Discovering Metric Temporal Business Constraints from Event Logs

Fabrizio Maria Maggi

University of Tartu, Estonia
f.m.maggi@ut.ee

Abstract. Process discovery aims at building process models using information retrieved from logs. Process characteristics play a significant role in the selection of a suitable process modeling language for describing process discovery results. Business processes characterized by high variability, in which participants have a lot of autonomy and flexibility in executing the process, are difficult to be described with procedural process modeling languages, since they explicitly represent in a model every possible path. Declarative languages, like Declare, alleviate this issue by defining a set of constraints between activities that must not be violated during the process execution instead of describing what to do step by step. Recently, several process discovery techniques have been proposed for extracting a set of Declare constraints from a log. However, no one of these techniques allows the user to exploit the time perspective often available in a log to discover "time-aware" Declare constraints. *Timed Declare* has already previously been introduced to monitor metric temporal constraints at runtime. In this paper, we use this semantics for discovering a set of Timed Declare constraints from an event log. We have implemented the proposed approach as a plug-in of the process mining tool ProM. We have validated the approach by using our plug-in to mine two real-life event logs.

Keywords: Process Discovery, Metric Temporal Logic, Event Correlations, Timed Declare.

1 Introduction

Process discovery is one of the three branches of the family of process mining techniques together with *conformance checking* and *process enhancement*. Through process discovery, it is possible to build from scratch a process model describing the behavior of a business process as recorded in an event log. Recently, several works have investigated advantages and disadvantages of using procedural or declarative process modeling languages to describe the results of a process discovery technique [13,14]. The results of these studies highlighted that the dichotomy *procedural versus declarative* reflects the nature of the process under examination. Procedural models like Petri nets, BPMN, and EPCs are more suitable to support business processes working in stable environments, in which participants have to follow predefined procedures, since they suggest step by step what to do next. In contrast, declarative process modeling languages, like Declare, provide process participants with a (preferably small) set of rules to be followed during the process execution. In this way, process participants have the flexibility to follow any path that does not violate these rules.

Declare is a declarative language introduced in [2] that combines a formal semantics grounded in Linear Temporal Logic (LTL) with a graphical representation for users.[1] A *Declare map* is a set of Declare constraints each one with its own graphical representation and LTL semantics (see [2] for a full overview of Declare). Recently, several process discovery techniques have been proposed for describing with a set of Declare constraints the behavior of a process as recorded in an event log [6,7,11,10]. However, no one of these techniques allows the user to extract "time-aware" Declare constraints from a log.

During the execution of a business process it is often extremely important to meet deadlines and optimize response times. To this aim, a Declare map can also include metric temporal constraints to guarantee the correct execution of a process in terms of *latencies* (related to events that cannot occur *before* a certain time, or must occur *after* a certain time) and *deadlines* (related to events that cannot occur *after* a certain time, or must occur *before* a certain time).

Timed Declare has already previously been introduced to monitor metric temporal constraints at runtime [16]. In this paper, we introduce an approach for discovering a set of Timed Declare constraints that are satisfied in a given log. When evaluating the satisfaction of a Timed Declare constraint, one often faces ambiguities in connecting events that "activate" the constraint (activations) and events that "fulfill" it (target events), since the activation of a constraint may potentially be associated to multiple target events. For example, consider traces $\mathbf{T}_1 = \langle a, b, c, b\rangle$ and $\mathbf{T}_2 = \langle a, a, b, b\rangle$ and the *response* constraint $\mathbf{G}(a \rightarrow \mathbf{F}b)$, meaning that if *a occurs*, then eventually *b follows* after a. It is unclear whether to associate the activation a of the constraint at $\mathbf{T}_1(1)$ with the occurrence of the target b at $\mathbf{T}_1(2)$ or $\mathbf{T}_1(4)$. We face similar ambiguity for the two activations of a in $\mathbf{T}_2$. If we want to discover Timed Declare constraints, we need to evaluate the time difference between the occurrence of a and its corresponding target b. A conservative approach, where we associate an occurrence of a with the closest occurrence of b could negatively affect the discovery results and lead to incorrect conclusions. Our proposed technique allows the user to guide the discovery task through event correlations to correctly evaluate the metric temporal constraints and to improve the quality of the discovered models.

