Where Did I Misbehave? Diagnostic Information in Compliance Checking

Elham Ramezani, Dirk Fahland, and Wil van der Aalst

Eindhoven University of Technology, the Netherlands
{e.ramezani,d.fahland,w.v.d.aalst}@tue.nl

Abstract. Compliance checking is gaining importance as today's organizations need to show that operational processes are executed in a controlled manner while satisfying predefined (legal) requirements. Deviations may be costly and expose the organization to severe risks. Compliance checking is of growing importance for the business process management and auditing communities. This paper presents a *comprehensive compliance checking approach based on Petri net patterns and alignments*. 50 control-flow oriented compliance rules, distributed over 15 categories, have been formalized in terms of Petri-net patterns describing the compliant behavior. To check compliance with respect to a rule, the event log describing the observed behavior is aligned with the corresponding pattern. The approach is *flexible* (easy to add new patterns), *robust* (the selected alignment between log and pattern is guaranteed to be optimal), and allows for both a *quantification of compliance* and *intuitive diagnostics* explaining deviations at the level of alignments. The approach can also handle resource-based and data-based compliance rules and is supported by *ProM* plug-ins. The applicability of the approach has been evaluated using various real-life event logs.

Keywords: compliance checking, process mining, conformance checking, Petri nets

1 Introduction

Business processes need to comply with regulations and laws set by both internal and external stakeholders. Failing to comply may be costly, therefore, organizations need to continuously check whether business processes are executed within the boundaries set by managers, governments, and other stakeholders. Deviations of the observed behavior from the specified behavior may point to fraud, malpractice, risks, and inefficiencies. Five types of compliance-related activities can be identified [23, 30, 19, 28]:

- *compliance elicitation*: determine the constraints that need to be satisfied (i.e., rules defining the boundaries of compliant behavior),
- *compliance formalization*: formulate precisely the compliance requirements derived from laws and regulations in compliance elicitation,
- *compliance implementation*: implement and configure information systems such that they fulfil compliance requirements,
- *compliance checking*: investigate whether the constraints will be met (forward compliance checking) or have been met (backward compliance checking), and

– *compliance improvement*: modify the processes and systems based on the diagnostic information in order to improve compliance.

There are two basic types of conformance checking: (1) *forward compliance checking* aims to design and implement processes where conformant behavior is enforced and (2) *backward compliance checking* aims to detect and localize non-conformant behavior. This paper focuses on backward compliance checking based on event data.

Compliance checking is gaining importance because of the availability of event data and new legislations. Major corporate and accounting scandals including those affecting Enron, Tyco, Adelphia, Peregrine and WorldCom have fueled the interest in more rigorous auditing practices. Legislation such as the Sarbanes-Oxley (SOX) Act of 2002 and the Basel II Accord of 2004 was enacted as a reaction to such scandals. At the same time, new technologies are providing opportunities to systematically observe processes at a detailed level by recording all process relevant events.

Process mining techniques [1] offer a means to more rigorously check compliance and ascertain the validity and reliability of information about an organization's core processes. To core challenge is to compare the prescribed behavior (e.g., a process model or set of rules) to observed behavior (e.g., audit trails, workflow logs, transaction logs, message logs, and databases). For example, in [3] it is shown how constraints expressed in terms of Linear Temporal Logic (LTL) can be checked with respect to an event log. In [25] both LTL-based and SCIFF-based (i.e., abductive logic programming) approaches are used to check compliance with respect to a declarative process model and an event log. Dozens of approaches have been proposed to check conformance given a Petri net and an event log [2, 8, 6, 7, 11, 13, 20, 26, 27, 31, 38]. Approaches such as in [31] replay the event log on the model while counting "missing" and "remaining" tokens. The former indicate observed, but disallowed behavior, and the latter indicate non-observed, but required behavior. State-of-the-art techniques in conformance checking retrieve this information by computing *optimal alignments* [2, 8] between traces in the event log and "best fitting" paths in the model.

Existing approaches to backwards compliance checking have two main problems. First of all, the *elicitation of compliance rules is not supported well*. End users need to map compliance rules onto expressions in temporal logic or encode the rules into a Petri-net-like process model. Second, existing checking techniques can discover violations but *do not provide useful diagnostics*. While forward compliance checking techniques [10, 18] employ pattern matching to highlight compliance violations in a model, such techniques are not applicable in backwards checking where not a model, but a log is given. Here, LTL-based checkers will classify a trace as non-compliant without providing detailed diagnostics and discard the remainder of the trace when the first deviation is detected.

To address these limitations we provide a *comprehensive collection of control-flow related compliance rules*. We identify 50 rules distributed over 15 categories. These compliance rules are formalized in terms of Petri net patterns. We apply the alignment technique developed in [2, 8] to analyze if the process execution (log) has been compliant with the compliance rules (Petri net patterns). If the observed behavior is consistent with the compliance rule, then the optimal alignment shows that all moves of the log can be mimicked by the corresponding Petri net pattern and vice versa. If this is not

possible because the compliance rule is violated, then the alignment shows the root cause of the deviation. This way, we are able to show *detailed diagnostics without false negatives* (non-conformant behavior remains undetected) and *false positives* (conformant behavior is classified as non-conforming because of an incorrect alignment of log and model/rule). The approach is *extendible*, i.e., to add a new type of rule, one just needs to add the Petri net pattern to our repository. Moreover, as shown in this paper, our approach can be used to support resource-based and data-based compliance rules.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 explains the notion of alignments to relate observed and modeled behavior. Our compliance rule framework is introduced in Section 4. Although the primary focus of this paper is on control-flow compliance rules, Section 5 illustrates that the approach also supports the other perspectives (e.g. resources and data). In Section 6 the approach is validated using a case study and the implementation in ProM is showcased. Section 7 concludes the paper.

2 Related Work

The importance of compliance management has been pointed out by various authors [5]. In [30] a life cycle is introduced to structure the process of compliance management. A comparative analysis over different compliance management solution frameworks is provided in [23].

