paper, we discussed an approach to model the so-called *data aware compliance rules*. These rules were realized by extending BPMN-Q. Including data aspects increased the expressiveness of the language. Nevertheless, formalizing these rules (queries), by mapping into PLTL, is not straightforward. Extra information, e.g. the notion of contradicting states, must be present. To explain violations, we applied temporal logic querying (TLQ). We demonstrated how feedback can be given — based on so-called anti-pattern queries that are derived automatically. To the best of our knowledge, we are the first to apply TLQ in the area of business process management.

The ability to provide explanations *why* a certain compliance rule is not satisfied has to be seen as a major step towards real-world applicability. Knowing just that a process violates a certain compliance rule is of limited use for common business scenarios. Owing to the intrinsic complexity of these scenarios, feedback on violations is crucial.

In future, we will investigate approaches for (semi) automated resolution of violations. In that case, other formalism has to be used as resolution of violation implies changes to the structure of the process models.

References

1. Weske, M.: Business Process Management. Springer, Heidelberg (2007)
2. United States Senate and House of Representatives in Congress: Sarbanes-Oxley Act of 2002. Public Law 107-204 (116 Statute 745) (2002)
3. Kharbili, M.E., de Medeiros, A.K.A., Stein, S., van der Aalst, W.: Business Process Compliance Checking: Current State and Future Challenges. In: MobIS, GI. LNI, vol. P-141, pp. 107–113 (2008)
4. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (1999)
5. Awad, A., Decker, G., Weske, M.: Efficient compliance checking using bpmn-q and temporal logic. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 326–341. Springer, Heidelberg (2008)
6. Awad, A.: BPMN-Q: A Language to Query Business Processes. In: EMISA, GI. LNI, vol. P-119, pp. 115–128 (2007)
7. Zuck, L.: Past Temporal Logic. PhD thesis, Weizmann Intitute, Israel (1986)
8. Chan, W.: Temporal-logic queries. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 450–463. Springer, Heidelberg (2000)
9. Awad, A., Decker, G., Lohmann, N.: Diagnosing and Repairing Data Anomalies in Process Models. In: 5th International Workshop on Business Process Design. LNBIP. Springer, Heidelberg (to appear, 2009)
10. Küster, J.M., Ryndina, K., Gall, H.: Generation of Business Process Models for Object Life Cycle Compliance. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 165–181. Springer, Heidelberg (2007)
11. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Inf. Softw. Technol. 50, 1281–1294 (2008)
12. Chechik, M., Gurfinkel, A.: TLQSolver: A temporal logic query checker. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 210–214. Springer, Heidelberg (2003)
13. Gurfinkel, A., Chechik, M., Devereux, B.: Temporal logic query checking: A tool for model exploration. IEEE Trans. Softw. Eng. 29, 898–914 (2003)
14. Bruns, G., Godefroid, P.: Temporal logic query checking. In: LICS, p. 409. IEEE Computer Society, Los Alamitos (2001)
15. Lu, R., Sadiq, S.W., Governatori, G.: Compliance aware business process design. In: ter Hofstede, A.H.M., Benatallah, B., Paik, H.-Y. (eds.) BPM Workshops 2007. LNCS, vol. 4928, pp. 120–131. Springer, Heidelberg (2008)
16. Goedertier, S., Vanthienen, J.: Designing Compliant Business Processes from Obligations and Permissions. In: Eder, J., Dustdar, S. (eds.) BPM Workshops 2006. LNCS, vol. 4103, pp. 5–14. Springer, Heidelberg (2006)
17. Goedertier, S., Vanthienen, J.: Compliant and flexible business processes with business rules. In: BPMDS. CEUR Workshop Proceedings, CEUR-WS.org, vol. 236 (2006)
18. Milosevic, Z., Sadiq, S.W., Orlowska, M.E.: Translating business contract into compliant business processes. In: EDOC, pp. 211–220. IEEE Computer Society, Los Alamitos (2006)
19. Yu, J., Manh, T.P., Han, J., Jin, Y., Han, Y., Wang, J.: Pattern based property specification and verification for service composition. In: Aberer, K., Peng, Z., Rundensteiner, E.A., Zhang, Y., Li, X. (eds.) WISE 2006. LNCS, vol. 4255, pp. 156–168. Springer, Heidelberg (2006)
20. Lui, Y., Müller, S., Xu, K.: A static compliance-checking framework for business process models. IBM Syst. J. 46, 335–362 (2007)

21. Governatori, G., Milosevic, Z.: Dealing with contract violations: formalism and domain specific language. In: EDOC, pp. 46–57. IEEE Computer Society, Los Alamitos (2005)
22. Governatori, G., Milosevic, Z., Sadiq, S.: Compliance checking between business processes and business contracts. In: EDOC, pp. 221–232. IEEE Computer Society, Los Alamitos (2006)
23. Sadiq, S.W., Governatori, G., Namiri, K.: Modeling control objectives for business process compliance. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 149–164. Springer, Heidelberg (2007)
24. Ryndina, K., Küster, J.M., Gall, H.C.: Consistency of Business Process Models and Object Life Cycles. In: Kühne, T. (ed.) MoDELS 2006. LNCS, vol. 4364, pp. 80–90. Springer, Heidelberg (2007)
25. Lu, R., Sadiq, S., Governatori, G.: Measurement of Compliance Distance in Business Processes. Inf. Sys. Manag. 25, 344–355 (2008)
26. Ghose, A., Koliadis, G.: Auditing business process compliance. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 169–180. Springer, Heidelberg (2007)
27. Flender, C., Freytag, T.: Visualizing the soundness of workflow nets. In: Algorithms and Tools for Petri Nets (AWPN 2006), University of Hamburg, Germany, Department Informatics Report 267, pp. 47–52 (2006)
28. Schroeder, A., Mayer, P.: Verifying interaction protocol compliance of service orchestrations. In: Bouguettaya, A., Krueger, I., Margaria, T. (eds.) ICSOC 2008. LNCS, vol. 5364, pp. 545–550. Springer, Heidelberg (2008)