We have implemented our proposed approach in a plug-in of the process mining tool ProM.[2] We have validated our discovery technique by using two real-life logs provided for the 2011 and 2012 BPI challenges [1,8] pertaining to the treatment of patients diagnosed with cancer in a large Dutch academic hospital and to a financial process in a Dutch financial institute.

The remainder of this paper is organized as follows. Section 2 presents some preliminaries about Declare, Timed Declare and event correlations. Section 3

[1] In the remainder, LTL refers to the version of LTL tailored towards finite traces.
[2] `www.processmining.org`

presents our algorithms for discovering Timed Declare constraints. Section 4 presents the validation of the approach. Finally, Section 5 concludes the paper.

2 Preliminaries

In this section, we introduce some preliminary notions. In particular, in Section 2.1, we give an overview of the Declare language. In Section 2.2 we introduce Timed Declare. In Section 2.3, we give a categorization of the correlation mechanisms used in this paper.

2.1 Declare

Declare is a declarative process modeling language introduced by Pesic and van der Aalst in [2]. A *Declare map* consists of a set of constraints which, in turn, are based on templates. Templates are parameterized classes of rules and constraints are their concrete instantiations. Here, we indicate template parameters with capital letters (see Tables 1 and 2) and real activities in their instantiations with lower case letters (e.g., constraint $\mathbf{G}(a \rightarrow \mathbf{F}b)$). Templates have a user-friendly graphical representation understandable to the user and their semantics are specified through LTL formulas. Each constraint inherits the graphical representation and semantics from its template.

For the sake of readability, in Tables 1 and 2, we use PLTL to define the semantics of Declare constraints, i.e., LTL augmented with past operators. The LTL rules used in this paper are constructed from propositional atoms by applying the future temporal operators $\mathbf{X}$ (next), $\mathbf{F}$ (future), $\mathbf{G}$ (globally), and $\mathbf{U}$ (until) in addition to the usual boolean connectives. Given a formula φ, $\mathbf{X}\varphi$ means that the next time instant exists and φ is true in the next time instant (strong next). $\mathbf{F}\varphi$ indicates that φ is true sometimes in the future. $\mathbf{G}\varphi$ means that φ is true always in the future. $\varphi\mathbf{U}\psi$ indicates that φ has to hold at least until ψ holds and ψ must hold in the current or in a future time instant.

PLTL extends LTL by introducing past operators. The past operators we use in this work are $\mathbf{Y}$ (yesterday), $\mathbf{O}$ (once), and $\mathbf{S}$ (since), which correspond to the future operators $\mathbf{X}$, $\mathbf{F}$, and $\mathbf{U}$ respectively. At any non-initial time, $\mathbf{Y}\varphi$ is true if and only if φ holds at the previous time instant. $\mathbf{O}\varphi$ indicates that φ is true at some past time instant (including the present time). $\psi\mathbf{S}\varphi$ is true if ψ holds somewhere in the past and φ is true from then up to now.

