Consistency Checking of Compliance Rules

Ahmed Awad, Matthias Weidlich, Mathias Weske

Business Process Technology Group
Hasso-Plattner-Institute, University of Potsdam, Germany
ahmed.awad,matthias.weidlich,mathias.weske@hpi.uni-potsdam.de

Abstract. Compliance checking of business process models against regulation is inevitable. Due to various sources of compliance requirements, a conflict of interest of such requirements is very likely. Thus, it is crucial to analyze the relation between compliance rules to discover any possible conflicts before even checking such rules against process models. Although this step is important in the compliance management life cycle, there is almost no work that studied this direction. In this paper, we start by checking for consistency between execution ordering compliance rules expressed in linear temporal logic (LTL), addressing control and data flow aspects. To achieve this, we rely on the generation of Büchi automaton from LTL formulas. However, we show that domain-specific knowledge is of crucial importance to draw correct conclusions.

Key words: Compliance Checking, Consistency Checking, Temporal Logic

1 Introduction

Compliance checking of business process models has received growing attention in recent years. Financial scandals in large public companies led to legislative initiatives like SOX [1]. The purpose of these initiatives is the enforcement of certain controls on business. Controls can be in the form of, e.g., *executing* activities in certain order or separation of duty regarding financial transactions (a person who issues an order must not be the one to grant it). In addition to regulations, compliance requirements might be enforced by domain specific initiatives, e.g., BASEL II [28] in the banking sector.

Since then, compliance of a process model at design time has been addressed by different approaches. In such approaches either the process model creation is guided by compliance rules (e.g., [17, 16, 18]) or existing process models are *verified* against compliance rules (e.g., [3, 12, 23, 31]). Still, most of the effort is spent on checking/enforcing compliance between a process model and a rule. Almost no work discussed how to automatically check consistency among various compliance rules. On the one hand, such inconsistencies might stem from ambiguous descriptions of compliance rules as they are typically described in natural language and, therefore, subject to interpretation. On the other hand, compliance requirements might originate from a variety of sources, e.g., in the form of regulations (SOX Act in the public sector), internal policies of the organization, or guidelines and best practices (e.g, BASEL II and guidelines for anti-money laundering [7] in the financial sector). Consequently, the chance for conflicting rules is very likely. For instance, consider two rules, one stating that "in general, money needs to be transfered before shipment of the product", whereas the second states that "shipment of the product might happen before payment, if credit card details are already known for the customer". Clearly, there is a conflict that might be attributed to *misinterpretation* of the compliance requirements. That is, certain assumptions (e.g., that credit card details are not known in case of the first rule) are not made explicit.

One might argue that these conflicts between compliance rules are revealed once a process model is checked for compliance. However, it does not seem reasonable to spend effort on such a check that is bound to fail. Moreover, the negative compliance result might cause even more time consuming investigations of the particular process, even though conflicts in the compliance rules caused the compliance violation. Therefore, it is essential to provide support for automated conflict detection for compliance rules.

Our contribution in this paper is an automated approach to check consistency among rules that specify execution ordering constraints, potentially enriched with data-related aspects. These rules are expressed as temporal logic formulas [30, 32]. Although the notion of consistency checking among rules expressed in temporal logic has already been addressed in literature, it has been applied solely in the context of hardware specifications [8]. While our work is inspired by these approaches of checking consistency, we show that they cannot be applied for compliance rules of business process models in a straight-forward manner. Instead, additional domain knowledge needs to be leveraged.

The remainder of this paper is organized as follows. Section 2 provides preliminaries for our work, i.e., basic notions of execution ordering compliance rules and their formal grounding used for model checking. Afterwards, Section 3 shows that consistency checking of compliance rules is not straight-forward and illustrates the need for considering domain knowledge. Based thereon, we introduce the notion of a business context that captures domain knowledge in Section 4. Section 5 shows how the business context is applied to detect conflicting and inconsistent rules. Further on, we review related work in Section 6 and conclude the paper in Section 7.

2 Preliminaries

In this section we introduce the background of our work. First, Section 2.1 briefly summarizes linear temporal logic with past operators (PLTL). Second, Section 2.2 summarizes our previous work in the area of compliance checking. Afterwards, Section 2.3 gives details on the formal background of compliance checking using model checking.

