

# Efficient Compliance Checking Using BPMN-Q and Temporal Logic

Ahmed Awad, Gero Decker, Mathias Weske

Business Process Technology Group  
Hasso-Plattner-Institute, University of Potsdam, Germany  
{ahmed.awad,gero.decker,weske}@hpi.uni-potsdam.de

**Abstract.** Compliance rules describe regulations, policies and quality constraints business processes must adhere to. Given the large number of rules and their frequency of change, manual compliance checking can become a time-consuming task. Automated compliance checking of process activities and their ordering is an alternative whenever business processes and compliance rules are described in a formal way. This paper introduces an approach for automated compliance checking. Compliance rules are translated into temporal logic formulae that serve as input to model checkers which in turn verify whether a process model satisfies the requested compliance rule. To address the problem of state-space explosion we employ a set of reduction rules. The approach is prototypically realized and evaluated.

## 1 Introduction

Business processes and their explicit representation in business process models are important assets to understand how companies work. 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.

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, e.g. reporting financial transactions to an external entity. The obligation of adhering to rules ranges from gaining competitive advantage to employing strategies that protect businesses from failure, e.g. Basel II (<http://www.Basel-II.info>) in the field of risk assessment in the banking sector; other regulations come as quality standards like ISO 9000. Regulations might also come with legal regulations like the Sarbanes-Oxley Act of 2002 [1]. Violation could lead to penalties, scandals and loss of business reputation.

The changing nature of rules (e.g. due to changes in policies) calls for checking business processes each time a rule is added or changed. As a result, organizations need to hire compliance experts auditing their process models. In the case of manual auditing, a considerable amount of time is consumed in identifying the set of process models affected by each rule before revising them for compliance; something that may lead to failure to meet the deadline for declaring compliance.

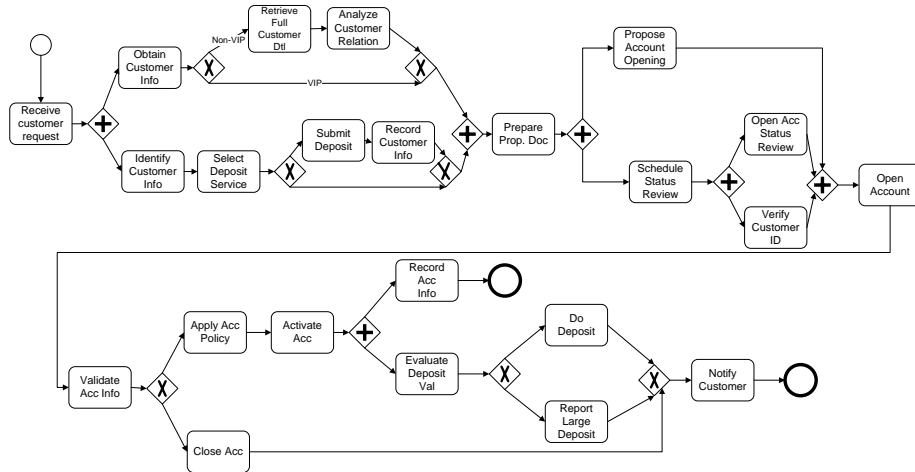


Fig. 1. Banking business process model adapted from [18]

The contribution of this paper is twofold. First, it presents an automated approach for checking compliance of business process models regarding ordering constraints for activities. Automation is achieved by expressing rules as queries in a visual language we developed earlier [2]. These queries determine the set of process models that are candidates for further compliance checking. We assume that candidate process models contain all activities mentioned in the rule. If this is not the case i.e. the process model contains only a subset of the activities in the rule, we simply know that the process model is no compliant. The hard part is to be sure that candidate process models do satisfy the rule. This activity is already very time consuming in manual auditing. Efficiently determining whether process models comply to the rules using model checking is the second contribution. Our approach is not limited to processes defined in BPMN, it can be applied to any graph based process definition language as described below.

The remainder of this paper is structured as follows. In Section 2 we discuss a scenario in the banking business and derive a set of rules. Section 3 discusses how we adapted BPMN-Q to express compliance rules as queries. Details of applying model checking to formally verify compliance is given in Section 4. Related work is reported in Section 5, before we conclude the paper in Section 6.

## 2 Compliance Example

In this section, we introduce a business process from the banking sector. It will serve as example throughout the paper. In the financial sector, banks are obliged to conform to several compliance rules. Consider the process model in Figure 1 (expressed in BPMN notation) for opening a bank account.

