# Diagnostic Information for Compliance Checking of Temporal Compliance Requirements

Elham Ramezani Taghiabadi, Dirk Fahland,
Boudewijn F. van Dongen, and Wil M.P van der Aalst

Eindhoven University of Technology, The Netherlands
{e.ramezani,d.fahland,b.f.v.dongen,w.m.p.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 or service level agreements. Deviations may be costly and expose an organization to severe risks. Compliance checking is of growing importance for the business process management and auditing communities. This paper presents an approach for checking compliance of observed process executions recorded in an event log to *temporal compliance requirements*, which restrict when particular activities may or may not occur. We show how temporal compliance requirements discussed in literature can be unified and formalized using a generic temporal compliance rule. To check compliance with respect to a temporal rule, the event log describing the observed behavior is aligned with the rule. The alignment then shows which events occurred out of order and which events deviated by which amount of time from the prescribed behavior. This approach integrates with an existing approach for control-flow compliance checking, allowing for multi-perspective diagnostic information in case of compliance violations. We show the feasibility of our technique by checking temporal compliance rules of real life event logs.

**Keywords:** compliance checking, process mining, data aware conformance checking, Petri nets.

## 1 Introduction

Processes supported by information systems need to comply with regulations, laws and service level agreements set by both internal and external stakeholders. Failing to comply may be costly, therefore organizations need to continuously check whether processes are executed within a given set of boundaries. Deviations of the observed behavior from the specified compliant behavior may point to fraud, malpractice, risks, and inefficiencies. Five types of compliance-related activities can be identified [8,23,7,19]: (1) *determine* compliance requirements that have to be satisfied by the particular process and system, (2) *formalize* compliance requirements in a suitable form, (3) *implement and configure* information systems such that they fulfil compliance requirements, (4) *check* whether compliance requirements are met, and (5) *improve* the process and underlying systems based on diagnostic information to improve compliance.

Compliance checking is gaining importance because of the availability of event data on one hand and changing legislations on the other hand. The organizations are not only required to obey the laws and regulations but often required to comply with standards and contractual obligations. In many standards such as clinical guidelines and constraints governing cooperative business such as service level agreements, time is of utmost significance. In processes that are subject to such guidelines and agreements, it is often extremely important to meet deadlines, optimize response time, stay compliant about durations, adhere to constraints about delay between activities and periodic repetition of actions. At the same time, new technologies are providing opportunities to systematically observe processes at a detailed level by recording all process relevant events.

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

The compliance requirements considered in this paper constrain at which *time* activities may, must or must not occur. Compliance violations regarding time cannot be detected using existing control-flow checking techniques such as [21]; as these techniques abstract from a concrete notion of time, both when specifying compliance rules and when checking reality recorded in the event log against specified behavior. Therefore for temporal compliance checking, it is required to express temporal constraints with an explicit notion of time, capture notion of time in the event log and compare the specified time in temporal constraints with time recorded in the event log.

This paper addresses the problem of backward compliance checking for temporal compliance requirements (i.e., compliance requirements restricting process time). We propose a technique for temporal compliance checking that seamlessly integrates with control-flow compliance checking. Most importantly, the technique provides detailed diagnostic information in case of non-compliant behavior: it shows for each case *which* events violated temporal requirements and *when* the event should have occurred to be compliant. Our temporal compliance checking technique leverages a recent *data-aware* conformance checking technique [13] that allows to check conformance of a log with respect to a *data-aware Petri net*. We show that every temporal compliance requirement discussed in literature (and many more) can be formalized in a simple data-aware Petri net, by making time a data attribute of the specification. The conformance checker [13] then compares the observed temporal behavior in the event data to the compliant temporal behavior specified in the Petri net. In case of deviations, the conformance checker highlights which events occurred out of order, and by how much time an event deviated from the compliant behavior. Moreover, we show how this temporal compliance checking can be combined with control-flow compliance checking to check complex compliance requirements involving control-flow and temporal aspects. The technique has been implemented as a ProM plug-in and has been applied in a case study using event data and compliance requirements for a real-life business process in the public sector.

The remainder of this paper is organized as follows. We recall conformance checking techniques for control-flow and data-flow in Sect. 2. In Sect. 3 introduces the problem of temporal compliance checking and our proposed solution. Experimental results are presented and discussed in Sect. 4. We discuss related work in Sect. 5 and conclude in Sect. 6.

## 2 Preliminaries

This section recalls basic conformance checking notions for control-flow conformance [1,3] and data-flow conformance [13] on which we build for temporal compliance checking.

**Logs.** Conformance checking relates behavior that *has happened* and was recorded in an event log $L$ to a formal specification $S$ that describes which behavior *should have happened*. In this context, an *event log* $L$ is a *set* of *traces*. Each trace $\sigma_L \in L$ describes a particular *case* (i.e., a process instance) as a sequence of *events* $\sigma_L = e_1 e_2 \ldots e_n$. An event $e$ has a number of attributes, typically referring to the *activity* executed or the *time* of its execution. For instance, we write $e.activity = a$ and $e.time = t$ to refer to the *value* of the activity and of the time attribute of event $e$, respectively; as a shorthand, we may write $e = (a, t)$. An example of a log trace is $\sigma_L = \langle (A, 1)(C, 3)(C, 7)(D, 12) \rangle$ stating $A$ happened at time 1, $C$ happened at 3, another $C$ happened at 7 and finally $D$ happened at 12.

**Specified Behaviors.** The specification of behavior that should have happened can be expressed in various ways, for instance as a Petri net [1] or as declarative constraints [12] or predicate logic. In essence, each specification describes a set $S$ of *admissible* behaviors, again being sequences of events having attributes. Typically, the set $S$ is very large, varying over all admissible attribute value combinations. The set $S$ is the semantic notion of all compliant traces. For example, a specification could describe the traces $S$ shown in Fig. 1(left). The traces state that $A$ happens at time 1, $B$ happens at time 4, $C$ happens any time between 1.1 and 7, and $D$ always happens 10 time units after $C$. Note that the ordering of $B$ and $C$ depends on the times of their occurrences.