2.1 Linear Temporal Logic with Past Operators (PLTL)

Linear Temporal Logic (LTL) allows expressing formulas about the future of systems. In addition to logical connectors $(\neg, \wedge, \vee, \rightarrow, \Leftrightarrow)$ and atomic propositions, it introduces temporal operators, such as *eventually* (F), *always* (G), *next* (X), and *until* (U). PLTL [32] extends LTL by operators that enable statements over the past. That is, it introduces the *previous* (P), *once* (O), *always been* (H), and *since* (S) operators. Although the past operators do not increase expressiveness of the formalism, they enable convenient specification of predicates over the past [20].

2.2 Compliance Rules for Business Process Models

Compliance rules might require the execution of certain activities or enforce ordering constraints for a set of activities. Thus, these rules focus purely on the control flow

	Rule Pattern	PLTL Formula
Control Flow	Global Scope Existence	$\mathsf{F}(executed(a))$
	Global Scope Absence	$\mathsf{G}(\neg executed(a))$
	Before Scope Absence	$\mathsf{G}(ready(b) \rightarrow \mathsf{H}(\neg executed(a)))$
	After Scope Absence	$\mathsf{G}(executed(a) \rightarrow \mathsf{G}(\neg executed(b)))$
	After Scope Existence	$\mathsf{G}(executed(a) \rightarrow \mathsf{F}(executed(b)))$
	Between Scope Absence	$\mathsf{G}(executed(a) \rightarrow (\neg executed(b))\ \mathsf{U}\ executed(c))$
	Before Scope Existence)	$\mathsf{G}(ready(b) \rightarrow \mathsf{O}(executed(a)))$
	Before Scope Absence	$\mathsf{G}(ready(c) \rightarrow (\neg executed(b))\ \mathsf{S}\ executed(a))$
Data	Data Flow	$\mathsf{G}(ready(a) \rightarrow \bigwedge_{d \in D}(\bigvee_{s \in S} state(d, s)))$
Mixed	Conditional After Scope Existence	$\mathsf{G}(executed(a) \wedge \bigwedge_{d \in D}(\bigvee_{s \in S} state(d, s))$ $\rightarrow \mathsf{F}(executed(b)))$
	Conditional After Scope Absence	$\mathsf{G}(executed(a) \wedge \bigwedge_{d \in D}(\bigvee_{s \in S} state(d, s))$ $\rightarrow (\neg executed(b))\ \mathsf{U}\ executed(c))$
	Conditional Before Scope Existence	$\mathsf{G}(ready(b)$ $\rightarrow \mathsf{O}(executed(a) \wedge \bigwedge_{d \in D}(\bigvee_{s \in S} state(d, s))))$
	Conditional Before Scope Absence	$\mathsf{G}(ready(c) \rightarrow (\neg executed(b))\ \mathsf{S}$ $(executed(a)) \wedge \bigwedge_{d \in D}(\bigvee_{s \in S} state(d, s)))$

Table 1. Mapping of Compliance Rule Patterns into PLTL

perspective of process models. These rules might be further classified according to the scope they consider, which might be *global*, *before*, *after*, or *between* [10]. Within a scope rules can require the *existence* or *absence* of activities. In [3, 6], we showed how compliance rules focusing on the control flow can be modeled in PLTL. We also showed how these rules can be verified using model checking techniques.

The aforementioned kinds of rules neglect data aspects of process models. It is well-known that this is not sufficient for real-world scenarios, as execution of an activity often changes data objects from one state to another [19]. Therefore, data aspects have to be taken into account in order to achieve holistic compliance checking, which we discussed in previous work [5].

The whole spectrum of compliance rules along with the respective PLTL expressions is illustrated in Table 1. With A as the set of activities of a process model, the PLTL expressions are based on the predicates $ready(a)$ and $executed(a)$. They capture the facts that an activity $a \in A$ is about to execute or has already executed, respectively. It is worth to mention that the predicate $executed(a)$ holds solely in *one* state. That is, the state that is reached by the execution of a. In other words, $executed(a)$ does *not* hold in all states after execution of activity a. In the line of the data access semantics defined in [2], we assume a set of data objects D and a set of data states S. Further on, the predicate $state(d, s)$ requires that a certain data object $d \in D$ is in state $s \in S$.