The process starts with "Receive customer request" to open an account. Customer information is obtained from the request and identified. In case of a non-VIP customer, her detailed information is retrieved and its relation to the

bank is analyzed. If the customer selects to open a deposit account, an extra deposit form must be submitted and the customer’s information is recorded. With all previous steps completed, a proposal document is prepared by a bank’s employee for further analysis. A proposal’s status review is scheduled along with the proposal of opening an account. Later on, the customer’s identity is verified and the status of the account is reviewed. An account is opened at that point followed by a validation of account information. If validation fails the account is closed. With valid account information, an account policy is applied before the account is activated. Finally the account information is recorded and the account is ready for transactions. Large deposit transactions (“Evaluate Deposit Val”) are reported to a central bank to track the possibility of money laundering. At the end of the process the customer is notified.

The process has to comply with several rules. Among them are rules to prevent money laundering. The rules demand that an account is not opened until certain checks have been completed. We selected the following two rules to guide the discussion in this paper.

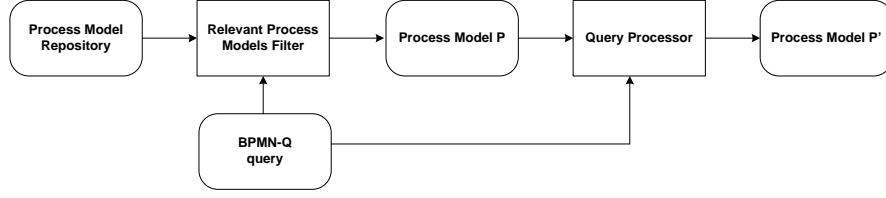
- Rule 1: Before opening an account, customer information must be obtained and verified. The rule delays the activity ”open account” until information about the customer has been verified e.g. checking absence from black lists containing undesirable customers. Violation of this rule may lead to opening an account for individuals that are known to be on the black list of e.g. the central bank.
- Rule 2: Whenever a customer requests to open a deposit account, customer information must be recorded before opening the account. Information must be recorded to ease future tracking of their transactions and identifying them in case they run large deposit transactions.

### 3 Declarative Representation of Compliance Rules

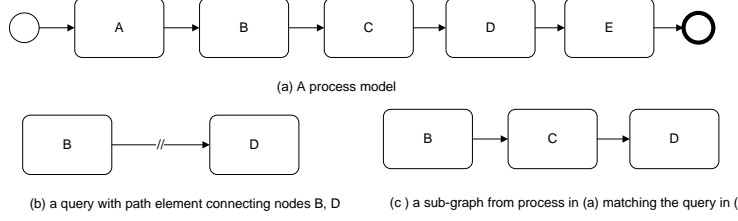
As mentioned in Section 1, the first contribution of this paper is the automated discovery of process models that are relevant to a given rule (see Definition 3). To determine these relevant process models, we express rules concerned with ordering of activities as queries in BPMN-Q. Before we go further with the details of using BPMN-Q to express compliance rules, we briefly introduce BPMN-Q. Next, we discuss how to adapt it for compliance checking.

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. Figure 2 sketches an overview of the steps of processing a query. In addition to the sequence flow edges in BPMN, BPMN-Q introduces the concept of path (see edge in Figure 3 (b)). When matching a process graph (like the one in Figure 3 (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 are contained (Figure 3 (c)).

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].



**Fig. 2.** Overview of query processing in BPMN-Q



**Fig. 3.** Example of a BPMN-Q query

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

- $\mathcal{N}$  is a finite set of nodes that is partitioned into the set of activities  $\mathcal{A}$ , the set of events  $\mathcal{E}$ , and the set of gateways  $\mathcal{G}$
- The set of events  $\mathcal{E}$  can be further partitioned into:
  - Start events  $\mathcal{E}^s$  i.e. nodes with no incoming edges.
  - Intermediate events  $\mathcal{E}^i$ .
  - End events  $\mathcal{E}^e$  i.e. nodes with no outgoing edges.
- $\mathcal{F} \subseteq (\mathcal{N} \setminus \mathcal{E}^e) \times (\mathcal{N} \setminus \mathcal{E}^s)$  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  $QG = (\mathcal{N}Q, \mathcal{A}Q, \mathcal{E}Q, \mathcal{G}Q, \mathcal{S}, \mathcal{PA})$  where

- $\mathcal{N}Q$  is a finite set of nodes that is partitioned into the set of activities  $\mathcal{A}Q$ , the set of events  $\mathcal{E}Q$ , and the set of gateways  $\mathcal{G}Q$
- $\mathcal{S} \subseteq \mathcal{N}Q \times \mathcal{N}Q$  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.