In this paper, we use Petri nets to specify the admissible behavior $S$ in a compact form. Figure 1(right) shows a Petri net variant, called *data-aware Petri net* [13], that specifies the above admissible sequences of events *including the values of the time attribute*. That is, the firing sequences of the Petri net $N_S$ is the set $S$ given in Fig. 1(left). In $N_S$, transitions have additional annotations that specify attribute values of the event $e$ that is created by the occurrence of a transition. For instance, the guard $[e.time = 1]$ of transition $A$ expresses that the event $e$ created by $A$ has the value 1 of its $time$ attribute. Likewise, an event created by $B$ has value 4 of its time attribute. The guard of $C$ permits any value between 1.1 and 7 for attribute $time$ of $C$. The relation between events of $C$ and $D$ is expressed by the help of *variable* $t_C$. The $time$ value of the most recent event produced by $C$ is stored in variable $t_C$, expressed by the *write statement* $\{t'_C = e.time\}$ annotated at $C$. The guard of $D$ then expresses that the $time$ value of $D$ has to be exactly 10 time units after the value stored in $t_C$. Note that the net in Fig. 1

$$
\begin{aligned}
S = \{ & \langle \text{(A,1)(C,1.1)(B,4)(D,11.1)} \rangle, \\
& \ldots, \\
& \langle \text{(A,1)(C,3.9)(B,4)(D,13.9)} \rangle, \\
& \ldots, \\
& \langle \text{(A,1)(B,4)(C,4.1)(D,14.1)} \rangle, \\
& \ldots, \\
& \langle \text{(A,1)(B,4)(C,7)(D,17)} \rangle \}
\end{aligned}
$$

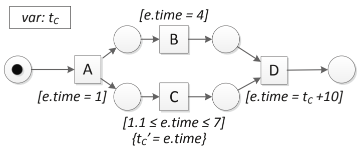

**Fig. 1.** Data-aware Petri net $N_S$ specifying admissible times of occurrences of activities

has no explicit notion of time; the annotations simply constrain numerical values of the event attribute $time$.

**Aligning Observed to Specified Behaviors.** An observed trace in a log may deviate from specified behaviors. For instance, the non-compliant trace $\sigma_L = \langle (A,1)(C,3)$ $(C,7)(D,12)\rangle$ is not specified by $N_S$ of Fig. 1. To understand where and how $\sigma_L$ deviates from the behaviors $S$, we *align* $\sigma_L$ to $S$, as follows [1,3,13].

The idea is to find a specified trace $\sigma_S \in S$ that is as similar as possible to $\sigma_L$; the differences between $\sigma_S$ and $\sigma_L$ then indicate deviations. We relate $\sigma_L$ to any trace $\sigma_S \in S$ by pairing events. Let $\mathcal{A}_L$ and $\mathcal{A}_S$ be the activity names of events in $L$ and $S$, respectively. Let $\ell : \mathcal{A}_S \to 2^{\mathcal{A}_L}$ map each specified activity to a set of log activities. Intuitively, an event $f$ of the specification $S$ pairs with an event $e$ in the log $L$ if $e.activity \in \ell(f.activity)$.

Let $\mathcal{E}_L$ and $\mathcal{E}_S$ be the universe of all possible log events and of all specification events, respectively. A *move* of ($\sigma_L$ and $S$) is a pair $(x, y) \in (\mathcal{E}_L \cup \{\gg\}) \times (\mathcal{E}_S \cup \{\gg\}) \setminus \{(\gg, \gg)\}$. For $x \in \mathcal{E}_L$ and $y \in \mathcal{E}_S$, we call

1. $(x, \gg)$ a *move on log*,
2. $(\gg, y)$ a *move on specification* $S$,
3. $(x, y)$ a *synchronous move* iff $x.activity \in \ell(y.activity)$ and $x.attr = y.attr$ for every other attribute $attr$ of $x$ and $y$, and
4. $(x, y)$ a synchronous move with *data deviation* iff $x.activity \in \ell(y.activity)$ and $x.attr \neq y.attr$ for some other attribute $attr$ of $x$ and $y$.

An *alignment* of a trace $\sigma_L \in L$ to $S$ is a sequence $\gamma = \langle (x_1, y_1) \ldots (x_n, y_n)\rangle$ of moves (of $\sigma_L$ and $S$) such that the projection $x_1 \ldots x_n$ to $\mathcal{E}_L$ is the original trace $\sigma_L$ (i.e., $\langle x_1 \ldots x_n\rangle|_{\mathcal{E}_L} = \sigma_L$), and the projection $y_1 \ldots y_n$ to $\mathcal{E}_S$ is a specified trace (i.e., $\langle y_1 \ldots y_n\rangle|_{\mathcal{E}_S} \in S$.

For example, the trace $\sigma_L = \langle (A,1)(C,3)(C,7)(D,12)\rangle$ has among others the following three alignments to the traces $S$ of Fig. 1: $\gamma_1 = \begin{array}{c|c|c|c|c} (A,1) & (C,3) & \gg & (C,7) & (D,12) \\ \hline (A,1) & (C,2) & (B,4) & \gg & (D,12) \end{array}$, $\gamma_2 = \begin{array}{c|c|c|c|c|c} (A,1) & \gg & (C,3) & \gg & (C,7) & (D,12) \\ \hline (A,1) & (C,2) & \gg & (B,4) & \gg & (D,12) \end{array}$, and $\gamma_3 = \begin{array}{c|c|c|c|c} (A,1) & (C,3) & \gg & (C,7) & (D,12) \\ \hline (A,1) & \gg & (B,4) & (C,7) & (D,17) \end{array}$. Alignments $\gamma_1$ and $\gamma_2$ yield the same specified trace $\langle (A,1)(C,2)(B,4)(D,12)\rangle$; $\gamma_3$ yields a different trace $\langle (A,1)(B,4)(C,7)(D,17)\rangle$.

**Diagnostic Information from Deviations.** All alignments differ in the kind of deviations they show. A move on log $(x, \gg)$ indicates that trace $\sigma_L$ had an event $x$ that was not supposed to happen according to specification $S$ whereas a move on specification $S$ $(\gg, y)$ indicates that $\sigma_L$ was missing an event that was expected according to $S$. Synchronous moves with data deviations indicate that the observed event had other attribute values than specified. As the alignment preserves the position relative to trace $\sigma_L$, we can locate the exact position where $\sigma_L$ had an event too much or missed an event compared to $S$.

According to $\gamma_1$, the $time$ attribute of the first $C$ is *wrong* (it should have been 2 instead of 3 according to the synchronous move with *data deviation*), $B$ should have occurred (according to the move on $S$), and the second $C$ should not have occurred (move on log). According to $\gamma_2$, none of the $C$ events was correct, but there should have been another $C$ event at time 2. According to $\gamma_3$, the second $C$ event was correct at time 7, the first $C$ event at time 3 was wrong, and $D$ should instead have occurred at time 17 (synchronous move with *data deviation*).

