Conformance Checking Based on Multi-Perspective Declarative Process Models

Andrea Burattin[a,*], Fabrizio M. Maggi[b], Alessandro Sperduti[c]

[a] *Institute of Computer Science, University of Innsbruck, Technikerstraße 21a, 6020 Innsbruck, Austria.*
[b] *Institute of Computer Science, University of Tartu, Liivi 2, 50409 Tartu, Estonia.*
[c] *Department of Mathematics, University of Padua, Via Trieste 63, 35121 Padova, Italy.*

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 behavior provided in the form of a process model. Recently, techniques for conformance checking based on declarative specifications have been developed. Such specifications are suitable to describe processes characterized by high variability. However, an open challenge in the context of conformance checking with declarative models is the capability of supporting multi-perspective specifications. This means that declarative models used for conformance checking should not only describe the process behavior from the control flow point of view, but also from other perspectives like data or time. In this paper, we close this gap by presenting an approach 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 using artificial and real-life event logs.

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

1. Introduction

The fast pace of change in markets is more and more imposing the definition and use of flexible information systems for supporting business processes of companies, and organizations in general. In fact, such dynamic markets make

*Corresponding author
Email addresses: andrea.burattin@uibk.ac.at (Andrea Burattin), f.m.maggi@ut.ee (Fabrizio M. Maggi), sperduti@math.unipd.it (Alessandro Sperduti)

the complete specification of a business process obsolete in a relatively short span of time, due to the need of bringing frequent modifications and updates to the process in order to accommodate for new market conditions and/or regulations. Notwithstanding these dynamic scenarios, 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. In particular, there are some process constraints that need to be satisfied for the process to be considered healthy and aligned with the current business goals.

Examples of such constraints that can be found in the literature are:

- a bank account is opened only in case risk is low (Awad et al., 2009a);
- in case the respondent bank rating review is rejected, an account must never be opened (Awad et al., 2009a);
- for payment runs with amount beyond 10,000 euros, the payment list has to be signed before being transferred to the bank and has to be filed afterwards for later audits (Ly et al., 2011);
- if an open item is not marked as cleared within 30 days, the bank details may be faulty. Thus, the bank details have to be (re)checked (Ly et al., 2011);
- a passenger ship leaving Amsterdam has to moor in Newcastle within 16 hours (Maggi et al., 2013b);
- if the size of a fishing boat is above 25 m (100 tons) and it is located at 541 of latitude and 8.51 of longitude, it cannot be engaged in fishing (Maggi et al., 2013b);
- if employing any electronic storage media other than optical disk technology (including CD-ROM), the member, broker, or dealer must notify its designated examining authority at least 90 days prior to employing such storage media (Giblin et al., 2006);
- if the PainScore of patient p is greater than 7 and the status is uninitialized then the status must be changed to initialized and a timer event is generated to treat patient p within 1 h (Middleton et al., 2009);
- if the first test terminates with a particular result code, then all the consequent executions of the test should return the same result code (Ly et al., 2015);
- orders of more than $1,000$ euros can only be approved by a senior manager (Ly et al., 2015);
- final approval of the assessment can only be granted by the manager that requested the assessment (Ly et al., 2015);
- every closed project must be validated by a person who did not participate in the project (4-Eyes principle, also called Separation of Duties) (Ly et al., 2015);
- in case the total number of users permissible on the server has reached the limit, access privilege to current potential user requires exception approval from IT administrator (Narendra et al., 2008);
- any [LightPathOperation (LPO)] ID appearing in any partition request must be different from any LPO ID appearing in any future concatenate request (Hallé & Villemaire, 2008).

Other constraints have been used in the context of real-life data analysis:

- if "hemoglobine foto-elektrisch" occurs in a trace and "Diagnosis Treatment Combination ID" is equal to 495326, then "hemoglobine foto-elektrisch" is followed eventually by "ureum" (3TU Data Center, 2011);
- "A_SUBMITTED-complete" and "A_PARTLYSUBMITTED-complete" are always performed by the same actor (3TU Data Center, 2012);
- if "Create fine" is not followed by "Payment" within two months then"Add penalty" must eventually occur (3TU Data Center, 2015).

These *multi-perspective* constraints describe the expected process behavior not only from the control flow point of view, but also from other perspectives like data or time.[1] These perspectives can be evaluated in isolation, but can also appear simultaneously in the same constraint.

Early works in conformance checking (Cook & Wolf, 1999; Rozinat & van der Aalst, 2008) mainly focused on the control flow perspective in the context of procedural models, i.e., on the functional dependencies among performed activities/tasks in the process, while abstracting from time constraints or data dependencies. 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 alignment-based approaches, where the conformance checking is performed by aligning both the modeled behavior and the behavior observed in the log (Adriansyah et al., 2011). Only recently, approaches able to deal with data dependencies have been developed (de Leoni & van der Aalst, 2013; Mannhardt et al., 2014).

Procedural specifications, such as Petri Nets (Desel & Esparza, 1995) or BPMN (Object Management Group, 2011), can be problematic to deal with when the process undergoes a frequent redefinition through time, while they are very effective for describing processes that are stable in time and predictable. In dynamic scenarios, declarative process description formalisms are preferred (Silva et al., 2014; Zugal et al., 2011; Pichler et al., 2011; van der Aalst et al., 2009; Haisjackl et al., 2013). A declarative approach allows the business analyst to focus on the formalization of few relevant and relatively stable through time constraints that the process should satisfy, avoiding the burden to model many process control-flow details that are fated to change through time. Through declarative formalisms the processes are kept under-specified so that few rules allow for multiple execution paths. In this way, the process representation is more robust to changeable behaviors and, at the same time, remains compact.

In (van der Aalst et al., 2009; Westergaard & Maggi, 2011; Pesic et al., 2007), the authors introduce a declarative process modeling language called *Declare* with formal semantics grounded in LTL (Linear Temporal Logic) (Pnueli, 1977).

[1] The notion of time taken into consideration in the control flow analysis is qualitative (i.e., based on the total order relation between events in a log). The additional perspective we want to deal with here is the one that takes into consideration a quantitative notion of time (e.g., conformance with respect to constraints based on timestamps like deadlines or latencies).

A Declare model is a set of Declare constraints defined as instantiations of Declare templates that are parameterized classes of properties. Since Declare models are focused on ruling out forbidden behavior, Declare is very suitable for defining compliance models that are used for checking that the behavior of a system complies certain regulations. The compliance model defines the rules related to a single instance of a process, and the expectation is that all the instances follow the model.

Conformance checking approaches based on the control flow perspective have been defined for declarative models in (Chesani et al., 2009; Montali et al., 2010; de Leoni et al., 2012, 2014a; Burattin et al., 2012). One of the most well known tools for conformance checking with declarative specifications is the *LTL Checker* presented for the first time in (van der Aalst et al., 2005). Recently, the additional data perspective has been considered in (Borrego & Barba, 2014), even if, in this work, the data perspective is not fully integrated with the control flow perspective. This means that the conformance checking on data is performed by specifying conditions on a set of global variables that must be satisfied during the process execution and are completely disconnected from the control flow. Consider, for example, a diabetic patient who must take sugar if his or her glucose level is below 50 mg/dl. When a test is completed and the measured level is 50 or less, then the patient must eventually take sugar. This constraint clearly relates information coming from data (glucose level below 50 mg/dl) to a behavior imposed on the control flow (eventually take sugar). The technique presented in (Borrego & Barba, 2014) does not allow for checking this type of constraints. In addition, with this technique it is not possible to impose temporal conditions like "the patient must promptly take sugar within 3 minutes after the glucose test". Therefore, efficient and fully integrated multi-perspective conformance checking proposals for declarative models are still missing.

In this paper, we aim at closing this gap by proposing a conformance checking approach based on Declare that allows for defining multi-perspective constraints jointly considering data, temporal, and control flow perspectives. To this aim, we formally define Multi-Perspective Declare (MP-Declare), an augmented version of Declare that, being based on Metric First-Order Linear Temporal Logic semantics, allows for the definition of activation, correlation, and time conditions to build constraints over traces.

We assess the validity of the proposed approach both on artificial and real-life event logs. Controlled artificial data, involving logs containing up to 5 million events, are used to test the scalability of the proposed approach, while a real-life event log is used to demonstrate its suitability to define and check multi-perspective constraints in real-life scenarios. In particular, through our evaluation, we want to answer the following research questions:

RQ1 What is the effectiveness of the proposed approach?