Global Scope rules require that a certain activity is either executed in all process instances (existence rule), or in $none$ of them (absence rule). An example for a global scope existence rule would be an insurance handling process, in which it is required that the case is $always$ archived.

Before / After Scope Absence rules require the absence of a certain activity either *before* or *after* the execution of another activity, i.e., mutual exclusion.

Leads to rules specify a causal dependency between two activities, such that one activity eventually follows the execution of another activity (existence rule). This case might also be interpreted as an *after scope existence* rule. Further on, a third activity might be requested not to occur in between (absence rule). As in case of the existence rule, the latter could be seen as a *between scope absence* rule.

Precedes rules specify a causal dependency between two activities, such that one activity has been executed whenever another activity is about to execute. Again, this kind of rules might be specified as an absence rule. That is, an activity has been executed whenever another activity is about to execute, while a third activity has not been executed in between.

Data Flow rules specify requirements for the execution of a certain activity in terms of combinations states of data objects. These requirements have to be met, whenever the activity is about to execute.

Conditional Leads to rules specify an execution ordering relation between activities that is further refined by data conditions. That is, a *leads to* rule as introduced above has to hold solely in case certain data conditions are met. Such a data condition might be considered in the case of existence rules as well as in case of absence rules.

Conditional Precedes rules also refine the aforementioned $precedes$ rules. That is, the causal dependency is required to hold solely if the data condition is met. Again, a data condition might be applied for existence rules and absence rules.

2.3 LTL Model Checking based on Büchi Automata

Model checking techniques can be applied to check compliance rules that are specified in LTL. In particular, both, the actual compliance rule as well as the behavioral model of the system, a process model, might be represented as Büchi automata. A Büchi automaton for an LTL formula represents a computation to satisfy the formula, if there is one. Approaches like [14, 13][1] are capable of generating a Büchi automaton for a given LTL formula. Based on the set of words that is accepted by the automata of the formula *and* the system, we are able to conclude whether the system satisfies the compliance rule.

3 Checking Consistency for Compliance Rules

This section introduces our approach of checking consistency be means of exemplary compliance rules. The overall setting is depicted in Fig. 1. In general, there might be more than one set of compliance rule, e.g., rules related to anti money laundering and rules related to BASEL II. Some of these rules are related, i.e., they address a common set of business processes. In order to enable consistency checking of these rules, we have to include extra knowledge from the domain. Finally, the conjunction of all LTL formulas is created and its corresponding Büchi automaton is checked for having accepting states.

[1] An implementation of the algorithm presented in [13] is available from `http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/index.php`

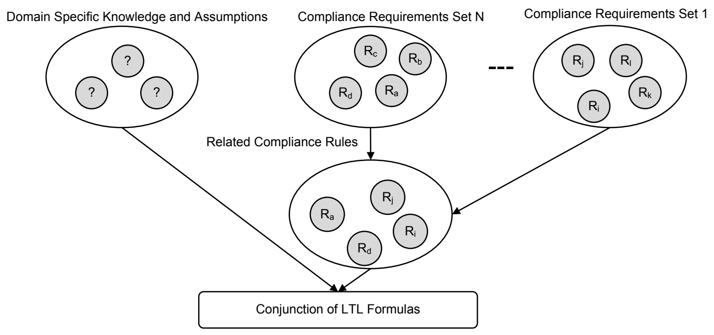

Domain Specific Knowledge and Assumptions
Compliance Requirements Set N
Compliance Requirements Set 1
Related Compliance Rules
Conjunction of LTL Formulas

Fig. 1. The Approach of Consistency Checking

In order to motivate the need for taking domain knowledge into account, we introduce a set of example compliance rules. Following on the model checking approach as introduced in the previous section, we illustrate the resulting Büchi automaton for the conjunction of compliance rules. Of course, one would assume the compliance rules to be consistent, if the Büchi automaton has accepting runs, whereas inconsistent rules are indicated by an automaton that has no accepting runs. However, our examples show that detection of inconsistencies is not as straight-forward as expected.

Example 1. (Cyclic Dependency) Imagine two rules that consider the ordering of two activities, one specifying the payment of goods and the other describing the sending of goods. Let the first rule R1 state that payment must precede sending goods, whereas the second rule R2 states the opposite, i.e., sending goods precedes payment. It is obvious that the objectives behind the two rules are totally different. In the first one we need to guarantee that the money has been transferred before the goods are sent out. With respect to the second rule, the objective might be to gain new customers by relaxed payment conditions. If we represent the payment activity with a and the sending goods activity with b, the corresponding LTL formulas would be $G(executed(b) \rightarrow O(executed(a)))$ and $G(executed(a) \rightarrow O(executed(b)))$. Here, O is the past time LTL operator *once*.

