Conformance Checking Based on Multi-Perspective Declarative Process Models

A. Burattin[a], F. M. Maggi[b], A. Sperduti[a]

[a]*University of Padua, Italy*
[b]*University of Tartu, Estonia*

Abstract

Process mining is a family of techniques that aim at analyzing business process execution data recorded in event logs. Conformance checking is a branch of this discipline embracing approaches for verifying whether the behavior of a process, as recorded in a log, is in line with some expected behaviors provided in the form of a process model. The majority of these approaches require the input process model to be procedural (e.g., a Petri net). However, in turbulent environments, characterized by high variability, the process behavior is less stable and predictable. In these environments, procedural process models are less suitable to describe a business process. Declarative specifications, working in an open world assumption, allow the modeler to express several possible execution paths as a compact set of constraints. Any process execution that does not contradict these constraints is allowed. One of the open challenges in the context of conformance checking with declarative models is the capability of supporting multi-perspective specifications. In this paper, we close this gap by providing a framework for conformance checking based on MP-Declare, a multi-perspective version of the declarative process modeling language Declare. The approach has been implemented in the process mining tool ProM and has been experimented in three real life case studies.

Keywords: Process Mining, Conformance Checking, Linear Temporal Logic, Business Constraints, Declare

1. Introduction

The need to develop information systems able to fully support business processes of companies, and organizations in general, is becoming more and more urgent because of the fast pace of change in markets. Such dynamic markets impose frequent modifications and updates to business processes, leading to a constant decrease, in terms of temporal span, to the life-cycle of a business process definition. In this context, one very important functionality that any process-aware information system should be able to support is *conformance checking*, i.e., the ability to verify whether the actual flow of work is conformant with the intended business process model. This is especially true in the case

of very complex processes, where the adoption of an imperative formalism to represent it, such as Petri Nets [1] or BPM Notation [2], may lead to so much intricate workflows (so called "spaghetti"-like workflows) to become basically impossible to even properly visualize the process for human inspection.

Early works in conformance checking (e.g., [3, 4, 5]) mainly focused on the control-flow perspective in the context of imperative models, i.e., on the functional dependencies among performed activities/tasks in the process, while abstracting from time constraints, data dependencies, and resource assignments. These works were mainly based on replaying the log on the model to compute, according to the proposed approach, the fraction of events or traces in the log that can be replayed by the model. An evolution of these approaches is given by align-based approaches, where the conformance checking is performed by aligning both the modeled behavior and the behavior observed in the log (e.g. [6]). Only recently, approaches able to deal with multiple perspectives have been developed [7, 8], as well as approaches that aim at being computationally efficient via a problem decomposition strategy [9, 10, 11, 12].

In the case in which the process in consideration is complex, however, it is much better to use a declarative formalism, such as Declare [13, 14, 15], to represent a set of constraints that must be satisfied throughout the process execution. In this way, the "spaghetti"-like workflows are avoided, and the obtained model is flexible enough to allow all behaviors that do not violate the defined constraints. Conformance checking approaches based on the control-flow perspective have been defined for declarative models as well (e.g. [16, 17, 18]). More recently the additional data perspective has been considered in [19, 20], even if in these works the data perspective is not fully integrated with the control flow perspective. Efficient and fully integrated multi-perspective conformance checking proposals for declarative models, however, are still missing.

In this paper, we aim at closing this gap by proposing a multi-perspective approach based on Declare where it is possible to define multi-perspective constraints jointly considering data, temporal, and control flow perspectives. In order to allow that, we formally define Multi-Perspective Declare (MP-Declare), an augmented version of Declare where, thanks to the use of Metric First-Order Linear Temporal Logic, it is possible to define activation, correlation, and time conditions to build constraints over traces.

A nice feature of MP-Declare is that, by construction, it allows the user to efficiently perform conformance checking over event logs. In fact, we show that it is possible to define a conformance checking algorithmic framework operating on constraint templates, that is linear in the number of traces, constraints, and in the number of events of each trace. Conformance checking for a specific template is then obtained via definition of template-dependent procedures within the framework, whose time complexity depends on the actual template. Overall, however, the time complexity is upper bounded in the worst case by a quadratic function.

We assess the validity of the proposed approach both on artificial and real event logs. Controlled artificial data, involving logs containing up to 5 million events, are used to prove the scalability of the proposed approach, while real

event logs generated by three real business processes are used to demonstrate the expressivity and flexibility of constraints defined via MP-Declare.

2. Related Work

The scientific literature reports several works in the field of conformance checking [21]. Typically, the term *conformance checking* refers to the comparison of observed behaviors – as recorded in an event log – with respect to a process model. In the past, most of the conformance checking techniques were based on procedural models. State of the art examples of these approaches are reported in [7, 22, 11, 12].

In recent years, an increasing number of researchers are focusing on the conformance checking with respect to declarative models. For example, in [16], an approach for compliance checking with respect to *reactive business rules* is proposed. Rules, expressed using Condec [23], are mapped to Abductive Logic Programming, and Prolog is used to perform the validation. The approach has been extended in [17], by mapping constraints to LTL, and evaluating them using automata. The entire work has been contextualized into the service choreography scenario.

Runtime monitoring for compliance checking has been studied also based on MFOTL, as reported in [24, 25]. In these cases, the focus is on security policy monitoring. On the one side the authors try to enforce security policies, on the other they perform monitoring. In order to enforce security policies, it is necessary to distinguish between *controllable* and *observable* activities and, under specific circumstances, terminate the systems in order to prevent policy violations. Concerning the monitoring, authors identified fragments of the used logic, to describe security policies insensitive with respect to the ordering of actions with equal timestamps. The authors assume to perform monitoring in a distributed systems, which have synchronized clocks with limited precision.

Another application domain that researchers used to assess the applicability of conformance checking techniques is the medical domain. In particular, Grando et al. [26, 27] used Declare to model medical guidelines and to provide semantic (i.e., ontology-based) conformance checking measures. However, in this analysis neither data nor time perspectives are taken into account.

In [18], the authors report an approach that can be used to evaluate the conformance of a log with respect to a Declare model. In particular, their algorithms compute, for each trace, whether a Declare constraint is violated or fulfilled. Using these statistics the approach allows the user to evaluate the "healthiness" of the log. The approach is based on the conversion of Declare constraints into automata and, using a so-called "activation tree", it is able to identify violations and fulfillments. The approach described in this work does not take into account the data and time perspective, but only the control-flow is analyzed.

The work described in [28, 29] consists in converting a Declare model into an automaton and perform conformance checking of a log with respect to the

generated automaton. The conformance checking approach is based on the concept of "alignment" and as a result of the analysis each trace is converted into the most similar trace that the model accepts.

In a recent work, reported in [20], the data perspective for conformance checking with Declare is expressed in terms of conditions on global variables disconnected from the specific Declare constraints expressing the control flow. This work does not take the temporal perspective into account. In contrast, we provide a formal semantics in which the data perspective, the temporal perspective and the control flow are connected with each others.

3. Preliminaries

In this section, we present the fundamental concepts required to understand the rest of the paper.

3.1. Process Mining and XES

The basic idea behind process mining is to discover, monitor and improve processes by extracting knowledge from data that is available in today's systems [5]. The starting point for process mining is an event log. XES (eXtensible Event Stream) [30, 31] has been developed as the standard for storing, exchanging and analyzing event logs.

Each event in a log refers to an activity (i.e., a well-defined step in some process) and is related to a particular case (i.e., a process instance). The events belonging to a case are ordered with respect to their execution times. Hence, a case (i.e., a trace) can be viewed as a sequence of events. Event logs may store additional information about events 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. In XES, data elements can be event attributes, i.e., data produced by the activities of a business process and case attributes, namely data that are associated to a whole process instance. In this paper, we assume that all attributes are globally visible and can be accessed/manipulated by all activity instances executed inside the case.

3.2. Metric First Order Temporal Logic

In this paper, we use Metric First Order Temporal Logic (MFOTL) first introduced in [32]. MFOTL extends propositional metric temporal logic [33] to merge the expressivity of first-order logic together with the MTL temporal modalities. We deal with a fragment of MFOTL where all traces are finite.

In the following, we call "structure" a triple $D = (\Delta, \sigma, \iota)$. Δ is the domain of the structure, i.e., an arbitrary set. σ is the signature of the structure, i.e., a triple $\sigma = (C, R, a)$, where C is a set of constant symbols, R is a set of relational symbols, and a is a function that specify the arity of each relational symbol. ι is the interpretation function of the structure that assigns a meaning to all the symbols in σ over the domain Δ.

4

Definition 1 (Timed temporal structure). *A timed temporal structure over the signature $\sigma = (C, R, a)$ is a pair (D, τ) where D is a finite sequence of structures $D = (D_1, \ldots, D_n)$ and $\tau = (\tau_1, \ldots, \tau_n)$ is a finite sequence of timestamps with $\tau_i \in \mathbb{N}$.[1] D is assumed to have constant domains, i.e., $\Delta_i = \Delta_{i+1}$, for all $1 \le i < n$. Each constant symbol in C has an interpretation that does not vary over the time. The sequence of timestamps τ is monotonically increasing, i.e., $\tau_i \le \tau_{i+1}$, for all $1 \le i < n$.*

We indicate with $I = [a, b)$ an interval, where $a \in \mathbb{N}$ and $b \in \mathbb{N} \cup \{\infty\}$, and with V a set of variables. To express MFOTL formulas, we use the syntax:

Definition 2 (MFOTL Syntax). *Formulas of MFOTL over a signature $\sigma = (C, R, a)$ are given by the grammar*

$$\phi ::= t_1 \approx t_2 \mid r(t_1, \ldots, t_{a(r)}) \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \exists x.\phi \mid \mathbf{X}_I\phi \mid \phi_1\mathbf{U}_I\phi_2 \mid \mathbf{Y}_I\phi \mid \phi_1\mathbf{S}_I\phi_2$$

where $\phi, \phi_1, \phi_2 \in$ MFOTL, $I = [a, b)$ is an interval, r is an element of R, x ranges over V, and $t_1, t_2, \ldots$ belong to $V \cup C$.

A valuation is a mapping $v : V \to \Delta$. With abuse of notation, if c is a constant symbol in C, we say that $v(c) = c$. For a valuation v, a variable $x \in V$, and $d \in \Delta$, $v[x/d]$ is the valuation that maps x to d and leaves unaltered the valuation of the other variables.