Tables 1 and 2 show the PLTL semantics of the Declare constraints used in this paper. 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 $\mathbf{T}_1 = \langle a, b, c, b \rangle$, $\mathbf{T}_2 = \langle a, a, b, b \rangle$, and $\mathbf{T}_3 = \langle b, b, c, d \rangle$, but not for $\mathbf{T}_4 = \langle a, b, a, c \rangle$ because, in this case, the second a is not followed by a b. Note that, in $\mathbf{T}_3$, the *response* constraint is satisfied in a trivial way because a never occurs. In this case, we say that the constraint is *vacuously satisfied* [9]. In [5], the authors introduce the notion of *behavioral vacuity detection* according to which a constraint is non-vacuously satisfied in a

Table 1. Semantics and graphical notation for positive relation constraints

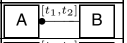
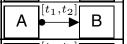
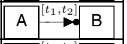
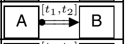
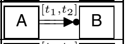
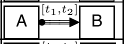
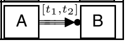
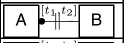
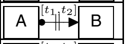
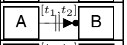
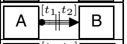
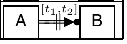
constraint	untimed semantics	timed semantics	notation
responded existence	$\mathbf{F}A \rightarrow \mathbf{F}B$	$\mathbf{G}(A \rightarrow (\mathbf{O}_{[t_1,t_2]}B \vee \mathbf{F}_{[t_1,t_2]}B))$	
response	$\mathbf{G}(A \rightarrow \mathbf{F}B)$	$\mathbf{G}(A \rightarrow \mathbf{F}_{[t_1,t_2]}B)$	
precedence	$\mathbf{G}(B \rightarrow \mathbf{O}A)$	$\mathbf{G}(B \rightarrow \mathbf{O}_{[t_1,t_2]}A)$	
alternate response	$\mathbf{G}(A \rightarrow \mathbf{X}(\neg A\mathbf{U}B))$	$\mathbf{G}(A \rightarrow \mathbf{X}(\neg A\mathbf{U}_{[t_1,t_2]}B))$	
alternate precedence	$\mathbf{G}(B \rightarrow \mathbf{Y}(\neg B\mathbf{S}A))$	$\mathbf{G}(B \rightarrow \mathbf{Y}(\neg B\mathbf{S}_{[t_1,t_2]}A))$	
chain response	$\mathbf{G}(A \rightarrow \mathbf{X}B)$	$\mathbf{G}(A \rightarrow \mathbf{X}_{[t_1,t_2]}B)$	
chain precedence	$\mathbf{G}(B \rightarrow \mathbf{Y}A)$	$\mathbf{G}(B \rightarrow \mathbf{Y}_{[t_1,t_2]}A)$	

Table 2. Semantics and graphical notation for negative relation constraints

constraint	untimed semantics	timed semantics	notation
not responded existence	$\mathbf{F}A \rightarrow \neg\mathbf{F}B$	$\mathbf{G}(A \rightarrow (\neg\mathbf{O}_{[t_1,t_2]}B \wedge \mathbf{F}_{[t_1,t_2]}B))$	
not response	$\mathbf{G}(A \rightarrow \neg(\mathbf{F}B))$	$\mathbf{G}(A \rightarrow \neg(\mathbf{F}_{[t_1,t_2]}B))$	
not precedence	$\mathbf{G}(B \rightarrow \neg(\mathbf{O}A))$	$\mathbf{G}(B \rightarrow \neg(\mathbf{O}_{[t_1,t_2]}A))$	
not chain response	$\mathbf{G}(A \rightarrow \neg(\mathbf{X}B))$	$\mathbf{G}(A \rightarrow \neg(\mathbf{X}_{[t_1,t_2]}B))$	
not chain precedence	$\mathbf{G}(B \rightarrow \neg(\mathbf{Y}A))$	$\mathbf{G}(B \rightarrow \neg(\mathbf{Y}_{[t_1,t_2]}A))$	

trace if the trace contains at least one activation of the constraint. An *activation* of a constraint in a trace is an event whose occurrence imposes, because of that constraint, some obligations on other events in the same trace. For example, a is an activation for the *response* constraint $\mathbf{G}(a \rightarrow \mathbf{F}b)$, because the execution of a forces b to be executed eventually.