**Computing Alignments.** The conformance checking problem in this setting is to find for a given trace $\sigma_L$ and specification $S$ a *best* alignment $\gamma$ of $\sigma_L$ to $S$ s.t. no other alignment has fewer non-synchronous moves or moves with data deviations. The techniques of [1,3] and [13] find such a best alignment using a cost-based approach: a cost-function $\kappa$ assigns each move $(x, y)$ a cost $\kappa(x, y)$ s.t. a synchronous move (without data deviations) has cost 0 and all other types of moves have cost $> 0$. The choice of costs depends on the particular domain and can be set for instance based on how likely a particular deviation is known to happen. By giving frequent deviations fewer costs than infrequent deviations, the best alignment is the one giving the *most probable* compliant trace. For example, assuming cost 10 for all non-synchronous moves (knowing control-flow is usually followed), cost 3 for data deviations by $C$ (typically happens as specified) and costs 1 for data deviations by $D$ (typically known to deviate), $\gamma_1$ has cost 23, $\gamma_2$ has cost 40, and $\gamma_3$ has cost 21, making $\gamma_3$ the best alignment to the most probable trace.

The A$^\star$-based search on the space of (all prefixes of) all alignments of $\sigma_L$ to $S$ described in [1,3] can be used to find a best alignment for $\sigma_L$ and $S$ (when attributes other than $activity$ are ignored). This approach is extended in [13] to find *data-aware alignments*; an ILP solver finds among all synchronous moves values for attribute of $S$ such that the data deviations are minimized. In the following, we apply alignments for temporal compliance checking.

## 3 Temporal Compliance Checking

This section presents our main contribution, an approach for checking temporal compliance on past executions recorded in event logs. We first recall different dimensions of compliance requirements. After that we present our approach that allows to integrate checking for control-flow and temporal compliance requirements together with a generic framework for formalizing various temporal requirements.

### 3.1 Compliance Requirements

Compliance requirements prescribe how internal or cross-organizational business processes have to be designed or executed. They originate from legislations and restrict one or several perspectives of a process (control flow, data flow, process time or organizational aspects), they can restrict each case individually or a group of cases, or prescribe properties of process executions or process design [21]. These different aspects of compliance give rise to the compliance rule framework shown in Fig. 2. A complex compliance requirement covering several perspectives of a process can be decomposed into smaller compliance rules, each covering a single aspect along the dimensions of this framework.

For example, a compliance requirement might state: “The treatment with antibiotics must be administered with one dose per day for 3 days in a row. After each cycle of 3 treatments, in case of necessity, the treatment can be extended for other cycles; but there should be delay of at least one week between two subsequent cycles of treatment”.

This requirement can be divided into three different compliance rules: (1) *(control flow)* "antibiotics must be administered in cycles of 3 occurrences", (2) *(process time)* "between two subsequent administration of antibiotics in a cycle, there should be one day delay", and (3) *(process time)* "between two subsequent cycles, there should be at least one week delay", each taking only one perspective (control flow or process time) into account. The compliance checking technique of [21] is able to check control-flow compliance rules, but does not provide a notion of *time* and therefore cannot check temporal compliance rules. In the following, we investigate the relation between the control-flow and time perspective, then present a new technique that supports time to check temporal compliance rules, and finally show how to integrate diagnostic information from both perspectives for comprehensive compliance diagnostics.

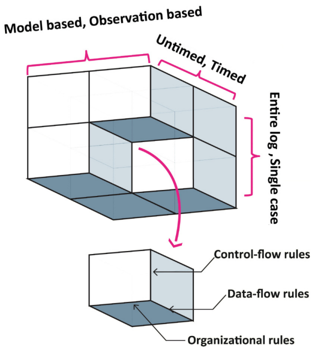

**Fig. 2.** Compliance Rule Framework [21]

## 3.2 Separating Temporal- and Control-Flow Compliance Checking

We conducted an extensive literature survey and identified numerous works [11,18,10,24,14,2] discussing temporal compliance rules and their formalization. Typically every compliance requirement restricting the process time implies a control flow compliance rule in addition to a temporal compliance rule. Even if the ordering of activities is not restricted in the compliance requirement, at least the existence of some activities is specified. In the general case, the control-flow rule constrains more than just the existence of activities, for instance that "antibiotics must be administered in cycles of 3 occurrences". This leads to a simple assumption for temporal compliance requirements: a temporal compliance rule constrains the occurrences of events specified in a given control-flow rule (e.g., "between two subsequent administration of antibiotics in a cycle, there should be one day delay"); a "larger" temporal rule simply implies a larger control-flow rule.

A control-flow rule may have to hold multiple times in a trace [16], based on repeated occurrences of events. For instance, if antibiotics are administered 6 times in total, the control-flow rule given above has to hold twice; the associated temporal rule has to hold whenever the control-flow rule occurs. We collected and categorized available temporal compliance rules (15 temporal compliance rules) in a framework distributed over 7 categories shown in Tab. 1.

Some example rules with their category are given next. The complete collection of compliance rules and their formalization is described in [22].

- *Repetition.Rule2*: "The delay between execution of two subsequent activities $A$ and $B$ within all occurrences of a control-flow rule, must be within $[\alpha, \beta]$ time units"
- *Instance Duration.Rule1*: "Every occurrence of a control flow rule must be completed within $[\alpha, \beta]$ time units since time $t$."
- *Validity.Rule1*: "Every activity $A$ within all occurrence of a control flow rule must be completed within $[\alpha, \beta]$ time units since it starts."

**Table 1.** Categorization of the 15 Temporal Compliance Rules

| Category (Rules) | Description |
|---|---|
| Instance Duration (2) | Limits the time length in which a control-flow rule instance must hold. [24] |
| Delay Between Instances (1) | Limits the delay between two subsequent instances of a control-flow rule [11,10,18,2] |
| Validity (3) | Limits the time length in which an activity can be executed.[11,10,18,24] |
| Time Restricted Existence (2) | Limits the execution time of an activity in calendar.[11,10,18] |
| Repetition (2) | Limits the delay between execution of two subsequent activities.[11,10,18,24,14,2] |
| Time Dependent variability(1) | Limits choice of a process path among several ones with respect to temporal aspects.[11,10,18,24] |
| Overlap (4) | Limits start and completion of an activity with respect to start and completion of another activity.[11,10,18,24] |

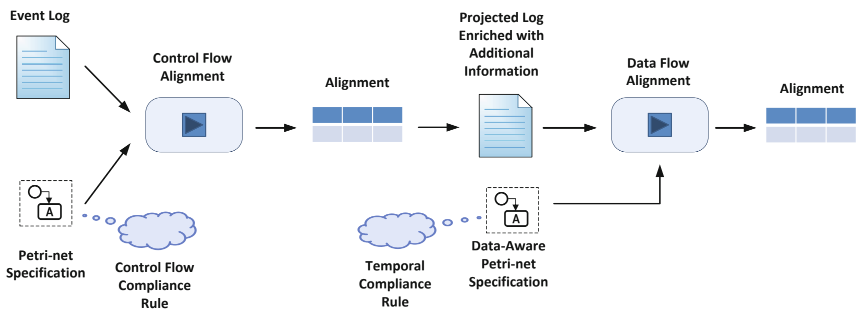

**Fig. 3.** Temporal Compliance Checking Overview

- *Time Restricted Existence.Rule1*: “Every activity $A$ within all occurrences of a control flow rule may only be executed at times $t_1, ..., t_n$”.
- *Time Dependent Variability.Rule1*: “Within all occurrences of a control flow rule, activity $B$ must be executed within $[\alpha_1, \beta_1]$ time units since time $t_1$ if $A$ has occurred within $[\alpha_2, \beta_2]$ time units since time $t_2$.”