Definition 3 (MFOTL Semantics). *Given (D, τ) a timed temporal structure over the signature $\sigma = (C, R, a)$ with $D = (D_1, \ldots, D_n)$, $\tau = (\tau_1, \ldots, \tau_n)$, ϕ a formula over S, v a valuation, and $1 \le i \le n$, we define $(D, \tau, v, i) \vDash \phi$ as follows:*

$$
\begin{aligned}
(D, \tau, v, i) &\vDash t \approx t' &\text{iff}\quad& v(t) = v(t') \\
(D, \tau, v, i) &\vDash r(t_1, \ldots, t_{a(r)}) &\text{iff}\quad& (v(t_1), \ldots, v(t_{a(r)}))) \in \iota(r) \\
(D, \tau, v, i) &\vDash (\neg\phi_1) &\text{iff}\quad& (D, \tau, v, i) \nvDash \phi_1 \\
(D, \tau, v, i) &\vDash \phi_1 \wedge \phi_2 &\text{iff}\quad& (D, \tau, v, i) \vDash \phi_1 \text{ and } (D, \tau, v, i) \vDash \phi_2 \\
(D, \tau, v, i) &\vDash \exists x.\phi_1 &\text{iff}\quad& (D, \tau, v[x/d], i) \vDash \phi_1, \text{ for some } d \in \Delta \\
(D, \tau, v, i) &\vDash \mathbf{Y}_I\phi_1 &\text{iff}\quad& i > 1, \tau_i - \tau_{i-1} \in I, \text{ and } (D, \tau, v, i-1) \vDash \phi_1 \\
(D, \tau, v, i) &\vDash \mathbf{X}_I\phi_1 &\text{iff}\quad& i < n, \tau_{i+1} - \tau_i \in I \text{ and } (D, \tau, v, i+1) \vDash \phi_1 \\
(D, \tau, v, i) &\vDash \phi_1\mathbf{S}_I\phi_2 &\text{iff}\quad& \text{for some } j \le i, \tau_i - \tau_j \in I, \\
& & & (D, \tau, v, j) \vDash \phi_2 \text{ and } (D, \tau, v, k) \vDash \phi_1 \\
& & & \text{for all } k \in [j+1, i+1) \\
(D, \tau, v, i) &\vDash \phi_1\mathbf{U}_I\phi_2 &\text{iff}\quad& \text{for some } j \ge i, \tau_j - \tau_i \in I, \\
& & & (D, \tau, v, j) \vDash \phi_2 \text{ and } (D, \tau, v, k) \vDash \phi_1 \\
& & & \text{for all } k \in [j, i)
\end{aligned}
$$

We add syntactic sugar for the normal connectives, such as $true \equiv \exists x.x \approx x$, $\phi_1 \vee \phi_2 \equiv \neg(\neg\phi_1 \wedge \neg\phi_2)$, $\forall x.\phi \equiv \neg\exists x.\neg\phi$ $\phi_1 \to \phi_2 \equiv (\neg\phi_1) \vee \phi_2$ and

---

[1] Note that every timestamp available in a XES log can be translated into an integer.

Table 1: Semantics for some Declare templates.

Template	LTL semantics	Activation
responded existence	$\mathbf{G}(A \rightarrow (\mathbf{O}B \vee \mathbf{F}B))$	A
response	$\mathbf{G}(A \rightarrow \mathbf{F}B)$	A
alternate response	$\mathbf{G}(A \rightarrow \mathbf{X}(\neg A \mathbf{U}B))$	A
chain response	$\mathbf{G}(A \rightarrow \mathbf{X}B)$	A
precedence	$\mathbf{G}(B \rightarrow \mathbf{O}A)$	B
alternate precedence	$\mathbf{G}(B \rightarrow \mathbf{Y}(\neg B \mathbf{S}A))$	B
chain precedence	$\mathbf{G}(B \rightarrow \mathbf{Y}A)$	B
not responded existence	$\mathbf{G}(A \rightarrow \neg(\mathbf{O}B \vee \mathbf{F}B))$	A
not response	$\mathbf{G}(A \rightarrow \neg\mathbf{F}B)$	A
not precedence	$\mathbf{G}(B \rightarrow \neg\mathbf{O}A)$	B
not chain response	$\mathbf{G}(A \rightarrow \neg\mathbf{X}B)$	A
not chain precedence	$\mathbf{G}(B \rightarrow \neg\mathbf{Y}A)$	B

$\phi_1 \leftrightarrow \phi_2 \equiv (\phi_1 \rightarrow \phi_2) \wedge (\phi_2 \rightarrow \phi_1)$. We also add temporal syntactic sugar, $\mathbf{F}_I\psi \equiv \text{true}\mathbf{U}_I\psi$ (timed future operator), $\mathbf{G}_I\psi \equiv \neg(\mathbf{F}_I(\neg\psi))$ (timed globally operator), $\mathbf{O}_I\psi \equiv \text{true}\mathbf{S}_I\psi$ (timed once operator) and $\mathbf{H}_I\psi \equiv \neg(\mathbf{O}_I(\neg\psi))$ (timed historically operator). The non-metric variants of the temporal operators are obtained by specifying $I = [0, \infty)$.

3.3. Declare

Declare is a declarative process modeling language originally introduced by Pesic and van der Aalst in [13, 14, 15]. Instead of explicitly specifying the flow of the interactions among process activities, Declare describes a set of constraints that must be satisfied throughout the process execution. The possible orderings of activities are implicitly specified by constraints and anything that does not violate them is possible during execution. In comparison with procedural approaches that produce "closed" models, i.e., all that is not explicitly specified is forbidden, Declare models are "open" and tend to offer more possibilities for the execution. In this way, Declare enjoys flexibility and is very suitable for highly dynamic processes characterized by high complexity and variability due to the turbulence and the changeability of their execution environments.

A Declare model consists of a set of constraints applied to activities. Constraints, in turn, are based on templates. Templates are patterns that define parameterized classes of properties, and constraints are their concrete instantiations (we indicate template parameters with capital letters and concrete activities in their instantiations with lower case letters). They have a graphical representation understandable to the user and their semantics can be formalized using different logics [34], the main one being LTL over finite traces, making them verifiable and executable. Each constraint inherits the graphical representation and semantics from its template. Table 1 summarizes some Declare

templates (the reader can refer to [13] for a full description of the language).

The *responded existence* template specifies that if A occurs, then B should also occur (either before or after A). The *response* template specifies that when A occurs, then B should eventually occur after A. The *precedence* template indicates that B should occur only if A has occurred before. Templates *alternate response* and *alternate precedence* strengthen the response and precedence templates respectively by specifying that activities must alternate without repetitions in between. Even stronger ordering relations are specified by templates *chain response* and *chain precedence*. These templates require that the occurrences of A and B are next to each other. Declare also includes some negative constraints to explicitly forbid the execution of activities. The *not responded existence* template indicates that if A occurs in a process instance, B cannot occur in the same instance. According to the *not response* template any occurrence of A cannot be eventually followed by B, whereas the *not precedence* template requires that any occurrence of B is not preceded by A. Finally, according to the *not chain response* and *not chain precedence*, A and B cannot occur one immediately after the other.

The major benefit of using templates is that analysts do not have to be aware of the underlying logic-based formalization to understand the models. They work with the graphical representation of templates, while the underlying formulas remain hidden. Declare is very suitable for specifying compliance models that are used to check if the behavior of a system complies with desired regulations. The compliance model defines the constraints related to a single process instance, and the overall expectation is that all instances comply with the model. Consider, for example, the *response* constraint $\mathbf{G}(a \rightarrow \mathbf{F}b)$. This constraint indicates that if a *occurs*, b must eventually *follow*. Therefore, this constraint is satisfied for traces such as $\mathbf{t}_1 = \langle a, a, b, c \rangle$, $\mathbf{t}_2 = \langle b, b, c, d \rangle$, and $\mathbf{t}_3 = \langle a, b, c, b \rangle$, but not for $\mathbf{t}_4 = \langle a, b, a, c \rangle$ because, in this case, the second instance of a is not followed by a b. Note that, in $\mathbf{t}_2$, the considered response constraint is satisfied in a trivial way because a never occurs. In this case, we say that the constraint is *vacuously satisfied* [35]. In [18], the authors introduce the notion of *behavioral vacuity detection* according to which a constraint is non-vacuously satisfied in a trace when it is activated in that trace. An *activation* of a constraint in a trace is an event whose occurrence imposes, because of that constraint, some obligations on other events (targets) in the same trace. For example, a is an activation for the *response* constraint $\mathbf{G}(a \rightarrow \mathbf{F}b)$ and b is a target, because the execution of a forces b to be executed, eventually. In Table 1, for each template the corresponding activation is specified.

An activation of a constraint can be a *fulfillment* or a *violation* for that constraint. When a trace is perfectly compliant with respect to a constraint, every activation of the constraint in the trace leads to a fulfillment. Consider, again, the response constraint $\mathbf{G}(a \rightarrow \mathbf{F}b)$. In trace $\mathbf{t}_1$, the constraint is activated and fulfilled twice, whereas, in trace $\mathbf{t}_3$, the same constraint is activated and fulfilled only once. On the other hand, when a trace is not compliant with respect to a constraint, an activation of the constraint in the trace can lead to a fulfillment but also to a violation (at least one activation leads to a violation). In trace $\mathbf{t}_4$,
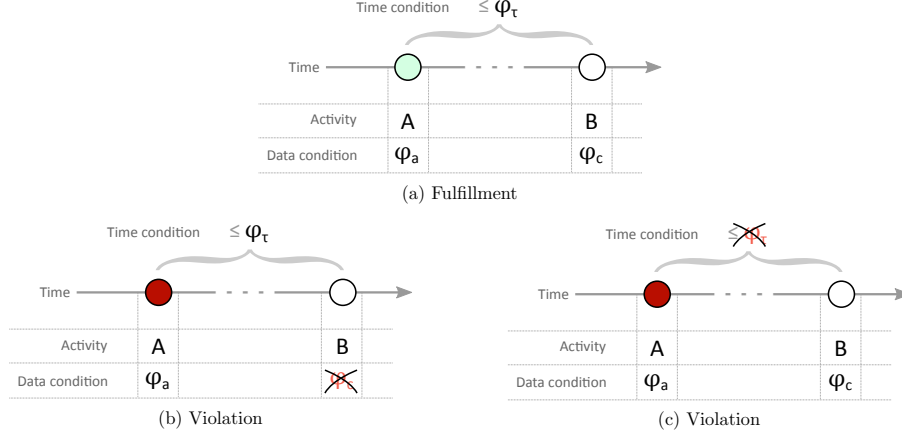
Figure 1: Fulfillment and violation scenarios for the *response* constraint between activities A and B. (a) reports a typical fulfillment scenario. In (b), the violation is due to the violation of the correlation condition φ_c. In (c), the violation is due to the violation of the time condition φ_τ.

for example, the response constraint $\mathbf{G}(a \to \mathbf{F}b)$ is activated twice, but the first activation leads to a fulfillment (eventually b occurs) and the second activation leads to a violation (b does not occur subsequently). An algorithm to discriminate between fulfillments and violations for a constraint in a trace is presented in [18]. Table 1 reports the activations for the main Declare templates.