Compliance management has gained wide interest from the Business Process Management (BPM) community. Compliance checking approaches can be mapped onto two main categories [22]:

- *Forward* compliance checking aims at ensuring compliant process executions. Processes can be constructed to be compliant [32] or verified whether they are compliant [24]. Alternatively, compliance requirements can be transformed into monitoring rules [12] or model annotations which then are used to enforce compliant process executions [17, 39]. Diagnostic information is obtained by pattern matching [10, 18].
- *Backward* compliance checking evaluates in hindsight whether process executions did comply to all compliance rules or when and where a particular rule was violated. A variety of conformance checking techniques have been proposed to quantify conformance and detect deviations based on an event log and process model (e.g., a Petri net) [2, 8, 6, 7, 11, 13, 20, 26, 27, 31, 38]. Also approaches based on temporal logic [3, 25] have been proposed to check compliance

In this paper we focus on backward compliance checking and assume an event log to be present. Compared to existing approaches we provide a comprehensive collection of compliance rules. Moreover, we focus on providing diagnostic information in backwards compliance checking.

3 Conformance Checking Based on Alignments

As will be shown in this paper, we provide a large repository of Petri net patterns modeling typical compliance rules. These rules can be instantiated for a particular process,

i.e., the abstract activities in the pattern are replaced by concrete activities also recorded in the event log. The log *complies* to the rule if each log trace is described by the Petri net pattern. In case a trace is not described, we want to locate *where* the trace deviates from the pattern. This section recalls basic notions and a recent technique [2, 8] for finding deviations between log traces and formal specification (a Petri net).

An *event log* is a *multiset* of *traces*. Each trace describes the life-cycle of a particular *case* (i.e., a *process instance*) as a sequence of *events*. An event often refers to the *activity* executed. However, event logs may store additional information about events. For example, many process mining techniques use extra information such as the *resource* (i.e., person or device) executing or initiating the activity, the *timestamp* of the event, or *data elements* recorded with the event (e.g., the size of an order).

From a formal point of view a trace σ_L is a sequence over an alphabet Σ_L, i.e., $\sigma_L \in {\Sigma_L}^*$. An event log L is a multiset of traces, i.e., $L \in \mathbb{B}({\Sigma_L}^*)$. The alphabet Σ_L is typically the set of activity names. However, when including additional perspectives, the alphabet may be extended to also contain information about data and resources. For example, $(prepare\ decision, start, John, gold, 50\ euro) \in \Sigma_L$ may refer to an event describing the start of activity "prepare decision" by John for a gold customer claiming 50 euro. The choice of Σ_L depends on the compliance rule that needs to be checked, e.g., for most control-flow related rules it is sufficient to record the activity name.

A Petri net pattern is essentially a specification prescribing compliant traces in a concise way. Technically, a *specification* $S \subseteq {\Sigma_S}^*$ is a finite set of traces over an alphabet Σ_S together with a mapping $\ell : \Sigma_S \to 2^{\Sigma_L} \cup \{\tau\}$ that relates each specification event in Σ_S to a set of log events in Σ_L or to τ. In this paper, S is the set of firing sequences of a Petri net and ℓ is the function that labels each transition with an activity name. S and ℓ can be described by other formalisms such as temporal logics as well.

We use the alignment approach described in [2, 8] to relate traces in the log (i.e., observed behavior) to traces of the specification (i.e., prescribed behavior). An optimal alignment of σ_L to S, roughly speaking, is a trace σ_S that is possible according to S and that is *as similar to* σ_L *as possible*. By comparing σ_L and σ_S, a business analyst gains an understanding on what has been done wrong in σ_L and what instead should have been done (to behave as shown in σ_S).

A given trace σ_S will be related to S by pairing events of σ_L to events of S. Formally, a *move* (of σ_L and S) is a pair $(x, y) \in (\Sigma_L \cup \{\gg\}) \times (\Sigma_S \cup \{\gg\}) \setminus \{(\gg, \gg)\}$. For $x \in \Sigma_L, y \in \Sigma_S$, we call $(x, \gg)$ a *move on log*, $(\gg, y)$ a *move on specification* S, and if $x \in \ell(y)$, then (x, y) is a *synchronous move*.

An *alignment* of a trace $\sigma_L \in {\Sigma_L}^*$ to S is a sequence $\gamma = \langle (x_1, y_1) \dots (x_n, y_n) \rangle$ of moves (of σ_L and S) such that the projection $x_1 \dots x_n$ to Σ_L is the original trace σ_L, i.e., $\langle x_1 \dots x_n \rangle|_{\Sigma_L} = \sigma_L$, and the projection $\langle y_1 \dots y_n \rangle|_{\Sigma_S} = \sigma_S \in S$ is described by the specification.

For example, for a specification $S = \{\langle a, b, c, d \rangle, \langle a, c, b, d \rangle\}$ with $\ell(x) = \{x\}$ the trace $\sigma_L = \langle a, c, c, d \rangle$ has (among others), the following two alignments with events of σ_L shown at the top and events of S shown at the bottom: $\gamma_1 = \frac{a|c|c|\gg|d}{a|c|\gg|b|d}$ and $\gamma_2 = \frac{a|\gg|\gg|c|c|d}{a|c|b|\gg|\gg|d}$.

Both alignments yield the same specified trace $\sigma_S = \langle a, c, b, d \rangle \in S$. However, γ_1 is preferable over γ_2 as it maximizes the number of synchronous moves. The conformance checking problem in this setting is to find for a given trace σ_L and specification S an *optimal* alignment γ of σ_L to S s.t. no other alignment has fewer non-synchronous moves (move on log only or move on specification only). The technique of [2, 8] finds such an optimal alignment using a cost-based approach: a cost-function κ assigns each move (x, y) a cost $\kappa(x, y)$ s.t. a synchronous move has cost 0 and all other types of moves have cost > 0. Then an A*-based search on the space of (all prefixes of) all alignments of σ_L to S is guaranteed to return an optimal alignment for σ_L and S.

In such an optimal alignment, a move on log $(x, \gg)$ indicates that the trace σ_L had an event x that was not supposed to happen according to the specification S whereas a move on specification $(\gg, y)$ indicates that σ_L was missing an event $\ell(y)$ that was expected according to S. As the alignment preserves the position relative to the trace σ_L, we can locate the exact position where σ_L had an event too much or missed an event compared to S.