**Definition 3.** A process graph  $PG = (\mathcal{N}, \mathcal{A}, \mathcal{E}, \mathcal{G}, \mathcal{F})$  is relevant to query graph  $QG = (\mathcal{N}Q, \mathcal{A}Q, \mathcal{E}Q, \mathcal{G}Q, \mathcal{S}, \mathcal{PA})$  iff  $\mathcal{A} \supseteq \mathcal{A}Q$

To address the execution ordering between activities, a process graph is divided into a set of execution paths. An execution path is a sequence of nodes starting from one of the process start node(s) and ending at one of its end node(s).

**Definition 4.** An execution path  $exp$  is a sequence of nodes  $(n_0, \dots, n_k)$  where  $n_0, \dots, n_k \in \mathcal{N}$ ,  $n_0 \in \mathcal{E}^s$  and  $n_k \in \mathcal{E}^e$ .  $EXP$  is the set of all execution paths in a given process graph.

We can determine the execution order between two nodes a, b in a process graph with respect to an execution path  $exp$  by finding the precedence between the *first occurrence* of node a and occurrence of node b. We emphasize on the *first occurrence* of node a because a might appear more than once in case the execution path includes loops. We need to be sure that a executed at least once before the execution of b.

**Definition 5.** An execution ordering relation between nodes on an execution path  $exp$  is defined as  $<_{exp} = \{(n', n'') \in \mathcal{N} \times \mathcal{N} : n' \in exp \wedge n'' \in exp \wedge \exists i, j (n' = n_i \wedge n'' = n_j \wedge i < j \wedge \nexists j' (n'' = n_{j'} \wedge j' < i))\}$ , where  $n \in exp$  means that the node  $n$  resides on the execution path  $exp$ .

The evaluation of path edge as shown in Figure 3 conforms to the following definition.

**Definition 6.** A function subgraph  $(a, b, p_i) := PSG'(N', E')$ , where  $p_i$  is a process graph and  $a, b \in \mathcal{N}_i$ , constructs the process sub-graph of  $p_i$  where:

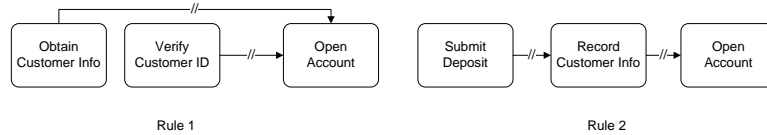
- $N' = \{x : \forall exp_i \in EXP_i ( a \in exp_i \wedge b \in exp_i \wedge x \in exp_i \wedge ( x = a \vee x = b \vee ( a <_{exp_i} x \wedge x <_{exp_i} b ) ) ) \}$
- $\forall x, y \in N' ( (x, y) \in \mathcal{F} \rightarrow (x, y) \in E' )$

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 7.

**Definition 7.** A process graph  $PG = (\mathcal{N}, \mathcal{A}, \mathcal{E}, \mathcal{G}, \mathcal{F})$  matches a query graph  $QG = (\mathcal{N}Q, \mathcal{A}Q, \mathcal{E}Q, \mathcal{G}Q, \mathcal{S}, \mathcal{P}A)$  iff:

- $\mathcal{N}Q \subseteq \mathcal{N}$ .
- $\mathcal{S} \subseteq \mathcal{F}$ .
- $\forall (n, m) \in \mathcal{P}A (subgraph(n, m, p) \neq \emptyset)$ .

To express rules as queries, we add an activity node in the query for each activity mentioned in the rule. To express the ordering relationship between two activities, a path element connects a source node in the query to a destination node. Figure 4 shows how the rules from Section 2 can be visually represented.

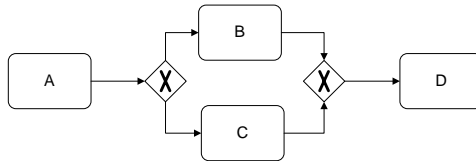


**Fig. 4.** Queries capturing rules

If BPMN-Q does not find a match i.e. it fails to find an execution path from *Record Customer Info* to *Open Account* then we are sure that the answer to rule 2 is “NO”. On the other hand if the matching succeeds and a path is found, we cannot be sure that this execution path is activated in all possible execution

scenarios. That is because the path element does not consider the semantics of control nodes in execution paths. We record this as a limitation that we will resolve in this paper.

Another limitation is the ability to express the direction of the execution dependency between activities. For instance, determining the order of execution between two activities A and B in Figure 5 by finding path from A to B, are we interested in being sure that every execution of activity A will *lead to* execution of Activity B, or on the other hand each time activity B is executed it must have been *preceded* by an execution of activity A. The two situations are different. In the first one we state a constraint over the future states of a process execution while in the second one we state this constraint over its past execution states. From this simple process fragment, we can see that activity B is preceded by



**Fig. 5.** A fragment of a process model to show difference between leads to and precedes concepts. A *precedes* B, but A does not *lead to* B

activity A but activity A does not lead to activity B.