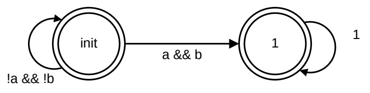


Fig. 2. Automaton for R1 and R2

The Büchi automaton generated for the conjunction of both formulas is illustrated in Fig. 2. Although R1 and R2 are clearly inconsistent, the respective Büchi automaton has accepting runs. In particular, the following runs are possible.

1. Neither a nor b are executed at all, which is represented by the transition from state $init$ to itself.
2. Both activities, a and b are executed in a single step, which is represented by the transition from $init$ to $state1$.

The phenomena that led to the first accepting run is called vacuous satisfiability [8]. The set of compliance rules would be satisfied by all process models that contain neither a nor b. Obviously, that is not a valid solution in our context, as these compliance rules are specified for process models that contain these activities.

The second accepting run is invalid in our context either. Common process description languages, e.g., the Business Process Modeling Notation (BPMN) or Event-Driven

Process Chains (EPC) assume interleaving semantics. That is, two activities cannot complete their execution at the very same time. That, in turn, is reflected in any behavioral model of a process model, which is derived by transformation to some intermediary representation, for instance, Petri nets (cf., [9]). These models do not contain a single state in which two activities finish their execution. For this reason, the second accepting run of the automaton in Fig. 2 is also invalid in our context.

Example 2. (Contradictions) Imagine two rules that have the same condition, but different implications. For instance, one rule R3 specifies that the reception of payment leads to the activity of sending goods by DHL. A second rule R4, in turn, states that after the reception of payment, the goods have to be sent by UPS. Further on, let a be the activity of receiving the payment, while the send activities are represented by b in case of DHL, and c in case of UPS, respectively. Then, the resulting LTL formula would be $G(executed(a) \rightarrow F(executed(b))) \wedge G(executed(a) \rightarrow F(executed(c)))$. The Büchi automaton, which was omitted due to its size, has accepting runs, owing to the vacuous satisfiability phenomena discussed above and state transitions with more than one activity being executed at a time. However, there are other accepting runs that might not be traced back to these phenomena. These runs correspond to scenarios, in which activity a is executed; thereafter activities b and c are executed in any order.

A process model fragment that shows such a behavior and, therefore, satisfies the compliance rules is illustrated in Fig. 3. However, this model would not be considered a valid solution for our compliance requirements from a business point of view. Goods might be send only once using either activity *Send Goods by DHL* or *Send Goods by UPS*. Therefore, consistency checking for these kinds of compliance rules requires further information about the business domain.

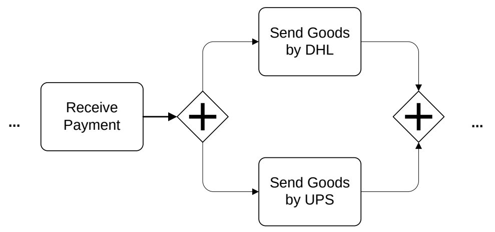


Fig. 3. A process fragment satisfying rules R3 and R4

Example 3. (Data Issues) The aforementioned aspects are also of relevance for data-related rules. As mentioned before, we assume a formalization of data flow for process models, such that a data object can be only one data state at a time. Thus, data states are exclusive.

Consider a compliance rule R5 that requires a purchase request to be in state $archived$ when the case is closed. Based on the predicates introduced in Section 2, this compliance requirement is expressed as $G(ready(a) \rightarrow state(d, archived))$ with a being the activity to close the case and d the data object representing the purchase request. Further on, a second rule R6 requires a purchase request to be either in state $accepted$, or $rejected$, i.e., $G(ready(a) \rightarrow state(d, accepted) \vee state(d, rejected))$, when the case is about to be closed.