In the remainder of this paper, we show how to leverage this approach to compliance checking. The optimal alignments between log and specification provide excellent diagnostic information and can be used to robustly quantify conformance. Thereby, the specification S will formally capture all traces that comply to a given compliance constraint. The alignment of a log trace σ_L to S can then clearly show where and how often σ_L deviates from the constraint.

4 Expressing Compliance Rules as Petri Net Patterns

In this section, we provide a framework for compliance rules together with an extensive list of control-flow rules in 15 different categories. Each rule has a comprehensive description and is formalized as a Petri net pattern. In Sect. 5 we generalize this approach to data- and resource-related rules.

4.1 Compliance Rule Framework

A compliance rule prescribes how an internal or cross-organizational business process has to be designed or executed. It originates in explicitly stated regulations and can refer to the individual perspectives of a business process (control-flow, data flow, organizational aspects) or a combination of several perspectives. We reviewed existing literature on compliance [4, 15, 9, 14, 21, 36, 19, 33, 35], collected the described rules, and categorized them. We found that a single rule usually is not concerned with only one perspective of a process, but with several perspectives. Based on this observation, we identified six orthogonal dimensions of compliance rules, into which each of the rules could be categorized. For example, the rule "After a claim of more than 3000 EUR has been filed, two different employees need to check the validity of the claim independently." is composed of 3 basic rules that refer to (1) *control-flow* ("After a claim has been filed, validity must be checked."), (2) *data-flow* ("A claim over 3000 EUR requires two validity checks."), and (3) the *organization* ("Multiple validity checks are carried out by different employees.").

Furthermore, a compliance rule can (4) impose *time-related* constraints (e.g., "Within 6 months the claim must be decided.") or can be *untimed*,(5) prescribe properties of a *single case* or of *multiple* cases (e.g., "20% of all claims require a detailed check."), and (6) prescribe properties of the *process design* (e.g., "The claim process must have a time-out event handler.") or properties of the process executions, which can be *observed* (i.e., recorded in an event log).

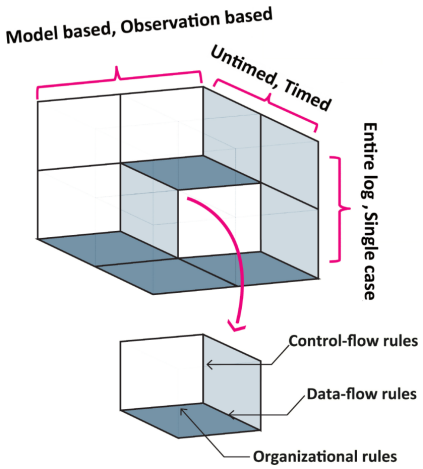

Fig. 1. Compliance Rule Framework

These six dimensions give rise to the framework shown in Fig. 1. In this paper, we present compliance rules for control-flow, data-flow as well as organizational aspects, where we focus on untimed, observation-based properties of individual cases. The remainder of this section presents control-flow compliance rules and their formalization. Section 5 presents data-flow rules and organizational rules and their combination with control-flow rules.

4.2 Control-flow compliance rules

Eliciting and *formalizing* compliance rules for a business process comprises determining the laws and regulations that are relevant for this process and formulating these compliance rules in an unambiguous, yet understandable manner [30]. Typically this requires to express a given informal requirement in a formal notation, a task an end user may not be capable of. To support elicitation, we provide end users with an *extensive library* of comprehensive compliance rules. Each rule has an informal, precise description and is accompanied by a mathematical formalization. The end user just has to pick the rule(s) that describe the given compliance requirement best; the accompanying formalization is then used for compliance checking.

We collected from literature [4, 15, 9, 14, 21, 36, 19, 33, 35] 50 compliance rules that concern the control-flow perspective of a process, and classified them further into 15 categories; see Tab. 1. Each category includes several compliance rules. For example, the *Existence* category defines 3 rules in total, e.g., "In each process execution, task A should be executed" and "In each process execution, Task A should not be executed." Each rule is parameterized over tasks (e.g., Task A) or numeric parameters (e.g., governing bounds for repetitions etc.).

Formalizing these rules requires to decide for a particular formalism. Some compliance rules prescribe behaviors that are easier to express in terms of logical formulas (each A is followed by a B), and some rules prescribe behaviors that are easier to express in a more operational model (A, B, and C happen twice directly in sequence with no other event in between). Our literature survey found both kinds of rules to be relevant, pitting logics (e.g., LTL) against operational models (e.g., Petri nets). This makes the choice for a formal language is a choice for convenience, familiarity and tool

Table 1. Control Flow Compliance Rule Set

Category (Rules)	Description
Existence (2)	Limits the occurrence or absence of a given event A within a scope. [4],[15],[9], [14],[21],[36],[33]
Bounded Existence (6)	Limits the number of times a given event A must or must not occur within a scope. [15],[14]
Bounded Sequence (5)	Limits the number of times a given sequence of events must or must not occur within a scope. [15],[14]
Parallel (1)	A specific set of events should occur in parallel within a scope. [33]
Precedence (10)	Limits the occurrence of a given event A in precedence over a given event B. [15],[33],[14],[36],[9],[19],[21],[4],[33]
Chain Precedence (4)	Limits the occurrence of a sequence of events $A_1, \ldots, A_n$ over a sequence of events $B_1, \ldots, B_n$. [15],[14],[21]
Response (10)	Limits the occurrence of a given event B in response to a given event A. [33],[14],[21],[15],[37],[9],[19]
Chain Response (4)	Limits the occurrence of a sequence of events $B_1, \ldots, B_n$ in response to a sequence of events $A_1, \ldots, A_n$. [15]
Between (7)	Limits the occurrence of a given event B between a sequence of events A and C. [14]
Exclusive (1)	Presence of a given event A mandates the absence of an event B. [15]
Mutual Exclusive (1)	Either a given event A or event B must exist but not none of them or both. [15],[34]
Inclusive (1)	Presence of a given event A mandates that event B is also present. [15]
Prerequisite (1)	Absence of a given event A mandates that event B is also absent. [15]
Substitute (1)	A given event B substitutes the absence of event A. [15]
Corequisite (1)	Either given events A and B should exist together or to be absent together. [15]

support. Because of tool support for conformance checking [8], we decided to formalize rules as *parameterized Petri net patterns*. Although being an unusual choice, we could formalize operational rules as well as declarative rules in a systematic and understandable way. The complete collection of compliance rules and their Petri net patterns is described in [29]; in the following we present a few characteristic rules and their formalization in terms of Petri net patterns.