In [7,11,10], algorithms for the discovery of Declare maps from event logs have been presented. In these works different notions of constraint support have been proposed. In this paper, we assume that the support of a constraint in a log is the percentage of traces in the given log in which the constraint is *non-vacuously satisfied.*

2.2 Timed Declare

We use Metric Temporal Logic (MTL) to define the semantics of Timed Declare constraints. We deal with a fragment of MTL where all traces are finite [15]. In Tables 1 and 2, we use the MTL future operators $\mathbf{X}_{[t_1,t_2]}$, $\mathbf{F}_{[t_1,t_2]}$, and $\mathbf{U}_{[t_1,t_2]}$. In addition we use the MTL past operators $\mathbf{Y}_{[t_1,t_2]}$, $\mathbf{O}_{[t_1,t_2]}$, and $\mathbf{S}_{[t_1,t_2]}$. Given a formula φ and the current time instant t, $\mathbf{X}_{[t_1,t_2]}\varphi$ means that the next time instant exists and falls into the interval $[t+t_1, t+t_2]$, and φ is true in the next time instant. $\mathbf{F}_{[t_1,t_2]}\varphi$ indicates that φ is true sometimes in the future in a time instant belonging to the interval $[t+t_1, t+t_2]$. $\varphi\mathbf{U}_{[t_1,t_2]}\psi$ indicates that φ has to hold at least until ψ holds and ψ must hold in a time instant belonging to the interval $[t+t_1, t+t_2]$. $\mathbf{Y}_{[t_1,t_2]}\varphi$ is true if φ holds at the previous time instant and this instant belongs to $[t-t_2, t-t_1]$. $\mathbf{O}_{[t_1,t_2]}\varphi$ indicates that φ is true at some past time instant (including the present time) falling into the interval $[t - t_2, t - t_1]$. $\psi \mathbf{S}_{[t_1,t_2]} \varphi$ is true if ψ holds somewhere in the past in a time instant belonging to the interval $[t - t_2, t - t_1]$ and φ is true from then up to now.

Tables 1 and 2 show the MTL semantics of the Timed Declare constraints used in this paper and their graphical notation. Consider, for example, the *response* constraint $\mathbf{G}(a \rightarrow \mathbf{F}_{[t_1,t_2]} b)$. This constraint indicates that if a occurs at a time instant t, b must eventually follow at a time instant belonging to the interval $[t{+}t_1, t{+}t_2]$. Note that negative constraints are used to model latency constraints, i.e., constraints stating that at least a minimum amount of time is needed to accomplish a certain operation [12]. For example, the *not response* constraint $\mathbf{G}(a \rightarrow \neg(\mathbf{F}_{[0,t_1]} b))$ indicates that if a occurs at a time instant t, it takes at least t_1 time units to execute b (b cannot be executed in the time interval $[t, t + t_1]$).

2.3 Event Correlations

We use the term event correlation to indicate a mechanism to link two events in a trace. Correlations are defined over event attributes and linked through *relationship operators* between them. For example, two events are correlated if they act upon common data elements of the process or if they are executed by the same resource etc. For a categorization of correlations we refer to [3] in which the following types of correlations are defined: (i) *Property-based correlation*, i.e., events are classified based on a function operating on their attributes. For example, all claim applications referring to an amount greater than 1000 euros are grouped together; (ii) *Reference-based correlation*, i.e., two events are correlated if an attribute of the first event (identifier attribute) and an attribute of the second event (reference attribute) have the same value; (iii) *Moving time-window correlation*, i.e., two events are correlated if they occur within a given duration of one another (e.g., one hour).

In this paper, we use a variation of reference-based correlation according to which two events are correlated if there is a function connecting an attribute of the first event with an attribute of the second event. This function can include operators such as *greater than*, *less than*, *equal to*, and *not equal to*. For example, an event of producing a document is correlated to an event of checking it if the resource that produces the document is different from the one that checks it.