In [18], the authors define two metrics to measure the conformance of an event log with respect to a constraint in terms of violations and fulfillments, called *violation ratio* and *fulfillment ratio* of the constraint in the log. These metrics are valued 0 if the log contains no activations of the considered constraint. Otherwise, they are evaluated as the percentage of violations and fulfillments of the constraint over the total number of activations.

Tools implementing process mining approaches based on Declare are presented in [36]. The tools are implemented as plug-ins of the process mining framework ProM.

4. MFOTL Semantics for Multi-Perspective Business Constraints

In this section, we introduce a multi-perspective version of Declare (MP-Declare). The version is similar to the ones in [37, 38], but we enrich it by allowing both time and data perspective. To do this, we use Metric First-Order Linear Temporal Logic (MFOTL). While many reasoning tasks are clearly undecidable for MFOTL, this logic is appropriate to unambiguously describe the semantics of the MP-Declare constraints we can use for conformance checking in our proposed algorithms.
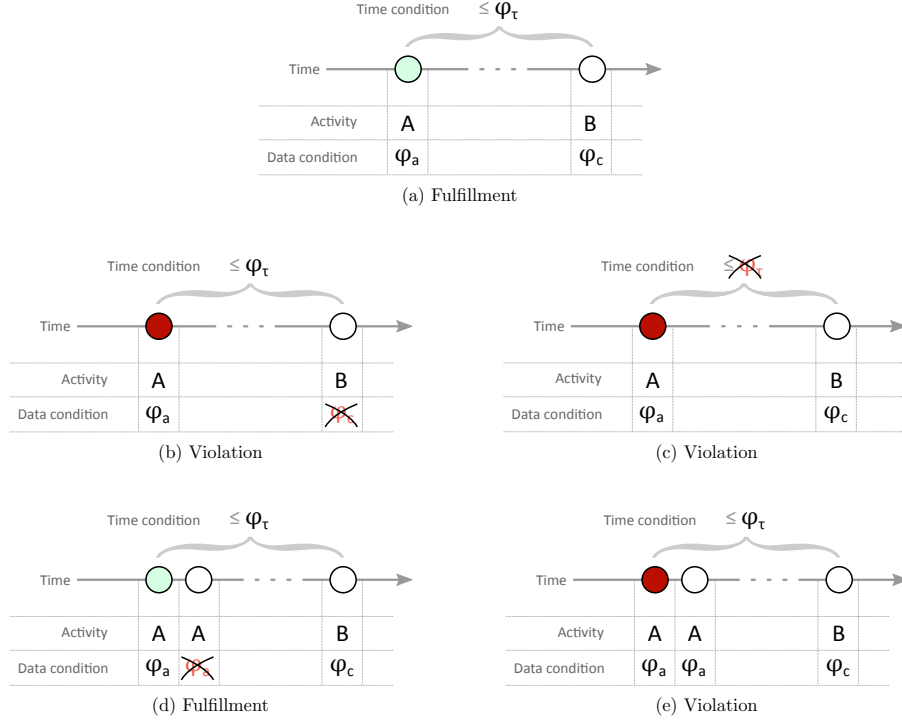
8

(a) Fulfillment

(b) Violation

(c) Violation

(d) Fulfillment

(e) Violation

Figure 2: Fulfillment and violation scenarios for the *alternate response* constraint between activities A and B. (a) reports a typical fulfillment scenario. In (b), the violation is due to the violation of the correlation condition φ_c. In (c), the violation is due to the violation of the time condition φ_τ. The activation in (d) is a fulfillment because the second occurrence of A does not satisfy the activation condition. In contrast, (e) reports a violation since, in this case, the second occurrence of A satisfies the activation condition.

To define the new semantics for Declare, we have to contextualize the definitions given in Section 3.2 in XES. Consider, for example, that the execution of an activity *pay* is recorded in an event log and, after the execution of *pay* at timestamp τ_i, the attributes *originator*, *amount*, and z have values *John*, 100, and *July*. In this case, the valuation of $(activityName, originator, amount, z)$ is $(pay, John, 100, July)$ in τ_i. Considering that in XES, by definition, the activity name is a special attribute always available, if $(pay, John, 100, July)$ is the valuation of $(activityName, originator, amount, z)$, we say that, when *pay* occurs, two special relations are valid $event(pay)$ and $p_{pay}(John, 100, July)$. In the following, we identify $event(pay)$ with the event itself *pay* and we call $(John, 100, July)$, the *payload* of *pay*.

The semantics for MP-Declare is shown in Table 2. Note that all the templates here considered have two parameters, an activation and a target (see also
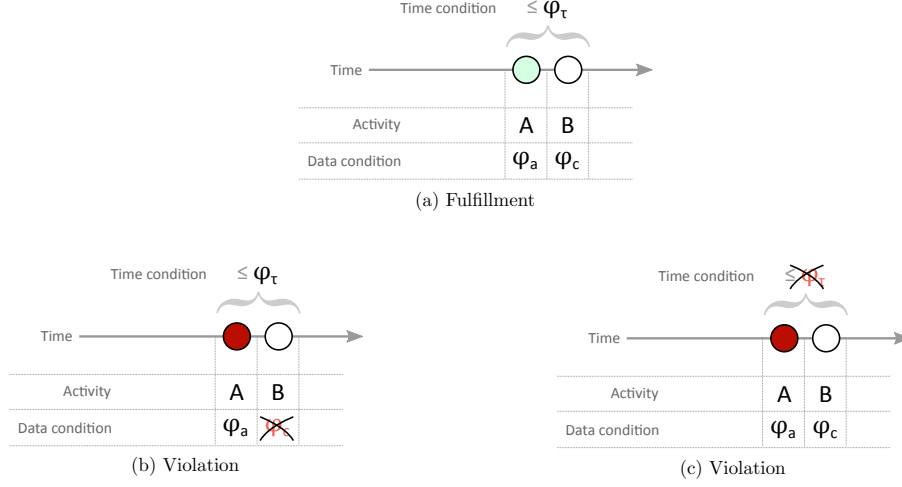
(a) Fulfillment

(b) Violation

(c) Violation

Figure 3: Fulfillment and violation scenarios for the *chain response* template between activities A and B. (a) reports a typical fulfillment scenario. Note that, in this case, the two events are contiguous. In (b), the violation is due to the violation of the correlation condition φ_C. In (c), the violation is due to the violation of the time condition φ_τ.

Table 1). As an example, we consider the response constraint "activity *pay* is always eventually followed by activity *get discount*" having *pay* as activation and *get discount* as target. The timed semantics of Declare, introduced in [37], is extended by requiring two additional conditions on data, i.e., the *activation condition* φ_a and the *correlation condition* φ_c. The activation condition is a relation (over the variables corresponding to the global attributes in the event log) that must be valid when the activation occurs. If the activation condition does not hold the constraint is not activated. In the case of the response template the activation condition has the form $p_A(x) \wedge r_a(x)$, meaning that when A occurs with payload x, the relation r_a over x must hold. For example, we can say that whenever *pay* occurs and *client type* is *gold* then eventually *get discount* must follow. In case *pay* occurs but *client type* is not *gold* the constraint is not activated. The correlation condition is a relation that must be valid when the target occurs. It has the form $p_B(y) \wedge r_c(x, y)$, where r_c is a relation involving, again, variables corresponding to the (global) attributes in the event log but, in this case, relating the valuation of the attributes corresponding to the payload of A and the valuation of the attributes corresponding to the payload of B. In our example, we can say that whenever *pay* occurs and *client type* is *gold* then eventually *get discount* must follow and the due amount corresponding to activity *get discount* must be lower than the one corresponding to activity *pay*. In the following, with abuse of notation we specify the interval characterizing the time perspective of a MP-Declare constraint ($I = [a, b)$) with φ_τ.

10

Table 2: Semantics for MP-Declare constraints.

Template	MFOTL Semantics
responded existence	$\mathbf{G}(\forall x.((A \wedge \varphi_a(x)) \rightarrow (\mathbf{O}_I(B \wedge \exists y.\varphi_c(x,y)) \vee \mathbf{F}_I(B \wedge \exists y.\varphi_c(x,y)))))$
response	$\mathbf{G}(\forall x.((A \wedge \varphi_a(x)) \rightarrow \mathbf{F}_I(B \wedge \exists y.\varphi_c(x,y))))$
alternate response	$\mathbf{G}(\forall x.((A \wedge \varphi_a(x)) \rightarrow \mathbf{X}(\neg(A \wedge \varphi_a(x))\mathbf{U}_I(B \wedge \exists y.\varphi_c(x,y)))))$
chain response	$\mathbf{G}(\forall x.((A \wedge \varphi_a(x)) \rightarrow \mathbf{X}_I(B \wedge \exists y.\varphi_c(x,y)))$
precedence	$\mathbf{G}(\forall x.((B \wedge \varphi_a(x)) \rightarrow \mathbf{O}_I(A \wedge \exists y.\varphi_c(x,y)))$
alternate precedence	$\mathbf{G}(\forall x.((B \wedge \varphi_a(x)) \rightarrow \mathbf{Y}(\neg(B \wedge \varphi_a(x))\mathbf{S}_I(A \wedge \exists y.\varphi_c(x,y))))$
chain precedence	$\mathbf{G}(\forall x.((B \wedge \varphi_a(x)) \rightarrow \mathbf{Y}_I(A \wedge \exists y.\varphi_c(x,y)))$
not responded existence	$\mathbf{G}(\forall x.((A \wedge \varphi_a(x)) \rightarrow \neg(\mathbf{O}_I(B \wedge \exists y.\varphi_c(x,y)) \vee \mathbf{F}_I(B \wedge \exists y.\varphi_c(x,y)))))$
not response	$\mathbf{G}(\forall x.((A \wedge \varphi_a(x)) \rightarrow \neg\mathbf{F}_I(B \wedge \exists y.\varphi_c(x,y))))$
not precedence	$\mathbf{G}(\forall x.((B \wedge \varphi_a(x)) \rightarrow \neg\mathbf{O}_I(A \wedge \exists y.\varphi_c(x,y)))$
not chain response	$\mathbf{G}(\forall x.((A \wedge \varphi_a(x)) \rightarrow \neg\mathbf{X}_I(B \wedge \exists y.\varphi_c(x,y)))$
not chain precedence	$\mathbf{G}(\forall x.((B \wedge \varphi_a(x)) \rightarrow \neg\mathbf{Y}_I(A \wedge \exists y.\varphi_c(x,y)))$

Graphical representations of three MP-Declare templates are reported in Figures 1, 2 and 3. In particular, these figures report the semantics for response, alternate response and chain response constraints. Each figure shows possible scenarios of violations and fulfillments for the corresponding constraint. A scenario is described reporting events as rounded circles. Each circle is associated to an activity (A, B, or C) and a data condition (either an activation condition φ_a or a correlation condition φ_c). The time condition φ_τ is reported above the horizontal curly bracket. Crossed data or time conditions indicate violated conditions. Red circles indicate events that are violations, green circles indicate fulfillments.