4.3 Petri net patterns for compliance rules

Bounded Existence of a Task (from category Bounded Existence). Description: "Task A should be executed exactly k times." If A occurs less than or more than k times, the rule is violated. For instance, for $k = 2$, the trace $\langle BCADBCAD \rangle$ complies to this rule and $\langle BCADBCAAD \rangle$ violates the rule.

Figure 2 shows the Petri net pattern that formalizes this rule. Task A is expressed as an A-labeled transition. Occurrences of any other transition than A are mimicked by the Ω-labeled transition. This way, the pattern abstracts from all other trace events that

are not described in the compliance rule. The transition F expresses that the end of the trace has been reached, i.e., it occurs *after* all other events of the trace occurred.

The pre-place P_k of A is initially marked with k tokens. As each occurrence of A consumes one token from P_k, A can occur at most k times. Also each occurrence of A produces one token on $Count$. By consuming k tokens from place $Count$, the final transition F can only occur (i.e., the trace can only complete) if A has occurred k times.

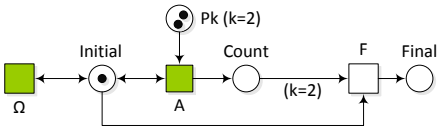

Fig. 2. Petri net pattern for rule "Bounded Existence of a Task."

Figure 2 also illustrates the basic principles of Petri net patterns for compliance rules. Each pattern has a dedicated place $Final$ that defines the final marking of the pattern. A trace σ *complies* to the pattern (its rule) iff after executing σ, final transition F is enabled, and its occurrence leads to the *final marking* that puts 1 token on place $final$ and all other places are empty. In Fig. 2, the arc from F to $Initial$ ensures that once F occurs, no other tokens remain in the net and A and Ω cannot occur anymore.

We use the alignment-based approach of Sect. 3 for checking whether a trace σ complies to this rule, by aligning σ to an occurrence sequence σ_S of the net of Fig. 2 where F is the last event of σ_S. To this end, we first have to map transitions of the pattern to events of σ by the labeling function ℓ. For example, the trace $\langle BCADBCAD\rangle$ could have the mapping $\ell(A) = \{A\}$, $\ell(\Omega) = \{B, C, D\}$, and $\ell(F) = \{\tau\}$ (the final transition F is always regarded as silent and not mapped to any trace event). For this labeling, the approach of Sect. 2 aligns $\sigma_1 = \langle BCADBCAD\rangle$ by $\gamma_1 = \frac{B|C|A|D|B|C|A|D|\gg}{\Omega|\Omega|A|\Omega|\Omega|\Omega|A|\Omega|F}$ and $\sigma_2 = \langle BCADBCAAD\rangle$ by $\gamma_2 = \frac{B|C|A|D|B|C|A|A|D|\gg}{\Omega|\Omega|A|\Omega|\Omega|\Omega|A|\gg|\Omega|F}$ and $\sigma_3 = \langle BCADBCD\rangle$ by $\gamma_3 = \frac{B|C|A|D|B|C|\gg|D|\gg}{\Omega|\Omega|A|\Omega|\Omega|\Omega|A|\Omega|F}$. Alignment γ_1 only contains synchronous moves of σ_1 and Petri net (except for final transition F) which means that σ_1 complies to the rule. In contrast, γ_2 contains a move on log $(A, \gg)$ on the third A. This move on log not only indicates that σ_2 violates the compliance rule, but the move $(A, \gg)$ also tells the exact location of the deviation, which is the third A. γ_3 contains a move on model $(\gg, A)$ that is required to get the Petri net pattern into its final marking. This move on model also indicates a violation of the rule as a missing second A in the σ_3.

This first rule constrains a single task. The following rules constrain *orderings* of several tasks and also shows the importance of the Ω-transition in the Petri net patterns.

Direct Precedence of a Task (from category Precedence). Description: "Every time B occurs, it should be directly preceded by A." If B occurs without a directly preceding A, the rule is violated. For instance, $\langle ACCAAC\rangle$ and $\langle ABCAAB\rangle$ comply to the rule, whereas $\langle ABACB\rangle$ violates the rule.

The pattern of Fig. 3(top left) formalizes this rule. It basically describes a cycle of A and B, so that B can only occur if A has preceded it. As there is no Ω-transition adjacent to place p, *A directly precedes* any B (occurrences of any other transition but B are excluded).

This pattern also demonstrates the power of the alignment-based approach and for checking whether a trace complies the rule. For instance, in the compliant trace $\sigma =$

$\langle ABCAABA \rangle$, the second and the last occurrence of A are aligned to the left-most transition A of Fig. 3 and only the first and third A that directly precede a B are aligned to enter the cycle. This capability of the alignment-based approach allows to design patterns with non-deterministic choices such as in Fig. 3, which gives greater flexibility when formalizing compliance rules.

Our approach also allows to derive *variants* of patterns. For instance, Fig. 3(top right) formalizes that "Every occurrence of B should be preceded by A (also several steps earlier)." The patterns presented so far assumed an occurrence of a task A to be represented as an atomic event A in the log. Fig. 3(bottom) formalizes the direct precedence rule for the case that task A is represented by two events $A - start$ (A_{st}) and $A - complete$ (A_{cmp}) indicating the start and completion of an ongoing activity. All Petri net patterns of our collection rules come in these two flavors and can be picked based on the setting. The next rule demonstrates that in this way, also intricate ordering constraints can be formalized with Petri nets in an intuitive way.

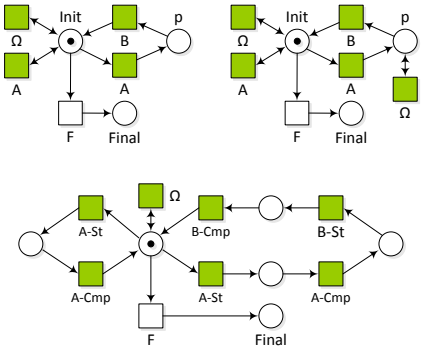

Fig. 3. Petri net patterns for precedence.

