Monitoring Business Process Compliance Using Compliance Rule Graphs

Linh Thao Ly[1], Stefanie Rinderle-Ma[2], David Knuplesch[1], and Peter Dadam[1]

[1] Institute of Databases and Information Systems, Ulm University, Germany
[2] Faculty of Computer Science, University of Vienna, Austria
{thao.ly,david.knuplesch,peter.dadam}@uni-ulm.de,
stefanie.rinderle-ma@univie.ac.at

Abstract. Driven by recent trends, effective compliance control has become a crucial success factor for companies nowadays. In this context, compliance monitoring is considered an important building block to support business process compliance. Key to the practical application of a monitoring framework will be its ability to reveal and pinpoint violations of imposed compliance rules that occur during process execution. In this context, we propose a compliance monitoring framework that tackles three major challenges. As a compliance rule can become activated multiple times within a process execution, monitoring only its overall enforcement can be insufficient to assess and deal with compliance violations. Therefore, our approach enables to monitor each activation of a compliance rule individually. In case of violations, we are able to derive the particular root cause, which is helpful to apply specific remedy strategies. Even if a rule activation is not yet violated, the framework can provide assistance in proactively enforcing compliance by deriving measures to render the rule activation satisfied.

1 Introduction

In many application domains of information systems, business process compliance is increasingly gaining importance. In healthcare, for example, medical guidelines and clinical pathways should be followed during patient treatments. In the financial sector, regulatory packages such as SOX or BASEL III have been introduced to strengthen customers' confidence in bank processes. Finally, collections of quality controls, e.g., Six Sigma or ITIL, are of particular importance for the internal control of business processes. In this paper we assume an example scenario that is set in bank accounting, where a variety of rules and policies exists (e.g., as a result of risk management). A selection of such rules is listed below:

- c_1 Conducting a payment run creates a payment list containing multiple items that must be transferred to the bank. Then, the bank statement must be checked for payment of the corresponding items. In order to avoid fraud and errors, the payment list must be transferred to the bank only once.
- c_2 For payment runs with amount beyond €10,000, the payment list has to be signed before being transferred to the bank and has to be filed afterwards for later audits.

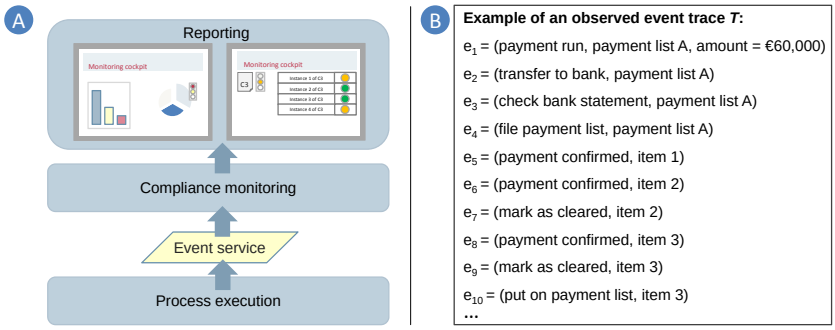

Fig. 1. General compliance monitoring architecture and events from the bank accounting case

c_3 When payment of an open item is confirmed in the bank statement, the item has to be marked as cleared eventually.

c_4 Once marked as cleared, the item must not be put on any payment list.

c_5 If an open item is not marked as cleared within 30 days, the bank details may be faulty. Thus, the bank details have to be (re)checked.

One way to ensure business process compliance is to verify the models of the affected business processes implemented within process-aware information systems against such imposed rules in order to achieve compliance by design. For that purpose, a multitude of approaches for business process model checking has been proposed in literature. However, design time checks are not quite sufficient as in many application domains, business process models are rarely documented or adhered to. To support business process compliance in such scenarios, it must be possible to monitor whether process executions (regardless whether an explicit process model exists) comply with imposed rules.

1.1 Challenges for Compliance Monitoring

Fig. 1A depicts a general compliance monitoring architecture. The events observed from process execution provide the basis for compliance monitoring. The actual low-level execution events may also be aggregated to meaningful events and then provided to the monitoring tier using event processing frameworks (e.g., complex event processing [1]). The monitoring tier, in turn, monitors compliance with imposed rules. It provides input to the reporting tier, where the results are visualized in accordance with the needs of the stakeholders involved. We assume that events such as the events from the bank accounting case listed in Fig. 1B can be provided by the event service. Clearly, the information that the monitoring tier is capable of providing constitutes the basis for presentation and visualization to stakeholders such as the process supervisor. This raises three fundamental challenges that we will discuss using the bank accounting case.

Challenge 1: Identification and monitoring of individual activations of a compliance rule Consider again compliance rule c_4. Then, a closer look at trace T from Fig. 1B reveals that c_4 is violated over T as *item 2* and *item 3* are already marked as cleared but *item 3* is still put on a payment list again. In order to effectively deal with such violations, it is not sufficient for the monitoring tier just to identify that c_4 is violated. Imagine that multiple other items are also marked as cleared within T. Then, it becomes very difficult to pinpoint the item(s) causing violations. Hence, the monitoring tier must provide fine-grained compliance feedback such that it becomes possible to pinpoint the violation.

In this example, two *activations* of c_4 are present in the execution, namely for *item 2* and *item 3*. We refer to an event pattern that activates a compliance rule as a rule activation. For example, c_4 becomes activated when an item is marked as cleared. As activities can be carried out multiple times, e.g., for different items, there can be multiple activations of a compliance rule in a process execution[3]. As the example also shows, the activations can be in different compliance states. For example, the activation for *item 3* is violated while the one for *item 2* is not. Therefore, the monitoring tier must be able to identify the rule activations and to provide information on their individual compliance state as the basis for effectively assessing and dealing with incompliance.

Challenge 2: Proactive prevention of violations Generally, each activation of a compliance rule can be in one of three compliance states in a stage of process execution: satisfied, violated, or violable. Satisfied and violated are permanent states. For example, the activation of c_4 for *item 3* is violated while the activation of c_3 for *item 2* is satisfied. A violable activation, however, can become both violated or satisfied depending on the events observed in the future.