Not surprisingly, the Büchi automaton for the conjunction of these rules has a lot of accepting runs (again, the figure was omitted due to its size) even though both rules are inconsistent. That results from the phenomena that were already discussed above for activities. That is, accepting runs violate the requirement of exclusive data states, require data objects to be in no state (vacuous satisfiability), or are invalid from a business perspective (e.g, the state of the purchase request might be set to both, $accepted$ and $rejected$). Again, inconsistencies between two compliance rules cannot be identified by solely testing the generated Büchi automaton for the absence of accepting runs.

4 The Business Context

It is of crucial importance to include correct assumptions about the system/domain under investigation in order to correctly decide about the consistency of the specification [8]. We also illustrated this need by our examples in Section 3. Thus, reasonable consistency checking has to consider business context information.

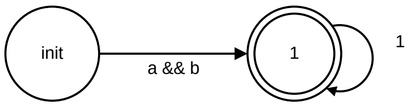
The business context describes the domain specific knowledge. It reflects the way a business provides its added value. Moreover, it could be seen as the unified view of the domain among the participants. The context is a global process independent description of the business activities. Processes are composed from individual business activities from the context. Each business activity provides a value by its own. For instance, an activity *Payback claim by bank transfer* provides the value of transferring the amount of money claimed by a customer as its *effect*. However, execution of such an activity is based on assumptions about the input, there are *preconditions*. Moreover, having another activity *Payback claim by a cheque*, it would be part of the domain knowledge that it is not possible for the two payment activities to be *both* executed in the same process.

To make the concept of a context more familiar, one might think of the notion of service registries [27] in service oriented architecture as a sub-concept for the business context. There, each service is described independently in terms of its inputs and outputs on a syntactical level. In a business context, business activities are the counterpart of services in a registry. We require at least the following information to be part of the business context in terms of relations between business activities. 1) Preconditions: for each activity we need to know the preconditions, other activities required to execute previously. 2) Effect: the value added by executing a certain activity. 3) Contradiction: the set of activities that cannot be allowed to execute within the same process instance.

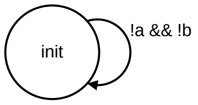
5 Detecting Compliance Conflicts

Given a set of compliance rules, it becomes necessary to check consistency among them. In particular, the business context (domain knowledge) has to be considered in the PLTL expressions in order to detect inconsistencies based on the respective Büchi automaton. Again, we assume a set of compliance rules to be inconsistent, if the Büchi automaton for the conjunction of the rules has no accepting runs. We illustrate the approach based on the examples presented in Section 3.

Example 1. (Cyclic Dependency) The first example contains two rules R1 and R2 that state a cyclic dependency between payment for goods, activity a and sending the goods, activity b, respectively. However, the Büchi automaton generated for the conjunction showed accepting runs, which required both activities not to be executed at all or to be



(a) Automaton for the conjunction of R1, R2, $F(executed(a))$, and $F(executed(b))$



(b) Automaton for the conjunction of R1, R2, $F(executed(a))$, $F(executed(b))$, and $G(\neg(executed(a) \wedge executed(b)))$

Fig. 4. Büchi automata for R1 and R2 with additional conditions

executed in the same step. In order to cope with the first phenomenon, that is, vacuous satisfiability, we specify an additional rule, $F(executed(a))$, that enforces the execution of payment for goods activity. That, in turn, results in the automaton that is illustrated in Fig.4(a), which still accepts the concurrent execution of both activities. As discussed in Section 3 such concurrent execution conflicts with the assumption of interleaving semantics for process models. Therefore, another predicate that prohibits concurrent execution, $G(\neg(executed(a) \wedge executed(b)))$, is added. As mentioned in Section 2 the predicate $executed(a)$ holds only for one state, i.e., the state that is reached by executing the activity a. The updated automaton is shown in Fig. 4(b). This automaton has no accepting run. At this point, we conclude that the rules R1 and R2 are *inconsistent* given our knowledge about the process modeling domain. Thus, the additional formula that prohibits concurrent execution of activities is of a general nature and has to be added for consistency checking of any activity execution compliance rules.