To correctly associate an activation and a target of a given constraint, correlations can be provided by a domain expert or, alternatively, they can be automatically discovered from event logs (as presented in [4]). For discovering event correlations from a log, in [4], the authors first generate all feasible correlations for the constraint under examination, i.e., correlations between comparable attributes of activations and targets of that constraint (comparable attributes are attributes having the same data type). The "goodness" of a (feasible) correlation is then evaluated based on its support.

In this paper, we define the support of a correlation (for positive relation constraints) as the ratio between the number of activations that, through the given correlation, can be linked to *at least* one target and the total number of activations. For negative relation constraints, we define the support of a correlation as the ratio between the number of activations for which there is *no* correlated target and the total number of activations. In this way, for negative relation constraints, the support of a correlation is higher if it allows us to decouple activations and possible targets instead of connecting them. Note that, if we define the support of a correlation in this way, the support of a correlation evaluated for a negative relation constraint can be derived from the support of the same correlation evaluated for the corresponding positive relation constraint. For example, if $support_{response(a,b)}(corr)$ is the support in a log of a correlation *corr* for a response constraint between activities a and b, we have that $support_{not\ response(a,b)}(corr) = 1 - support_{response(a,b)}(corr)$.

3 Discovering Timed Declare Constraints from Logs

In this section, we describe the proposed algorithms for the discovery of Timed Declare constraints from event logs. Each of them can be used to discover constraints referring to different Declare templates. Every algorithm takes as input a log to be processed. Moreover, the presented algorithms work on a given candidate constraint (without time information: the time intervals to be associated to each candidate are determined in a later stage). Therefore, they assume that a list of candidate constraints has been created beforehand. The list of candidate constraints can include all the possible instantiations of a template with each possible combination of event names in the given log (see [11]). This approach can be optimized and the number of candidate constraints can be reduced by using a seminal Apriori algorithm as presented in [10]. Every algorithm requires as input also a correlation to link each activation of the candidate constraint under examination with the corresponding target.

The algorithms produce as output the percentage of traces in the log in which each candidate constraint is non-vacuously satisfied (the support of the candidate constraint in the log) and a vector containing the time distances between each activation and the corresponding target. The support value is used to filter out candidate constraints that are non-vacuously satisfied in a low percentage of log traces (with respect to a user-defined threshold). The time distances are used to identify the time interval $[t_1, t_2]$ characterizing the discovered metric temporal constraint (see Section 3.4).

In the following sections, we describe the algorithms we use for the discovery of positive relation constraints. The same algorithms can be used also for the discovery of negative relation constraints taking into account that, as explained in Section 3.4, the mechanisms for the definition of the time intervals are different in the two cases.

3.1 Timed Response, Precedence, and Responded Existence

The algorithm presented in this section (Algorithm 1) can be used for the discovery of *timed response* constraints. A similar algorithm (with small modifications) is able to discover *timed precedence* and *timed responded existence* constraints.

Algorithm 1. Discovery algorithm for timed response.

```
Input: $log$, the event log to be processed; $(a, t)$, activation and target of a candidate constraint; $corr$ a
       selected correlation for linking each activation with the corresponding target
1  define vector $timeDistances$ containing the time distances between each activation and the corresponding
   targets
2  $validTracesNumber := 0$
3  foreach $trace \in log$ do
4      define vector $pendingActivations_{trace}$ containing events corresponding to pending activations in trace
       $trace$
5      $activated := false$
6      foreach $event \in trace$ do
7          if $event.name = a$ then
8              $activated := true$
9              $pendingActivations_{trace}$.add($event$)
10         if $event.name = t$ then
11             foreach $p \in pendingActivations_{trace}$ do
12                 if $isValid(corr, p, event)$ then
13                     $pendingActivations_{trace}$.remove($p$)
14                     $timeDistances$.add($|p.timestamp - event.timestamp|$)
15     if $pendingActivations_{trace}.size = 0$ && $activated$ then
16         $validTracesNumber++$
17 $support := validTracesNumber/log.size$
Output: $support$, the support of the candidate constraint; $timeDistances$
```

The inputs of this algorithm are an event log *log*, a candidate constraint with activation a and target t and a feasible correlation *corr*. The outputs of the algorithm are vector *timeDistances* containing the time distances between each activation of the candidate constraint and the corresponding targets, and the support of the candidate constraint *support*, i.e., the ratio between the number of traces in which the constraint is non-vacuously satisfied ($validTracesNumber$) and the total number of traces in the event log.