The dependency between control-flow rules and temporal rules raises a challenge for temporal compliance checking: we first have to identify the different occurrences of a control-flow rule, for which then the temporal rule can be checked. This gives rise to our approach shown in Fig. 3.

We decompose a complex compliance requirement into a control-flow rule and a temporal rule. The event log is first aligned to the control-flow rule using the technique of Sect. 2 to identify control-flow violations in terms of missing or inserted events; this alignment will also distinguish multiple occurrences of the same control-flow rule within one trace. For each alignment, we enrich the log with information about multiple occurrences of rules and control-flow violations. The enriched log is then used to check temporal compliance using the data-aware alignments of Sect. 2.

However, there is a small challenge in decomposing control-flow and temporal compliance checking. In case a control-flow violation can be attributed to different occurrences of a control-flow rule, there exist more than one alignment of a trace to the control-flow rule. Picking the wrong control-flow alignment could introduce false positives on temporal compliance, which we eliminate as follows: we compute for each trace all its control-flow alignments; each alignment leads to an enriched trace *variant* for which we compute temporal compliance; the trace variant with the best temporal compliance (containing only real violations) is returned and all other variants (containing false positives) are discarded.

Aligning control-flow and temporal checking has an advantage regarding diagnostic information. A severe control-flow violation implies a violation of the temporal compliance rule. Checking temporal compliance alone might obscure insights into the nature of the violation. By integrating control-flow and temporal compliance checking, we can present more meaningful diagnostic information to a user. For this, the existing control-flow checking technique has to be extended, as we describe next.

## 3.3 Extended Control-Flow Compliance Checking

A control-flow compliance checking technique based on alignments was presented in [21]. In this technique, a compliance rule is formalized as a Petri net that describes all admissible sequences of activities. A collection of over typical 50 control-flow compliance rules and their formalization as parameterized Petri net patterns are available from a comprehensive repository [22]. In this section, we show how to extend the patterns of [21] to distinguish multiple occurrences of the same rule in a trace. Checking control-flow compliance of a log to an extended pattern then allows to enrich the log with information needed for temporal compliance checking.

For example, the Petri net pattern of Fig. 4 formalizes the compliance rule "activity $A$ must be executed in groups of $k$ occurrences" (shown for $k = 3$). The core of the rule is formalized in the grey-shaded part between transitions $I_{st}$ and $I_{cmp}$. The rule becomes $active$ when $I_{st}$ occurs. Then activity $A$ has to occur 3 times before the rule can complete (each time $A$ occurs, one token is taken from $todo$ and put on $done$). In between, arbitrary other activities can occur, expressed by transition $\Omega$. The compliance rule may hold multiple times in a trace; this behavior is captured by the cycle involving $I_{st}$ and $I_{cmp}$. Whenever $I_{st}$ occurs, it puts 3 tokens in the place $todo$ which activates a new *instance* of the rule "activity $A$ occurs in groups of 3"; the instance completes with transition $I_{cmp}$ which removes all tokens from $done$ and puts a token on $p_1$. The entire Petri net of Fig. 4 thus allows for multiple instances of the compliance rule, each instance is framed by the $I_{st}$ and $I_{cmp}$ transitions; between two instances arbitrary other activities are allowed as expressed by the $\Omega$-transition attached to place $p_1$. The net has a dedicated place *Initial* and a place *Final*, a *Start* and an *End* transition. A compliant behavior takes the net from the initial marking to the final marking (just one token in $Final$) showing arbitrary many instances of the compliance

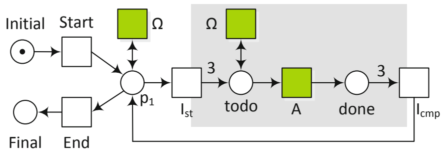

**Fig. 4.** Petri net formalizing a control-flow rule

rule. Every Petri net of [21] formalizing a control-flow rule can be extended in this way to distinguish multiple instances of the same rule in a trace; see [22] for the complete pattern repository.

The alignments of Sect. 2 can be used to check compliance of a trace to the compliance rule “activity $A$ must be executed in groups of $k$ occurrences”. Assume the trace $\sigma = \langle (B,1)(A,2)(A,30)(A,54)(A,100)(C,123)(A,162)(D,173)\rangle$ to be given. For control-flow compliance checking, we ignore attributes other than $activity$ and thus align the trace $\sigma = \langle BAAAACAD\rangle$ to the net of Fig. 4. When aligning $\sigma$, $A$ maps to $A$ and $\Omega$ maps to all other events $B, C, D$ which are not relevant for the rule. Additionally, we assume transitions $Start$, $End$, $I_{st}$ and $I_{cmp}$ to be *silent* so that moves on the specification (without corresponding event in $\sigma$) have cost 0. The approach of Sect. 2 yields a best alignment $\gamma_1 = \begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c|c|c} \gg & B & \gg & A & A & A & \gg & \gg & A & C & \gg & A & \gg & D & \gg \\ \hline Start & \Omega & I_{st} & A & A & A & I_{cmp} & I_{st} & A & \Omega & A & A & I_{cmp} & \Omega & End \end{array}$ showing 1 instance of the rule with 3 occurrences of $A$ and 1 instance of the rule with 2 occurrences of $A$. The alignment also shows a missing event $A$ in the second instance by the move on specification $(\gg, A)$. Note that $\sigma$ has 6 more best alignments to the net of Fig. 4 varying in where the move $(\gg, A)$ is placed. The subsequent steps (shown for $\gamma_1$) would have to be executed for each of these alignments.