The state violable can have very different semantics depending on the rule activations. Consider for example the activation of c_3 for *item 1* (referred to as ACT1) and the activation of c_4 for *item 2* (referred to as ACT2). Then, both are violable as ACT1 can become violated if *item 1* is not marked as cleared in the future and ACT2 can become violated if *item 2* is put on a payment list again. Obviously, different measures are necessary to render ACT1 and ACT2 satisfied. In order to proactively prevent violations, the monitoring tier has to make the state violable more transparent to process supervisors. In particular, support with regard to how to render an activation satisfied is desirable. Being aware that ACT1 can be rendered satisfied by marking *item 1* as cleared, a process supervisor may, for example, schedule this task. To our best knowledge, challenge 2 has not been addressed yet.

Challenge 3: Root cause identification in case of violations The identification of violations is only a first step. A rule activation can become violated, when required events were not observed or prohibited events were observed within a particular scope. Particularly for more complex rules, a violation can

[3] Note that there can also be rules that are only activated once, e.g., cardinality rules. We consider them as being activated upon the process start.

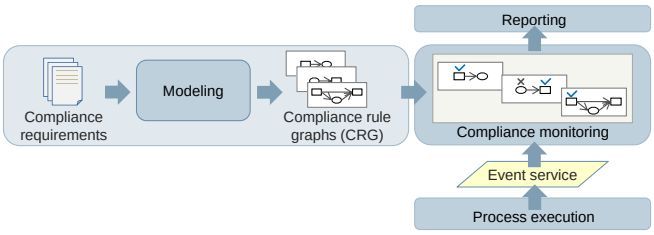

Fig. 2. Fundamental architecture of the SeaFlows compliance monitoring framework

have multiple causes, each of which may require different compensation mechanisms. Therefore, being able to identify the root cause of a violation would facilitate the application of adequate compensation.

1.2 Contributions

In this paper, we propose an approach developed in the SeaFlows project[4] that tackles these three challenges. The basic architecture of the framework is depicted in Fig. 2. We adopt a graph-based compliance rule language referred to as Compliance Rule Graphs (CRG) [2]. Our approach enables to "instantiate" a CRG each time a new activation of it is observed and thus to individually monitor the activations. As a result, feedback can not only be provided on the overall enforcement of a compliance rule but also on its activations. Our monitoring approach is based on a pattern matching mechanism that exploits the structure of CRGs for monitoring and introduces markings to indicate observed event patterns that are relevant to the compliance rule to be checked. From the markings, it is possible to derive measures to proactively prevent violations in case of violable rule activations, for example to derive pending activities. For violated activations, we can derive the root cause from the markings.

In the following, we first introduce necessary fundamentals on CRGs in Sect. 2. The monitoring framework is then presented in Sect. 3. Sect. 4 introduces our proof-of-concept implementation. In Sect. 5, we discuss related work. Sect. 6 summarizes the paper and provides an outlook on future research.

2 Compliance Rule Graph Fundamentals

For our monitoring framework we adopted compliance rule graphs (CRG), a graph-based compliance rule modeling language [2], as the graph structure has some advantages that we can exploit for monitoring as we will later show. The CRG language adopts the typical rule structure (i.e., if some conditions apply, then some consequence must also apply). As we address compliance rules on the occurrence, absence, and ordering relations of events, the rule antecedent

[4] www.seaflows.de

and rule consequences are constituted by event patterns, respectively. Their explicit structure makes it easier to comprehend CRGs than for example complex logic formulas. For brevity reasons, we will focus on compliance rules with only one consequence event pattern in this paper. Our approach is, however, also applicable to compliance rules with multiple consequence patterns.

Based on the assumption adopted from graph-based process modeling languages that a graph is a suitable representation for expressing occurrence and ordering relations of events, the event patterns associated with the rule antecedent and the rule consequence are modeled by means of directed graphs. In order to distinguish between the antecedent and the consequence, they are modeled using designated node types (condition and consequence node types). Thus, from their looks, CRGs are acyclic directed graphs with different node types with a graph fragment describing the rule antecedent pattern and a graph fragment describing the rule consequence pattern.

These event patterns can be modeled using occurrence nodes, i.e., nodes that represent the occurrence of events of associated type and properties (e.g., data conditions) and edges constraining their ordering (similar to what we are used to from process modeling)[5]. Beside occurrence nodes, absence nodes representing the absence of certain events can be used to further refine the event patterns. By combining these modeling primitives, it is possible to model sophisticated event patterns, which can serve as antecedent or consequence of a CRG. Def. 1 provides a basic definition of CRGs.

Definition 1 (Compliance rule graph). *A compliance rule graph is a 7-tuple* $R = (N_A, N_C, E_A, E_C, E_{AC}, nt, p)$ *where:*

- N_A *is a set of nodes of the antecedent graph of* R*,*
- N_C *is a set of nodes of the consequence graph of* R*,*
- E_A *is a set of directed edges connecting nodes of* N_A*,*
- E_C *is a set of directed edges connecting nodes of* N_C*,*
- E_{AC} *is a set of directed edges connecting nodes of the antecedent and the consequence graph of* R*,*
- $nt : N_A \cup N_C \rightarrow \{\texttt{AnteOcc}, \texttt{AnteAbs}, \texttt{ConsOcc}, \texttt{ConsAbs}\}$ *is a function assigning a node type to the nodes of* R*, and*
- p *is a function assigning a set of properties (e.g., activity type, data conditions) to each node of* R*.*

Assuming an existing event model, Fig. 3A depicts the modeling of the CRG for compliance rule c_1 in two steps. Apparently, c_1 is activated by the *payment run* creating a *payment list*. This is modeled through the corresponding `AnteOcc` node. As c_1 requests the *payment run* to be followed by the event *transfer to bank* and the subsequent event *check bank statement* for the created *payment list*, `ConsOcc` nodes are used to model this consequence pattern. In a second step, the consequence CRG is refined to capture the condition that the *payment list* must

[5] Note that the antecedent pattern can also be left empty to model for example cardinality constraints.