Example 2. (Contradictions) . The second example introduces two rules R3 and R4 with the same condition (reception of payment), activity a, but different implications (sending goods by DHL, activity b, or sending goods by UPS, activity c, respectively). However, enforcing the knowledge about non-concurrent execution of activities a, b, and c, as was discussed in Example 1, still generates a Büchi automaton that accepts all runs in which activity a is executed; thereafter, activities b and c are executed in any order. However, according to the business context, we figure out that activities *Send Goods by DHL* and *Send Goods by UPS* contradict each other. This fact has to be reflected in consistency checking. That is, both activities are considered to be mutually exclusive. Therefore, we add the following two LTL formulas to the conjunction, $G(executed(b) \rightarrow G(\neg executed(c)))$ and $G(executed(c) \rightarrow G(\neg executed(b)))$. The resulting Büchi automaton has no accepting run. Again, the automaton was omitted due to its size. We conclude that there cannot be a process model that would satisfy R3 and R4 under the information given by the business context. Thus, we detected an inconsistency between both compliance rules.

Similar to the two examples above, data access semantics for data elements have to be reflected in the LTL formulas for consistency checking. That is, for our case, the fact that each data object can assume a single value (state) at a time has to be considered. Moreover, the knowledge about contradicting data values has to be incorporated.

So far, we showed through examples how consistency checking between different compliance rules can be reduced to a check of accepting runs (accepting states) in a Büchi automaton generated from an LTL formula. Using the business context, additional knowledge was injected in the LTL formula in order to reflect domain-specific compliance requirements. Based thereon, we generalize the approach to any given set of compliance rules for which consistency checking is required. The requirement of separate execution states is captured as follows.

Definition 1 (Separate Execution States). *For any set of activities A, each activity completes its execution in a separate execution state, i.e.,* $\bigwedge_{a,b \in A, a \neq b} G(\neg(executed(a) \wedge executed(b)))$.

Non-vacuous satisfiability for compliance rules has to be ensured. Therefore, we introduce the following LTL formula.

Definition 2 (Non Vacuous Satisfiability). *Process models that would satisfy compliance rules regarding a set of activities A have to satisfy them non-vacuously. That is, $\bigwedge_{a \in A} F(executed(a))$, if and only if a appears in the condition of a compliance rule.*

Further on, contradicting activities as well as mutual exclusive data states as defined by the business context have to be treated accordingly.

Definition 3 (Contradicting Activities). *if two activities a and b are known to be contradicting, exclusive, from the business context, the LTL formula to avoid contradicting activities is defined as $G(executed(a) \rightarrow G(\neg executed(b)))$.*

Definition 4 (Mutual Exclusive Data States). *Let D be the universal set of data objects, S the set of all data states, and S_d the set of all possible data states for a specific data object $d \in D$. For each data object $d \in D$, the LTL formula that captures the mutual exclusion between data states is given by $\bigwedge_{s \in S_d} G(state(d,s) \rightarrow \bigwedge_{s' \in S_d \setminus \{s\}} \neg state(d,s'))$.*

Algorithm 1 describes our approach to check consistency among a given set of compliance rules. The algorithm takes a set of compliance rules and a business context as input. Based thereon, the additional LTL formulas are generated in order to meet the requirements introduced above. Finally, the algorithm decides consistency by analyzing the existence of an accepting run of the resulting Büchi automaton.

6 Related Work

A large body of research on compliance checking of business process models has already been published. Our primary interest is the work on execution ordering compliance checking. The research in this area can be classified into two groups: compliance by design and compliance checking of existing models. The idea of compliance by design is to enforce process model compliance already at the stage of design [16, 17, 21, 25, 26]. Here, violations to a compliance rule are highlighted while a process model is created.

The other branch of research employs model checking techniques to verify that existing process models satisfy the compliance rules [3, 12, 23, 31]. Based thereon, explanation of how violation could occur [5] has been tackled. Further on, resolution strategies have been presented in order to resolve compliance violations automatically [15, 4].

Resolution of violations can be driven by the severity of violations. In this sense an interesting method was introduced in [22]. This formal approach enables measuring the