To align temporal compliance checking with control-flow compliance checking, we enrich the original trace $\sigma$ with information about rule instances and control-flow deviations, as follows. (1) Translate each move of the alignment $\gamma$ into a log event, where each event originating in a non-synchronous move is marked by a special “move” attribute. (2) Enrich each event of a synchronous move with all attributes of the original event in trace $\sigma$. (3) Provide each event of a move on specification with missing attributes, in particular an event without a $time$ attribute gets the $time$ value of the directly preceding event (except $Start$ and $I_{st}$ which get the $time$ value of the succeeding event).

For example, using alignment $\gamma_1$ we enrich trace $\sigma$ given above to the traces $\sigma_{\gamma,1} = \langle (Start,1)(B,1)(I_{st},2)(A,2)(A,30)(A,54)(I_{cmp},54)(I_{st},100)(A,100)(C,123)(A, 123, missing)(A,162)(I_{cmp},162)(D,173)(End,173)$. This traces now contains enough information to check temporal compliance.

## 3.4 Formalizing and Checking Temporal Compliance Rules

This section introduces a new technique to check temporal compliance. We express temporal compliance rules using the *data-aware Petri nets* of Sect. 2, which allow to describe and constrain *time* in processes. The corresponding alignments of a data-aware Petri net to a log [13] will then allow to check for temporal compliance and provide detailed diagnostic information. For the alignment, we use the enriched log provided by the technique of Sect. 3.3. This enriched log distinguishes all different instances of the control-flow rule that underlies the temporal rule to be checked. In addition, it allows to integrate diagnostic information for violations in the control-flow and the temporal perspective.

**Formalizing Temporal Constraints.** We explain the formalization of temporal compliance rules by the first temporal rule of our running example. The concrete temporal rule reads “Between two subsequent administration of antibiotics in a cycle, there should be

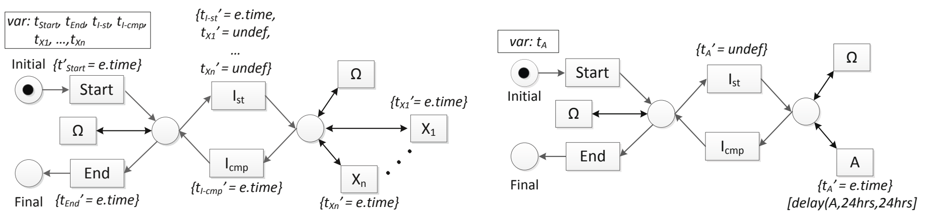

**Fig. 5.** Temporal Petri-net Pattern, generic (left) and instantiated (right)

one day delay," which we abstract to "the delay between two subsequent executions of activity $A$ in an instance of a control-flow rule, must be within $[\alpha, \beta]$ time units." The data-aware Petri net of Fig. 5(right) formalizes this rule.

The Petri net has a very simple control-flow structure that just distinguishes begin and end of a trace (places *Initial* and *Final*), and whether the trace is *within* an instance of a control-flow rule (after $I_{st}$ occurred) or *outside* a control-flow rule (after $I_{cmp}$ occurred). Transitions labeled $\Omega$ allow occurrences of all other activities not constrained by the temporal rule. The actual temporal aspect is described by the variable $t_A$ and the data annotations at transition $A$ and $I_{st}$. Annotation $\{t'_A = e.time\}$ at $A$ ensures that $t_A$ holds the timestamp of the most recent occurrence of activity $A$. The most important annotation is the guard $[delay(A, \alpha, \beta)]$ defined by $delay(A, \alpha, \beta) \equiv t'_A \in [t_A{+}\alpha, t_A{+}\beta] \vee t_A = undef$. The guard states that the time $t'_A$ of the current occurrence of $A$ has to be in the interval $[t_A + \alpha, t_A + \beta]$, where $t_A$ is the timestamp of the most recent occurrence of $A$. As the rule only ranges over occurrences of $A$ within the same instance of the control-flow rule, we have to take special care for the first occurrence of $A$ in an instance. The annotation at $I_{st}$ initializes $t_A = undef$ so that the guard of $A$ also holds for the first $A$. By setting parameters $A =$ *antibiotic administration* and $\alpha = \beta = 24$ hours, the pattern of Fig. 5 formalizes the given temporal rule.

**Checking Temporal Compliance.** We check compliance of a trace to the formalized rule on the enriched log trace obtained in Sect. 3.3, for instance trace $\sigma_{\gamma,1}$. The data-aware alignment technique explained in Sect. 2 compares the time stamp of events in $\sigma_{\gamma,1}$ with admissible time stamps defined in the guards of the data-aware Petri net and will give a data-aware alignment $\gamma_1^t$, with the least cost, as follows: $\gamma_1^t = \frac{(Start,1)|(B,1)|(I_{st},2)|(A,2)|\ldots}{(Start,1)|(\Omega,1)|(I_{st},2)|(A,2)|\ldots}$

$$\frac{\ldots|\textbf{(A,30)}|(A,54)|(I_{cmp},54)|(I_{st},100)|(A,100)|(C,123)|\textbf{(A,123,missing)}|\textbf{(A,162)}|(I_{cmp},162)|(D,173)|(End,173)}{\ldots|\textbf{(A,26)}|(A,54)|(I_{cmp},54)|(I_{st},100)|(A,100)|(\Omega,123)|\textbf{(A,124,missing)}|\textbf{(A,148)}|(I_{cmp},162)|(\Omega,173)|(End,173)}.$$

As is shown in the alignment $\gamma_1^t$ the second $A$ in the first instance occurred 28 time units after the preceding $A$, which violates the temporal rule. The data-aware alignment in addition returns the time at which the event should have occurred at the bottom row of the alignment. In the same way, two deviations in the second iteration are highlighted. However, the "correct" timestamps 124 and 148 suggested by the alignment have to be inspected carefully as in the second instance the second $A$ was missing in the original log (a control-flow violation indicated by the attribute value $missing$). Recall from Sect. 3.2 that we may have to check several enriched variants of the same original trace (differing in control-flow violations); after checking all variants, the one with the least temporal violations is returned and all other are discarded.

**A Generic Temporal Pattern.** As said in Sect. 3.2, we identified 15 generic temporal compliance rules [22]. Each rule can be formalized in a data-aware Petri net similar to Fig. 5(right). The generic pattern is shown in Fig. 5(left). It permits to constrain occurrences of $n$ generic activities $X_1, \dots, X_n$, as well as the $\mathit{Start}$ and $\mathit{End}$ of a trace and start and end of each instance (by $I_{st}$ and $I_{cmp}$). Each formalization of a compliance rule assigns a guard to one or more transitions of the pattern, depending on the particular temporal property. We show some more formalizations next.