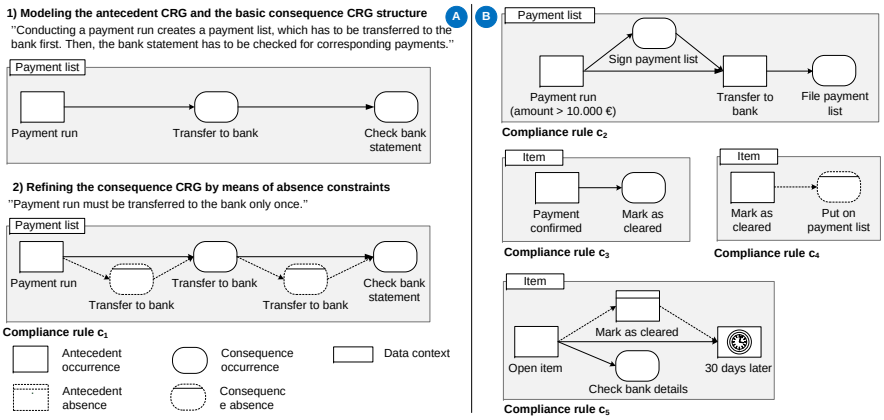

Fig. 3. Step by step modeling CRG for c_1 (A) and CRGs for c_2 - c_5 (B)

be transferred to the bank only once. This condition is captured by `ConsAbs` nodes signifying the requested absence of additional events of type *transfer to bank*. In this manner, we can also model the other compliance rule examples by means of CRG as depicted in Fig. 3 B. For example, the CRG for compliance rule c_5 expresses that in case an *open item* event is followed by a time event representing 30 days later without having recorded *mark as cleared*, the bank details have to be checked. Note that we assume an event processing framework that can deliver such time events.

Intuitively, an event pattern of a CRG (regardless of whether it is an antecedent or a consequence pattern) matches a set of events if the occurrence nodes and the ordering relations match a set of nodes and the absence constraints expressed through the absence nodes are satisfied. If the antecedent pattern is composed from only `AnteAbs` nodes, then there can be only one match of the antecedent (if the absence constraints apply). Each match of the antecedent event pattern constitutes an *activation* of the corresponding CRG. For example, the sequence of the events e_1 (*payment run* with amount beyond €10,000) and e_2 (*payment list A* is transferred to the bank) from Fig. 1 constitutes a match of the antecedent of c_2. CRGs with empty antecedent pattern are activated upon the process start.

Definition 2 (Semantics of CRGs). *Let $R = (N_A, N_C, \dots)$ be a CRG and $\sigma =< e_1, \dots, e_n >$ be an execution trace. Then, R is satisfied over σ iff:*

- *for R with non-empty antecedent pattern holds: for each match of the antecedent pattern of R in σ, there is also a corresponding match of R's consequence pattern in σ and*
- *for R with empty antecedent pattern holds: there is also a match of R's consequence pattern in σ.*

Due to their explicit structure, verbalization, a technique known from business rule modeling, can be easily realized for CRGs. While CRG is a compo-

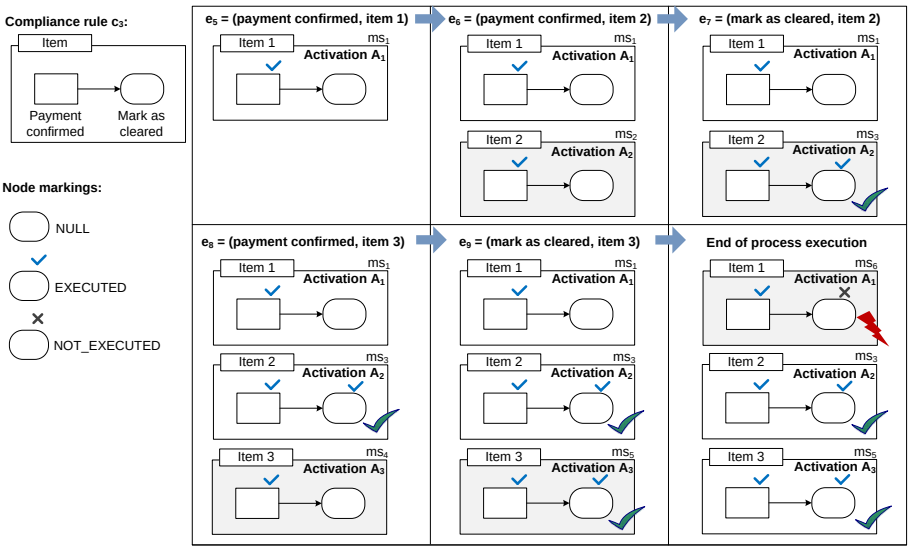

Fig. 4. Observed patterns with regard to CRG c_3 when processing events e_5 - e_9

sitional language, we can also use CRGs to model frequent rule patterns, for example [3]. Due to space limitations, we abstain from going into further details on the properties of CRGs (e.g., syntactic correctness, further modeling primitives). Further details can be found in [2]. In this paper, we only focus on a subset of the CRG language that is sufficient to illustrate our monitoring framework.

3 Compliance Monitoring

Generally, it would be possible to monitor compliance with an imposed CRG by transforming it into an automaton or by generating event queries (e.g., for complex event processing [1]) from it. The benefits and drawbacks of these approaches are discussed in Section 5. For example, addressing challenge 1 is cumbersome when employing the automaton approach [4]. The beauty of our approach is that no transformation of the modeled compliance rules (in the form of CRGs) into other representations is necessary in order to enable monitoring. A CRG is instead monitored by exploiting its graph structure. Thus, feedback can be provided specifically on the basis of the structure of the very CRG leaving no gap between the modeled rule and the feedback mechanism.

The basic idea behind the approach is illustrated by Fig. 4 using the example of CRG c_3. Fig. 4 depicts event patterns that are relevant to c_3 and that become observable in different stages of process execution when processing the events of trace T from Fig. 1. Instead of textually describing these patterns, we use the graph structure of CRGs and node markings that indicate whether or not an event was observed to capture these observable patterns. New patterns are marked in grey color.