- **RQ1.1** What are the violations that can be detected using multi-perspective constraints (absolute effectiveness)?
- **RQ1.2** What are the violations that can be detected using multi-perspective

constraints that are not identifiable with the existing control flow-based conformance checking approaches (relative effectiveness)?

RQ2 How does the execution time of the proposed approach compare to that of similar existing approaches (efficiency)?

The rest of the paper is structured as follows. In Section 2, we discuss the related work. Section 3 provides some background notions needed to understand the main contribution of the paper. In Section 4, we introduce the semantics of Multi-Perspective Declare based on Metric First-Order Linear Temporal Logic. In Section 5, we discuss the proposed conformance checking algorithms. Section 6 and 7 discuss the implementation of the approach and its evaluation, respectively. Section 8 concludes the paper and spells out directions for future work.

2. Related Work

The scientific literature reports several works in the field of conformance checking (van der Aalst, 2011). 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 (Cook & Wolf, 1999; Rozinat & van der Aalst, 2008; Adriansyah et al., 2011; de Leoni & van der Aalst, 2013; Mannhardt et al., 2014; van der Aalst, 2012, 2013; de Leoni et al., 2014b; Munoz-Gama et al., 2014; Knuplesch et al., 2010; Awad et al., 2009b; Taghiabadi et al., 2013, 2014).

In (Cook & Wolf, 1999; Rozinat & van der Aalst, 2008), for the first time, the concept of conformance checking with respect to (procedural) process models was investigated. In (Adriansyah et al., 2011), the authors introduce conformance checking augmented with the notion of alignments. Alignment-based approaches have also been used in (de Leoni & van der Aalst, 2013; Mannhardt et al., 2014; Taghiabadi et al., 2013, 2014), where the authors propose techniques for conformance checking with respect to time- and data-aware procedural models. In all these cases, however, authors applied conformance checking techniques to event logs and procedural models that cannot be applied to declarative models. As discussed in the introduction and demonstrated in (Silva et al., 2014; Zugal et al., 2011; Pichler et al., 2011; van der Aalst et al., 2009; Haisjackl et al., 2013), declarative specification are useful to capture process behaviors in turbulent environments where several execution paths are allowed.

In the above techniques, conformance checking is meant to be a post-mortem analysis. Other approaches, like those reported in (Knuplesch et al., 2010; Awad et al., 2009b), deal with compliance checking at design-time, before the process is deployed and executed. In (Knuplesch et al., 2010), the authors present an approach for checking procedural compliance models in cross-organizational business processes. These models are limited to the control flow perspective. In

(Awad et al., 2009b), a process model is queried and checked with respect to (procedural) compliance queries expressed in BPMN-Q.

In recent years, an increasing number of researchers are focusing on the conformance checking with respect to declarative models. For example, in (Chesani et al., 2009), an approach for compliance checking with respect to *reactive business rules* is proposed. Rules, expressed using Declare, are mapped to Abductive Logic Programming, and Prolog is used to perform the validation. The approach has been extended in (Montali et al., 2010), by mapping constraints to LTL, and evaluating them using automata. The entire work has been contextualized into the service choreography scenario. In (Grando et al., 2012, 2013), Grando et al. used Declare to model medical guidelines and to provide semantic (i.e., ontology-based) conformance checking measures. In (Burattin et al., 2012), 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 work described in (de Leoni et al., 2012, 2014a) 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. One of the most well known tools for conformance checking with declarative specifications is the *LTL Checker* presented for the first time in (van der Aalst et al., 2005).

However, in all these approaches neither data nor time perspectives are taken into account. In addition, automata-based approaches to conformance checking are very difficult to extend to the multi-perspective case since the verification of data-aware constraints faces a combinatorial blow up of the state-space, commonly known as the state explosion problem.

In a recent work, reported in (Borrego & Barba, 2014), 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. In other words, data constraints are not bound to control flow constraints and thus it is not possible to bind the behavior to specific data attributes. Moreover, the 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 other.

In (Westergaard & Maggi, 2012; Maggi & Westergaard, 2014), the authors propose an approach for checking Declare constraints augmented with temporal conditions, whereas, in (De Masellis et al., 2014), an approach for monitoring data-aware Declare constraints at runtime is presented. These approaches focus separately on the time perspective (Westergaard & Maggi, 2012; Maggi & Westergaard, 2014) and on the data perspective (De Masellis et al., 2014) and

it is not possible to express constraints involving multiple perspectives at the same time. In addition, a full-fledged implementation of such approaches is not available.

The result of our state of the art analysis is that conformance checking techniques for multi-perspective declarative process models are missing. With this work, we aim at filling this gap. In particular, first, we describe and formalize MP-Declare (i.e., Declare augmented with data and time constraints). Then, we provide an algorithmic framework to efficiently check the conformance of MP-Declare with respect to event logs.

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 (van der Aalst, 2011). The starting point for process mining is an event log. XES (eXtensible Event Stream) (IEEE Task Force on Process Mining, 2013; Verbeek et al., 2010) 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, each case follows a specific sequence of events (i.e., a trace). 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, i.e., 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 (Chomicki, 1995). MFOTL extends propositional metric temporal logic (MTL) (Koymans, 1990) to merge the expressivity of first-order logic together with the MTL temporal modalities. Since a business process is supposed to finish sooner or later, 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.

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 $\tau_i \in \mathbb{N}$.[2] D is assumed to have constant domains, i.e., $\Delta_i = \Delta_{i+1}$, for all $1 \leq i < n$. 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., $\tau_i \leq \tau_{i+1}$, for all $1 \leq i < n$.*

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 $1 \leq i \leq n$, we define $(D, \tau, v, i) \vDash \phi$ as follows:*

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

[2]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**
existence	$\mathbf{F}A$	A
absence	$\neg\mathbf{F}A$	A
choice	$\mathbf{F}A \vee \mathbf{F}B$	A, B
exclusive choice	$(\mathbf{F}A \vee \mathbf{F}B) \wedge \neg(\mathbf{F}A \wedge \mathbf{F}B)$	A, B
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

We add syntactic sugar for the normal connectives, such as $true \equiv \exists x.x \approx x$, $\phi_1 \vee \phi_2 \equiv \neg(\neg\phi_1 \wedge \neg\phi_2)$, $\forall x.\phi \equiv \neg\exists x.\neg\phi$ $\phi_1 \rightarrow \phi_2 \equiv (\neg\phi_1) \vee \phi_2$ and $\phi_1 \leftrightarrow \phi_2 \equiv (\phi_1 \rightarrow \phi_2) \wedge (\phi_2 \rightarrow \phi_1)$. 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), and $\mathbf{O}_I\psi \equiv \text{true}\mathbf{S}_I\psi$ (timed once 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 (van der Aalst et al., 2009; Pesic et al., 2007). 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 provides 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 instan-

tiations (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 (Montali et al., 2010), 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 (van der Aalst et al., 2009) for a full description of the language).

Templates *existence* and *absence* require that A occurs at least once and never occurs in every process instance, respectively. Templates *choice* and *exclusive choice* indicate that A or B occur eventually in each process instance. The exclusive choice template is more restrictive because it forbids A and B to occur both in the same process instance. 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 can only occur 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.

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* (Kupferman & Vardi, 2003). In (Burattin et al., 2012), 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 $\mathbf{t}_4$, for example, the response constraint $\mathbf{G}(a \rightarrow \mathbf{F}b)$ 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 (Burattin et al., 2012).

In (Burattin et al., 2012), 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. In our experiments, we also use *activation sparsity* introduced in (Burattin et al., 2012), which measures the percentage of events in the log that are activations for the constraint.

Tools implementing process mining approaches based on Declare are presented in (Maggi, 2013). The tools are implemented as plug-ins of the process mining toolkit ProM.

4. MFOTL Semantics for Multi-Perspective Business Constraints

In this section, we introduce a multi-perspective version of Declare (MP-Declare) based on 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 MP-Declare constraints. To demonstrate that the subset of MFOTL formulas needed to describe the semantics of MP-Declare constraints is decidable, in Section 7.1, we show that the validity of an MP-Declare constraint in a trace can be algorithmically verified. In Section 7.2, we also show that the computational complexity of our proposed algorithmic framework is suitable to guarantee good performances also when it is applied to large logs and models (this is also confirmed by the benchmark analysis illustrated in Section 6).