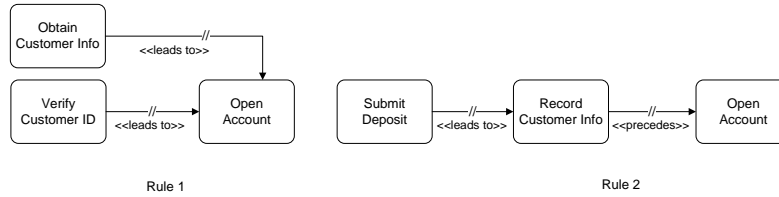
The two concepts of *precedes* and *leads to* are not distinguishable in queries. To overcome these limitations we decided to:

- Extend BPMN-Q with *precedes* and *leads to* qualifiers to solve the second limitation.
- Use Model checking as a formal approach to verify constraints against process sub-graphs to solve the first limitation.

To visually differentiate between the *precedes* and *leads to* semantics we simply added them under the arc representing the path operator like  $\llprecedes\gg$  and  $\llleads\ to\gg$  respectively in a way similar to the stereotypical extension in UML. Now queries from Figure 4 will look as in Figure 6. The formalism behind these two concepts is given within the context of this section along with necessary supporting definitions. To say that node A in a process graph leads to (see Definition 8) node B, we just need to be sure that every execution path going through A also goes through B. On the other hand node A precedes (Definition 9) node B only if all execution paths gone through B have gone through node A before.

**Definition 8.** Node  $m$  leads to node  $n$  iff  $\forall exp \in EXP(m \in exp \Rightarrow n \in exp \wedge m <_{exp} n)$

**Definition 9.** Node  $m$  precedes node  $n$  iff  $\forall exp \in EXP(n \in exp \Rightarrow m \in exp \wedge m <_{exp} n)$



**Fig. 6.** Queries refined

We can read queries as follows:

- Rule 1: The execution of "Obtain Customer Info" and "verify customer ID" always leads to execution of "Open Account" i.e. *leads to (Obtain Customer Info, Open Account) and leads to (Verify Customer ID, Open Account)*.
- Rule 2: The execution of submit deposit leads to recording customer info, which must precede opening an account i.e. *leads to (Submit Deposit, Record Customer Info) and precedes (Record Customer Info, Open Account)*.

The effects of these extensions are more than just a visual differentiation between the *leads to* and *precedes* concepts. One further effect is the temporal expression generated from each rule, shown in the next section. Another effect is on the query itself. In case of *precedes* paths we add a start event to the query graph (in case the query does not already have one) and a path operator between the start event and the destination of the *precedes* operator. This path is added to allow the query to match all possible paths from the beginning of the process to the destination activity of the *precedes* operator in order to give the model checker the possibility to find violations (if any).

## 4 Efficient Analysis using Temporal Logic

This section discusses how compliance checking on BPMN process models can be carried out. The compliance rules are formulated as BPMN-Q queries. Our approach can be divided into the following steps, which are also illustrated in Figure 7.

1. **Retrieval of BPMN sub-graphs.** A query processor takes a BPMN-Q query as input and retrieves a number of BPMN sub-graphs from a BPMN process model repository. Only those process models are considered that structurally match the BPMN-Q query.
2. **Graph reduction.** The sub-graphs are reduced, mainly removing activities and gateways that are not relevant to the query.
3. **Petri net generation.** The reduced sub-graphs are translated into Petri nets.
4. **State space generation.** The Petri nets are checked for boundedness and the reachability graph is calculated.
5. **Generation of temporal logic formulae.** The BPMN-Q query is translated into temporal logic formulae.

- Model checking.** The finite state machines and the temporal logic formulae are fed into a model checker. The model checker verifies whether the temporal logic formulae are respected by the given state machines. As a result, it is detected which process models comply to the initial BPMN-Q queries.

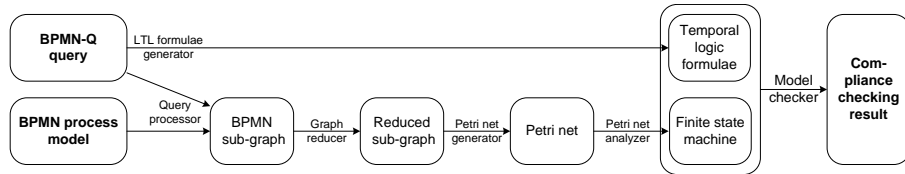


Fig. 7. Compliance checking approach

### Retrieval of BPMN Sub-graphs

The major role of BPMN-Q is to select the set of process models which are relevant for the query. As a first step, BPMN-Q selects all process models in each of which the set of activity nodes is a superset for the activities in the query. With each of these processes, the ordering between activities, expressed in the query as sequence flows and/or paths, are tested. If the process graph fails to satisfy any of these, it is dropped from the answer set. For more details about query processing of BPMN-Q, please refer to [2].