The rule "The delay between execution of two subsequent instances of a control-flow rule, must be within $[\alpha, \beta]$ time units." (which expresses the second temporal rule of our running example of Sect. 3.1.) The formalization of this rule instantiates Fig. 5(left) with $n = 0$ (no activity $X_i$), variables $t_{I_{st}}$ and $t_{I_{cmp}}$ and the guard $\mathit{delay2}(I_{cmp}, I_{st}, \alpha, \beta) \equiv t'_{I_{st}} \in [t_{I_{cmp}} + \alpha, t_{I_{cmp}} + \beta] \vee t_{I_{cmp}} = \mathit{undef}$ assigned to transition $t_{I_{st}}$. This way, $I_{st}$ is only allowed to occur between $t_{I_{cmp}} + \alpha$ and $t_{I_{cmp}} + \beta$ where $t_{I_{cmp}}$ is the last time $I_{cmp}$ occurred (if there was a last occurrence). Checking temporal compliance of $\sigma_{\gamma,1}$ of Sect. 3.3 to this rule for $\alpha = 7$ days and $\beta = \infty$ (and mapping all activities to $\Omega$), we obtain the following data-aware alignment:

$$\gamma_3^t = \frac{(\mathit{Start},1)\big|(B,1)\big|(I_{st},2)\big|(A,2)\big|(A,30)\big|(A,54)\big|(I_{cmp},54)\big|(I_{st},\boldsymbol{100})\big|(A,100)\big|(C,123)\big|\dots}{(\mathit{Start},1)\big|(\Omega,1)\big|(I_{st},2)\big|(\Omega,2)\big|(\Omega,30)\big|(\Omega,54)\big|(I_{cmp},54)\big|(I_{st},\boldsymbol{242})\big|(\Omega,100)\big|(\Omega,123)\big|\dots}.$$

The alignment $\gamma_3^t$ highlights a deviation for the start of the second instance of "3 *administrations of antibiotics*." According to the log, the second administration started just 46 hours after the preceding treatment where the rule requires a delay of at least 1 week (= 168 hours); the correct time is shown in the bottom row of the alignment.

Also compliance rules requiring the absence of an activity in a particular interval can be formalized: "No activity $A$ within all instances of a control flow pattern may be executed within $[\alpha, \beta]$ time units since time $t$." For this temporal rule, the generic temporal pattern of Fig. 5, has $n = 1$ transition. The guard for this temporal rule is: $\mathit{negation\_activity\_execution}(X_1, t, \alpha, \beta) \equiv t'_{X_1} \notin [t + \alpha, t + \beta]$. Here, the time $t$ can be a fixed time, or the time of some other activity (e.g., include $X_2$ in the pattern and define $t = t_{X_2}$).

Many temporal compliance requirements found in literature combine several constraints on the relation between the *start* and *completion* of two different activities; for instance, "within all instances of a control flow rule, activity $B$ must start within $[\alpha_{st}, \beta_{st}]$ time units after activity $A$ starts, and activity $B$ must complete within $[\alpha_{cmp}, \beta_{cmp}]$ time units before activity $A$ completes." For this temporal rule, the generic temporal pattern of Fig. 5, has $n = 4$ transitions labeled $A_{st}$, $A_{cmp}$, $B_{st}$ and $B_{cmp}$ expressing the start and completion of $A$ and $B$, respectively. The pattern uses the generic guard $\mathit{after}(X, Y, \alpha, \beta) \equiv t'_Y \in [t_X + \alpha, t_X + \beta] \vee t_X = \mathit{undef}$ twice: once as $\mathit{after}(A_{st}, B_{st}, \alpha_{st}, \beta_{st})$ at transition $B_{st}$ and once as $\mathit{after}(B_{cmp}, A_{cmp}, \alpha_{cmp}, \beta_{cmp})$ at transition $A_{cmp}$. Other combinations of this temporal constraint can be expressed in the same way varying the parameters of the guards.

Similarly, all other identified temporal compliance constraints identified in literature can be formalized by instantiating the generic temporal pattern of Fig. 5; see [22] for details. Each formalization is then eligible for temporal compliance checking using data-aware alignments. Our temporal compliance checking technique is not limited to predefined control-flow rules and temporal rules, but is extendible. In Sect. 4 we show

how we can adapt a generic temporal compliance rule for compound and complex temporal restrictions.

## 4 Experimental Results

Our temporal compliance checking technique is implemented in Process Mining Toolkit ProM, available from `www.promtools.org`, and was applied in a case study on real-life logs. We briefly discuss the implementation in ProM and then provide details on the case study.

**Implementation in ProM.** The temporal compliance checker is available in the package *Compliance* that provides 2 user-friendly plugins for temporal compliance checking. The first plugin provides control-flow compliance checking as described in Sect. 3.3: it takes as input a log and returns compliance diagnostics in form of an alignment, the control-flow rule to check compliance for is picked by the user from a rule repository[22] using a wizard. The second plugin takes the control-flow alignment, produces an enriched log and then checks temporal compliance of the log to a temporal rule that can be specified by the user through a wizard. The resulting alignment then provides diagnostic information by showing control-flow compliance violations and temporal violations projected into the events of the original log.

**Case Study.** We applied this implementation of the compliance checker in a case study for checking compliance of the building permit processes of five Dutch municipalities.

The municipalities may carry out the building permit process in different ways, as long as it is compliant to a number of regulations issued by the Dutch legislative. To test the feasibility of our temporal compliance checking technique, we selected a rather involved temporal compliance requirement that combines static and dynamic temporal aspects.

The compliance requirement was given informally: "Every application must be processed within at most 8 weeks from the date of a submitted request. If during the processing of the request, the organization requires additional information from the applicant, the time interval between asking for additional information and providing the information by the client must be added to the 8 weeks." This requirement is decomposed into two control-flow compliance rules and one temporal compliance rule:

**Control-Flow Compliance Rule 1:** "Every time activity $A$ is executed, it must be followed eventually by activity $D$."

**Control-Flow Compliance Rule 2:** "The sequence of activities $B$ and $C$ may only be executed after the execution of activity $A$ and before the execution of activity $D$."

**Temporal Compliance Rule:** "The delay between execution of two subsequent activities $A$ and $D$ in all instances of a control-flow pattern, must be $[\alpha, \beta + \beta_2]$ time units since time $t$, where $\beta_2$ is the time between executing $B$ and $C$."