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. 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 (Westergaard & Maggi, 2012), 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 an MP-Declare constraint ($I = [a, b)$) with φ_τ. In addition, we suppose that the first event of a case occurs at time τ_0, A occurs at time τ_A, and B occurs at time τ_B.

We give now a detailed description of the templates listed in Table 2. Template *existence* requires that A occurs at least once with the activation condition φ_a holding true, and $\tau_A \in [\tau_0 + a, \tau_0 + b)$ in every process instance. Template *absence* requires that A never occurs with the activation condition φ_a holding true, and $\tau_A \in [\tau_0 + a, \tau_0 + b)$ in every process instance. Templates *choice* and *exclusive choice* indicate that A with the activation condition φ_a holding true, and $\tau_A \in [\tau_0 + a, \tau_0 + b)$ occurs, or B with the activation condition φ_a holding true, and $\tau_B \in [\tau_0 + a, \tau_0 + b)$ occurs in every process instance. The exclusive choice template is more restrictive because it forbids A and B to occur both in the same process instance under the specified conditions.

The *responded existence* template indicates that, if A occurs with the activation condition φ_a holding true, then B should also occur (either before or after A) with the correlation condition φ_c holding true, and $\tau_B \in [\tau_A - b, \tau_A - a)$ (if B occurs before A), or $\tau_B \in [\tau_A + a, \tau_A + b)$ (if B occurs after A). The *response* template specifies that, when A with the activation condition φ_a holding true occurs, then B should eventually occur after A with the correlation condition φ_c holding true, and $\tau_B \in [\tau_A + a, \tau_A + b)$. The *precedence* template indicates that B with the activation condition φ_a holding true can occur only if A has occurred before with the correlation condition φ_c holding true, and $\tau_A \in [\tau_B - b, \tau_B - a)$. Template *alternate response* specifies that, when A with the activation condition φ_a holding true occurs, then B should eventually occur

Table 2: Semantics for MP-Declare constraints.

Template	**MFOTL Semantics**
existence	$\mathbf{F}_I(A \wedge \exists x.\varphi_a(x))$
absence	$\neg\mathbf{F}_I(A \wedge \exists x.\varphi_a(x))$
choice	$\mathbf{F}_I(A \wedge \exists x.\varphi_a(x)) \vee \mathbf{F}_I(B \wedge \exists x.\varphi_a(x))$
exclusive choice	$(\mathbf{F}_I(A \wedge \exists x.\varphi_a(x)) \vee \mathbf{F}_I(B \wedge \exists x.\varphi_a(x))) \wedge$ $\neg(\mathbf{F}_I(A \wedge \exists x.\varphi_a(x)) \wedge \mathbf{F}_I(B \wedge \exists x.\varphi_a(x)))$
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	$\mathbf{G}(\forall x.((B \wedge \varphi_a(x)) \rightarrow \neg\mathbf{Y}_I(A \wedge \exists y.\varphi_c(x,y)))$

after A with the correlation condition φ_c holding true, and $\tau_B \in [\tau_A + a, \tau_A + b)$. Another occurrence of A with the activation condition φ_a holding true is not allowed in $[\tau_A, \tau_B)$. Template *alternate precedence* indicates that B with the activation condition φ_a holding true can occur only if A has occurred before with the correlation condition φ_c holding true, and $\tau_A \in [\tau_B - b, \tau_B - a)$. Another occurrence of B with the activation condition φ_a holding true is not allowed in $[\tau_A, \tau_B)$. The *chain response* template specifies that, when A with the activation condition φ_a holding true occurs, then B should occur immediately after A with the correlation condition φ_c holding true, and $\tau_B \in [\tau_A + a, \tau_A + b)$. The *chain precedence* template indicates that B with the activation condition φ_a holding true can occur only if A has occurred immediately before with the correlation condition φ_c holding true, and $\tau_A \in [\tau_B - b, \tau_B - a)$.

The *not responded existence* template indicates that if A occurs with the activation condition φ_a holding true, then B cannot occur with the correlation condition φ_c holding true, and $\tau_B \in [\tau_A - b, \tau_A - a)$, or $\tau_B \in [\tau_A + a, \tau_A + b)$. According to the *not response* template, when A with the activation condition φ_a holding true occurs, then B cannot eventually occur after A with the correlation condition φ_c holding true, and $\tau_B \in [\tau_A + a, \tau_A + b)$. The *not precedence*

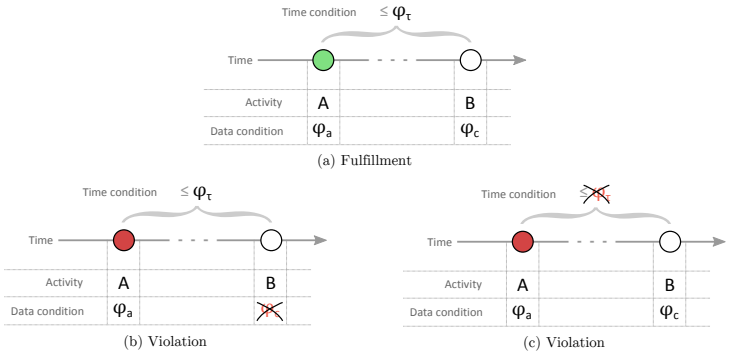


Figure 1: Fulfillment and violation scenarios for the *response* template. (a) reports a typical fulfillment scenario. In (b), the violation is due to the violation of the correlation condition φ_c. In (c), the violation is due to the violation of the time condition φ_τ.

template requires that B with the activation condition φ_a holding true cannot occur if A has occurred before with the correlation condition φ_c holding true, and $\tau_A \in [\tau_B - b, \tau_B - a)$. According to the *not chain response* template, when A with the activation condition φ_a holding true occurs, then B cannot occur immediately after A with the correlation condition φ_c holding true, and $\tau_B \in [\tau_A + a, \tau_A + b)$. Finally, the *not chain precedence* template indicates that B with the activation condition φ_a holding true cannot occur if A has occurred immediately before with the correlation condition φ_c holding true, and $\tau_A \in [\tau_B - b, \tau_B - a)$.

Graphical representations of the semantics 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 templates. Each figure shows possible scenarios of violations and fulfillments for the corresponding template. 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.

The *response* template 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* template 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* template 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.

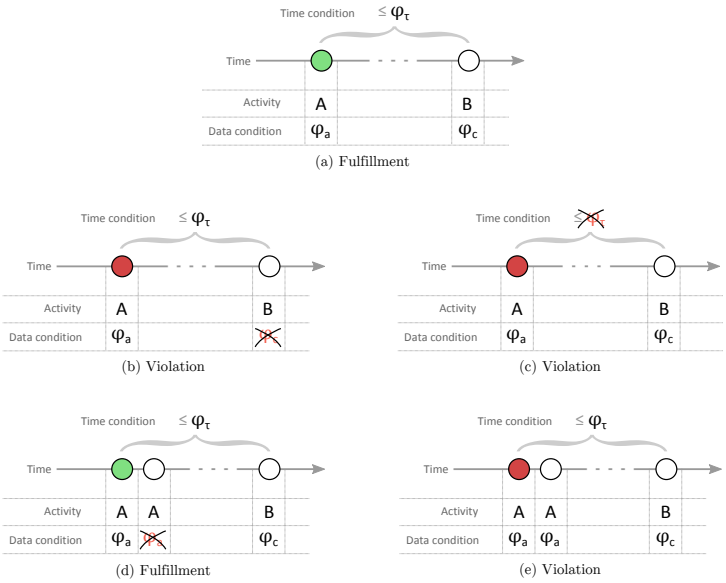


Figure 2: Fulfillment and violation scenarios for the *alternate response* template. (a) reports a typical fulfillment scenario. In (b), the violation is due to the violation of the correlation condition φ_c. In (c), the violation is due to the violation of the time condition φ_τ. The activation in (d) 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.