The result of queries (rules) 1 and 2 in Figure 6 against the process model in Figure 1 is shown in Figure 8 where all nodes between "Obtain Customer Info" and "Open Account" are included in the result. Since the activity "Verify Customer ID" already resides on the path from "Obtain Customer Info" to "Open Account", the evaluation of path from "Verify Customer ID" to "Open Account" will not introduce new nodes or edges to the result.

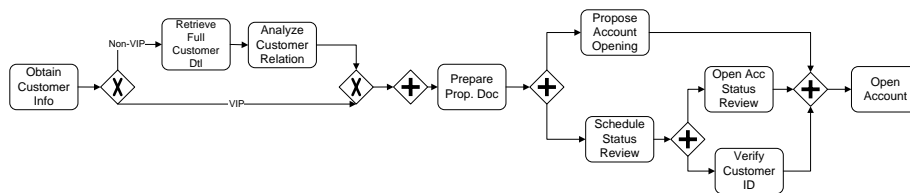


Fig. 8. Process graph matching rule 1

We can notice in the result of query 2 (as shown in Figure 9) that nodes preceding the activities "Submit Deposit" and "Record Customer Info" were also included. This is due to the implicit inclusion of a start node with a path to "Open Account". This inclusion occurred because of the *precedes* between "Record Customer Info" and "Open Account" as discussed in Section 3.

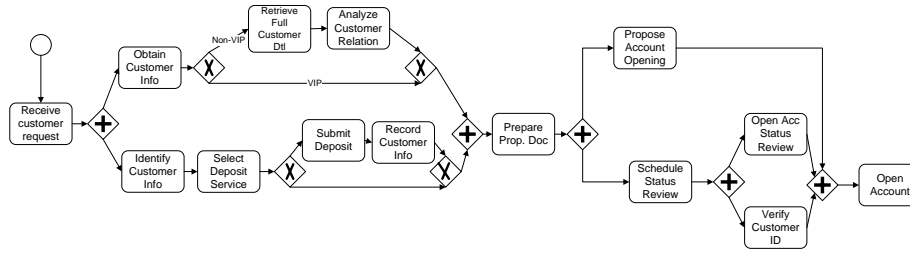


Fig. 9. Process graph matching rule 2

## Graph Reduction

Reduction rules have been successfully used either as a stand alone approach [23, 24], or as an engineering approach used to reduce the complexity of the process models [27, 19] to verify correctness of process models. We adopt the reduction approach to reduce the state space for model checking in a way to work around the state space explosion [3]. Unlike the aforementioned approaches which focused on simplifying the underlying control graph, our approach respects and depends on the set of activities included in the rule to be verified. So, the reduction result differs depending on the rule to be verified.

We have already discussed the difference between precedes and leads to dependencies. As a result, the reduction rules applied must respect these dependencies. This is especially important when considering decision points (XOR/OR-splits) and merging behavior (XOR/OR-joins).

- R1: Reduction of Activities, and intermediate events: all activities that are not of interest to the compliance rule (query) are removed.
- R2: Reduction of Structured Blocks. A block of split gateway that is directly connected to a join gateway of the same type and both have no other connections is replaced with a sequence flow edge.
- R3: Merging of similar gateways. if two splits or two joins of the same type follow each other, for example an XOR-split A is followed by XOR-split B and B has no other incoming edges, B is removed and the sequence flow edge between A and B. A inherits all outgoing sequence flow edges of B.
- R4: Reduction of structured loops. Loops having an XOR/OR-join node as an input and an XOR/OR-split as an output with backward edge from split node to join node are reduced by removing the backward edge.
- R5: Reduction of Single activity parallel blocks. Whenever an activity that is of interest to the compliance rule (query) lies in parallel block, after the reduction of other activities that are parallel to it, parallel block is removed and the activity is retained connected with nodes before and after the block. if the parallel block contains in only one of its branches activities of interest that are somehow in a nested structure, the direct edges from the AND-Split to the AND-join are removed.
- R6: Reduction of Single output AND-Split and single input join nodes. Due to the nature of pattern matching based query processing of BPMN-Q

(see [2] section 6), and to application of other reduction rules, a situation where an AND-Split with single outgoing edge or a join gateway that is either preceded or followed by an Activity of interest to the query may occur. The rule replaces the node with a single sequence flow between the source of its incoming sequence flow and the destination of its outgoing sequence flow.