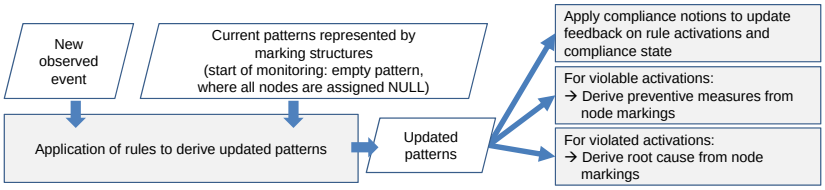

Fig. 5. Updating compliance feedback when a new event is observed

Example 1 Consider Fig. 4 and c_3. Then, after observing e_1-e_5, the pattern "*payment confirmed* for *item 1* without subsequent *mark as cleared* yet" becomes observable in the execution trace. This pattern is captured by ms_1 using the node state `EXECUTED` to indicate that *payment confirmed* was already observed. When observing e_6, a similar pattern can be formed for *item 2*. Obviously, ms_1 and ms_2 each constitutes an activation of c_3. When observing e_7, ms_2 is no longer current but instead, the situation can be represented by ms_1 and ms_3, where ms_3 reflects the pattern "*payment confirmed* for *item 2* with subsequent *mark as cleared*". Thus, ms_3 constitutes a satisfied activation of c_3. Observation of e_8 yields a new pattern for *item 3* represented by ms_4. This pattern is replaced by ms_5 after *item 3* is marked as cleared (event e_9). If the process execution would be terminated, ms_1 would be no longer current for activation A_1 as *item 1* will not be marked as cleared. Thus, ms_1 is updated to ms_6. Altogether, after execution of e_1-e_9 and the termination of the process execution, the compliance with c_3 is reflected by the patterns ms_6, ms_3, and ms_5, where each of this represents an activation of c_3. While the activations A_2 and A_3 are satisfied as the required events were observed, activation A_1 for *item 1* is violated as it was not marked as cleared as indicated by the `CONSOCC` node marked with `NOTEXECUTED`.

In the example, we built the observable patterns manually. Inspired by pattern matching, our monitoring framework automatically identifies all activations of a CRG to be checked and further tries to identify a match of the consequence pattern in the execution trace. For that purpose, it builds such patterns based on the graph structure of the CRG to be checked that become observable in the partial execution trace as illustrated in the example. However, in the monitoring framework, these patterns are not built from scratch each time a new event is observed. Instead, new observable patterns are *derived* from existing patterns by applying defined rules when observing a new event. Fig. 5 summarizes the overall procedure when a new event is observed.

Each relevant observable pattern represented by marking the CRG as illustrated in the example is stored in a data structure called *marking structure* (in brief `MS`). From the node semantics and the node markings, it can be derived whether a `MS` represents a rule activation. Also the individual compliance state of the rule activation can be determined this way (challenge 1). For a violable rule activation, measures can be derived from the node markings to proactively enforce the satisfaction, for example, the pending activities can be identified

(challenge 2). In case of violation, the node markings further enable root cause analysis without additional cost (challenge 3).

In the following, we first formalize the MSs and introduce formal notions to assess them with regard to compliance in Section 3.1. In Section 3.2, we introduce the algorithm for deriving updated observed patterns from old patterns when a new execution event is observed. Finally, we further show how the challenges 2 and 3 can be dealt with in Section 3.3.

3.1 CRG Markings and Compliance Notions

In Fig. 4, we already introduced the node markings that are used to indicate whether or not an event was observed: A CRG node n marked with NULL signifies that no matching event is observed yet. Regardless of the node type, a CRG node n in a pattern marked with EXECUTED means that a matching event has been observed. A CRG node n marked with NOTEXECUTED means that the associated event has not been and will not be observed (e.g., when the window for an event to occur has elapsed). Using the node markings, we can use the specific CRG structure to express relevant patterns that become observable in the execution trace. During compliance monitoring for a CRG, each such pattern is captured in a data structure called *marking structure* (MS), e.g., ms_1 in Fig. 4. In particular, a MS captures a (potential) activation of a CRG observable from the trace. It contains a marking for each antecedent node of the CRG and multiple markings for the CRG's consequence pattern[6]. Def. 3 formalizes the notion of MSs.

Definition 3 (CRG MS). *Let $R = (N_A, N_C, \dots)$ be a CRG and $NodeStates :=$ $\{\text{NULL}, \text{EXECUTED}, \text{NOTEXECUTED}\}$ be the set of execution states of CRG nodes. Then, a CRG MS of R is defined as a 3-tuple*
$ms := (ns_A, ev_A, \{(ns_C^1, ev_C^1), \dots, (ns_C^k, ev_C^k)\})$ where

- *$ns_A : N_A \rightarrow NodeStates$ is a function assigning an* execution state *to each node of A.*
- *ev_A is a function assigning an observed execution event (or a dummy event in case $ns_A(n) \in \{\text{NULL}, \text{NOTEXECUTED}\}$) to each node of A. We denote (ns_A,ev_A) as* ANTEMARK *of R.*
- *$ns_C^i : N_C \rightarrow NodeStates$ is a function assigning an* execution state *to each node of C.*

[6] The rationale behind this is that depending on the particular CRG, the pattern matching procedure may has to try different options to form a match of the consequence pattern out of the events contained in the execution trace. Consider for example the rule "After A, there has to a B that is not followed by a C". Then, assuming that an A is present in the trace, the first subsequent B may not lead to a match of the consequence as it can still be followed by a C. In this case, it becomes necessary to also explore other options, which results in multiple markings for the consequence pattern. This only becomes necessary for the particular case of CONSOCC nodes with direct CONSABS successors and is taken into account by our pattern matching mechanism (cf. Section 3.2).

– ev^i_C *is a function assigning an observed execution event (or a dummy event in case* $ns^i_C(n) \in \{\textsc{Null}, \textsc{NotExecuted}\}$*) to each node of* C*. We denote* (ns^i_C, ev^i_C) *as* ConsMark *of* R*.*

Example 2 Consider ms_1 from Fig. 4:

– $ms_1 = (ns_A, ev_A, \{(ns_C, ev_C)\})$ with
– $ns_A(payment\ confirmed) = \textsc{Executed}$, $ev_A(payment\ confirmed) = e_5$,
– $ns_C(mark\ as\ cleared) = \textsc{Null}$, and $ev_C(mark\ as\ cleared) = no\ event$.