Algorithm 1 Consistency Checking for a Set of Compliance Rules

```
Require: A set of compliance rules $R$ expressed in LTL.
Require: Domain specific knowledge.
 1: Let $A$ be the set of all activities mentioned in compliance rules.
 2: Let $A_{condition} \subseteq A$ be the set of activities mentioned in the condition part of the rules.
 3: Let $D$ be the set of data objects mentioned in the rules.
 4: Let $S$ be the set of data states for the data objects in $D$.
 5: Let $M$ be the set of LTL formulas that will be conjuncted and checked for consistency.
 6: $M = R$
 7: for all $a \in A_{condition}$ do
 8:     $M = M \cup \{F(a)\}$    // Handling activities
 9: end for
10: for all $a \in A$ do
11:     for all $b \in A$ do
12:         if $a \neq b$ then
13:             $M = M \cup \{G(\neg(a \wedge b))\}$    // Including knowledge about execution environment
14:         end if
15:     end for
16:     Let $C_a$ be the set of contradicting activities to activity a according to the context.
17:     for all $c \in C_a$ do
18:         $M = M \cup \{G(a \rightarrow G(\neg c))\}$    // Contradictions between activities
19:     end for
20: end for
21: for all $d \in D$ do
22:     Let $S_d \subseteq S$ be the set of states data object $d$ can assume.
23:     for all $s \in S_d$ do
24:         $M = M \cup \bigcup_{s' \in S_d \wedge s' \neq s} \{G(state(d,s) \rightarrow \neg state(d,s'))\}$ // One data state at a time
25:         LET $C_{(d,s)}$ be the set of contradicting data states of s.
26:         $M = M \cup \bigcup_{s' \in C_{(d,s)}} \{G(state(d,s) \rightarrow G(\neg state(d,s')))\}$ // State contradictions
27:     end for
28: end for
29: Let $BA = LTL2BA(\bigwedge_{m \in M} m)$
30: if $BA$ has accepting run then
31:     return $true$
32: else
33:     return $false$
34: end if
```

degree of compliance. Once a compliance rule is specified, the approach tells the degree of compliance for a given process model on the scale from 0 to 1.

In the above mentioned approaches, the need to check consistency of compliance rules was not addressed. Moreover, the consistency of the requirement of a rule with the way business is conducted, e.g. the business context, was not discussed.

Recently, several requirements frameworks for business process compliance management have been proposed. In [24] the authors formulate requirements for compliance management tools. The requirements address the issues of lifetime compliance. The focus is also on the requirements to languages expressing compliance rules, on the rules priority, and on validation of process models against rules during design time and runtime.

Another framework was proposed in [11]. The requirement to check consistency among several rules complements these frameworks. Moreover, the approach we presented in this paper is a realization of this requirement.

The review of related work illustrates that the problem of business process compliance is pressing. However, it reveals that in the area of compliance rules consistency checking not much has been done yet.

Declarative business process modeling is a way to allow flexibility in processes. Processes are modeled by specifying a set of execution ordering constraints on a set of activities [29]. In that approach, constraints are mapped onto LTL formulas; which are used to generate an automaton to both guide the execution and monitor it. Albeit for a different purpose, this approach is similar to our work in terms of mapping LTL formulas to Büchi automata. However, in our approach, we just need to check the resulting automata for accepting states/runs as means to decide about consistency between compliance rules.

7 Conclusion

Consistency checking between compliance rules and the way business is conducted, i.e. the business context, on the one hand and among a set of compliance rules on the other hand, is an integral step in the compliance management lifecycle. In this paper, we demonstrated a formal approach to check such consistency. Checking is done by mapping compliance rules, represented as PLTL formulas, into Büchi automata. In order to correctly decide about consistency, domain knowledge was reflected. Therefore, further PLTL formulas were added to the set of compliance rules. Rules are consistent under the domain knowledge, if and only if the Büchi automaton has accepting runs.

In future, we would consider approaches for managing inconsistencies among rules. One approach could be to prioritize rules. Another would be deriving consistent subsets of rules.

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, G. Decker, and N. Lohmann. Diagnosing and repairing data anomalies in process models. In *BPD*, LNBIP. Springer-Verlag, 2009.
3. A. Awad, G. Decker, and M. Weske. Efficient Compliance Checking Using BPMN-Q and Temporal Logic. In *BPM*, volume 5240 of *LNCS*, pages 326–341. Springer, 2008.
4. A. Awad, S. Smirnov, and M. Weske. Towards Resolving Compliance Violations in Business Process Models. In *GRCIS*. CEUR-WS.org, 2009.
5. A. Awad, M. Weidlich, and M. Weske. Specification, verification and explanation of violation for data aware compliance rules. In *ICSOC/ServiceWave*, volume 5900 of *LNCS*, pages 500–515, 2009.
6. A. Awad and M. Weske. Visualization of compliance violation in business process models. In *BPI*, LNBIP. Springer-Verlag, 2009.
7. F. S. Commission. Guidelines on anti-money laundering & counter-financing of terrorism, 2007.