The *response* constraint in Figure 1 indicates that, if A occurs at time τ_A with φ_a holding true, B must occur at some point $\tau_B \in [\tau_A + a, \tau_A + b)$ with φ_c holding true. The *alternate response* constraint in Figure 2 specifies that, if A occurs at time τ_A with φ_a holding true, B must occur at some point $\tau_B \in [\tau_A + a, \tau_A + b)$ with φ_c holding true. A is not allowed in the interval $[\tau_A, \tau_B]$ if φ_a is true. Any event different from A is allowed and, also, A is allowed if φ_a is false. The *chain response* constraint in Figure 3 indicates that, if A occurs at time τ_A with φ_a holding true, B must occur next at some point $\tau_B \in [\tau_A + a, \tau_A + b)$ with φ_c holding true.

5. Conformance Checking Algorithms

As stated in the previous section, with MP-Declare, it is possible to express Declare constraints taking into account also the temporal and the data perspectives. As an example, it is possible to express constraints like:

- *activity A must occur between 10 and 11 hours before activity B*;

- *if activity A writes a variable x with value <1000, then B must occur after two days.*

Therefore, using this language, it is possible to define multi-perspective compliance models that can be used for several purposes like, for example, for representing Service Level Agreements (SLAs). In this context, it would be useful to provide the user with techniques to detect whether cases are actually fulfilling the required set of constraints or not. In this section, we present algorithms to check the conformance of an event log with respect to a MP-Declare model.

The proposed approach for the conformance checking of MP-Declare constraints is based on several procedures. The main component is described in the `CheckLogConformance` procedure, reported in Algorithm 1. This algorithm requires as input a log and a MP-Declare model (i.e., a set of MP-Declare constraints). Then, it iterates through all traces and, for each constraint, it computes the violations and the fulfillments by calling the `CheckTraceConformance` procedure. `CheckTraceConformance`, described in Algorithm 2, takes as input a trace and a constraint and generates the set of violating and fulfilling events for that specific constraint in that specific trace. The basic idea of this procedure is to iterate through all the events of the trace and, for each of them, call specific template-dependent operations (lines 5-11).

Algorithm 1: `CheckLogConformance`

Input: *Log*: an event log
  Model: a model
Output: A set of violating and fulfilling traces/constraints

1 Let *fulfill* and *viol* be maps that, given a trace and a constraint, return the set of fulfilling and violating events

2 **foreach** *trace* ∈ *Log* **do**
3   **foreach** *constr* ∈ *Model* **do**
4   *viol, fulfill* ← `CheckTraceConformance`(*trace, constr*)
        `// Algorithm 2`
5   *viol* [*trace*][*constr*] ← *viol*
6   *fulfill* [*trace*][*constr*] ← *fulfill*
7   **end**
8 **end**
9 **return** *viol, fulfill*

The described algorithms might be seen as a general "framework" that can be used for conformance checking with respect to different templates. Each template that needs to be verified must properly define the following required operations:

- *opening*: this procedure is called once per trace, before starting the analysis of the first event of the trace;

- *fulfillments*: this procedure is called for each event of the trace and is supposed to return the set of fulfillments that have been observed so far; modifications to the set of activations are allowed as well;

Algorithm 2: `CheckTraceConformance`

Input: *trace*: a trace

        $c = \langle templ, A, T, \varphi_a, \varphi_c, \varphi_\tau \rangle$: a constraint

Output: Set of violating and fulfilling events

---

1 $pending \leftarrow \emptyset$

2 $fulfillments \leftarrow \emptyset$

3 $violations \leftarrow \emptyset$

    `/* All the following calls are allowed to make side effects`
       `on the provided parameters`   `*/`

4 $templ.opening()$ `/* Opening template operations */`

5   **foreach** $e \in trace$ **do**

6 $templ.fulfillment(e, trace, pending, fulfillments, T, \varphi_a, \varphi_c, \varphi_\tau)$

7 $templ.violation(e, trace, pending, violations, T, \varphi_c, \varphi_\tau)$

8 $templ.activation(e, A, pending, \varphi_a)$

9   **end**

10 $templ.closing(pending, fulfillments, violations)$ `/* Closing template`
    `operations */`

11   **return** $violation, fulfillments$

- *violations*: this procedure is called for each event of the trace and is supposed to return the set of violations that have been observed so far; modifications to the set of activations are allowed as well;

- *activation*: this procedure is called for each event of the trace and is supposed to update the set of activations that have been observed so far (i.e., whether the current event is a new activation or not);

- *closing*: this procedure is called once per trace, after all the events have been analyzed.

In this paper, we illustrate the procedures for three templates, i.e., *response*, *alternate response*, and *chain response*. We consider these three specifications sufficiently representative in order to provide a clear idea of the capabilities of our framework.[2] In each procedure, given the set of all possible activities $\mathcal{A}$, we define a constraint as a tuple: $c = \langle template, A, T, \varphi_a, \varphi_c, \varphi_\tau \rangle$, where *template* indicates which template the constraint is referring to, *template* $\in$ {*existence, absence, choice, responded existence, . . .*}; $A \subseteq \mathcal{A}$ is the nonempty set of activations; $T \subseteq \mathcal{A}$ is the nonempty set of targets; φ_a and φ_c indicate, respectively, the activation and the correlation condition; and φ_τ represents the time condition. We also use the functions $verify(\varphi_a, A)$, $verify(\varphi_c, A, B)$, and $verify(\varphi_\tau, A, B)$. The first function evaluates φ_a with respect to the attributes

---

[2]All the procedures for conformance checking based on MP-Declare have been implemented and are publicly available (see Section 6).

reported in A. The second function evaluates φ_c with respect to the attributes defined in A and B. The third function compares the timestamps attached to A and B in order to see whether φ_τ is satisfied or not. As already mentioned, each event recorded in an event log brings a payload of attributes. In the description of the algorithms, we use the $\pi_a(e)$ operator to get the value of an attribute a of an event e. For example, we use $\pi_{activity}(e)$ to select the activity name associated to e.

The first template we consider is *response* and the corresponding procedures are reported in Table 3. The *opening* procedure does nothing. The *fulfillment* procedure checks whether the input event refers to a target. If this is the case, then all pending activations that can be correlated to this target (in case the time and the correlation conditions are satisfied) become fulfillments. The *activation* procedure checks whether the input event refers to an activation of the constraint and the activation condition φ_a is satisfied (in this case the event has to be added to the set of pending activations). Violations are identified in the *closing* procedure (the *violation* procedure is not used in this case). Here, all pending activations that do not have a corresponding target when the entire trace has been processed become violations.

Response

---

template.opening()

    1 do nothing

---

template.fulfillment$(e, trace, pending, fulfillments, T, \varphi_a, \varphi_c, \varphi_\tau)$

    1 if $\pi_{activity}(e) \in T$ **then**
    2   **foreach** $act \in pending$ **do**
    3   **if** $verify(\varphi_c, act, e)$ **and** $verify(\varphi_\tau, act, e)$ **then**
    4 $pending \leftarrow pending \setminus \{act\}$
    5 $fulfillments \leftarrow fulfillments \cup \{act\}$
    6   **end**
    7   **end**
    8 end

---

template.violation$(e, trace, pending, violations, T, \varphi_c, \varphi_\tau)$

    1 do nothing   `/* Actual violations are not identified here */`

---

template.activation$(e, A, pending, \varphi_a)$

    1 if $\pi_{activity}(e) \in A$ **and** $verify(\varphi_a, e)$ **then**
    2 $pending \leftarrow pending \cup \{e\}$
    3 end

---

template.closing$(pending, fulfillments, violations)$

    1 foreach $act \in pending$ **do**
    2 $pending \leftarrow pending \setminus \{act\}$
    3 $violations \leftarrow violations \cup \{act\}$
    4 end

Table 3: Procedure specifications for the *response* constraint.

The procedures for the *alternate response* template are reported in Table 4. In particular, *opening* defines a new data structure (*possibleTargets*) that will be used by the other procedures. The *fulfillment* procedure starts by checking whether the input event refers to an activation and the activation condition is satisfied. If this is the case, the procedure checks whether there is exactly one pending activation and at least one possible target. If this is the case, if for at least one possible target the time and the correlation conditions are satisfied, the pending activation becomes a fulfillment (*fulfillment*, lines 6-8). If the activity referring to the input event is a target, the event is added to the set of possible targets (*fulfillment*, line 14). The *violation* procedure also starts by checking whether the input event refers to an activation and the activation condition is satisfied. If this is the case, the procedure checks whether there is exactly one pending activation. If this is the case, the pending activation becomes a violation (the pending activation cannot be a fulfillment because, in this case, the invocation of the *fulfillment* procedure moves it from the pending set to the fulfillment set). The *activation* procedure checks whether the input event refers to an activation and the activation condition is satisfied. In this case, the set of possible targets is reset to the empty value and the event is returned to be added to the set of pending activations. The *closing* procedure verifies that if there is a pending activation, this activation can be correlated at least to one possible target. If this is the case (if the time and the correlation conditions are satisfied), then the activation becomes a fulfillment (*closing*, line 7), otherwise it is marked as a violation (*closing*, line 11).