As illustrated by Fig. 4, in each stage of process execution, the compliance with an imposed CRG can be reflected by a set of MSs. To assess these MSs, we can benefit from the node semantics and the node markings. Generally, for a CRG's antecedent or consequence pattern composed from occurrence and absence nodes, a match in the execution trace is found if all events associated with occurrence nodes are observed (i.e., marked as Executed) and for all absence nodes, no matching events were observed (i.e., marked as NotExecuted). If the antecedent is marked accordingly, the corresponding MS constitutes an activation of the CRG. Recall that a rule activation is satisfied if a match of the CRG's consequence can also be found in the execution trace. Thus, the activation is satisfied if the MS also contains a ConsMark that is marked as described. Def. 4 formalizes this intuition:

Definition 4 (Compliance Notions for MSs). *Let* $R = (N_A, N_C, \dots)$ *be a CRG and* $ms = (ns_A, ev_A, \{(ns^1_C, ev^1_C), \dots, (ns^k_C, ev^k_C)\})$ *be a* MS *of* R*. Then,*

– *we will say* ms *is* activated *if the following holds:*
  • $\forall n \in N_A : nt(n) = \textsc{AnteOcc} \Rightarrow ns_A(n) = \textsc{Executed} \wedge$
    $\forall n \in N_A : nt(n) = \textsc{AnteAbs} \Rightarrow ns_A(n) = \textsc{NotExecuted}$

For an activated ms*, we further distinguish between the following states:*

– ms *is* satisfied *if the following holds:*
  • $\exists ns^i_C, i \in \{1, \dots, k\}$:
    $(\forall n \in N_C : nt(n) = \textsc{ConsOcc} \Rightarrow ns^i_C(n) = \textsc{Executed}) \wedge$
    $(\forall n \in N_C : nt(n) = \textsc{ConsAbs} \Rightarrow ns^i_C(n) = \textsc{NotExecuted})$
– ms *is* violated *if the following holds:*
  • $\forall ns^i_C, i \in \{1, \dots, k\}$:
    $(\exists n \in N_C : nt(n) = \textsc{ConsOcc} \Rightarrow ns^i_C(n) = \textsc{NotExecuted}) \vee$
    $(\exists n \in N_C : nt(n) = \textsc{ConsAbs} \Rightarrow ns^i_C(n) = \textsc{Executed})$
– *Otherwise,* ms *is considered* violable.

Example 3 Consider again Fig. 4. Then,

– ms_1, ms_3, and ms_6 are all activated, i.e., they constitute activations of c_3.
– ms_1 is violable, ms_3 is satisfied, while ms_6 is violated.

Def. 4 enables us to assess obtained MSs. Altogether, this enables the process supervisor to get an overview on rule activations in the process execution and provides basic information on their compliance state. In Section 3.3, we will further discuss how the monitoring framework can be used to assist the process supervisor in identifying the root cause for violations and even in preventing violations. Before that, we first introduce the pattern matching mechanism operating on MSs behind our framework in Section 3.2.

3.2 The Pattern Matching Mechanism

As mentioned, each MS represents a (potential) activation of the CRG. As indicated in Fig. 5, the monitoring of a CRG starts with a MS where all nodes are assigned Null (i.e., no events observed yet). How to derive updated MSs from existing MSs when a new event is observed? The pattern matching mechanism of the framework is based on three considerations:

1: The objective is to identify all rule activations present in the execution trace. For that purpose, it becomes necessary to explore different options to form a match of the CRG's antecedent pattern out of the events in the trace in the pattern matching process.

2: For each MS, the objective is further to identify a match of the consequence CRG (cf. Def. 2). For that purpose, we try to explore only one option if possible to increase the efficiency. Alternative options are only explored if necessary.

3: Exploit the ordering of nodes for pattern matching: A node can only match an event, if the node and event specification match and the node is not yet assigned to another event. Additionally, also matching events for relevant predecessors must have already been found. In particular, for AnteOcc and AnteAbs nodes, AnteOcc predecessors must be already marked as Executed. For ConsOcc and ConsAbs nodes, AnteOcc and ConsOcc predecessors must be marked as Executed.

Based on these considerations, algorithm 1 derives updated patterns from a MS when an event is observed. The outer loop implements consideration 1. The inner loop in line 19 updates the observable patterns with regard to the consequence CRG (represented by the ConsMarks). It further implements consideration 2, as only for particular nodes, namely ConsOcc nodes with direct ConsAbs successors, alternative options have to be explored.

Example 4 Fig. 6A applies algorithm 1 to c_2 over the events $< e_1, e_2, e_4 >$[7]. New MSs are highlighted. The monitoring starts with ms_1. When e_1 is observed, application of algorithm 1 yields both ms_1 and ms_2. Here, ms_1 enables the recognition of future activations of c_2, while ms_2 explores whether e_1 leads to a rule activation[8]. So far, no activation of c_2 is observed yet. When observing

[7] e_3 is irrelevant to c_2.

[8] Note that when manually conducting pattern matching (cf. Fig. 4), we would typically not identify ms_1. However, such not yet matching patterns are important to enable automatically deriving updated patterns from existing patterns.

Algorithm 1 Deriving updated patterns from a MS over an event

$R = (A, C, \dots, \dots)$ is a CRG; e is an observed event; $MS_{Res} = \emptyset$;
$ms = (ns_A, ev_A, \{(ns_C^1, ev_C^1), \dots, (ns_C^k, ev_C^k)\})$ is a MS of R;
{CRG nodes that match e (cf. consideration 3):}
$N_{AnteOcc}$ is the set of ANTEOCC, $N_{AnteAbs}$ is the set of ANTEABS nodes matching e ;
$N^i_{ConsOcc}, i = 1, \dots, k$; is the set of CONSOCC nodes matching e of (ns_C^i, ev_C^i);
$N^i_{ConsAbs}, i = 1, \dots, k$; is the set of CONSABS nodes matching e of (ns_C^i, ev_C^i);
for all $Q \in \mathcal{P}(N_{AnteOcc})$ **do**
  create a copy ms' of ms;
  for all $n \in Q$ **do**
    $ns'_A(n) =$ EXECUTED; $ev'_A(n) = e$;
    mark all ANTEABS predecessors of n with $ns'_A(n) =$ NULL as NOTEXECUTED;
    for all CONSMARKS (ns_C^i, ev_C^i) of ms' **do**
      mark all CONSOCC predecessors of n with $ns_C^i(n) =$ NULL as NOTEXECUTED;
      mark all CONSABS predecessors of n with $ns_C^i(n) =$ NULL as NOTEXECUTED;
    end for
  end for
  for all n with $n \in N_{AnteAbs} \wedge ns'_A(n) =$ NULL **do**
    $ns'_A(n) =$ EXECUTED; $ev'_A(n) = e$;
  end for
  $CM = \emptyset$;
  for all CONSMARKS $cm = (ns_C^i, ev_C^i)$ of ms' **do**
    $N = N^i_{ConsOcc} \backslash \{n \in N_C \mid nt(n) =$ CONSOCC $\wedge\, ns_C^i(n) =$ NOTEXECUTED$\}$;
    $D = \{n \in N \mid n$ *has no direct* CONSABS *successor* $\}$;
    $I = N \backslash D$;
    for all $Q = D \cup T, T \in \mathcal{P}(I)$ **do**
      create a copy $cm' = (ns'^i_C, ev'^i_C)$ of cm;
      for all $n \in Q$ **do**
        $ns'^i_C(n) =$ EXECUTED; $ev'^i_C(n) = e$;
        mark all CONSABS predecessors of n with $ns'^i_C(n) =$ NULL as NOTEXECUTED;
      end for
      for all n with $n \in N^i_{ConsAbs} \wedge ns'^i_C(n) =$ NULL **do**
        $ns'^i_C(n) =$ EXECUTED; $ev'^i_C(n) = e$;
      end for
      $CM = CM \cup \{cm'\}$;
    end for
  end for
  set CONSMARKS of ms'=CM;
  $MS_{res} = MS_{res} \cup \{ms'\}$;
end for
return MS_{res};