We formalized the two control-flow compliance rules by instantiating a corresponding Petri net pattern from the compliance rule repository in [22]. Their generic parameters were mapped as follows: $A$ = *submit request*, $B$ = *request additional information*, $C$ = *receive additional information* and $D$ = *publish result*. The formalization of the

**Table 2.** Temporal Compliance Violations

| Municip. | Cases | Violations | | |
|---|---|---|---|---|
| | | # | delay (months) avg. | max. |
| M1 | 257 | 51 | 3 | 8 |
| M2 | 166 | 37 | 4 | 15 |
| M3 | 353 | 54 | 3 | 10 |
| M4 | 269 | 38 | 3 | 11 |
| M5 | 319 | 53 | 4 | 9 |

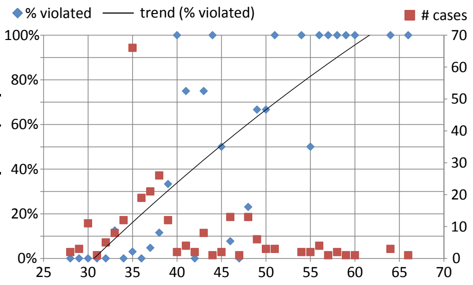

**Fig. 6.** Violations vs. handover of work

temporal compliance rule was derived by instantiating the generic temporal pattern of Fig. 5(left) as follows. The temporal rule requires 4 activities $A$, $B$, $C$, and $D$ and a variation of the guard $delay2$ introduced earlier. The guard $delay3(A, B, C, D, \alpha, \beta) \equiv t'_D \in [t_A + \alpha, t_A + \beta + (t_C - t_B)] \vee t_A = undef$ is assigned to transition $D$ with $\alpha = 0$ and $\beta = 8\,weeks$. As activities $B$ and $C$ are optional, we have to provide valid time stamps in variables $t_B$ and $t_C$ in each iteration. Therefore, the instance start transition $I_{st}$ of the temporal pattern (Fig. 5(left)) initializes both variables to 0, i.e., $\{t'_B = 0, t'_C = 0\}$, making their difference 0 if $B$ and $C$ are absent.

In order to check compliance of the building permit process to these requirements, we obtained five event logs, each coming from a different municipality. Each log was extracted from the municipality's case handling system and contained all activities performed for a case together with time stamps and resource information. In total we obtained 1408 cases as shown in Table 2 together containing 35352 events. Cases had 37 events in average and 97 events at most, distributed over 178 different event classes.

We first checked for compliance violations of the control-flow rules, followed by temporal compliance checking. In our analysis of the control-flow violations, we found 4 real violations and 40 *false positives* out of total number of 1408 cases in all municipalities. Most of the *false positives* occurred due to inaccurate time stamps and mistakes in data entries by a human user. The remaining real violations were mostly caused by the publication of the result before the municipality processed the additional information provided by the applicants. In general, the control-flow violations have not been severe because the process under analysis is quite standardized in all 5 municipalities. However the temporal violations in all municipalities seem to be significant. Table 2 shows the result of temporal compliance checking in different municipalities.

Based on the diagnostic information we got from the temporal compliance checking, we investigated the cause of the high number of temporal violations. We found that in compliant cases requests for additional information were issued no later than 2 months after receiving the application. In all violating cases, requests for additional information were made only later than 2 months after receiving the application, i.e., when compliance was violated already. This suggests that the process primarily needs to be improved in the initial phase when employees gather and assess information about a particular application.

Another influential factor in increasing the violations, is the number of handovers among employees working on a case. The diagram of Fig. 6 shows the distribution of cases and the percentage of violating cases over the number of resource handovers happening in a case, for *M1*. The red squares indicate for a particular value $x$ on the $x$-axis how many cases of *M1* had $x$ handovers of work; the blue diamond at $x$ indicate the percentage of cases with $x$ handovers of work that had a compliance violation. The share of violations increases as handovers increase. This observation suggests that less compliance violations occur when an employee handles several subsequent activities of a case. Though, further analysis on organizational structure and division of work in *M 1* is required. A process oriented division of work could decrease the number of handovers. In addition, a job rotation programme could enrich the skill set of employees to be able to execute more activities with respect to one case, hence decreasing the number of handovers and improve compliance.

Applying the technique presented in this paper, we were able to check compliance of all the traces in the event logs rather than being limited to sample based compliance checking. The technique is fast and works on large event logs because we can focus on events relevant to a specific compliance rule and abstract from all other events. The remaining effort for a human user is in formalizing the compliance requirements and analyzing results. The effort in formalizing requirements was kept low in our case study. We could pick available control-flow compliance patterns from an existing repository. A wizard helped in selecting the right pattern. The main effort was in expressing the temporal compliance requirement as the guard $delay3(A, B, C, D, \alpha, \beta)$ presented earlier; once the guard was identified, the constraint could quickly be formalized by instantiating the generic pattern through a wizard. Note that this formalization need to be done once. Checking could then be continued for all cases automatically.

The technique could identify, locate, and determine the extent of deviations. These diagnostic information can be used by the business analyst to analyze the cause of the deviations.

## 5 Related Work

Existing work in temporal compliance checking primarily focuses on verification at design time or at run time.

It is possible to derive temporal properties of acyclic process models by annotating tasks with intervals of execution and waiting times; execution times and waiting times of the entire process can then be derived by interval computations and compared against predefined constraints of total execution times [5]. In addition, the time-critical paths of a process model can be computed [20]. In a similar fashion, the approach in [15] formulates temporal constraints in terms of deadlines for completing an activity (relative to another activity). Reasoning on time intervals is used to verify whether a constraint is violated.

For verifying that a process with loops satisfies a general time-related constraint, typically temporal model checking techniques are applied. The properties of interest are *metric* temporal constraints, e.g., deadline on execution of activities in a business process. *Metric temporal logic* (MTL), a temporal logic with metric temporal constraints, can express typical compliance requirements as presented in this paper. Unfortunately, the model checking problem for MTL is undecidable over models with infinite traces [9]. By introducing so called observers on atomic propositions, the problem whether a process model, given as a timed transition system (TTS), satisfies an

MTL formula becomes decidable by a reduction to LTL model checking [2]. This approach allows to check temporal compliance of a real-time extension of Dwyer's specification patterns [4]. A similar approach is followed in [6] for checking whether an extended CCSL (Clock Constraint Specification Language) specification holds in a timed Petri net; CCSL is less expressive than the constraints that can be expressed and checked with our technique.

An alternative approach to describe temporal constraints is *timed Declare* [24] in which LTL-like constraints are extended with the notion of time. By a translation to timed automata, such constraints can be monitored at runtime to evaluate whether a process instance might or will violate a temporal constraint. A similar approach is proposed in [17].

In comparison, the technique presented in this paper focuses on backwards checking of temporal constraints in execution logs. The generic Petri net pattern proposed in Sect. 3.4 is capable to express all temporal constraints that we encountered in the works discussed above, and other temporal constraints such as cyclic temporal constraints not discussed elsewhere; see [22] for a detailed discussion. Our technique detects *all* temporal violations in a trace, not just the first temporal violation encountered as it happens in model checking approaches. In case of violations also the compliant behavior (when a non-compliant event should have happened) is returned as diagnostic information.