Direct Precedence or Simultaneous Occurrences of Tasks. "Task A must always be executed simultaneously or directly before task B." In case of atomic tasks this rule is identical to the preceding rule, in case of ongoing tasks, A and B can overlap in time. For example, the trace $\langle A_{st}B_{st}CA_{cmp}DB_{cmp} \rangle$ complies to this rule whereas $\langle A_{st}A_{cmp}CB_{st}DB_{cmp} \rangle$ violates this rule.

The Petri net pattern of Fig. 4 formalizes this rule. The case where A strictly precedes B (A ends before B starts) is formalized by the lower cycle of the net. More interestingly, the case where A and B occur simultaneously is formalized by the upper cycle (white transitions are silent). There is no cycle that permits B without a preceding or simultaneous A. If there is no B or B just occurred, any events but B_{st} and B_{cmp} may occur. This is also the situation when the pattern may terminate. The replay-based approach of Sect. 3 aligns traces to this pattern as explained for "Direct Precedence of a Task."

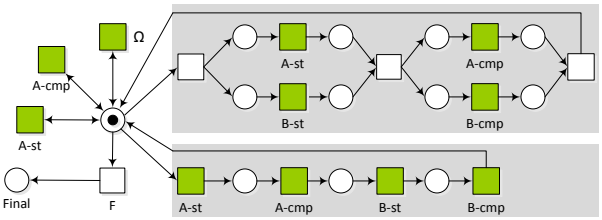

Fig. 4. Petri net pattern for "Direct Precedence or Simultaneous Occurrences of Tasks."

The preceding rules constrained control-flow in one specific dimension (ordering or number of occurrences). The next rule shows that also mixed rules occur, and how to formalize mixed rules by reusing concepts of the preceding patterns.

Bounded Existence of Sequence of Tasks (from category Bounded Sequence). Description: "The direct sequence of tasks $\langle AB \rangle$ (B exactly after A) should not occur more than k times." If $\langle AB \rangle$ occurs for the $k+1$-st time, the rule is violated. For instance, for $k = 2$, $\langle CABACBABC \rangle$ complies to this rule and $\langle CABBCABABC \rangle$ violates the rule.

The pattern of Fig. 5 formalizes this rule by combining concepts of "Bounded Existence" (Fig. 2) with concepts of "Direct Precedence" (Fig. 3). A consecutive sequence $\langle AB \rangle$ is expressed as a cycle in the pattern. The complete cycle may occur at most k times because of preplace P_k of B. A $k+1$-st occurrence of A is permitted as it does not complete $\langle AB \rangle$, yet. After A occurred, there may be arbitrary further occurrences of A; a subsequent B still yields a direct sequence $\langle AB \rangle$, any other transition interrupts this sequence (Ω brings the token back to $Initial$).

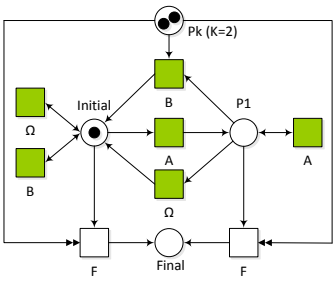

Fig. 5. Petri net pattern for "Bounded Existence of Sequence of Tasks."

The pattern can go to the final marking at any point (that is, if there are no other events to be executed). Here, the *reset arcs* from P_k to the final transitions F ensure that all pending tokens are removed from the net (e.g., if $\langle AB \rangle$ never occurred).

Aligning the violating trace $\sigma = \langle CABBCABABC \rangle$ to this pattern yields alignment $\gamma = \frac{C|A|B|B|C|A|B|A|B|C|\gg}{\Omega|A|B|B|\Omega|A|B|A|\gg|\Omega|F}$ where the move on log $(B, \gg)$ at the third occurrence of $\langle AB \rangle$ arises because there is no token on P_k left (and hence B is not enabled in the pattern), i.e., this move on log indicates the violation. These patterns demonstrate some of the key concepts and basic building blocks that we used to formalize all 50 compliance rules in terms of Petri net patterns.

5 Compliance to Data and Organizational Aspects

So far we presented a comprehensive collection of control-flow compliance rules and their formalization as Petri net patterns. These rules cover the *control-flow* dimension of the compliance rule framework introduced in Fig. 1. In this section, we show how the pattern-based approach to compliance rules of Sect. 5 can also be applied to check compliance with respect to *data* and to *organizational aspects*, which constitute two other dimensions of the framework. As before, we consider single-case observation-based untimed compliance rules.

5.1 Data Flow Compliance Rules

A typical example of a data flow compliance rule is to **Restrict data values permitted for a task**. For example, "A discount of 10% is granted if the customer is a gold customer; 5% are granted if the customer is a silver customer." A rule of this kind prescribes that task *grant* refers to 2 attributes e.g., *customer status* and *percentage*. When task *grant discount* occurs, these attribute values need to be logged in the corresponding

event such as $(grant, John, gold, 10\%)$ (see Sect. 3); otherwise compliance cannot be checked in hindsight.

When checking compliance to this rule, it is not just sufficient to check whether $grant$ occurred, but we need to check whether $grant$ occurred with the right attribute values. To this end slight changes in actual Petri net pattern and labeling ℓ that relates Petri net transitions to events are required. Figure 6(top) shows the Petri net pattern for this rule. It contains two transitions $grant$ that are further distinguished by the attribute value combinations that are permitted by this task.

Recall from Sect. 3 that each pattern also has a labeling function $\ell(.)$ that maps transitions to sets of events. In contrast to Sect. 4, a transition is not mapped to an event name, but to a *combination of name and attribute values*. For instance, the mapping $\ell(grant10\%gold) = \{(grant, x, y, z) \mid y = gold, z = 10\%\}$ maps transition $grant10\%gold$ only to $grant$ events which have $gold$ and 10% as their attribute values, correspondingly for $grant5\%silver$. Other occurrences of $grant$ (with other attribute value combinations) are *disallowed* by mapping Ω only to events other than $grant$, e.g., $\ell(\Omega) = \{(a, x, y, \ldots) \mid a \neq grant\}$. This mapping ℓ and the pattern of Fig. 6(bottom) together formalize the compliance rule. For example, trace $\langle$(*add item*, $x, 10EUR$) (*add item*, $y, 32EUR$) (*grant*, Joe, $gold, 10\%$)$\rangle$ complies to this rule whereas aligning trace $\langle$(*add item*, $x, 10EUR$) (*add item*, $y, 32EUR$) (*grant*, Jim, $silver, 10\%$)$\rangle$ to the this rule yields a move on log $((grant, Jim, silver, 10\%), \gg)$ indicating that Jim was granted a wrong discount.