e_2, ms_1 remains unaffected. However, ms_2 results in ms_2 and ms_3. Here, ms_2 enables the recognition of possible future rule activations in combination with e_1. The compliance notions (cf. Def. 4) reveal that ms_3 constitutes an activation of c_2, A_1, that is already VIOLATED as the payment list was not signed before being transferred to the bank. In this case, the process supervisor can be notified. If in practice the activity *transfer to bank* can be put on hold, the system could even suspend its execution being aware that the activity leads to incompliance. Despite the violation, the monitoring of A_1 can still be continued. Thus, when observing e_4, ms_3 yields ms_4. Activation A_1 is still violated, but nevetheless due to one but not two causes as we will later discuss in Section 3.3.

Example 5 Fig. 6B applies algorithm 1 to c_4 over the events $< e_7, e_9, e_{10} >$. As *mark as cleared* occurs twice the execution (as e_7 for *item 2* and as e_9 for

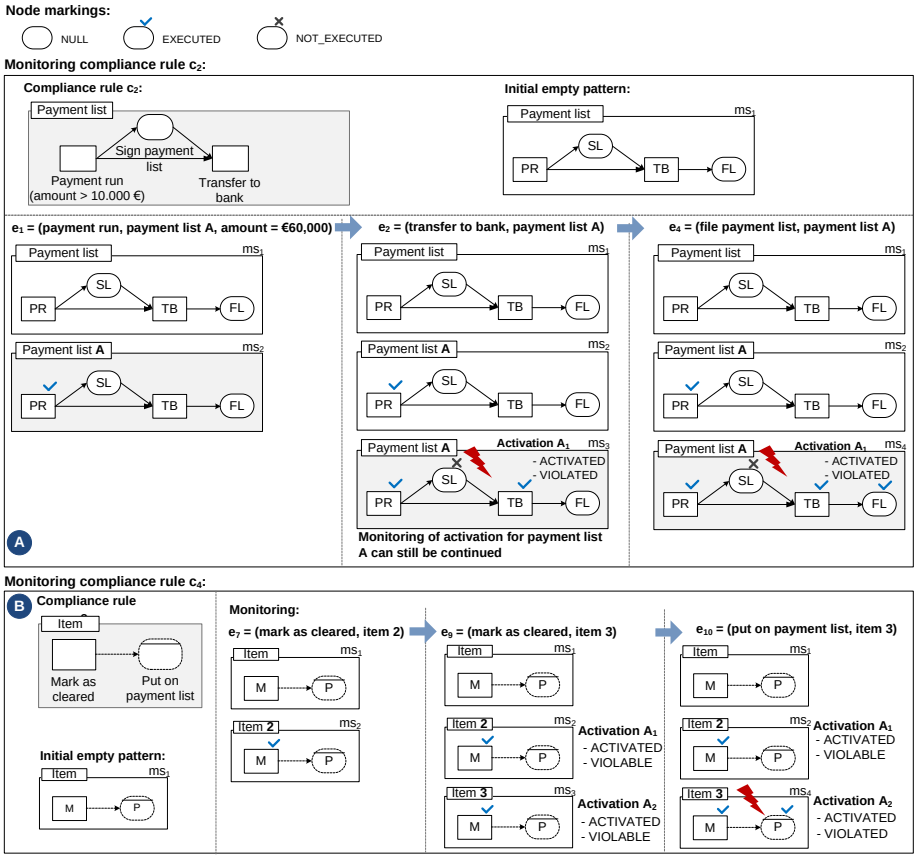

Fig. 6. Monitoring c_2 over $< e_1, e_2, e_4 >$ (A) and c_4 over $< e_7, e_9, e_{10} >$ (B)

item 3), compliance monitoring reveals two activations of c_4, namely A_1 and A_2, after observing e_7 and e_9. At that stage, both activations are VIOLABLE. However, *item 3* is later put on a payment list again (indicated by e_{10}). As a result, activation A_2 becomes VIOLATED as the absence constraint is violated.

3.3 Prevention of Violations and Root Cause Analysis

Prevention of violations A VIOLABLE rule activation can become both SATISFIED or VIOLATED depending on future events. Due to its graph notation, such a MS can be presented to the process supervisor if required in order to identify measures to prevent a violation. Additionally, the system can assists in preventing violations by deriving concrete actions in order to render the activation SATISFIED. Based of Def. 4, the rule activation becomes SATISFIED when a match of the consequence is found. Thus, from a ConsMark (ns_C, ev_C) that can still lead to

a match of the consequence, we can derive actions to satisfy the corresponding activation as follows:

- Each ConsOcc node n with $ns_C(n) = \text{Null}$ represents a still **pending activity**.
  Possible action: Schedule the pending activity[9].
  Example: Consider ms_1 from Fig. 4. Then, ConsOcc node *mark as cleared* is pending as it is still marked as Null. To satify this activation, the corresponding activity can be scheduled, for example, by putting it into an agent's worklist.
- Each ConsAbs node n with $ns_C(n) = \text{Null}$ that does not have any ConsOcc predecessors still in state Null represents an *active absence constraint*. The absence of the corresponding event is necessary in order for this ConsMark to constitute a match of the consequence CRG.
  Possible action: Deactivate the corresponding activity until n is marked as NotExecuted (e.g., when the window of n elapsed).
  Example: Consider ms_2 from Fig. 6B. Then, ConsAbs node *put on payment list* represents an active absence constraint. As no ConsOcc nodes are pending, immediate end of process execution would render this activation satisfied. To enforce compliance, the activity *put on payment list* can be deactivated for *item 2*.