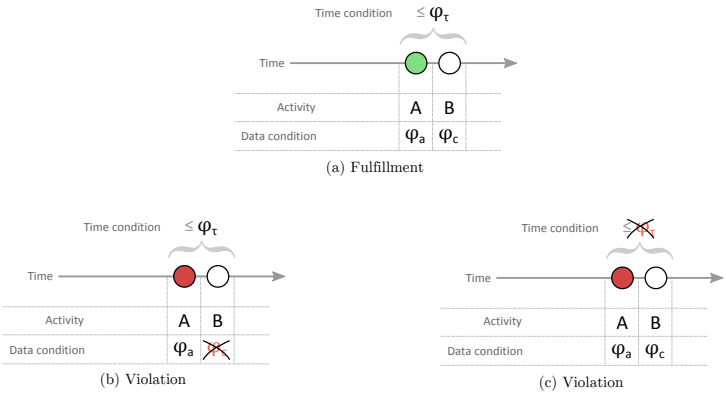


Figure 3: Fulfillment and violation scenarios for the *chain response* template. (a) reports a typical fulfillment scenario. Note that, in this case, the two events are contiguous. In (b), the violation is due to the violation of the correlation condition φ_c. In (c), the violation is due to the violation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

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 of a business process. 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 is 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 an MP-Declare model.

Note that, in order to keep this section clear and understandable, here we just report the pseudocode of the algorithms without considering possible optimizations. In Section 6, we discuss in detail the optimizations used to improve the performance of the implemented conformance checker.

5.1. General Framework

The proposed approach for the conformance checking of MP-Declare constraints is based on several procedures. The main component iterates through all traces of the input log and, for each constraint of the input MP-Declare model, computes violations and fulfillments by calling the `CheckTraceConformance` procedure. This procedure, described in Algorithm 1, takes as inputs 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 4, 6, 9, 12, 14).

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 and the set of pending activations to remove;
- *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 and the set of pending activations to remove;

Algorithm 1: `CheckTraceConformance`

Input: *trace*: a trace
$c = \langle t, A, T, \varphi_a, \varphi_c, \varphi_\tau \rangle$: a constraint
Output: Set of violating and fulfilling events

$pending \leftarrow \emptyset$
$fulfillments \leftarrow \emptyset$
$violations \leftarrow \emptyset$
/* Opening operations */
$t.opening()$
foreach $e \in trace$ **do**
  /* Fulfillment check */
  $\Delta_f, \Delta_p \leftarrow t.fulfillment(e, trace, pending, fulfillments, T, \varphi_a, \varphi_c, \varphi_\tau)$
  $fulfillments \leftarrow fulfillments \cup \Delta_f$
  $pending \leftarrow pending \setminus \Delta_p$
  /* Violation check */
  $\Delta_v, \Delta_p \leftarrow t.violation(e, trace, pending, violations, T, \varphi_c, \varphi_\tau)$
  $violations \leftarrow violations \cup \Delta_v$
  $pending \leftarrow pending \setminus \Delta_p$
  /* Activation check */
  $pending \leftarrow pending \cup t.activation(e, A, pending, \varphi_a)$
end
/* Final check and closing operations */
$\Delta_f, \Delta_v \leftarrow t.closing(pending, fulfillments, violations)$
$fulfillments \leftarrow fulfillments \cup \Delta_f$
$violations \leftarrow violations \cup \Delta_v$
return *violation*, *fulfillments*

- *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. Possible new fulfillments or violations are returned.

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.[3] 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, $template \in$

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

{*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.

Note that, in the proposed algorithms, we use this definition of constraint just for readability purposes. This definition still reflects an MFOTL formula in a straightforward manner. We also use functions *verify*(φ_a, A), *verify*(φ_c, A, B), and *verify*(φ_τ, A, B). The first function evaluates φ_a with respect to the attributes 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. These functions play an important role in the verification of constraints involving multiple perspectives. In fact, through these functions the validity of the conditions on data and timestamps is assessed. 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.

5.2. Template-Dependent Procedures

The first template we consider is *response* and the corresponding procedures are reported in Algorithm 2. 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  Δ_f, Δ_p ← ∅
2  if π_activity(e) ∈ T then
3  |  foreach act ∈ pending do
4  |  |  if verify(φ_c, act, e) and verify(φ_τ, act, e) then
5  |  |  |  Δ_f ← Δ_f ∪ {act}
6  |  |  |  Δ_p ← Δ_p ∪ {act}
7  |  |  end
8  |  end
9  end
10 return Δ_f, Δ_p
```

Response *(continued from previous page)*

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

```
1 return ∅, ∅            /* Actual violations are not identified here */
```

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

```
1 if π_activity(e) ∈ A and verify(φ_a, e) then
2 |   return {e}
3 end
4 return ∅
```

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

```
1 return ∅, pending  /* No fulfillment updates.  All pending activations
  are now violations */
```

Algorithm 2: Procedures for the *response* template.

The procedures for the *alternate response* template are reported in Algorithm 3. 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 7-8). If the activity referring to the input event is a target, the event is added to the set of possible targets (*fulfillment*, line 15). 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 8), otherwise it is marked as a violation (*closing*, line 12).

Alternate Response

template.opening()

```
1 define possibleTargets ← ∅ as a data structure available throughout the entire
  CheckTraceConformance algorithm
```

Alternate Response *(continued from previous page)*

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

```
1  Δ_f, Δ_p ← ∅
2  if π_activity(e) ∈ A and verify(φ_a, e) then
3  |  if |possibleTargets| ≥ 1 and |pending| = 1 then
4  |  |  act ← element ∈ pending            /* There is only one element */
5  |  |  foreach p ∈ possibleTargets do
6  |  |  |  if verify(φ_c, act, p) and verify(φ_τ, act, p) then
7  |  |  |  |  Δ_f ← Δ_f ∪ {act}
8  |  |  |  |  Δ_p ← Δ_p ∪ {act}
9  |  |  |  |  break                /* It is possible to exit the loop */
10 |  |  |  end
11 |  |  end
12 |  end
13 end
14 if e ∈ T then
15 |  possibleTargets ← possibleTargets ∪ {e}
16 end
17 return Δ_f, Δ_p
```

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

```
1  Δ_v, Δ_p ← ∅
2  if π_activity(e) ∈ A and verify(φ_a, e) then
3  |  if |pending| = 1 then
4  |  |  act ← element ∈ pending            /* There is only one element */
5  |  |  Δ_v ← Δ_v ∪ {act}
6  |  |  Δ_p ← Δ_p ∪ {act}
7  |  end
8  end
9  return Δ_v, Δ_p
```

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

```
1  if π_activity(e) ∈ A and verify(φ_a, e) then
2  |  possibleTargets ← ∅
3  |  return {e}
4  end
5  return ∅
```

Alternate Response *(continued from previous page)*

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

```
1  Δ_f, Δ_v ← ∅
2  if |pending| = 1 then
3  |  targetFound ← false
4  |  act ← element ∈ pending                    /* There is only one element */
5  |  foreach p ∈ possibleTargets do
6  |  |  if verify(φ_c, act, p) and verify(φ_τ, act, p) then
7  |  |  |  targetFound ← true
8  |  |  |  Δ_f ← Δ_f ∪ {act}
9  |  |  end
10 |  end
11 |  if not targetFound then
12 |  |  Δ_v ← Δ_v ∪ {act}
13 |  end
14 end
15 return Δ_f, Δ_v          /* Both fulfillment and violation are updated */
```

Algorithm 3: Procedures for the *alternate response* template.

The procedures for the *chain response* template are reported in Algorithm 4. 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
```

Chain Response *(continued from previous page)*

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

```
1 Δ_f, Δ_p ← ∅
2 if |pending| = 1 then
3 |   act ← element ∈ pending // There is only one element
4 |   if π_activity(e) ∈ T and verify(φ_c, act, e) and verify(φ_τ, act, e) then
5 |   |   Δ_f ← Δ_f ∪ {act}
6 |   |   Δ_p ← Δ_p ∪ {act}
7 |   end
8 end
9 return Δ_f, Δ_p
```

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

```
1 Δ_v, Δ_p ← ∅
2 if |pending| = 1 then
3 |   act ← element ∈ pending // There is only one element
4 |   if π_activity(e) ∉ T or not verify(φ_c, act, e) or not verify(φ_τ, act, e) then
5 |   |   Δ_v ← Δ_v ∪ {act}
6 |   |   Δ_p ← Δ_p ∪ {act}
7 |   end
8 end
9 return Δ_v, Δ_p
```

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

```
1 if π_activity(e) ∈ A and verify(φ_a, e) then
2 |   return {e}
3 end
4 return ∅
```

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

```
1 return ∅, pending  /* No fulfillment updates.  All pending activations
  are now violations */
```

Algorithm 4: Procedures for the *chain response* template.