Alternate Response

---

template.opening()

   1 **define** *possibleTargets* $\leftarrow \emptyset$ as a data structure available throughout the entire `CheckTraceConformance` algorithm

---

template.fulfillment(e, *trace*, *pending*, *fulfillments*, T, φ_a, φ_c, φ_τ)

   1 **if** $\pi_{activity}(e) \in A$ **and** *verify*(φ_a, e) **then**
   2   **if** $|possibleTargets| \geq 1$ **and** $|pending| = 1$ **then**
   3 $act \leftarrow element \in pending$ **// There is only one element**
   4   **foreach** $p \in possibleTargets$ **do**
   5   **if** *verify*(φ_c, act, p) **and** *verify*(φ_τ, act, p) **then**
   6   *fulfillments* $\leftarrow$ *fulfillments* $\cup \{act\}$
   7   *pending* $\leftarrow$ *pending* $\setminus \{act\}$
   8   **break** **// It is possible to exit the loop**
   9   **end**
  10   **end**
  11   **end**
  12 **end**
  13 **if** $e \in T$ **then**
  14   *possibleTargets* $\leftarrow$ *possibleTargets* $\cup \{e\}$
  15 **end**

Alternate Response *(continued from previous page)*

template.violation$(e, trace, pending, violations, T, \varphi_c, \varphi_\tau)$

  1 **if** $\pi_{activity}(e) \in A$ **and** *verify*(φ_a, e) **then**
  2   **if** $|pending| = 1$ **then**
  3 $act \leftarrow element \in pending$ // There is only one element
  4 $pending \leftarrow pending \setminus \{act\}$
  5 $violations \leftarrow violations \cup \{act\}$
  6   **end**
  7 **end**

template.activation$(e, A, pending, \varphi_a)$

  1 **if** $\pi_{activity}(e) \in A$ **and** *verify*(φ_a, e) **then**
  2 $possibleTargets \leftarrow \emptyset$
  3 $pending \leftarrow pending \cup \{e\}$
  4 **end**

template.closing$(pending, fulfillments, violations)$

  1 **if** $|pending| = 1$ **then**
  2 $targetFound \leftarrow$ false
  3 $act \leftarrow element \in pending$ // There is only one element
  4   **foreach** $p \in possibleTargets$ **do**
  5   **if** *verify*(φ_c, act, p) **and** *verify*(φ_τ, act, p) **then**
  6 $targetFound \leftarrow$ true
  7 $fulfillments \leftarrow fulfillments \cup \{act\}$
  8   **end**
  9   **end**
 10   **if not** $targetFound$ **then**
 11 $violations \leftarrow violations \cup \{act\}$
 12   **end**
 13 **end**

Table 4: Procedure specifications for the *alternate response* constraint.

The procedures for the *chain response* template are reported in Table 5. As for the response template, *opening* does nothing. The *fulfillment* and the *violation* procedures verify whether there is exactly one element in the set of pending activations. In this case, they check whether the input event refers to a target and the time and correlation conditions are fulfilled. If this is the case, the pending activation becomes a fulfillment, otherwise it is marked as a violation. The *activation* procedure checks whether the input event refers to an activation and the activation condition is satisfied (in this case the event has to be added to the set of pending activations). The *closing* procedure checks whether there is still a pending activation when the entire trace has been processed. In this case, the pending activation becomes a violation.

Chain Response

---

template.opening()

  1 do nothing

---

template.fulfillment(e, *trace*, *pending*, *fulfillments*, T, φ_a, φ_c, φ_τ)

  1 if $|pending| = 1$ **then**
  2 $act \leftarrow element \in pending$ **// There is only one element**
  3   **if** $\pi_{activity}(e) \in T$ **and** $verify(\varphi_c, act, e)$ **and** $verify(\varphi_\tau, act, e)$ **then**
  4 $pending \leftarrow pending \setminus \{act\}$
  5 $fulfillments \leftarrow fulfillments \cup \{act\}$
  6   **end**
  7 end

---

template.violation(e, *trace*, *pending*, *violations*, T, φ_c, φ_τ)

  1 if $|pending| = 1$ **then**
  2 $act \leftarrow element \in pending$ **// There is only one element**
  3   **if** $\pi_{activity}(e) \notin T$ **or not** $verify(\varphi_c, act, e)$ **or not** $verify(\varphi_\tau, act, e)$ **then**
  4 $pending \leftarrow pending \setminus \{act\}$
  5 $violations \leftarrow violations \cup \{act\}$
  6   **end**
  7 end

---

template.activation(e, A, *pending*, φ_a)

  1 if $\pi_{activity}(e) \in A$ **and** $verify(\varphi_a, e)$ **then**
  2 $pending \leftarrow pending \cup \{e\}$
  3 end

---

template.closing(*pending*, *fulfillments*, *violations*)

  1 foreach $act \in pending$ **do**
  2 $pending \leftarrow pending \setminus \{act\}$
  3 $violations \leftarrow violations \cup \{act\}$
  4 end

Table 5: Procedure specifications for the *chain response* constraint.

The algorithms for the other templates specified in Table 2 can be very easily derived from the ones described in this section. In particular, the algorithms for the *precedence*, the *alternate precedence* and the *chain precedence* are the same as the ones described for *response*, *alternate response* and *chain response* respectively. The only difference is that, for the precedence templates, the traces in the input log have to be parsed from the end to the beginning. Similarly, the algorithms for checking the negative templates are the same as the ones described for the corresponding negative templates. In this case, every fulfillment for a positive template becomes a violation for the corresponding negative template and vice versa.

From the computational complexity point of view, it is worthwhile noting that the complexity of Algorithm 1 and Algorithm 2 is linear in the number of traces, constraints, and in the number of events of each trace. The complexity of the template-dependent procedures, instead, depends on the actual template. Specifically, with respect to the procedures of each constraint reported in this paper, we have the following complexities:

- Response: *opening*, *violation*, and *activation* are constant; *fulfillment* and *closing* have linear complexity on the number of pending activations for the current trace (which is at most the number of events on the trace);

- Alternate Response: *opening*, *violation*, and *activation* are constant; *fulfillment* and *closing* are linear on the number of possible targets (which is at most the number of events on the trace);

- Chain Response: *opening*, *fulfillment*, *violation* , *activation* are constant; *closing* is linear on the number of pending activations for the current trace (which is at most the number of events on the trace).

6. Implementation and Benchmarks

This section provides some details on the implementation of the approach and a benchmark analysis on different scenarios.

6.1. Implementation Details

The entire approach has been implemented as a plug-in of the process mining toolkit ProM.[3] In particular, the plug-in receives as input an event log and a model and evaluates the conformance of the log with respect to the model. It is interesting to note that, in the current implementation, the processing of each trace is independent from all the others. Also, the analysis of a constraint in the reference model is independent from all the others. For this reason, it is possible to parallelize and distribute the analysis over different computational nodes and drastically improve the performances. The results of the tests reported in this paper, however, do not benefit from such a possibility and our tests sequentially evaluate each constraint on each trace.

The conformance checking results are presented using a ProM plug-in called "Analysis Result Visualizer". This visualizer is composed of three main windows. The first window consists of a summary of the statistics computed for each constraints (e.g., number of activations, number of violations and number of fulfillments) on the entire log. This window is shown in Figure 4.

The second window (shown in Figure 5a) provides a more detailed view. This window is divided into three columns. The leftmost column contains a list of all the cases with information on case id, number of activations in the case, and number of fulfillments and violations. The central column contains the list

---

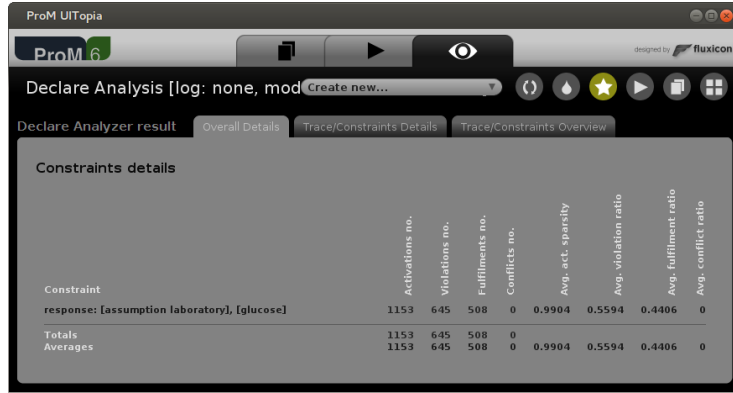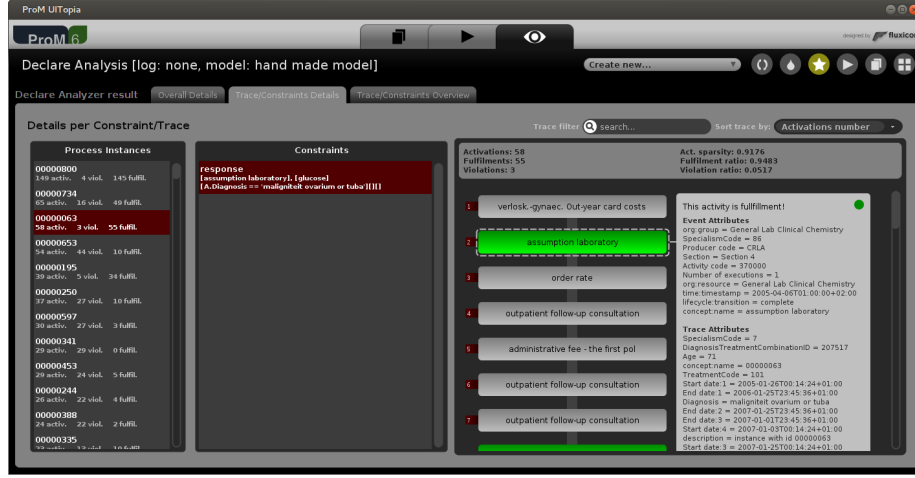[3]The software can be downloaded from `http://www.promtools.org/prom6`.

Figure 4: Overall details window with the result summary.

of constraints in the reference model. When a case and a constraint are selected, in the list in the rightmost column of the window, a representation of the case appears. In this representation, each event is drawn as a rectangle. Green-painted rectangles represent fulfillments, red-painted boxes represent violations. It is possible to move the mouse cursor over each rectangle to see the complete set of attributes belonging to the event.

The third window (shown in Figure 5b) also lists all cases. Here, each event of a case is represented as a small box that can be gray, green (in case the event is a fulfillment), or red (in case the event is a violation). This visualization is also called "birdview" since it provides a high-level overview of the constraints and allows the user to quickly identify possible issues. When the mouse is moved over an event, a pop-up showing the corresponding activity name appears. In both the second and the third window, it is possible to sort cases based on different parameters (name of the case, number of activations, number of violations, and number of fulfillments), or interactively search for cases with a specific case id.

6.2. Benchmarks

In order to gain some insights on the computational feasibility of our implementation, we run several tests in different possible scenarios. In particular, we tested our implementation against logs with different sizes and different trace lengths. We generated traces with 10, 20, 30, 40, and 50 events and, for each of these lengths, we generated logs with 25 000, 50 000, 75 000, and 100 000 traces. Therefore, in total, we used 20 logs. The number of events contained in each log is reported in Table 6. In addition, we designed 10 Declare models. In particular, we prepared two models with 10 constraints, one only containing constraints on the control-flow (without conditions on data and time), and another one including real multi-perspective constraints (with conditions on time and data). We followed the same procedure to create models with 20, 30, 40, and 50 constraints.