8. P. Dasgupta. *A Roadmap for Formal Property Verification*. Springer, 2006.
9. R. M. Dijkman, M. Dumas, and C. Ouyang. Semantics and Analysis of Business Process Models in BPMN. *Inf. Softw. Technol.*, 50(12):1281–1294, 2008.
10. M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *ICSE*, pages 411–420, New York, NY, USA, 1999. ACM.
11. M. El Kharbili, S. Stein, I. Markovic, and E. Pulvermüller. Towards a Framework for Semantic Business Process Compliance Management. In *GRCIS*, pages 1–15, June 2008.
12. A. Förster, G. Engels, T. Schattkowsky, and R. Van Der Straeten. Verification of Business Process Quality Constraints Based on VisualProcess Patterns. In *TASE*, pages 197–208. IEEE Computer Society, 2007.
13. P. Gastin and D. Oddoux. Fast LTL to Büchi Automata Translation. In *CAV*, volume 2102 of *LNCS*, pages 53–65. Springer, 2001.
14. R. Gerth, D. D. Eindhoven, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification Testing and Verification*, pages 3–18. Chapman & Hall, 1995.
15. A. Ghose and G. Koliadis. Auditing Business Process Compliance. In *ICSOC*, volume 4749 of *LNCS*, pages 169–180. Springer, 2007.
16. S. Goedertier and J. Vanthienen. Compliant and Flexible Business Processes with Business Rules. In *BPMDS*, volume 236 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.
17. S. Goedertier and J. Vanthienen. Designing Compliant Business Processes from Obligations and Permissions. In *(BPD)*, volume 4103 of *LNCS*, pages 5–14. Springer Verlag, 2006.
18. M. Kähmer, M. Gilliot, and G. Muller. Automating privacy compliance with expdt. In *CEC/EEE*, pages 87–94. IEEE, 2008.
19. J. M. Küster, K. Ryndina, and H. Gall. Generation of Business Process Models for Object Life Cycle Compliance. In *BPM*, volume 4714 of *LNCS*, pages 165–181. Springer, 2007.
20. F. Laroussinie, N. Markey, and P. Schnoebelen. Temporal logic with forgettable past. In *LICS*, pages 383–392. IEEE Computer Society, 2002.
21. R. Lu, S. Sadiq, and G. Governatori. Compliance Aware Business Process Design. In *BPM Workshops*, volume 4928 of *LNCS*, pages 120–131. Springer, 2007.
22. R. Lu, S. Sadiq, and G. Governatori. Measurement of Compliance Distance in Business Processes. *Inf. Sys. Manag.*, 25(4):344–355, 2008.
23. Y. Lui, S. Müller, and K. Xu. A Static Compliance-checking Framework for Business Process Models. *IBM SYSTEMS JOURNAL*, 46(2):335–362, 2007.
24. L. T. Ly, K. Göser, S. Rinderle-Ma, and P. Dadam. Compliance of Semantic Constraints – A Requirements Analysis for Process Management Systems. In *GRCIS*, pages 16–30. CEUR-WS.org, June 2008.
25. Z. Milosevic, S. Sadiq, and M. Orlowska. Translating Business Contract into Compliant Business Processes. In *EDOC*, pages 211–220. IEEE Computer Society, 2006.
26. K. Namiri and N. Stojanovic. Pattern-Based Design and Validation of Business Process Compliance. In *OTM Conferences (1)*, volume 4803 of *LNCS*, pages 59–76. Springer, 2007.
27. OASIS. Universal Description Discovery and Integration UDDI, 2004.
28. B. C. on Banking Supervision. Basel ii accord, 2004.
29. M. Pesić. *Constraint-Based Workflow Management System: Shifting Control to Users*. PhD thesis, Technische Universiteit Eindhoven, 2008.
30. A. Pnueli. The temporal logic of programs. In *SFCS*, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society.
31. J. Yu, T. P. Manh, J. Han, Y. Jin, H. Y., and J. Wang. Pattern Based Property Specification and Verification for Service Composition. In *WISE*, volume 4255 of *LNCS*, pages 156–168. Springer, 2006.
32. L. Zuck. *Past Temporal Logic*. PhD thesis, Weizmann Intitute, Israel, August 1986.