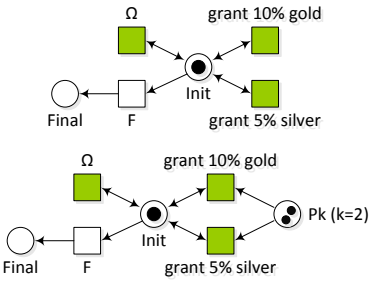

Fig. 6. Petri net pattern to 'Restrict data values permitted for a task'.

Note that data-flow compliance is essentially formalized by further distinguishing transitions in the Petri net patterns, and by defining the right mapping from transitions to events. This permits to *combine control-flow rule and data-flow compliance rules* also formally, e.g., the pattern of Fig. 6(bottom) formalizes that "A discount (of 10% for gold customers and 5% for silver customers) is given at most twice per case."

5.2 Compliance to Organizational Aspects

Separation of Duty (4-eyes principle). The perhaps best known compliance rule states that "Of two sequential tasks A and B, if A was performed by user R, then B must not be performed by R." Here, each task has a particular attribute *performed by* (or role) which takes as values user names or roles. Technically, the role attribute is a special data attribute: a log event $(check, Sue)$ describes that Sue performed activity $check$. A trace $\sigma_1 = \langle(receive, Tom)(check, Sue)(notify, Sue)(pay, Tom)\rangle$ complies to the 4-eye principle for tasks $check$ and pay whereas $\langle(receive, Tom)(check, Sue)(notify, Sue)$ $(pay, Sue)\rangle$ violates the principle.

Figure 7 shows the Petri net pattern that formalizes this compliance rule. It distinguishes two cases (as indicated by the upper and lower grey-shaded rectangle. Each case describes one compliant role assignment to tasks A and B, either A is performed

by R, then B not by R, or vice versa. In this compliance rule, once R performed A, it always has to perform A and may never perform B (within the same trace). Hence the choice for either case is permanent in the pattern as well. The pattern may terminate at any point in time, and all other tasks (except for A and B with the chosen role assignments) may occur at any point in time. As for data-flow compliance, the labeling $\ell(.)$ is crucial to relate patterns of the transitions to events: $\ell(A,R) = \{(x,y) \mid x = A, y = R\}, \ell(A,not\text{-}R) = \{(x,y) \mid x = A, y \neq R\}, \ell(B,R) = \{(x,y) \mid x = B, y = R\}, \ell(B,not\text{-}R) = \{(x,y) \mid x = B, y \neq R\}, \ell(\Omega) = \{(x,y) \mid x \notin \{A,B\}, y \neq R\}$.

Each user gives rise to a different labeling that has to be checked separately from other labelings. When checking compliance of trace σ_2 given above w.r.t. tasks $check$, pay, and user Joe, the alignment-based approach of Sect. 3 returns a move on log $((pay, Sue), \gg)$ indicating that the pay task should not have been performed by Sue (as it is not allowed by the pattern).

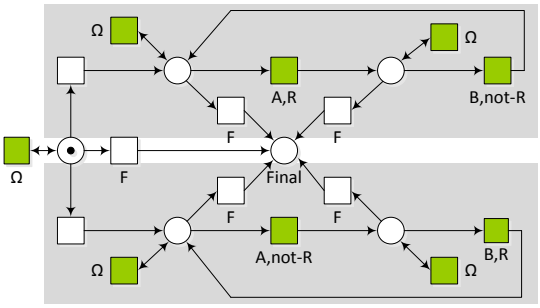

Fig. 7. Petri net pattern for "Separation of Duty."

Altogether, compliance to data-flow and to organizational aspects is orthogonal to control-flow compliance and builds on mapping Petri net transitions to events based on a combination of event name and attributes. This also allows to formalize and check rules that depend on mixture of control-flow, data-flow and organizational aspects.

6 Experimental Results

We have evaluated the technique on real-life log taken from the financial system of a large Dutch hospital. The log contained over 150000 events from over 700 different activities in 1150 cases, each case representing a patient. The log is obtained from financial system of the hospital in the period of 2005 to 2008. Beside anonymizing the log, all other data in log is preserved including event names, involved resources etc. We first describe the implementation used in this evaluation and then report on some compliance rules relevant to this process and the results we obtained for them.

6.1 Implementation in ProM

The presented technique is implemented in the *Compliance* package of the Process Mining Toolkit ProM 6, available from `http://www.processmining.org/`. The packet provides Petri net patterns for the control-flow compliance rules discussed in this paper. The "*Check Compliance*" plugin takes a log as input. Then the user can pick from a list of available compliance rules those rules against which the log shall be checked. For each rule to check, the user then configures its parameters, mostly by mapping events to task names of the rule. Then the conformance checker of Sect. 3 is called to align the log to the rule's Petri net pattern. The resulting alignment is shown to

the user. Each aligned trace is shown in a separate row and deviations are highlighted: a move on log indicates an event occurred which did not comply to the rule, a move on model indicates which event skipped in log such that log does not comply to the rule. Several figures in the next section show these alignments.

To ease presentation of our results, we abstract long sequences of events that are not relevant to the compliance rule (i.e., which are mapped to Ω-transitions), to shorter sequences. This way, order and relative position of compliance-relevant events are preserved while irrelevant details are abstracted from.

6.2 Case Study Constraints and Results

In the case study we followed the standard use case for compliance checking: (1) check relevant regulations and elicit respective compliance requirements, (2) for each requirement, identify the patterns that precisely express the requirement from the rule collection in Tab. 1, (3) take the corresponding Petri net pattern and map its transitions to the events in the given log, and (4) run the conformance checker.

In the following, we describe our findings for three compliance requirements that were derived from the financial department's internal policies and medical guidelines.