- R7: Reduction of start events. Depending on whether the query contains start events (either explicitly by the user or implicitly added as described in 3, we reduce start events in case two or more start events are the input for an AND-join. We remove the set of start events along with sequence flow edges to the AND-join and the AND-join itself and introduce a single start event and a sequence flow edge from that event to the destination of the outgoing sequence flow edge of the AND-join.
- R8: Reduction of single activity selection block. The application of this rule depends on the type of path operator the activity is involved in. The reduction rule is applicable only if this activity is involved in only *leads to* paths as a source, otherwise we cannot apply the rule.
- R9: Reduction of BPMN-specific activities. This includes the MI activity, loop activity and ad-hoc activities. For MI and ad-hoc activities we assume that there is only one token produced from the activity after all running instances complete. In case of Loop activities we follow the mapping shown in [4].

Rules from R1 to R4 are adapted from previous work using reduction rules to verify correctness [27, 19, 23]. Unlike other reduction approaches, the reduced graph always contains the set of activities mentioned in the query graph.

Applying reduction rules to the process graph of Figure 8 will result in reduced graph shown in Figure 10. We elaborate more details on applying reduc-

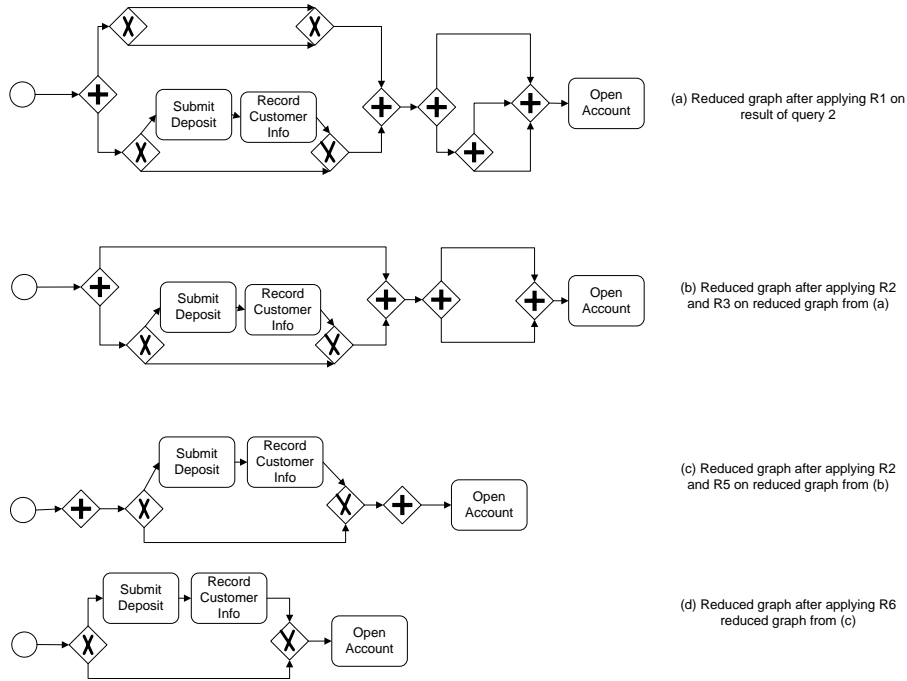


**Fig. 10.** Reduced graph for rule 1

tion rules to the result of query 2 shown in Figure 9. R1 is applied to remove all activities and intermediate events that are not of interest to the compliance rule. The result is shown in Figure 11 (a). Applying rules R2, R3 to the graph resulting from (a) yields the reduced graph in (b). Applying R2 again removes the parallel block immediately before the "Open Account" activity. A special case of R5 (as discussed earlier) is applied where the direct edges from the first AND-Split to the AND-join are removed resulting in graph (c). A final application of R6 produces graph in (d).

## Generation of Temporal Logic Formulae

Linear Temporal Logic allows expressing formulae about the future of systems. In addition to logical connectors ( $\neg, \vee, \wedge, \rightarrow, \Leftrightarrow$ ) and atomic propositions, it introduces temporal quantifiers (always, eventually, next, until). The temporal



**Fig. 11.** Reduced graph for rule 2

operator *eventually* is of direct correspondence to the *leads to* concept. On the other hand, the translation of the *precedes* into a temporal expression in LTL would be complex. We used Past linear time temporal logic PLTL [17, 29] as it has introduced the counterpart temporal quantifiers (always in past, once in the past, previous state, since) to allow expressing formulae about the past states of a system. Although these quantifier did not increase the expressiveness of LTL, it made expressing formulae about the past exponentially succinct than in pure-future LTL [16]. Since the representation of these temporal quantifiers is not standardized, we mention the notation we use throughout this paper. For future states G(all future states), X(next state), F(eventually), U(Until). For past states H(all past states), Y(previous state), O(Once in the past), S(Since). It is straightforward to relate the concept of *precedes* to the temporal operator O and the concept of *leads to* to the operator F.