19

(a) Window with the conformance checking details for a single case and constraint.

(b) Birdview-like window showing an overview of fulfillments and violations for some cases.

Figure 5: Windows used to inspect the conformance checking results by focusing on single cases.

		Number of log traces			
		25 000	50 000	75 000	100 000
Events per trace	10	250 000	500 000	750 000	1 000 000
	20	500 000	100 0000	1 500 000	2 000 000
	30	750 000	150 0000	2 250 000	3 000 000
	40	100 0000	200 0000	3 000 000	4 000 000
	50	125 0000	250 0000	3 750 000	5 000 000

Table 6: Number of events for each log.

We checked each log against each model, and we repeated the procedure five times, in order to get the average execution times for each configuration. To provide more accurate results, the times reported here are measured without considering the time needed to generate the graphical visualization (we perform the tests on a custom command-line version of ProM). All tests have been performed using two machines (part of a cluster) randomly, with the following hardware configurations: *(i)* 4 x Eight-Core Intel(R) Xeon(R) CPU E5-4640 0 @ 2.40GHz; *(ii)* 2 x Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz.

Figure 6 provides a graphical representation of the average execution times for the analysis of all models and logs. In particular, the graph on top reports the execution times using models with control-flow based constraints. The graph at the bottom reports the execution times using real multi-perspective models (with conditions on time and data). In Figure 7 and in Figure 8, we also report the average execution times (and standard deviations) required to analyze all models and logs but we provide different views on the data. In particular, in Figure 7, the execution times are grouped based on the number of traces in the logs. The graph on the left-hand side reports the execution times using models with control-flow based constraints, the one on the right-hand side reports the execution times using multi-perspective constraints. In Figure 8, the execution times are grouped based on the number of events in each trace.

As the statistics clearly show, the time required to perform the analysis directly depends both on the number of events in each trace, and on the actual size of the log. However, the execution times evaluated using models with control-flow based constraints seem to be more influenced by the number of events in each traces. We believe that this is due to the additional costs needed for starting up the data validation engine in case of multi-perspective models. In particular, it is necessary to restart such engine for each trace and the additional time required is so high that it becomes impossible to see the differences in terms of performances for traces of different lengths. In general, it is worthwhile noting that the most expensive configuration (a model with 50 multi-perspective constraints, and a log with 100 000 traces and 5 000 000 events) requires, on average, 255 369 milliseconds, i.e., about 4.2 minutes. This proves the scalability of our approach.

7. Case Studies

This section provides three case studies on real datasets. The first one is based on an event log provided by an academic hospital, the second one is a case study provided by a financial institution and the third one is based on a dataset provided by a bank.

7.1. A Large Academic Hospital

We have conducted a case study by using the BPI challenge 2011 event log [39]. This log pertains to a healthcare process and, in particular, contains the executions of a process related to the treatment of patients diagnosed with cancer in a large Dutch academic hospital. The whole event log contains 1 143 cases
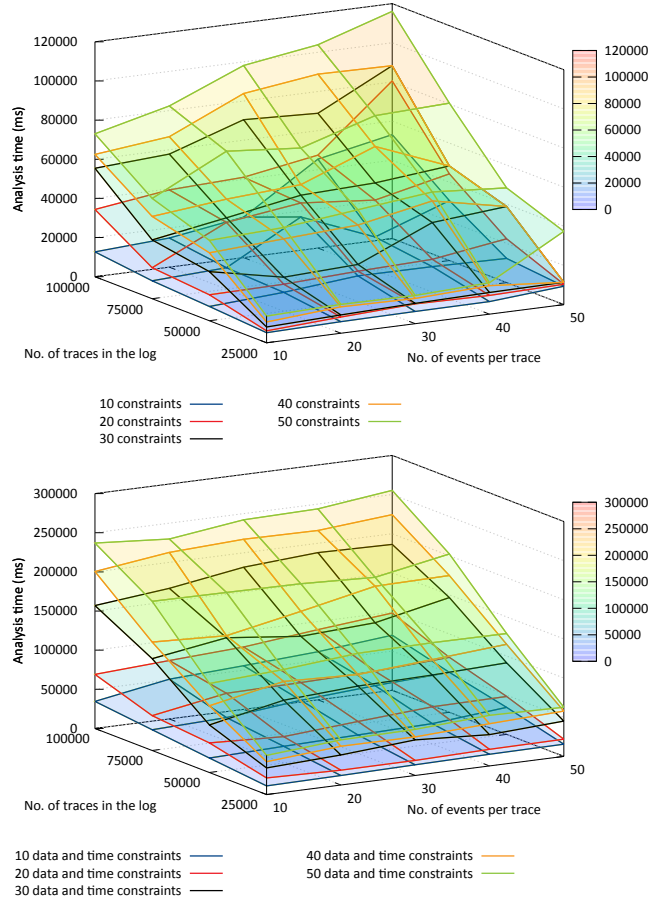
Figure 6: Execution times in milliseconds required to process logs with different number of traces of different lengths. The plot on top refers to models with control-flow constraints. The plot at the bottom refers to models with control-flow, data and time constraints.

Table 7: Reference constraints used to analyze the log from the BPI challenge 2011.

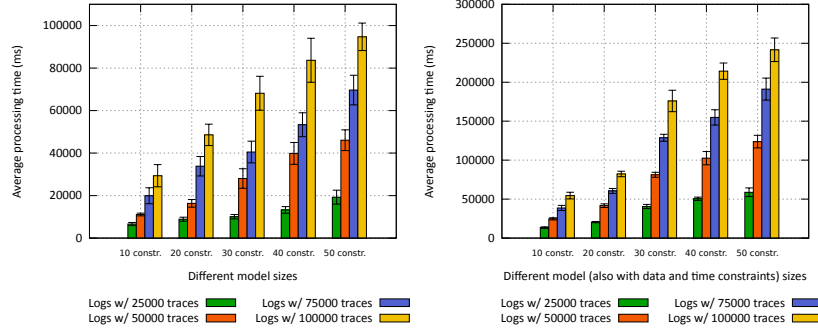| Id	Constraint	1st param.	2nd param.	Activation condition	Correlation condition	Time condition
1	Precedence	ca-125 using meia	outpatient follow-up consultation	A.Diagnosis == 'maligniteit ovarium or tuba'	–	0,15,d
2	Precedence	First outpatient consultation	telephone consultation	–	A.org:group == T.org:group	–

Figure 7: Execution times in milliseconds grouped based on the number of traces in the logs. The plot on the left hand side refers to models with control-flow constraints. The plot on the right hand side refers to models with control-flow, data and time constraints.
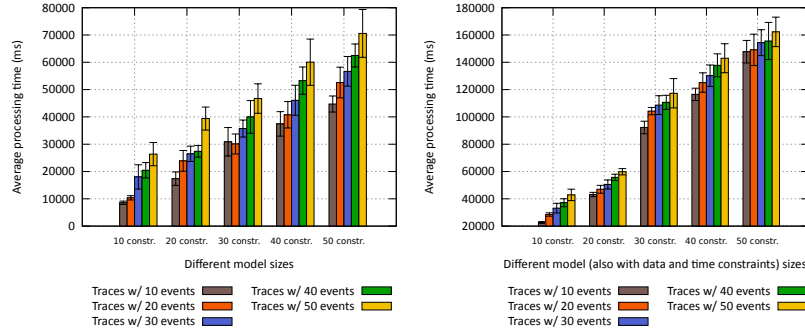

Figure 8: Execution times in milliseconds grouped based on the number of events in each trace. The plot on the left hand side refers to models with control-flow constraints. The plot on the right hand side refers to models with control-flow, data and time constraints.

Table 8: Conformance checking results using the log from the BPI challenge 2011.

Id	Act.no.	Viol.no.	Fulfill.no.	Avg.act.sparsity	Avg.viol.ratio	Avg.fulfill.ratio
1	343	242	101	0.9844	0.7055	0.2945
2	1 286	546	740	0.9677	0.4246	0.5754

Table 9: Execution times using the log from the BPI challenge 2011.

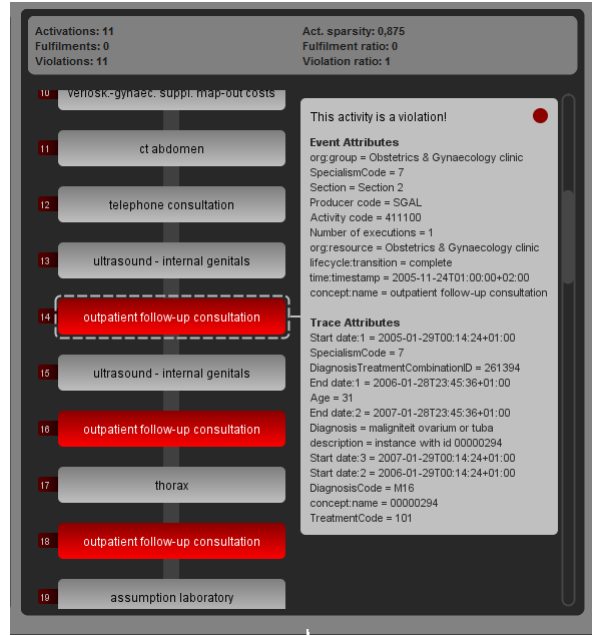| Id	Avg.execution time (milliseconds)
1	1 759
2	1 828

Figure 9: Example of violations for constraint 1.

and 150 291 events distributed across 623 event classes (i.e., each event refers to one of 623 different possible activities). Each case describes the treatment of a different patient. The event log contains domain specific attributes that are both case attributes and event attributes in addition to the standard XES attributes. For example, `Age`, `Diagnosis`, and `Treatment code` are case attributes and `Activity code`, `Number of executions`, `Specialism code`, and `Group` are event attributes. As mentioned in Section 3.1, in our analysis all the attributes are considered visible for all the activities and we suppose that an activity overwrites the old values of all the event attributes attached to it.