As the pattern matching mechanism employs a rather greedy strategy to identify a match of the consequence, the thus derivied action chains constitute the "shortest" ways to enforce compliance.

Root cause identification In a similar manner to violation prevention, the root cause of a violated rule activation can be easily derived from a ConsMarks. Generally, an activation can become violated if required events do not occur or / and prohibited events occur during process execution. These causes are also reflected in the MS. For a ConsMark (ns_C, ev_C) of a violated MS, we can identify why it does not constitute a match of the consequence CRG:

- Each ConsOcc node n with $ns_C(n) = \text{NotExecuted}$ represents a **required activity missing** in the pattern.
  Example: Consider ms_4 from Fig. 6A. Then, the missing event *sign payment list* before transferring *payment list A* to the bank can be precisely identified as the root cause for the violation of rule activation A_1.
- Each ConsAbs node n with $ns_C(n) = \text{Executed}$ represents an **prohibited activity observed** in the pattern.
  Example: Consider ms_4 from Fig. 6B. Then, the prohibited event *put on payment list* for *item 3* is identified as the root cause for the violation.

[9] Since CRGs are acyclic, we can further derive a process to be scheduled by adopting the ordering relations of the occurrence nodes in case multiple activities are pending.

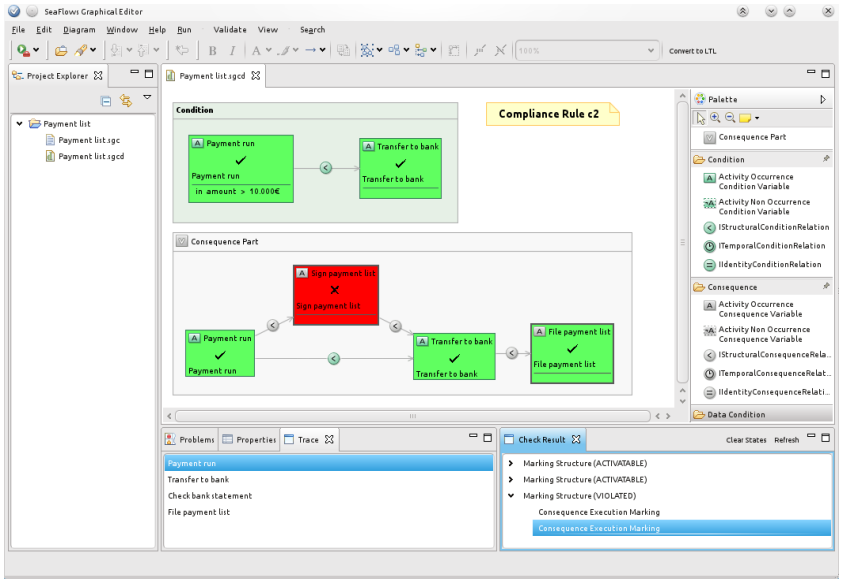

Fig. 7. Execution of c_2 over $< e_1, e_2, e_4 >$ and ms_4 as one of the resulting MSs

4 Implementation

We impemented our monitoring approach in the SeaFlows Toolset [5] that comprises a variety of tools for supporting compliance throughout the process lifecycle. The SeaFlows Compliance Monitor is integrated into the AristaFlow BPM Suite that is based on ADEPT [6]. CRGs are modeled using a graphical editor and stored as XML files. For convenient compliance rule modeling, the SeaFlows Compliance Rule Editor allows for modeling parametrized compliance rules patterns that can be reused whenever a rule with a similar structure is required. Fig. 7 shows the rule activation captured by MS ms_4 (cf. Fig. 6A) obtained when observing the event sequence $< e_1, e_2, e_4 >$ from Fig. 1. The root cause of the violation is visualized directly in the CRG by using the node markings and the color highlighting. This enables process supervisors to easily pinpoint violations and to apply root cause specific remedies, for example initiate an audit activity as the transferred payment list was not signed before being transferred to the bank.

5 State-of-the-art

Most work addressing process monitoring focus on data consistency or process performance (e.g., KPI monitoring or business activity monitoring). Several approaches address root cause analysis in the context of design time verification [7, 8, 5]. Generally, for a predefined set of rule patterns, possible types of violations

can be anticipated, which can be used for root cause analysis. However, runtime monitoring of complex rules on the occurrence, absence, and ordering of particular events necessitates more advanced strategies. We distinguished three classes of monitoring approaches addressing constraints on the behavior of events. In addition, compliance monitoring is also related to *conformance checking*.

Automaton-based monitoring One approach to monitor compliance with imposed rules is to use an automaton that reaches an accepting state if the rule to be checked is satisfied. As compliance rules are typically not modeled as automaton, they first have to be modeled using a formalism, such as linear temporal logic (LTL), from which an automaton can be generated. To hide the complexity of LTL from the modeler, graph notations for frequently used constraint patterns based on the work of Dwyer and Corbett [3], such as ConDec [9], were suggested. Maggi et al. [10] suggest a monitoring approach based on LTL and colored automata. It includes information about the accepting states of the automata of the individual constraints in a global automaton representing the conjunction of all imposed constraints. The latter is important to identify whether constraints are conflicting. In case a violation occurs, the monitoring can still be continued. Generally, challenge 1, the support of individual rule activations, is cumbersome to tackle using automaton-based approaches as this would require an additional instantiation mechanism. In addition, it is a non-trivial task to derive meaningful information from a non-accepting state of a generated automaton in case of violations (e.g., root cause).

Logic-based monitoring Some approaches employ logic formalisms to conduct monitoring. In [4], Montali et al. introduce an event calculus formalization for ConDec [9] constraints, which supports the identification of constraint activations. While this approach can also deal with temporal scopes, the formalization was done for existence, absence, and response constraints. Alberti et al. [11] report on monitoring contracts expressed as rules using the notion of happened and expected events. At runtime, events are aggregated in a knowledge base and reasoning is employed to identify violations. It seems that proactive prevention of violations and root cause analysis were not addressed by these approaches.