The algorithms for the *existence* and the *choice* templates are straightforward. 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 positive templates. In this case, every fulfillment for a positive template becomes a violation for the corresponding negative template and vice versa.

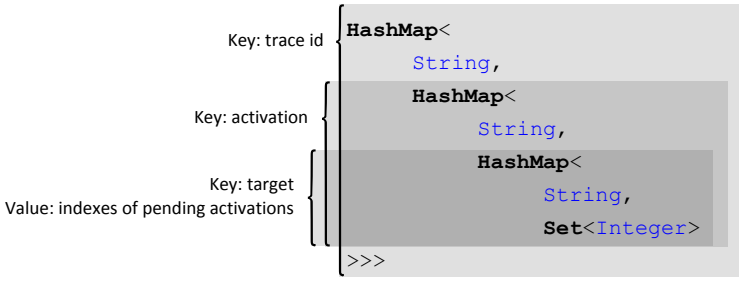


Figure 4: Java fragment with the type definition of the data structure storing the indexes of pending activations for different response constraints in different traces.

6. Implementation and Optimizations

The entire approach has been implemented as a plug-in of the process mining toolkit ProM called *Declare Analyzer*.[4] In particular, the plug-in receives as inputs an event log and a model, and evaluates the conformance of the log with respect to the model. In the implementation of the tool, we devised and applied several optimizations. The source code of the plug-in is available online.[5] Here, we mention some of the improvement strategies we implemented.

In Algorithm 1, one of the inputs is a constraint. However, in the actual implementation, this procedure is not invoked for each constraint, but only once for all constraints in the model that are instantiations of the same template. We store in constant time access data structures, the information related to all constraints instantiations of the same template thus reducing the number of invocations of the algorithm. As an example, the data structure to store pending activations for the response template is reported in Figure 4.

In addition, traces that do not contain any activation for the template under examination are not taken into consideration, and, also, cases referring to the same trace are analyzed only once. We would also like to mention that the presented algorithms have been designed to exploit the multi-core technologies nowadays available (either local or distributed over several computational nodes). Indeed, the processing of each trace is independent from all the others. Also, the analysis of each template in the model is independent from all the others. Therefore, it is possible to parallelize and distribute the analysis over different computational nodes and drastically improve the performances.

7. Evaluation

In this section, we first investigate the correctness and the computational complexity of the proposed algorithms. Then, we evaluate the efficiency of the

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

[5]The source code can be downloaded from https://svn.win.tue.nl/trac/prom/browser/Packages/DeclareAnalyzer/Trunk.

proposed approach and its applicability to a real-life scenario.

7.1. Correctness

The proposed algorithms correctly check the conformance of the corresponding MP-Declare templates. We investigate only the *response* template. However, it is possible to derive similar proofs for all the other templates.

Consider the *response* constraint between activities a and b. Then, the constraint provided to the `CheckTraceConformance` (Algorithm 1) is $c = \langle response, \{a\}, \{b\}, \varphi_a, \varphi_c, \varphi_\tau \rangle$. This constraint is fulfilled in a trace if, every time a occurs with φ_a holding true, b occurs eventually after a with φ_τ and φ_c holding true. In this proof, we analyze all the possible behaviors that are significant to check the conformance of the considered constraint given its MFOTL semantics.

Algorithm 1, first, calls the *opening* procedure of *response* (Algorithm 2), which does nothing. Then, it iterates over all the events of the trace. We can, therefore, distinguish two macro situations, i.e., whether the trace contains an activation of the considered constraint or not:

Case 1: the trace does not contain any instance of a (which is the only activation of the constraint). Then, *pending* is always empty, since the *activation* procedure always returns the empty set (Algorithm 2, *activation*, line 4). Because of that, in Algorithm 1, *fulfillments* and *violations* will always be empty, since *fulfillment* iterates over *pending* (Algorithm 2, *fulfillment*, line 3) and *violation* just returns the empty set. Finally, *closing* returns *pending*, which is empty. The result is that *violations* and *fulfillments* are both empty. This result is in line with the MFOTL semantics of the constraint, i.e., the constraint is satisfied, but never activated in the given trace.

Case 2: the trace contains an activation of the constraint (with φ_a holding true). The iteration over the events of the trace (Algorithm 1, line 5) does nothing until event e referring to the activation is reached. When such an event is reached, the *activation* procedure (Algorithm 2, *activation*, line 2) produces the addition of the event to *pending* (Algorithm 1, line 12). Then, the main iteration continues. At this stage, we can have two possible behaviors, and, therefore, we have to distinguish two additional sub-cases. The precondition holding for both these sub-cases is that *pending* already contains event e referring to a:

Case 2.1: an instance of b occurs after a with φ_τ and φ_c holding true. As previously said, when the main loop (Algorithm 1, line 5) reaches the event referring to b, *pending* already contains event e referring to a. The *fulfillment* procedure is called, and the event referring to b is recognized as a target event (Algorithm 2, *fulfillment*, line 2). Therefore, *fulfillment* adds e to both Δ_f and Δ_p (Algorithm 2, *fulfillment*, lines 5-6). These two sets are returned to Algorithm 1, which, in lines 7-8, adds e to *fulfillments* and removes e from *pending*. *pending* is now empty and, therefore, the result of the procedure is

that the constraint is activated and fulfilled once. This result is in line with the MFOTL semantics of the constraint.

Case 2.2: no instance of b occurs after a with φ_τ and φ_c holding true. In this case, *pending* already contains event e referring to a, but the main loop (Algorithm 1, line 5) never reaches a target event (Algorithm 2, *fulfillment*, line 2), which is the condition of the *fulfillment* procedure needed to convert pending activations into fulfillments. Therefore, `CheckTraceConformance` exits the loop (Algorithm 1, line 13) with *pending* still containing event e. The procedure *closing* is called and returns *pending* (containing e), which is then added to *violations* (Algorithm 1, lines 14, 16). Therefore, the procedure terminates returning one violation. This result is in line with the MFOTL semantics of the constraint.

7.2. Computational Complexity

From the computational complexity point of view, Algorithm 1 is called for each trace in the input log and for each constraint in the input MP-Declare model (i.e., $|Log| \cdot |Model|$ times) and its complexity is linear in the number of events in each trace ($|Log(t)|$, for trace t). In particular, *opening* and *closing* are called $|Log| \cdot |Model|$ times; *fulfillment*, *violation*, and *activation* are called $|Log| \cdot |Model| \cdot \sum_{t \in Log} |Log(t)|$ times.

The complexity of the template-dependent procedures, instead, depends on the actual template. In particular, we have the following complexities:

- Response: *opening*, *violation*, *activation*, and *closing* are constant; *fulfillment* has linear complexity in the number of pending activations in the current trace (which is at most the number of events in the trace);
- Alternate Response: *opening*, *violation*, and *activation* are constant; *fulfillment* and *closing* are linear in the number of possible targets in the current trace (which is at most the number of events in the trace);
- Chain Response: all procedures require a constant amount of time.

7.3. Benchmarks

In order to gain some insights on the computational feasibility of our implementation (**RQ2**), 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 3. In addition, we created 10 Declare models. In particular, we created two models with 10 response constraints, one only containing constraints on the control flow (without conditions on data and time), and another one including multi-perspective constraints (with conditions on time and data). We followed the same procedure to create models with 20,

		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	1 000 000	2 000 000	3 000 000	4 000 000
	50	1 250 000	2 500 000	3 750 000	5 000 000

Table 3: Number of events for each log.

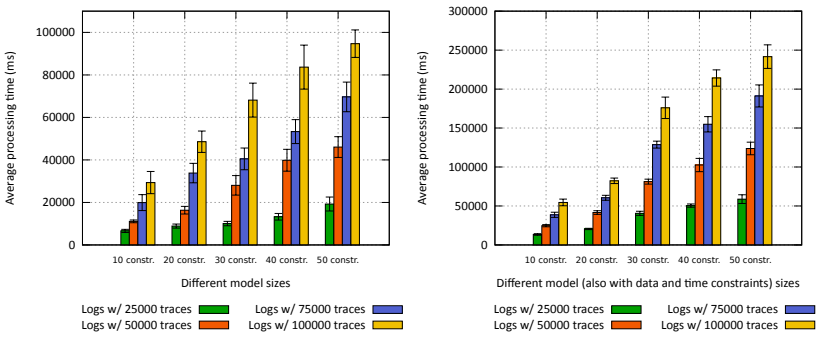


Figure 5: Execution times in milliseconds grouped based on the number of traces in the logs. The plot on the left hand side refers to models with control flow constraints. The plot on the right hand side refers to models with control flow, data and time constraints. Note the different scales between the two graphs.