Compliance Requirement 1. "The hospital should register each visiting patient and prevent duplicate registrations for a patient." This requirement is formalized by the compliance rule "Event *First Visit Registration* should occur exactly once per case." from the category '*bounded existence*' of Table 1. The corresponding pattern is shown in Fig. 2 (left). To obtain a reliable result, we needed to filter the data for patients who started their treatment between 2005 and 2008. We checked compliance for 640 cases and identified 622 compliant and 18 non-compliant cases.

Fig. 8. Non-compliant case for '*First Visit Registration* should occur exactly once.'

Figure 8 shows diagnostic information for a non-compliant case. As described above, ProM maps the trace on to the compliance-influencing events: *First Visit Registration* occurs twice, where the first occurrence is compliant (highlighted green) and the second one should not have been in the trace (highlighted yellow, *move on log*).

Compliance Requirement 2. The patients are sent to the hospital for specialized treatment. Therefore a basic X-ray scan has to be performed after a patient was registered. This requirement is formalized by two rules: "Event *x-ray* should occur at least once per case." and "Every time the event '*x-ray*' occurs, '*First Visit Registration*' should have happened before *x-ray*." Figure 9 shows compliance results for the second rule, which is formalized by Fig. 3(top right); we found 104 compliant cases out of 640.

An example of a non-compliant case is shown in Fig. 9. The relevant sequence of events in this case is $\langle \ldots x\text{-}ray \ldots First\,Visit\,Registration \ldots \rangle$. The compliance checker identified the *x-ray* event as an event that should not have happened (highlighted yellow, *move on log*) because it occurred before the $First\,Visit\,Registration$ event). In addition, *the checker highlights the position where x-ray should have occurred* as a move on model (highlighted purple).

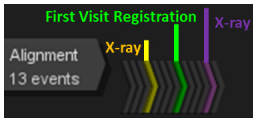

Fig. 9. Non-compliant case for '*First Registration Visit* precedes *x-ray*.'

Compliance Requirement 3. "For safety reasons, either a *CT-Scan* or an *MRI test* of an organ should be taken from a patient but not both." The corresponding compliance rule from the *Exclusive* category has the Petri net pattern of Fig. 10(top).

We checked this rule and identified 1092 compliant cases out of 1150. Fig. 10(bottom) shows diagnostic information for one non-compliant case. The relevant sequence of events for this case is $\langle ...CT.MRI...CT...MRI... \rangle$. The Occurrence of *CT* together with *MRI* is a violation. In addition, the diagnostic information provided by the checker clearly shows *several* violations due to several occurrences of *CT* (*move on log*, highlighted yellow).

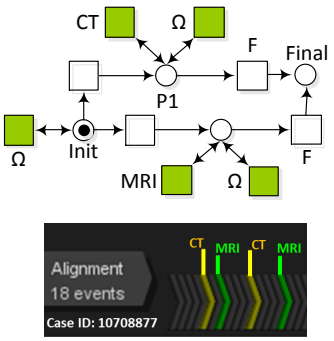

Fig. 10. Petri net pattern and a non-compliant case for 'CT-Scan and MRI Test exclude each other.'

Altogether we could identify and precisely locate various compliance deviations from given compliance rules in a real-life log.

7 Conclusion

Today's organizations need to comply to an increasing set of laws and regulations. Compliance requirements are often described in natural language which makes checking compliance a difficult task. In this paper we provided a first comprehensive collection of control-flow compliance rules which allow to formally capture a large set of compliance requirements. Moreover we presented a robust technique for backwards compliance checking which enables us to provide diagnostic information in case of violations. The technique is also applicable to check compliance of artifact-centric processes [16]. The approach is supported by ProM plugins and we tested our techniques using real-life logs and compliance requirements.

The unusual choice of formalizing compliance rules as Petri nets rather than logics posed no difficulties. Yet, we can foresee benefits from a mixed formalization of declarative rules by logics and operational rules by Petri nets. Note that in no situation, the end user is confronted with the formalization of the rule, but picks rules by their informal description.

We also showed that our approach also can handle organizational rules and data-flow rules to constrain individual tasks. Handling constraints across several tasks requires to generalize the technique, in particular the underling conformance checker [8]. Also, the mapping between event attributes and transitions is cumbersome and currently specified at a technical level using concrete values; a more user-friendly approach to specify organizational and data-flow rules is required.

Thus, future work aims at exploring the compliance rule framework (Fig. 1) further and extending the compliance rule set (Table 1) for other dimensions, i.e., with collections of compliance rules restricting data-flow, process resource, and process time.

Acknowledgements. We thank R.Mans, A.Adriansyah and C.Stahl for their substantial support in writing this paper. The research leading to these results has received funding from the European

Community's Seventh Framework Programme FP7/2007-2013 under grant agreement n° 257593 (ACSI).