To investigate the behavior of the process as recorded in the log, we have used the constraints shown in Table 7. The idea behind constraint 1 is that the tumor marker "ca-125" is used in the follow-up of patients diagnosed with ovarian cancer as an indicator of the evolution of the tumor. For this reason, we would expect that, if the diagnosis for a patient is "maligniteit ovarium", the follow-up consultation is preceded by the analysis of this tumor marker. In addition, we require a time condition indicating that this analysis should not come too early with respect to the follow-up. As shown in Table 8, constraint 1 has 343 activations. This means that there are 343 occurrences of `outpatient follow-up consultation` associated with a `Diagnosis` equal to `maligniteit ovarium or tuba`. As shown in Table 8, around 70% of these activations are violations. One of the reasons why there are so many violations in the log for this constraint is that there can be several follow-ups in a case and some

24

of them are not correlated with the "ca-125" test but with other tests. In Figure 9, it is possible to see some violations for constraint 1. For example, the selected event `outpatient follow-up consultation` is an activation for the constraint since, in its payload, the value for `Diagnosis` is `maligniteit ovarium or tuba`. However, this activation is probably connected with the computed tomography abdomen and/or the ultrasound test done immediately before.

The idea behind constraint 2 is that the first consultation for a patient in the hospital cannot be a telephone consultation. We also add a correlation condition to understand if every telephone consultation is preceded by a first consultation in the same department. There is no activation condition for this constraint. This means that every time `telephone consultation` occurs, the constraint is activated. The constraint has 1 286 activations. Around 42% of these activations are violations. Some of these violations are due to the occurrence of telephone consultations preceded by a first consultation in a different department. In addition, it is also worth to highlight that the log we are using for this case study is an excerpt derived from a larger log and it contains several cases that are truncated both at the beginning and at the end. This can be also the reason of violations for this constraint.

In Table 9, we show the execution times needed for checking the constraints in this case study.[4] [5] For each of them, the execution time is lower that 2 seconds. This confirms that the scalability of our tool.

7.2. A Dutch Financial Institution

The second case study we discuss is based on the application of the proposed approach to the event log provided for the BPI challenge 2012 and taken from a Dutch financial institute [40]. The event log pertains to an application process for personal loans or overdrafts. It contains 262 200 events distributed across 36 event classes and includes 13 087 cases. The amount requested by the customer is indicated in the case attribute `AMOUNT_REQ`. In addition, the log contains the standard XES attributes for events.

For this case study, we have used the constraints shown in Table 10. Some of these constraints involve some specific transactional states (a.k.a. event types) of an activity. For example, the parameters specified for constraint 7-10 are `W_Valideren aanvraag-SCHEDULE` and `W_Valideren aanvraag-START`. When an event type is not specified, like in the case of constraint 3-6, the event type considered by default is "complete".

With constraint 3, we want to understand how many submitted applications are eventually accepted. As shown in Table 11, there are 13 087 submissions of which only 5 113 are eventually accepted (around 39%). Using constraint 4, we

---

[4]The execution times in all the tables of this section are averaged over 5 runs.

[5]All the experiments described in this section have been performed on a machine with an Intel(R) Core(TM) i7-2670QM CPU @ 2.20GHz (limiting the execution to just one core), 8 GB of RAM and the Oracle Java virtual machine installed on a GNU/Linux Ubuntu operating system.

can understand that the majority of these accepted applications (around 79%) are accepted in less than 24 hours from the submission. Using constraints 5 and 6, we can understand how the requested amount affects the application. In

Table 10: Reference constraints used to analyze the log from the BPI challenge 2012.

Id	Constraint	1st param.	2nd param.	Activation condition	Correlation condition	Time condition
3	Response	A_SUBMITTED	A_ACCEPTED	–	–	–
4	Response	A_SUBMITTED	A_ACCEPTED	–	–	0,24,h
5	Response	A_SUBMITTED	A_ACCEPTED	A.AMOUNT_REQ >= 10 000	–	–
6	Response	A_SUBMITTED	A_ACCEPTED	A.AMOUNT_REQ < 10 000	–	–
7	Response	W_Valideren aanvraag-SCHEDULE	W_Valideren aanvraag-START	–	–	–
8	Response	W_Valideren aanvraag-SCHEDULE	W_Valideren aanvraag-START	–	A.org:resource != T.org:resource	–
9	Response	W_Valideren aanvraag-SCHEDULE	W_Valideren aanvraag-START	–	A.org:resource != T.org:resource	0,7,d
10	Response	W_Valideren aanvraag-SCHEDULE	W_Valideren aanvraag-START	–	A.org:resource != T.org:resource	0,24,h
11	Response	W_Valideren aanvraag-START	W_Valideren aanvraag-COMPLETE	–	–	–
12	Response	W_Valideren aanvraag-START	W_Valideren aanvraag-COMPLETE	–	A.org:resource == T.org:resource	–
13	Response	W_Valideren aanvraag-START	W_Valideren aanvraag-COMPLETE	–	A.org:resource == T.org:resource	0,1,h
14	Response	W_Valideren aanvraag-START	W_Valideren aanvraag-COMPLETE	–	A.org:resource == T.org:resource	0,15,m

Table 11: Conformance checking results using the log from the BPI challenge 2012.

Id	Act.no.	Viol.no.	Fulfill.no.	Avg.act.sparsity	Avg.viol.ratio	Avg.fulfill.ratio
3	13 087	7 974	5 113	0.8596	0.6093	0.3907
4	13 087	9 036	4 051	0.8596	0.6905	0.3095
5	6 847	3 601	3 246	0.9585	0.5259	0.4741
6	6 240	4 373	1 867	0.9211	0.7008	0.2992
7	5 023	51	4 972	0.9909	0.0102	0.9898
8	5 023	236	4 787	0.9909	0.047	0.953
9	5 023	263	4 760	0.9909	0.0524	0.9476
10	5 023	2 897	2 126	0.9909	0.5767	0.4233
11	7 891	2	7 889	0.9863	0.0003	0.9997
12	7 891	6	7 885	0.9863	0.0008	0.9992
13	7 891	228	7 663	0.9863	0.0289	0.9711
14	7 891	3 355	4 536	0.9863	0.4252	0.5748

particular, when the requested amount is lower than 10 000 the acceptance rate is almost 30%. The acceptance rate is higher if the requested amount is greater or equal to 10 000 (almost half of the applications is accepted in this case).

With constraints 7-14, we analyze the validation of the applications. With constraint 7, we can see that almost 99% of the scheduled validations are eventually started. In 95% of the cases, the resource that schedules the validation is not the same resource that starts this activity (see constraint 8). In addition, in around 94% of the cases, a scheduled validation is started within 7 days from the scheduling (constraint 9) and in almost half of the cases the validation is started only 24 hours after the scheduling. Constraint 11 indicates that almost 100% of the validations that have been started are also completed, and almost in all the cases the resource that starts the validation is the same resource that

Table 12: Execution times using the log from the BPI challenge 2012.

Id	Avg.execution time (milliseconds)
3	2 772
4	3 220
5	3 261
6	3 205
7	3 196
8	3 100
9	3 212
10	3 146
11	2 176
12	3 210
13	3 241
14	3 258

(a) Example of fulfillment `W_Valideren aanvraag-START` at position 35.

(b) A correlated target `W_Valideren aanvraag-COMPLETE` at position 36 executed by the same resource.

Figure 10: Example of fulfillment for constraint 13.

(a) Example of violation W_Valideren aanvraag–START at position 37.

(b) A possible target W_Valideren aanvraag–COMPLETE occurs more than 1 hours after.

Figure 11: Example of violation for constraint 13; W_Valideren aanvraag–COMPLETE occurs outside the required time interval (too late).

(a) Example of fulfillment W_Valideren aanvraag–START at position 39.

(b) Corresponding target executed by the same resource.

Figure 12: Example of fulfillment for constraint 13; W_Valideren aanvraag–START at position 39 is followed by W_Valideren aanvraag–COMPLETE within the required time interval.

Table 13: Reference constraints used to analyze the log from the BPI challenge 2014.

Id	Constraint	1st param.	2nd param.	Activation condition	Correlation condition	Time condition
15	Not response	Open	Reopen	–	–	–
16	Not response	Open	Reopen	–	A.org:resource != T.org:resource	–
17	Response	Open	Closed	–	–	–
18	Response	Open	Closed	–	–	0,12,h
19	Response	Open	Closed	A.KMnumber == 'KM0000611'	–	0,12,h
20	Response	Open	Closed	A.KMnumber == 'KM0002043'	–	0,12,h

Table 14: Conformance checking results using the log from the BPI challenge 2014.

Id	Act.no.	Viol.no.	Fulfill.no.	Avg.act.sparsity	Avg.viol.ratio	Avg.fulfill.ratio
15	46 607	2 121	44 486	0.8468	0.0455	0.9545
16	46 607	510	46 097	0.8468	0.0109	0.9891
17	46 607	449	46 158	0.8468	0.0096	0.9904
18	46 607	24 392	22 215	0.8468	0.5234	0.4766
19	446	386	60	0.9993	0.8655	0.1345
20	773	48	725	0.9969	0.0621	0.9379

completes this activity (see constraint 12). In 97% of the cases, the validation is done in at most 1 hour (constraint 13), and in more than half of the cases it is completed in less than 15 minutes (constraint 14).
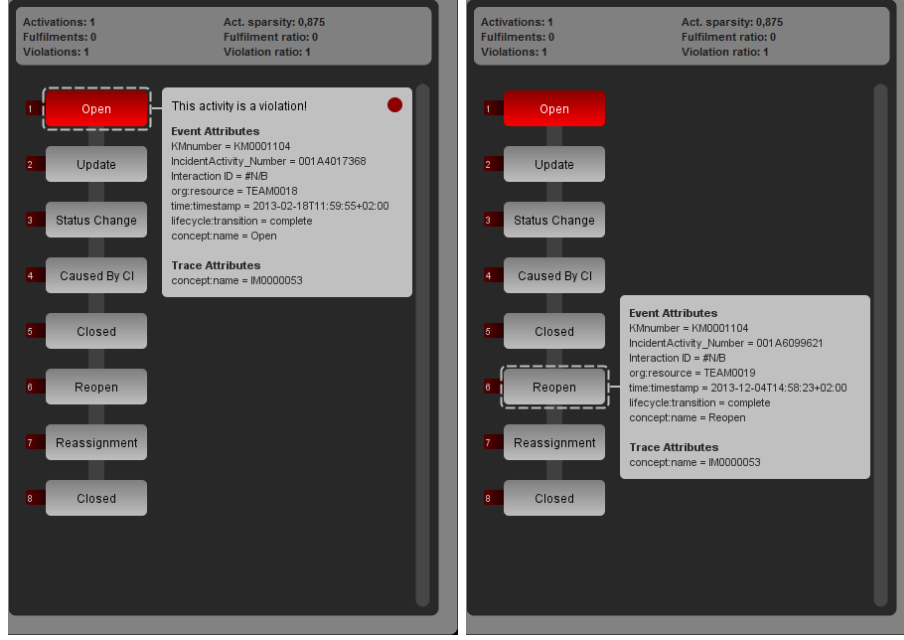
In Figure 10 and 12, we show two fulfillments for constraint 13 (the activations with the correlated targets). 12 shows a violation for the same constraint. In Table 12, we show the execution times needed for checking the constraints in this case study. Also in this case, like in the first case study here presented, the execution time is low (between 2 and 3 seconds on average).