30, 40, and 50 constraints. Models and logs are publicly available (Burattin, 2015).

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 performed the tests on a custom command-line version of ProM). All these tests have been performed using two machines, 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 5 and Figure 6 provide a graphical representation of the average execution times (and standard deviations) for the analysis of all models and logs. In particular, in Figure 5, 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 6, 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

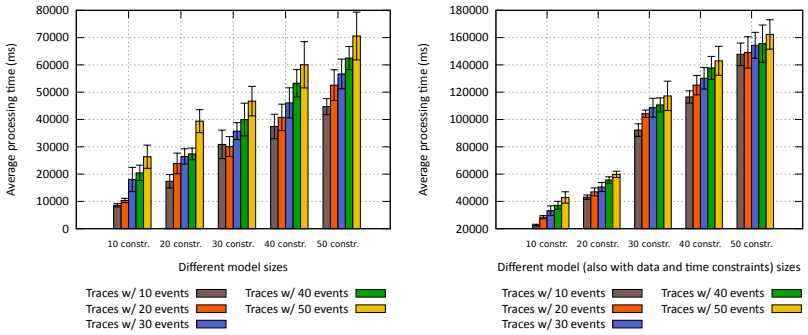


Figure 6: Execution times in milliseconds grouped based on the number of events in each trace. The plot on the left hand side refers to models with control flow constraints. The plot on the right hand side refers to models with control flow, data and time constraints. Note the different scales between the two graphs.

directly depends both on the number of events in each trace, and on the actual size of the log. Moreover, the evaluation using models with data and time constraints require a higher execution time. This is due to the execution time required by the data validation engine for the evaluation of the data conditions. Indeed, in order to process the activation and correlation conditions (expressed in the algorithms with the *verify* function calls), we use the external tool JEval.[6] To use the tool's APIs, it is necessary to declare a set of variables and their corresponding values. Then, JEval parses the given condition (which is provided as a string), replaces the variables with the proper values, and computes the actual boolean result. This procedure requires additional execution time.

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 shows the applicability of the proposed approach to very large datasets.

Finally, we compared our approach with the approaches reported in (van der Aalst et al., 2005), (Burattin et al., 2012), and (de Leoni et al., 2014a), when using control flow-based constraints. In particular, we executed tests using models with 10, 20, 30, and 40 constraints. We used logs with 10, 20, 30, and 40 events per trace and 25 000, 50 000, 75 000, and 100 000 traces. We repeated each test five times. Figure 7 reports the average execution time for each configuration. For these benchmarks we used a Windows 7 system equipped with an Intel Core i7-2620M CPU @ 2.70GHz and 8GM RAM. As the tables show, the execution times needed using the approach described in this paper are comparable with the ones needed using the approach described in (van der Aalst et al., 2005) for small models and small logs containing short traces. However, for larger models

[6] See `http://jeval.sourceforge.net/` for more information.

Activities per trace	Number of traces															
	25k	50k	75k	100k	25k	50k	75k	100k	25k	50k	75k	100k	25k	50k	75k	100k
10	8231	11194	16704	23874	10262	17363	28155	39122	14034	25122	37643	54157	16339	33228	51415	67922
20	9872	16788	20548	27484	14246	24742	33637	45087	16611	32008	43771	63255	18586	35396	56069	77541
30	10783	14889	24595	37000	14907	25250	41022	55255	18043	31977	52428	67898	21466	39727	68244	87763
40	10449	16857	32963	48968	16199	25128	46579	71514	17759	38039	67557	85610	23756	45608	74128	98059
	10 constraints				20 constraints				30 constraints				40 constraints			

(a) Average execution times needed using the approach described in this paper.

Activities per trace	Number of traces															
	25k	50k	75k	100k	25k	50k	75k	100k	25k	50k	75k	100k	25k	50k	75k	100k
10	6845	10798	15126	19490	10084	17984	26395	35337	13809	24601	38257	52915	16829	33793	51283	71036
20	9747	18636	25291	33418	15663	28764	41792	66503	21831	42179	60943	90817	26844	53823	85613	118342
30	11594	21921	36361	48432	20979	40710	64138	89395	28551	59427	90218	128791	38491	75975	121646	166047
40	16848	31921	49240	72168	28613	54516	81432	117665	38301	80768	123190	162680	49430	104833	156434	214903
	10 constraints				20 constraints				30 constraints				40 constraints			

(b) Average execution times needed using the approach described in (van der Aalst et al., 2005).

Activities per trace	Number of traces															
	25k	50k	75k	100k	25k	50k	75k	100k	25k	50k	75k	100k	25k	50k	75k	100k
10	39019	71676	109400	140578	70774	142007	216459	283474	104286	205496	308463	417423	149836	297285	432670	591135
20	48688	92418	139692	182761	91086	180027	268252	357053	131686	267219	399685	530822	180010	364182	544565	1139182
30	55074	110742	165741	223938	112676	211318	326705	440152	162559	322656	484683	873701	221423	438969	663939	5081806
40	66481	133349	194523	262991	125094	254571	386999	512352	190167	377549	564961	3320700	260348	514649	2185486	> 10M
	10 constraints				20 constraints				30 constraints				40 constraints			

(c) Average execution times needed using the approach described in (Burattin et al., 2012).

Figure 7: Execution times needed to check control flow-based constraints using three different conformance checking approaches available in the literature.

and logs, our approach outperforms the one described in (van der Aalst et al., 2005). To check the conformance of a log containing 100 000 traces of length 40 against a model containing 40 constraints, the former requires about 1 minute and 30 seconds, the latter about 3 minutes and 30 seconds. In addition, our approach outperforms the approach described in (Burattin et al., 2012): using the smallest logs, our approach is more than 5 times faster, and to process the largest log, the approach described in (Burattin et al., 2012) requires more than 3 hours of processing (compared to 1 minute and 30 seconds required by our approach). We do not report benchmarks for the approach described in (de Leoni et al., 2014a), since it takes 26 minutes to check the conformance of 500 traces against 20 constraints (the smallest log considered in our tests contains 25 000 traces).

7.4. Application to a Dutch Financial Institution

Through the proposed approach, it is possible to effectively detect violations of multi-perspective constraints (**RQ1**). The log we used to support this claim is a real-life log provided for the BPI challenge 2012 and taken from a Dutch financial institute (3TU Data Center, 2012). The event log pertains to an application process for personal loans or overdrafts.[7] It contains 262 200 events distributed across 36 event classes and includes 13 087 traces. The amount requested by

[7] See (Bautista et al., 2012) for the English translation and a detailed description of the event names.

the customer is indicated in the case attribute AMOUNT_REQ. In addition, the log contains the standard XES attributes for events (e.g., activity name, timestamp, resource).

We have evaluated the conformance of this log with respect to the constraints shown in Table 4. These constraints have not been provided by the financial institute, but they are realistic and derived by the analysis reports submitted

Table 4: Reference constraints used to analyze the log from the BPI challenge 2012.

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

Table 5: Conformance checking results using the log from the BPI challenge 2012.

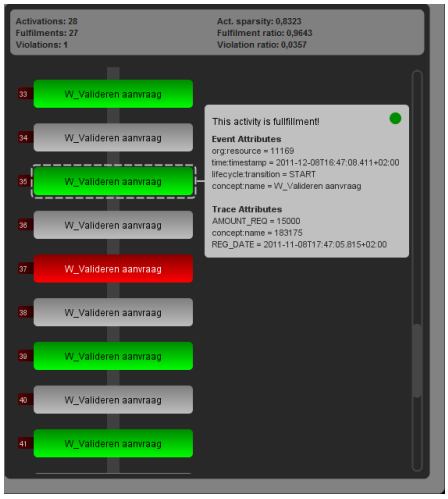
Id	Act.no.	Viol.no.	Fulfill.no.	Avg.act.sparsity	Avg.viol.ratio	Avg.fulfill.ratio
1	13 087	7 974	5 113	0.8596	0.6093	0.3907
2	13 087	9 036	4 051	0.8596	0.6905	0.3095
3	6 847	3 601	3 246	0.9585	0.5259	0.4741
4	6 240	4 373	1 867	0.9211	0.7008	0.2992
5	5 023	51	4 972	0.9909	0.0102	0.9898
6	5 023	236	4 787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

Table 6: Execution times using the log from the BPI challenge 2012.