Violation pattern based monitoring Incompliance with rules on the occurrence, absence, and ordering of events can also be detected by querying the (partial) execution trace for violation patterns. To conduct the querying, existing frameworks and technologies such as complex event processing (CEP) [1] are applicable. In [8], Awad et al. introduce anti-patterns for basic rule patterns such as precedence. While this approach addresses design time verification of process models, the anti-patterns can also be applied to query the execution trace. For simple compliance rules or basic relations (as for example introduced in [12]), all violation patterns can be anticipated. However, for more complex compliance rules on the occurrence, absence, and ordering of events that can be violated in multiple ways, automatic computation of violation patterns to identify all posssible violations becomes a real challenge. This has not been addressed yet.

In their work, Giblin et al. [13] developed the REALM rule model. For REALM patterns, such as "y must occur within time t after x", they provide transformations into ACT correlation rules, which can be used for detecting relevant event patterns. Event processing techonologies are further used by numerous compliance monitoring frameworks to detect violations, e.g., in the COMPAS project [14, 15]. How the event queries are generated from complex compliance rules is not the focus of these approaches.

Conformance checking Conformance checking investigates whether a process model and process logs are conform to each other. Generally, the conformance can be tested for example by replaying the log over the process model. To tackle this, several approaches were proposed [16, 17] that introduce techniques and notions such as fitness and appropriateness to also quantify conformance. Implementations of these approaches (e.g., the Conformance Checker) are available in the process mining framework ProM [18]. Conformance checking and compliance rule monitoring exhibit major differences that require different techniques. For example, compliance rules are typically declarative while process models are mostly procedural. In addition, most of the work on conformance checking operates a posteriori. However, these approaches provide good inspiration, for example to develop metrics for quantifying compliance. Weidlich et al. show in [12] how to derive event queries for monitoring process conformance from a process model. They employ a behavioral profile that serves as an abstraction of the process model. The profile captures three relations among the activities of a process model (e.g., strict order relation). For these relations, event monitoring queries can be generated (cf. discussion on violation pattern based monitoring).

6 Summary and Outlook

In this paper, we addressed three major challenges in the context of monitoring the compliance with imposed rules. Our framework enables the identification of all activations of a compliance rule. In case of a compliance rule is violated, it becomes possible to pinpoint the rule activations involved. In addition, as our framework does not require any transformations into other representations in order to conduct the monitoring, feedback and diagnosis can be given specifically based on the corresponding rule structure. In particular, we can derive the root cause for a violation from the node markings of the particular rule. Even for rule activations that are not yet permanently violated, we can derive actions (in particular pending activities and active absence constraints) that can be helpful to process supervisors to proactively prevent violations. The validity of the approach was shown based on our proof-of-concept implementation. Our monitoring approach is not restricted to CRGs but can also be adapted to deal with other graph-based rule languages. We also conducted research to further increase the efficiency of our approach, for example by pruning paths to be explored using domination rules. In future work, we will further address the interplay of CRGs, for example conflicting rules.

References

1. Jacobsen, H.A., Muthusamy, V., Li, G.: The PADRES event processing network: Uniform querying of past and future events. it - Information Technology (2009) 250–261
2. Ly, L.T., Rinderle-Ma, S., Dadam, P.: Design and verification of instantiable compliance rule graphs in process-aware information systems. In: Int'l Conf. on Advanced Information Systems Engineering. (2010) 9–23
3. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Proc. ICSE'99. (1999) 411 – 420
4. Montali, M., Maggi, F., Chesani, F., Mello, P., van der Aalst, W.: Monitoring business constraints with the event calculus. Technical report, Universita degli Studi di Bologna (2011)
5. Ly, L.T., Knuplesch, D., Rinderle-Ma, S., Göser, K., Pfeifer, H., Reichert, M., Dadam, P.: Seaflows toolset - compliance verication made easy for process-aware information systems. In: Proc. CAiSE'10 Forum. Volume 72 of LNBIP., Springer (2010) 76–91
6. Rinderle, S., Reichert, M., Dadam, P.: Flexible support of team processes by adaptive workflow systems. Distributed and Parallel Databases **16** (2004) 91–116
7. Elgammal, A., Turetken, O., van den Heuvel, W., Papazoglou, M.: On the formal specification of regulatory compliance: A comparative analysis. In: Proc. ICSOC'10 Workshops. (2010)
8. Awad, A., Weske, M.: Visualization of compliance violation in business process models. In: Proc. BPI'09. (2009)
9. Pesic, M., van der Aalst, W.M.P.: A declarative approach for flexible business processes management. In Eder, J., Dustdar, S., eds.: Business Process Management Workshops. Volume 4103 of LNCS., Springer (2006) 169–180
10. Maggi, F., Montali, M., Westergaard, M., van der Aalst, W.: Monitoring business constraints with linear temporal logic: An approach based on colored automata. In: Proc. BPM 2011. (2011)
11. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Montali, M., Torroni, P.: Expressing and verifying business contracts with abductive logic. In: Normative Multi-agent Systems. Number 07122 in Dagstuhl Seminar Proceedings (2007)
12. Weidlich, M., Ziekow, H., Mendling, J., Günter, O., Weske, M., Desai, N.: Event-based monitoring of process execution violations. In: Proc. CAISE 2011. (2011)
13. Giblin, C., Müller, S., Pfitzmann, B.: From regulatory policies to event monitoring rules: Towards model-driven compliance automation. Technical Report Research Report RZ-3662, IBM Research GmbH (2006)
14. Holmes, T., Mulo, E., Zdun, U., Dustdar, S.: Model-aware monitoring of soas for compliance. In Dustdar, S., Li, F., eds.: Service Engineering. Springer (2011) 117–136
15. Birukou, A., D'Andrea, V., Leymann, F., Serafinski, J., Silveira, P., Strauch, S., Tluczek, M.: An integrated solution for runtime compliance governance in SOA. In: Proc. ICSOC'10. (2010)
16. van der Aalst, W.M.P., de Medeiros, A.K.A.: Process mining and security: Detecting anomalous process executions and checking process conformance. Electr. Notes Theor. Comput. Sci. **121** (2005) 3–21
17. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. **33** (2008) 64–95
18. van der Aalst, W., et al.: ProM 4.0: Comprehensive support for real process analysis. In: Proc. ICATPN 2007. Volume 4546 of LNCS., Springer (2007) 484–494