References

1. Aalst, W.M.P.v.d.: Process Mining - Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. Aalst, W.M.P.v.d., Adriansyah, A., Dongen, B.F.v.: Replaying history on process models for conformance checking and performance analysis. WIREs Data Mining Knowl Discov 2, 182–192 (2012)
3. Aalst, W.M.P.v.d., Beer, H.T.d., Dongen, B.F.v.: Process mining and verification of properties: An approach based on temporal logic. In: OTM Conferences 2005. LNCS, vol. 3760, pp. 130–147. Springer (2005)
4. Aalst, W.M.P.v.d., Hee, K.M.v., Werf, J.M.v.d., Kumar, A., Verdonk, M.: Conceptual Model for Online Auditing. Decision Support Systems 50(3), 636–647 (2011)
5. Abdullah, N.S., Sadiq, S.W., Indulska, M.: Information systems research: Aligning to industry challenges in management of regulatory compliance. In: PACIS 2010. p. 36. AISeL (2010)
6. Adriansyah, A., Dongen, B.F.v., Aalst, W.M.P.v.d.: Towards Robust Conformance Checking. In: BPI 2010. LNBIP, vol. 66, pp. 122–133. Springer (2011)
7. Adriansyah, A., Sidorova, N., Dongen, B.F.v.: Cost-based Fitness in Conformance Checking. In: ACSD 2011. pp. 57–66. IEEE (2011)
8. Adriansyah, A., Dongen, B.F.v., Aalst, W.M.P.v.d.: Conformance checking using cost-based fitness analysis. In: EDOC 2011. pp. 55–64. IEEE (2011)
9. Awad, A., Decker, G., Weske, M.: Efficient compliance checking using bpmn-q and temporal logic. In: BPM 2008. LNCS, vol. 5240, pp. 326–341. Springer (2008)
10. Awad, A., Weske, M.: Visualization of compliance violation in business process models. In: Business Process Management Workshops. pp. 182–193 (2009)
11. Calders, T., Guenther, C., Pechenizkiy, M., Rozinat, A.: Using Minimum Description Length for Process Mining. In: SAC 2009. pp. 1451–1455. ACM Press (2009)
12. Christopher Giblin, S.M., Pfitzmann, B.: Research report: From regulatory policies to event monitoring rules: Towards model-driven compliance automation. Tech. rep., IBM Research GmbH, Zurich Research Laboratory, Switzerland (2006)
13. Cook, J., Wolf, A.: Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model. ACM Transactions on Software Engineering and Methodology 8(2), 147–176 (1999)
14. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: ICSE 1999. pp. 411–420 (1999)
15. Elgammal, A., Türetken, O., Heuvel, W.J.v.d., Papazoglou, M.P.: Root-cause analysis of design-time compliance violations on the basis of property patterns. In: ICSOC 2010. LNCS, vol. 6470, pp. 17–31 (2010)
16. Fahland, D., de Leoni, M., van Dongen, B.F., van der Aalst, W.M.P.: Conformance Checking of Interacting Processes with Overlapping Instances. In: BPM. LNCS, vol. 6896, pp. 345–361 (2011)
17. Fötsch, D., Pulvermüller, E., Rossak, W.: Modeling and verifying workflow-based regulations. In: ReMo2V 2006. CEUR Workshop Proceedings, vol. 241 (2007)
18. Ghose, A., Koliadis, G.: Auditing Business Process Compliance. In: ICSOC. Lecture Notes in Computer Science, vol. 4749, pp. 169–180 (2007)

19. Giblin, C., Liu, A.Y., Müller, S., Pfitzmann, B., Zhou, X.: Regulations expressed as logical models (realm). In: JURIX 2005. Frontiers in Artificial Intelligence and Applications, vol. 134, pp. 37–48. IOS Press (2005)
20. Goedertier, S., Martens, D., Vanthienen, J., Baesens, B.: Robust Process Discovery with Artificial Negative Events. Journal of Machine Learning Research 10, 1305–1340 (2009)
21. Gruhn, V., Laue, R.: Patterns for timed property specifications. Electr. Notes Theor. Comput. Sci. 153(2), 117–133 (2006)
22. Kharbili, M.E., Medeiros, A.K.A.d., Stein, S., Aalst, W.M.P.v.d.: Business process compliance checking: Current state and future challenges. In: MobIS 2008. LNI, vol. 141, pp. 107–113. GI (2008)
23. Kharbili, M.: Business process regulatory compliance management solution frameworks: A comparative evaluation. In: APCCM 2012. CRPIT, vol. 130, pp. 23–32. ACS (2012)
24. Lu, R., Sadiq, S.W., Governatori, G.: Compliance aware business process design. In: BBM Workshops 2007. LNCS, vol. 4928, pp. 120–131. Springer (2008)
25. Montali, M., Pesic, M., Aalst, W.M.P.v.d., Chesani, F., Mello, P., Storari, S.: Declarative Specification and Verification of Service Choreographies. ACM Transactions on the Web 4(1), 1–62 (2010)
26. Munoz-Gama, J., Carmona, J.: A Fresh Look at Precision in Process Conformance. In: BPM 2010. LNCS, vol. 6336, pp. 211–226. Springer (2010)
27. Munoz-Gama, J., Carmona, J.: Enhancing Precision in Process Conformance: Stability, Confidence and Severity. In: CIDM 2011. IEEE (2011)
28. Pitzmann, B., Powers, C., Waidner, M.: Ibm's unified governance framework (ugf). Tech. rep., IBM Research Division, Zurich (2007)
29. Ramezani, E., Fahland, D., Aalst, W.M.P.v.d.: Diagnostic information in compliance checking. Tech. rep., BPM Center Report BPM-12-11, BPMcenter.org (2012)
30. Ramezani, E., Fahland, D., Werf, J.M.E.M.v.d., Mattheis, P.: Separating compliance management and business process management. In: BPM Workshops 2011. LNBIP, vol. 100, pp. 459–464. Springer (2012)
31. Rozinat, A., Aalst, W.M.P.v.d.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. 33(1), 64–95 (2008)
32. Sadiq, S.W., Governatori, G., Namiri, K.: Modeling control objectives for business process compliance. In: BPM 2007. LNCS, vol. 4714, pp. 149–164. Springer (2007)
33. Schleicher, D., Grohe, S., Leymann, F., Schneider, P., Schumm, D., Wolf, T.: An approach to combine data-related and control-flow-related compliance rules. In: SOCA 2011. pp. 1–8. IEEE (2011)
34. Schleicher, D., Anstett, T., Leymann, F., Schumm, D.: Compliant business process design using refinement layers. In: OTM Conferences 2010. LNCS, vol. 6426, pp. 114–131. Springer (2010)
35. Schleicher, D., Fehling, C., Grohe, S., Leymann, F., Nowak, A., Schneider, P., Schumm, D.: Compliance domains: A means to model data-restrictions in cloud environments. In: EDOC. pp. 257–266 (2011)
36. Schumm, D., Leymann, F., Ma, Z., Scheibler, T., Strauch, S.: Integrating compliance into business processes: Process fragments as reusable compliance controls. In: MKWI 2010. Universitätsverlag Göttingen (2010)
37. Schumm, D., Türetken, O., Kokash, N., Elgammal, A., Leymann, F., Heuvel, W.J.v.d.: Business process compliance through reusable units of compliant processes. In: ICWE Workshops 2010. LNCS, vol. 6385, pp. 325–337. Springer (2010)
38. Weerdt, J.D., Backer, M.D., Vanthienen, J., Baesens, B.: A Robust F-measure for Evaluating Discovered Process Models. In: CIDM 2011. pp. 148–155. IEEE (2011)
39. Wolter, C., Meinel, C.: An approach to capture authorisation requirements in business processes. Requir. Eng. 15(4), 359–373 (2010)