Id	Avg.execution time (milliseconds)
1	2 772
2	3 220
3	3 261
4	3 205
5	3 196
6	3 100
7	3 212
8	3 146
9	2 176
10	3 210
11	3 241
12	3 258

to the BPI challenge. Some of the considered constraints involve some specific transactional states (a.k.a. event types) of an activity. For example, the parameters specified for constraint 5-8 are `W_Valideren aanvraag-SCHEDULE` and `W_Valideren aanvraag-START`. When an event type is not specified, like in the case of constraint 1-4, the event type considered by default is "complete". These examples show that constraints can not only be used to identify improper process executions, but also to highlight (positive or negative) deviances with respect to some properties of interest.

For example, with constraint 1, we want to understand how many submitted applications are eventually accepted. As shown in Table 5, there are 13 087 submissions of which only 5 113 are eventually accepted (around 39%). Using



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

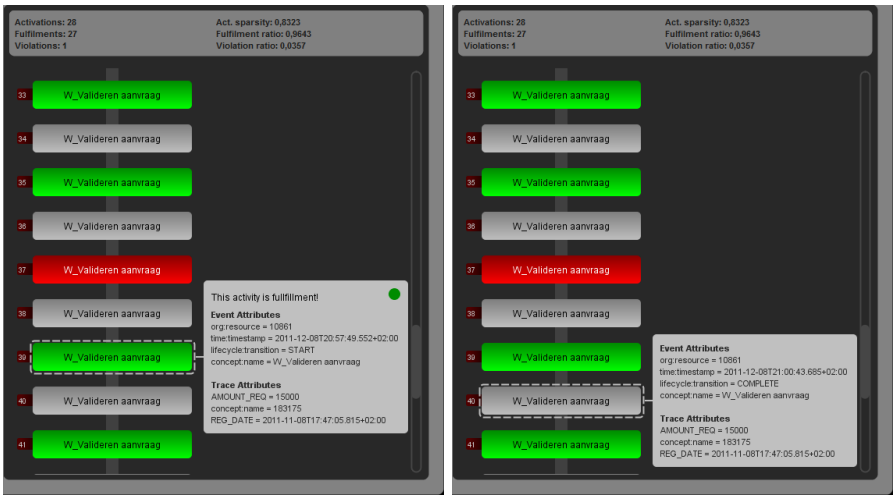
(b) A correlated target `W_Valideren aanvraag-COMPLETE` at position 36 executed by the same resource.

Figure 8: Example of fulfillment for constraint 11.

(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 9: Example of violation for constraint 11; `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 10: Example of fulfillment for constraint 11; `W_Valideren aanvraag-START` at position 39 is followed by `W_Valideren aanvraag-COMPLETE` within the required time interval.

constraint 2, we can understand that the majority of these accepted applications (around 79%) are accepted in less than 24 hours from the submission. Using

constraints 3 and 4, we can understand how the requested amount affects the application. In 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 5-12, we analyze the validation of the applications. With constraint 5, 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 6). In addition, in around 94% of the cases, a scheduled validation is started within 7 days from the scheduling (constraint 7) and in almost half of the cases the validation is started only 24 hours after the scheduling (constraint 8). Constraint 9 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 completes this activity (see constraint 10). In 97% of the cases, the validation is done in at most 1 hour (constraint 11), and in more than half of the cases it is completed in less than 15 minutes (constraint 12).

In Figure 8 and 10, we show two fulfillments for constraint 11 (the activations with the correlated targets). Figure 10 shows a violation for the same constraint. This type of violations could not be identified by control flow-based conformance checking approaches. The above remarks provide an answer to research questions **RQ1.1** and **RQ1.2**.

In Table 6, we show the execution times (averaged over 5 runs) needed for checking the constraints.[8] For each of them, the execution time is low (between 2 and 3 seconds on average). This confirms the applicability of our tool to real-life logs.

8. Conclusion and Future Work

In this work, we propose an approach 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. In particular, we describe and discuss in detail an algorithmic framework 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. An evaluation of the tool has been carried out using both real-life and synthetic logs. The evaluation shows the applicability of our implementation to real-life settings. In

[8] 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.

particular, it shows that the tool provides more fine-grained feedback than control flow-based conformance checking approaches and allows the user to identify violations that cannot be identified by taking into consideration only the control flow perspective. In addition, the provided benchmarks show the applicability of the tool to large logs and models.

The current graphical syntax for MP-Declare is similar to the one used in (Maggi et al., 2013a). However, an evaluation of the usability and effectiveness of the graphical notation would require a cognitive dimensions-based experimentation, which is out of the scope of this paper and is an avenue for future work. In addition, 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 approach suitable to be used in online settings. For this type of analysis, it becomes useful to identify violations deriving from the interplay of two or more constraints for early detection of violations. To this aim, the constraints cannot only be checked independently of one another, but also as a conjunction of formulas. Also, we plan to improve both the algorithmic implementation and the visual representation of the results. For example, the current execution time can be improved by identifying implicit dependencies between constraints, thus providing, in those situations, answers in less time (e.g., if we recognize that a constraint does not hold, we could immediately state that stronger constraints cannot hold as well). The usability and effectiveness of the visual representation of the results needs to be validated with users. Indeed, it represents a critical and fundamental issue for the real-world deployment of our proposed approach. Such a representation should guide the final users towards an easy and as-informative-as-possible identification of problems in the execution of a business process.

References

3TU Data Center (2011). BPI Challenge 2011 Event Log. http://dx.doi.org/10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54. URL: http://dx.doi.org/10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54. doi:10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54.

3TU Data Center (2012). BPI Challenge 2012 Event Log. http://dx.doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f. URL: http://dx.doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f. doi:10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f.

3TU Data Center (2015). Road Traffic Fine Management Process. Doi:10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5.

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

van der Aalst, W. M. P. (2011). *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. (1st ed.). Springer Publishing Company, Incorporated.

van der Aalst, W. M. P. (2012). Decomposing process mining problems using passages. In *Application and Theory of Petri Nets - 33rd International Conference, Petri Nets 2012* (pp. 72–91).

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

van der Aalst, W. M. P., de Beer, H. T., & van Dongen, B. F. (2005). Process mining and verification of properties: An approach based on temporal logic. In *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE, OTM Confederated International Conferences CoopIS, DOA, and ODBASE 2005, Agia Napa, Cyprus, October 31 - November 4, 2005, Proceedings, Part I* (pp. 130–147). URL: http://dx.doi.org/10.1007/11575771_11. doi:10.1007/11575771_11.

Adriansyah, A., van Dongen, B. F., & van der Aalst, W. M. P. (2011). Conformance checking using cost-based fitness analysis. In *Proceedings of the 15th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2011* (pp. 55–64).

Awad, A., Weidlich, M., & Weske, M. (2009a). Specification, verification and explanation of violation for data aware compliance rules. In *Service-Oriented Computing, 7th International Joint Conference, ICSOC-ServiceWave 2009, Stockholm, Sweden, November 24-27, 2009. Proceedings* (pp. 500–515). URL: http://dx.doi.org/10.1007/978-3-642-10383-4_37. doi:10.1007/978-3-642-10383-4_37.

Awad, A., Weidlich, M., & Weske, M. (2009b). Specification, Verification and Explanation of Violation for Data Aware Compliance Rules. In *Proceedings of the 7th International Joint Conference on Service-Oriented Computing (ICSOC-ServiceWave 2009)* (pp. 500–515). volume 5900 of *LNCS*.

Bautista, A. D., Wangikar, L., & Akbar, S. M. K. (2012). *Process Mining-Driven Optimization of a Consumer Loan Approvals Process – The BPIC 2012 Challenge*. Technical Report CKM Advisors 711 Third Avenue, Suite 1806, New York, NY, USA. URL: http://www.win.tue.nl/bpi/_media/2012/bautista.pdf.

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

Burattin, A. (2015). Artificial datasets for multi-perspective Declare analysis. http://dx.doi.org/10.5281/zenodo.20030. URL: http://dx.doi.org/10.5281/zenodo.20030. doi:10.5281/zenodo.20030.

Burattin, A., Maggi, F. M., van der Aalst, W. M. P., & Sperduti, A. (2012). Techniques for a Posteriori Analysis of Declarative Processes. In *2012 IEEE 16th International Enterprise Distributed Object Computing Conference* (pp. 41–50). IEEE.