7.3. Rabobank

The case study we illustrate in this section has been provided for the BPI challenge 2014 by Rabobank Netherlands Group ICT [41]. The log we use pertains to the management of calls or mails from customers to the Service Desk concerning disruptions of ICT-services. The log contains 46 616 cases,

Table 15: Execution times using the log from the BPI challenge 2014.

Id	Avg.execution time (milliseconds)
15	4 294
16	5 093
17	5 240
18	5 055
19	4 861
20	5 398

(a) Example of violation `Open` at position 1. (b) A forbidden event `Reopen` occurs after `Open`.

Figure 13: Example of violation for constraint 16; `Open` is followed by an event `Reopen` associated to a different resource.

466 737 events referring to 39 different event classes. There are 242 originators and domain specific event attributes like `KM number`, `Interaction ID` and `IncidentActivity_Number`. For this case study, we have used the constraints shown in Table 13.

As shown in Table 14, constraint 15 has 46 607 activations and 44 486 fulfillments. This allows us to understand that in around 95% of open calls are not reopened afterwards. This percentage is even higher if we require that an open call cannot be eventually reopened by the same resource (see constraint 16). Indeed, this is true in almost 99% of the cases.

Around 99% of the open calls are eventually closed (see constraint 17). Around half of them are closed within 12 hours (constraint 18). The "KM number" in this case study identifies the characteristics of a call to understand how urgent the corresponding problem is. The checks on rules 19 and 20 show that the calls corresponding to the number `KM0002043` are, in general, more urgent than the ones corresponding to the number `KM0000611`. Indeed, over 446 calls corresponding to the KM number `KM0000611` only 60 are closed within 12 hours. On the other hand, over 773 calls corresponding to the KM number `KM0002043`, 725 are closed within 12 hours.

Figure 13 shows a violation for constraint 16. The selected event `Open` is fol-

30

lowed by a forbidden event `Reopen` (associated to a different resource). Table 15 shows that the execution times for this case study range from 4 to 5 seconds.

8. Conclusion and Future Work

In this work, we propose a framework for checking the conformance of event logs with respect to MP-Declare models. MP-Declare is an extension of the declarative process modeling language Declare that allows the modeler to specify constraints over the data associated to the control-flow and over the "time dimension" of a business process. We describe and discuss in detail how the proposed framework can be used to define algorithms for conformance checking based on MP-Declare. Our proposal has been implemented in the process mining tool ProM. The implemented software covers the entire set of MP-Declare templates. In addition, the conformance checker can also be used with standard Declare. A wide experimentation has been carried out using both real-life and synthetic logs. These case studies prove the applicability of our implementation in realistic settings. Although it is extremely important to recognize deviances *a-posteriori*, in some particular contexts, it would be also useful to detect violations on-the-fly as they occur. To this aim, in the near future we are planning to make the proposed framework suitable to be used in online settings.

References

[1] T. Murata, Petri nets: Properties, analysis and applications., in: Proceedings of the IEEE, 1989, pp. 541–580.

[2] O. M. G. (OMG), Business Process Model and Notation (BPMN) Version 2.0, Tech. rep. (jan 2011).

[3] J. E. Cook, A. L. Wolf, Software process validation: Quantitatively measuring the correspondence of a process to a model, ACM Trans. Softw. Eng. Methodol. 8 (2) (1999) 147–176.

[4] A. Rozinat, W. M. P. van der Aalst, Conformance checking of processes based on monitoring real behavior, Inf. Syst. 33 (1) (2008) 64–95.

[5] W. M. P. van der Aalst, Process Mining: Discovery, Conformance and Enhancement of Business Processes, 1st Edition, Springer Publishing Company, Incorporated, 2011.

[6] A. Adriansyah, B. F. van Dongen, W. M. P. van der Aalst, Conformance checking using cost-based fitness analysis, in: Proceedings of the 15th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2011, 2011, pp. 55–64.

[7] M. de Leoni, W. M. van der Aalst, Aligning Event Logs and Process Models for Multi-Perspective Conformance Checking: An Approach Based on Integer Linear Programming, in: International Conference on Business Process Management, Springer Berlin Heidelberg, 2013, pp. 113–129.

[8] F. Mannhardt, M. de Leoni, H. A. Reijers, W. M. van der Aalst, Balanced multi-perspective checking of process conformance, Tech. Rep. BPM-14-07, BPM Center (2014).

[9] W. M. P. van der Aalst, Decomposing process mining problems using passages, in: Application and Theory of Petri Nets - 33rd International Conference, Petri Nets 2012, 2012, pp. 72–91.

[10] W. M. P. van der Aalst, Decomposing petri nets for process mining: A generic approach, Distributed and Parallel Databases 31 (4) (2013) 471–507.

[11] M. de Leoni, J. Munoz-Gama, J. Carmona, W. M. P. van der Aalst, Decomposing alignment-based conformance checking of data-aware process models, in: On the Move to Meaningful Internet Systems: OTM 2014 Conferences - Confederated International Conferences: CoopIS, and ODBASE 2014, 2014, pp. 3–20.

[12] J. Munoz-Gama, J. Carmona, W. M. P. van der Aalst, Single-entry single-exit decomposed conformance checking, Inf. Syst. 46 (2014) 102–122.

[13] W. van der Aalst, M. Pesic, H. Schonenberg, Declarative Workflows: Balancing Between Flexibility and Support, Computer Science - R&D (2009) 99–113.

[14] Declare (2008).
URL http://declare.sf.net

[15] M. Pesic, H. Schonenberg, W. van der Aalst, DECLARE: Full Support for Loosely-Structured Processes, in: EDOC 2007, pp. 287–298.

[16] F. Chesani, P. Mello, M. Montali, F. Riguzzi, M. Sebastianis, S. Storari, Checking Compliance of Execution Traces to Business Rules, in: Business Process Management Workshops, 2009, pp. 134–145.

[17] M. Montali, M. Pesic, W. M. van der Aalst, F. Chesani, P. Mello, S. Storari, Declarative specification and verification of service choreographiess, ACM Transactions on the Web 4 (1) (2010) 1–62.

[18] A. Burattin, F. M. Maggi, W. M. P. van der Aalst, A. Sperduti, Techniques for a Posteriori Analysis of Declarative Processes, in: 2012 IEEE 16th International Enterprise Distributed Object Computing Conference, IEEE, 2012, pp. 41–50.

[19] M. Montali, F. Chesani, P. Mello, F. M. Maggi, Towards data-aware constraints in declare, in: Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC '13, 2013, pp. 1391–1396.

[20] D. Borrego, I. Barba, Conformance checking and diagnosis for declarative business process models in data-aware scenarios, Expert Systems with Applications 41 (11) (2014) 5340–5352.

[21] W. M. P. van der Aalst, Process Mining: Discovery, Conformance and Enhancement of Business Processes, Springer Berlin / Heidelberg, 2011.

[22] A. Adriansyah, Aligning observed and modeled behavior, Phd thesis, Technische Universiteit Eindhoven (2014).

[23] M. Pešić, W. M. P. van der Aalst, A Declarative Approach for Flexible Business, in: Business Process Management, Springer Berlin Heidelberg, 2006, pp. 169–180.

[24] D. Basin, V. Jugé, F. Klaedtke, E. Zlinescu, Enforceable Security Policies Revisited, ACM Transactions on Information and System Security 16 (1) (2013) 1–26.

[25] D. Basin, M. Harvan, F. Klaedtke, E. Zlinescu, Monitoring Data Usage in Distributed Systems, IEEE Transactions on Software Engineering 39 (10) (2013) 1403–1426.

[26] M. A. Grando, W. M. P. van der Aalst, R. S. Mans, Reusing a Declarative Specification to Check the Conformance of Different CIGs, in: Business Process Management Workshops, Springer Berlin Heidelberg, 2012, pp. 188–199.

[27] M. A. Grando, M. H. Schonenberg, W. M. P. van der Aalst, Semantic-Based Conformance Checking of Computer Interpretable Medical Guidelines, in: International Joint Conference, BIOSTEC, Vol. 273 of Communications in Computer and Information Science, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 285–300.

[28] M. D. Leoni, F. M. Maggi, W. M. P. van der Aalst, Aligning Event Logs and Declarative Process Models for Conformance Checking, in: Business Process Management, Springer Berlin / Heidelberg, 2012, pp. 82–97.

[29] M. de Leoni, F. M. Maggi, W. M. van der Aalst, An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data, Information Systems (2014) 1–20.

[30] IEEE Task Force on Process Mining: XES Standard Definition, 2013.

[31] H. M. W. Verbeek, J. C. A. M. Buijs, B. F. van Dongen, W. M. P. van der Aalst, XES, XESame, and ProM 6, in: Information Systems Evolution - CAiSE Forum, Vol. 72, 2010, pp. 60–75.

[32] J. Chomicki, Efficient checking of temporal integrity constraints using bounded history encoding, ACM Trans. Database Syst. 20 (2) (1995) 149–186.

[33] R. Koymans, Specifying real-time properties with metric temporal logic, Real-Time Systems 2 (4) (1990) 255–299.

[34] M. Montali, M. Pesic, W. M. P. van der Aalst, F. Chesani, P. Mello, S. Storari, Declarative Specification and Verification of Service Choreographies, ACM Transactions on the Web 4 (1).

[35] O. Kupferman, M. Vardi, Vacuity Detection in Temporal Model Checking, Int. Journal on Software Tools for Technology Transfer (2003) 224–233.

[36] F. M. Maggi, Declarative process mining with the declare component of prom, in: BPM (Demos), 2013.

[37] M. Westergaard, F. M. Maggi, Looking into the future: Using timed automata to provide a priori advice about timed declarative process models, in: Proc. of CoopIS, LNCS, Springer, 2012.

[38] R. D. Masellis, F. M. Maggi, M. Montali, Monitoring data-aware business constraints with finite state automata, in: International Conference on Software and Systems Process 2014, ICSSP, 2014, pp. 134–143.

[39] 3TU Data Center, BPI Challenge 2011 Event Log (2011). `doi:doi:10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54`.

[40] 3TU Data Center, BPI Challenge 2012 Event Log (2012). `doi:10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f`.

[41] 3TU Data Center, BPI Challenge 2014 Event Log (2014). `doi:10.4121/uuid:c3e5d162-0cfd-4bb0-bd82-af5268819c35`.