## 6 Conclusion

In this paper, we provided an approach for temporal compliance checking of behavior recorded in execution logs. We developed a generic technique for formalizing all kinds of temporal compliance constraints, including all temporal compliance constraints documented in literature. In addition, we provide a general temporal compliance checking technique based on alignments. Our technique separates control-flow and temporal compliance checking to the possible extent, and provides integrated diagnostic information about both control-flow violations and temporal compliance violations. In particular, our technique is capable of finding all violations in a trace and highlights what the most likely compliant behavior should have been.

We provide a repository of compliance rules and an implementation of our compliance checker in the *Compliance* package of ProM. The software has been tested in a case study involving real-life logs from five Dutch municipalities. The results are encouraging: we were able to uncover various violations and no performance issues were encountered.

Future research aims at making the approach more user friendly. Eliciting and formalizing compliance rules, and mapping compliance diagnostics back to the original data is still a challenging step. Hence, higher-level compliance languages and more intuitive diagnostics are needed for end-users.

**Acknowledgements.** We thank Massimiliano de Leoni, Joos C.A.M.Buijs for their substantial support in writing this paper.

## References

1. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying history on process models for conformance checking and performance analysis. Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery 2(2), 182–192 (2012)
2. Abid, N., Dal Zilio, S., Le Botlan, D.: Real-time specification patterns and tools. In: Stoelinga, M., Pinger, R. (eds.) FMICS 2012. LNCS, vol. 7437, pp. 1–15. Springer, Heidelberg (2012)
3. Adriansyah, A., van Dongen, B., van der Aalst, W.M.: Conformance Checking Using Cost-Based Fitness Analysis. In: EDOC 2011, pp. 55–64 (2011)
4. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: ICSE, pp. 411–420 (1999)
5. Eder, J., Panagos, E., Rabinovich, M.: Time constraints in workflow systems. In: Jarke, M., Oberweis, A. (eds.) CAiSE 1999. LNCS, vol. 1626, pp. 286–300. Springer, Heidelberg (1999)
6. Ge, N., Pantel, M., Crégut, X.: Formal specification and verification of task time constraints for real-time systems. In: Margaria, T., Steffen, B. (eds.) ISoLA 2012, Part II. LNCS, vol. 7610, pp. 143–157. Springer, Heidelberg (2012)
7. Giblin, C., Liu, A.Y., Müller, S., Pfitzmann, B., Zhou, X.: Regulations expressed as logical models (realm). In: Frontiers in Artificial Intelligence and Applications, JURIX 2005, vol. 134, pp. 37–48. IOS Press (2005)
8. Kharbili, M.: Business process regulatory compliance management solution frameworks: A comparative evaluation. In: APCCM 2012. CRPIT, vol. 130, pp. 23–32. ACS (2012)
9. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Systems 2(4), 255–299 (1990)
10. Lanz, A., Weber, B., Reichert, M.: Time patterns in process-aware information system - a pattern based analysis - revised version. Tech. Rep. UIB-2009, University of Ulm, Germany (2009)
11. Lanz, A., Weber, B., Reichert, M.: Workflow time patterns for process-aware information systems. In: Bider, I., Halpin, T., Krogstie, J., Nurcan, S., Proper, E., Schmidt, R., Ukor, R. (eds.) BPMDS 2010 and EMMSAD 2010. LNBIP, vol. 50, pp. 94–107. Springer, Heidelberg (2010)
12. de Leoni, M., Maggi, F.M., van der Aalst, W.M.P.: Aligning Event Logs and Declarative Process Models for Conformance Checking. In: Barros, A., Gal, A., Kindler, E. (eds.) BPM 2012. LNCS, vol. 7481, pp. 82–97. Springer, Heidelberg (2012)
13. Leoni, M., Aalst, W.: Aligning event logs and process models for multi-perspective conformance checking: An approach based on integer linear programming. Tech. Rep. BPM Center Report BPM-13-05, BPMcenter.org (2013)
14. Li, H., Yang, Y.: Verification of temporal constraints for concurrent workflows. In: Yu, J.X., Lin, X., Lu, H., Zhang, Y. (eds.) APWeb 2004. LNCS, vol. 3007, pp. 804–813. Springer, Heidelberg (2004)
15. Li, H., Yang, Y.: Dynamic checking of temporal constraints for concurrent workflows. Electronic Commerce Research and Applications 4(2), 124–142 (2005)
16. Ly, L.T., Rinderle-Ma, S., Knuplesch, D., Dadam, P.: Monitoring business process compliance using compliance rule graphs. In: Meersman, R., Dillon, T., Herrero, P., Kumar, A., Reichert, M., Qing, L., Ooi, B.-C., Damiani, E., Schmidt, D.C., White, J., Hauswirth, M., Hitzler, P., Mohania, M. (eds.) OTM 2011, Part I. LNCS, vol. 7044, pp. 82–99. Springer, Heidelberg (2011)
17. Montali, M.: Specification and Verification of Declarative Open Interaction Models. LNBIP, vol. 56. Springer, Heidelberg (2010)

18. Niculae, C.C.: Time patterns in workflow management systems. Tech. Rep. BPM-11-04, BPM Center Report, BPMcenter.org (2011)
19. Pitzmann, B., Powers, C., Waidner, M.: Ibm's unified governance framework (ugf). Tech. rep., IBM Research Division, Zurich (2007)
20. Pozewaunig, H., Eder, J., Liebhart, W.: epert: Extending pert for workflow management system. In: ADBIS, pp. 217–224. Nevsky Dialect (1997)
21. Ramezani, E., Fahland, D., van der Aalst, W.M.P.: Where did I misbehave? diagnostic information in compliance checking. In: Barros, A., Gal, A., Kindler, E. (eds.) BPM 2012. LNCS, vol. 7481, pp. 262–278. Springer, Heidelberg (2012)
22. Ramezani, E., Fahland, D., van Dongen, B., van der Aalst, W.: Diagnostic information in temporal compliance checking. Tech. rep. BPM Center Report BPM-12-17, BPMcenter.org (2012)
23. Ramezani, E., Fahland, D., van der Werf, J.M., Mattheis, P.: Separating Compliance Management and Business Process Management. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM Workshops 2011, Part II. LNBIP, vol. 100, pp. 459–464. Springer, Heidelberg (2012)
24. Westergaard, M., Maggi, F.M.: Looking into the future: Using timed automata to provide a priori advice about timed declarative process models. In: CoopIS 2012. Springer (2012)