Chesani, F., Mello, P., Montali, M., Riguzzi, F., Sebastianis, M., & Storari, S. (2009). Checking Compliance of Execution Traces to Business Rules. In *Business Process Management Workshops* (pp. 134–145).

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

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

De Masellis, R., Maggi, F. M., & Montali, M. (2014). Monitoring data-aware business constraints with finite state automata. In *International Conference on Software and Systems Process 2014, ICSSP* (pp. 134–143).

Desel, J., & Esparza, J. (1995). *Free Choice Petri Nets*. New York, NY, USA: Cambridge University Press.

Giblin, C., Müller, S., & Pfitzmann, B. (2006). *From regulatory policies to event monitoring rules: Towards model-driven compliance automation*. Technical Report Report RZ 3662.

Grando, M. A., van der Aalst, W. M. P., & Mans, R. S. (2012). Reusing a Declarative Specification to Check the Conformance of Different CIGs. In *Business Process Management Workshops* (pp. 188–199). Springer Berlin Heidelberg.

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

Haisjackl, C., Zugal, S., Soffer, P., Hadar, I., Reichert, M., Pinggera, J., & Weber, B. (2013). Making sense of declarative process models: Common strategies and typical pitfalls. In *Enterprise, Business-Process and Information Systems Modeling - 14th International Conference, BPMDS 2013, 18th International Conference, EMMSAD 2013, Held at CAiSE 2013, Valencia, Spain, June 17-18, 2013. Proceedings* (pp. 2–17).

Hallé, S., & Villemaire, R. (2008). Runtime monitoring of message-based workflows with data. In *12th International IEEE Enterprise Distributed Object Computing Conference, ECOC 2008, 15-19 September 2008, Munich, Germany* (pp. 63–72).

IEEE Task Force on Process Mining (2013). *XES Standard Definition*. Technical Report http://www.xes-standard.org/xesstandarddefinition.

Knuplesch, D., Ly, L. T., Rinderle-ma, S., Pfeifer, H., & Dadam, P. (2010). On Enabling Data-Aware Compliance Checking of Business Process Models. In *Proceedings of the 29th international conference on Conceptual modeling* (pp. 332–346). Springer.

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

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

de Leoni, M., & van der Aalst, W. M. (2013). 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* (pp. 113–129). Springer Berlin Heidelberg.

de Leoni, M., Maggi, F. M., & van der Aalst, W. M. (2014a). An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data. *Information Systems*, *47*, 258–277.

de Leoni, M., Maggi, F. M., & van der Aalst, W. M. P. (2012). Aligning Event Logs and Declarative Process Models for Conformance Checking. In *Business Process Management* (pp. 82–97). Springer Berlin / Heidelberg.

de Leoni, M., Munoz-Gama, J., Carmona, J., & van der Aalst, W. M. P. (2014b). 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* (pp. 3–20).

Ly, L. T., Maggi, F. M., Montali, M., Rinderle-Ma, S., & van der Aalst, W. M. P. (2015). Compliance monitoring in business processes: Functionalities, application, and tool-support. *Inf. Syst.*, *54*, 209–234.

Ly, L. T., Rinderle-Ma, S., Knuplesch, D., & Dadam, P. (2011). Monitoring business process compliance using compliance rule graphs. In *On the Move to Meaningful Internet Systems: OTM 2011 - Confederated International Conferences: CoopIS, DOA-SVI, and ODBASE 2011, Hersonissos, Crete, Greece, October 17-21, 2011, Proceedings, Part I* (pp. 82–99). URL: http://dx.doi.org/10.1007/978-3-642-25109-2_7. doi:10.1007/978-3-642-25109-2_7.

Maggi, F. M. (2013). Declarative Process Mining with the Declare Component of ProM. In *BPM (Demos)*. ceur-ws.org volume 1021.

Maggi, F. M., Dumas, M., García-Bañuelos, L., & Montali, M. (2013a). Discovering data-aware declarative process models from event logs. In *Business Process Management - 11th International Conference, BPM 2013, Beijing, China, August 26-30, 2013. Proceedings* (pp. 81–96).

Maggi, F. M., Mooij, A. J., & van der Aalst, W. M. P. (2013b). Analyzing vessel behavior using process mining. In *Situation Awareness with Systems of Systems* (pp. 133–148). URL: http://dx.doi.org/10.1007/978-1-4614-6230-9_9. doi:10.1007/978-1-4614-6230-9_9.

Maggi, F. M., & Westergaard, M. (2014). Using timed automata for *a Priori* warnings and planning for timed declarative process models. *Int. J. Cooperative Inf. Syst.*, *23*. URL: http://dx.doi.org/10.1142/S0218843014400036. doi:10.1142/S0218843014400036.

Mannhardt, F., de Leoni, M., Reijers, H. A., & van der Aalst, W. M. (2014). *Balanced Multi-Perspective Checking of Process Conformance*. Technical Report BPM-14-07 BPM Center.

Middleton, G., Peyton, L., Kuziemsky, C., & Eze, B. (2009). A framework for continuous compliance monitoring of eHealth processes. In *World Congress on Privacy, Security, Trust and the Management of e-Business, 2009. CONGRESS '09* (pp. 152–160).

Montali, M., Pesic, M., van der Aalst, W. M. P., Chesani, F., Mello, P., & Storari, S. (2010). Declarative Specification and Verification of Service Choreographies. *ACM Transactions on the Web*, *4*, 1–62.

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

Narendra, N., Varshney, V., Nagar, S., Vasa, M., & Bhamidipaty, A. (2008). Optimal control point selection for continuous business process compliance monitoring. In *IEEE International Conference on Service Operations and Logistics, and Informatics, 2008. IEEE/SOLI 2008* (pp. 2536–2541). volume 2.

Object Management Group (2011). *Business Process Model and Notation (BPMN) Version 2.0*. Technical Report. URL: http://taval.de/publications/BPMN20.

Pesic, M., Schonenberg, H., & van der Aalst, W. (2007). DECLARE: Full Support for Loosely-Structured Processes. In *EDOC 2007* (pp. 287–298).

Pichler, P., Weber, B., Zugal, S., Pinggera, J., Mendling, J., & Reijers, H. A. (2011). Imperative versus declarative process modeling languages: An empirical investigation. In *BPM Workshops* (pp. 383–394).

Pnueli, A. (1977). The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)* (pp. 46–57). IEEE.

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

Silva, N. C., de Oliveira, C. A. L., Albino, F. A. L. A., & Lima, R. M. F. (2014). Declarative versus imperative business process languages - A controlled experiment. In *ICEIS 2014 - Proceedings of the 16th International Conference on Enterprise Information Systems, Volume 3, Lisbon, Portugal, 27-30 April, 2014* (pp. 394–401). URL: http://dx.doi.org/10.5220/0004896203940401. doi:10.5220/0004896203940401.

Taghiabadi, E. R., Fahland, D., Van Dongen, B. F., & van der Aalst, W. M. P. (2013). Diagnostic information for compliance checking of temporal compliance requirements. In *25th International Conference, CAiSE 2013* (pp. 304–320). Springer Berlin Heidelberg. doi:10.1007/978-3-642-38709-8_20.

Taghiabadi, E. R., Gromov, V., Fahland, D., & van der Aalst, W. M. P. (2014). Compliance Checking of Data-Aware and Resource-Aware Compliance Requirements. In *Confederated International Conferences: CoopIS, and ODBASE 2014* (pp. 237–257). Springer Berlin Heidelberg. doi:10.1007/978-3-662-45563-0_14.

Verbeek, H. M. W., Buijs, J. C. A. M., van Dongen, B. F., & van der Aalst, W. M. P. (2010). XES, XESame, and ProM 6. In *Information Systems Evolution - CAiSE Forum* (pp. 60–75). volume 72.

Westergaard, M., & Maggi, F. M. (2011). Declare: A tool suite for declarative workflow modeling and enactment. In *BPM (Demos)*. ceur-ws.org volume 820.

Westergaard, M., & Maggi, F. M. (2012). Looking into the future: Using timed automata to provide a priori advice about timed declarative process models. In *Proc. of CoopIS* LNCS. Springer.

Zugal, S., Pinggera, J., & Weber, B. (2011). The impact of testcases on the maintainability of declarative process models. In *BMMDS/EMMSAD* (pp. 163–177).