The generation of PLTL formulae from queries is straight forward. The following listing summarizes the translation into PLTL.

- A *leads to* B is translated to  $A \rightarrow F(B)$ .
- A *precedes* B is translated to  $B \rightarrow O(A)$ .

All generated formulae for different path constructs are conjuncted together and surrounded by the G operator to express the meaning of *in all possible execution scenarios*, be sure that the formulae are satisfied.

Query of rule 1 will generate the following temporal formula

$$G( (\text{Obtain Customer Info} \rightarrow F(\text{Open Account})) \wedge (\text{Verify Customer ID} \rightarrow F(\text{Open Account})))$$

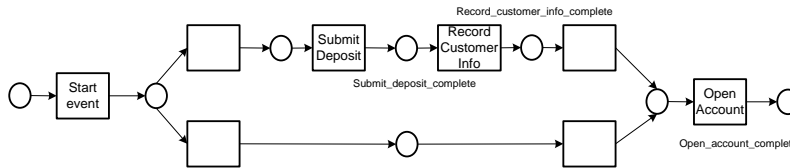
Model checking this formula against the reduced graph of Figure 10 will succeed, i.e. the process model complies with the rule. The query of rule 2 will generate the following temporal formula

$$G( (\text{submit deposit} \rightarrow F(\text{record customer info})) \wedge (\text{open account} \rightarrow O(\text{record customer info})))$$

Model checking this formula against the reduced graph of Figure 11 will fail, i.e. there are some execution scenarios that do not satisfy this formula.

### Petri Net and State Space Generation

We need to generate, from the (reduced) graph, the finite state machine that will be, along with the PLTL expression, the input to model checker. In fact, we need to be sure that the state machine is finite, otherwise model checking is not feasible [3, 8]. In order to generate the state machine and determine its finiteness, we have to give formal execution semantics to the different constructs of BPMN. We follow the approach introduced in [4]. In this approach a mapping of a subset of BPMN constructs (for example OR-join is not addressed) to Petri nets is given. We have implemented their algorithm in our tool, as will be shown later. The state machine is then obtained by the reachability graph of the Petri net. Finiteness of the state machine reduces to the boundedness of the net, so utilizing a Petri net tool to determine this property. Figure 12 is the generated Petri net for the reduced graph of Figure 11.



**Fig. 12.** Petri net for reduced process graph of Figure 11

Output places of transitions corresponding to activities are of interest to indicate the execution of the activity — see labeled places of Figure 12. Further reductions on the level of a Petri net are possible provided that these places are not merged.

Table 1 shows the average running time (in milliseconds) of the model checker (MC) for checking rule 2 with and without reduction. Also, reduction time is

presented. The query of rule 2 matched 5 process models in the repository. One of them was excluded from model checking because it suffered from a deadlock.

Process Model Id	No. Nodes	MC without reduction	MC with reduction
A	39	151 ms	110 ms
B	32	52 ms	31 ms
C	38	52 ms	48 ms
D	35	68 ms	52 ms

**Table 1.** Comparison of average running time of model checker with and without using reduction

## Tool Support

We have implemented a prototypical tool chain for our approach. As process modelling environment we use Oryx<sup>1</sup>, a web-based BPMN editor developed in our research group. We implemented a Petri net generator as an integrated component in BPMN-Q. LoLA [25]<sup>2</sup> is used for checking boundedness and absence of deadlocks. LoLA is also used for producing the finite state machine. We implemented a PLTL generator returning the temporal logic formulae. As model checker we opted for NuSMV<sup>3</sup> due to its support for PLTL expressions.

## 5 Related Work

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 [26].

*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 [14], 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 [13] 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

<sup>1</sup> See <http://oryx-editor.org>

<sup>2</sup> See [http://www.teo.informatik.uni-rostock.de/lst\\_tpp/lola/](http://www.teo.informatik.uni-rostock.de/lst_tpp/lola/)

<sup>3</sup> See <http://nusmv.irst.itc.it/>

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. In [20] a more comprehensive framework where a categorization of control objective along with corresponding compliance patterns were discussed. They assume that process models are by default are not compliant. Later on, they are adapted and enriched with controls to be compliant with support of monitoring of violation at runtime.

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 [15] to formally measure the compliance between a business contract and a business process. In [12] 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 [18], 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 [28] 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.

Using queries for generating PLTL formulae, the retrieval of sub graphs to be tested and the application of reduction rules for simplifying the state space are unique properties of our approach.

## 6 Conclusion

In this paper we have presented an approach for compliance checking for BPMN process models using BPMN-Q queries. As centerpiece of the analysis, a model checker is used. The approach is not limited to BPMN process models, it can be applied to any process modelling language with formal execution semantics. The usage of BPMN-Q to express compliance rules was not limited to the graphical representation of the rules. BPMN-Q as a query language helps identify the set of process models that are subject to compliance checking, a task that is too time-consuming if done manually. The usage of reduction was to simplify the state

space for the model checker, especially for large process models — which is the case in real world process models. For small process models, model checking can be applied directly without the overhead of reduction. This approach is effective under the assumption that business process models really reflect the way business is carried out. Although we assume that relevant (candidate) process models have to contain all activities mentioned in a rule, we still see our approach effective since the hard task is to check the ordering between activities. If a process model contains a non-empty subset of activities in a rule, we can conclude it non-compliant without further checking.

In the current version of our approach, we are able to give yes/no answers for the compliance between rule(s) and process models. As a limitation of our approach, the detailed analysis provided by the model checker in the case of non-compliance cannot be taken advantage of. This is due to the fact that we applied reductions. This means that generated counter examples by the model checker do not reflect real execution scenarios in process models. An important assumption behind the use of reduction rules is the relaxation of usage of temporal operators next X, and previous Y in queries. It has been pointed in literature that the X operator is of little interest when verifying properties of process models [5, 7].

Currently, most research in the area of compliance checking focuses on verification of control flow aspects. As future work, we intend to extend BPMN-Q with the capability of querying data objects and verification of their states as pre-conditions for activities.

## 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*, pages 115–128, 2007.
3. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.
4. R. M. Dijkman, M. Dumas, and C. Ouyang. Semantics and Analysis of Business Process Models in BPMN. *Information and Software Technology (IST)*, 2008.
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. J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34(2):85–107, 1997.
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. A. Ghose and G. Koliadis. Auditing business process compliance. In *Service-Oriented Computing ICSOC 2007*, pages 169–180. Springer Berlin / Heidelberg, 2007.
13. S. Goedertier and J. Vanthienen. Compliant and Flexible Business Processes with Business Rules. In *7th Workshop on Business Process Modeling*, 2006.
14. S. Goedertier and J. Vanthienen. Designing Compliant Business Processes from Obligations and Permissions, 2nd Workshop on Business Processes Design (BPD'06), Proceedings. In *Business Process Management Workshops*, 2006.
15. G. Governatori, Z. Milosevic, and S. Sadiq. Compliance checking between business processes and business contracts. In *EDOC '06*, pages 221–232, Washington, DC, USA, 2006. IEEE Computer Society.
16. S. Hornus and P. Schnoebelen. On solving temporal logic queries. In *AMAST '02: Proceedings of the 9th International Conference on Algebraic Methodology and Software Technology*, pages 163–177, London, UK, 2002. Springer-Verlag.
17. F. Laroussinie and P. Schnoebelen. A hierarchy of temporal logics with past. *Theoretical Computer Science*, 148(2):303–324, 1995.
18. Y. Lui, S. Mller, and K. Xu. A static compliance-checking framework for business process models. *IBM SYSTEMS JOURNAL*, 46(2):335–362, 2007.
19. 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.
20. K. Namiri and N. Stojanovic. Pattern-based design and validation of business process compliance. In R. Meersman and Z. Tari, editors, *OTM Conferences (1)*, volume 4803 of *Lecture Notes in Computer Science*, pages 59–76. Springer, 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 *BPM*, pages 149–164, 2007.
23. W. Sadiq and M. E. Orłowska. Applying graph reduction techniques for identifying structural conflicts in process models. In *CAiSE '99*, pages 195–209, London, UK, 1999. Springer-Verlag.
24. W. Sadiq and M. E. Orłowska. Analyzing process models using graph reduction techniques. *Inf. Syst.*, 25(2):117–134, 2000.
25. 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.
26. 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.
27. B. F. van Dongen, W. M. P. van der Aalst, and H. M. W. Verbeek. Verification of eps: Using reduction rules and petri nets. In *CAiSE*, pages 372–386, 2005.
28. 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.
29. L. Zuck. *Past Temporal Logic*. PhD thesis, Weizmann Intitute, Rehovet, Israel, August 1986.