For each trace *trace* in *log*, $pendingActivations_{trace}$ is a vector containing the pending activations in *trace* for the candidate constraint under examination, i.e., the activations that do not have a corresponding target. For each event *event* in *trace*, *event* is a pending activation if its event name is equal to a (*event* is an activation and is not associated to any target yet). In this case *event* is added to $pendingActivations_{trace}$ and the constraint is activated in *trace* (lines 8 and 9). On the other hand, if the event name of *event* is equal to t, *event* is a possible target corresponding to one of the pending activations in $pendingActivations_{trace}$. In this case, if a pending activation p in $pendingActivations_{trace}$ can be correlated to *event* through correlation *corr* (the algorithm checks if this is the case through function $isValid$), then p is removed from the set of pending activations and the time distance between p and the corresponding target *event* is added to *timeDistances* (lines 13 and 14). The candidate constraint under examination is non-vacuously satisfied in *trace*, if, when all the events in *trace* have been processed, the boolean variable *activated* is true (*trace* contains at least one activation of the candidate constraint) and all the activations in *trace* have a corresponding target (i.e., if $pendingActivations_{trace}$ is empty).

The algorithm for the discovery of *timed precedence* constraints is similar to the one described for timed response. The difference is that, for timed precedence constraints, we iterate each trace in the event log (line 6) from the last event to the first one. In the algorithm for the discovery of *timed responded existence*

Algorithm 2. Discovery algorithm for timed alternate response.

```
Input: $log$, the event log to be processed; $(a, t)$, activation and target of a candidate constraint; $corr$ a
       selected correlation for linking each activation with the corresponding target
define vector $timeDistances$ containing the time distances between each activation and the corresponding targets
$validTracesNumber := 0$
foreach $trace \in log$ do
    define vector $pendingActivations_{trace}$ containing events corresponding to pending activations in trace $trace$
    define vector $possibleTargets_{trace}$ containing possible target events corresponding to a pending activation
    $violated := false$
    $activated := false$
    foreach $event \in trace$ do
        if $event.name = a$ then
            $activated := true$
            if $possibleTargets_{trace}.size \geqslant 1$ && $pendingActivations_{trace}.size = 1$ then
                $targetFound := false$
                $previousAct := element \in pendingActivations_{trace}$
                foreach $p \in possibleTargets_{trace}$ do
                    if $isValid(corr, previousAct, p)$ then
                        $targetFound := true$
                        $timeDistances$.add($|previousAct.timestamp - p.timestamp|$)
                if $!targetFound$ then
                    $violated := true$
            if $possibleTargets_{trace}.size = 0$ && $pendingActivations_{trace}.size = 1$ then
                $violated := true$
            if $possibleTargets_{trace}.size \geqslant 1$ then
                $pendingActivations_{trace}$.removeAll()
            $possibleTargets_{trace}$.removeAll()
            $pendingActivations_{trace}$.add($event$)
        if $event.name = t$ then
            $possibleTargets_{trace}$.add($event$)
    if $possibleTargets_{trace}.size \geqslant 1$ && $pendingActivations_{trace}.size = 1$ then
        $targetFound := false$
        $previousAct := element \in pendingActivations_{trace}$
        foreach $p \in possibleTargets_{trace}$ do
            if $isValid(corr, previousAct, p)$ then
                $targetFound := true$
                $timeDistances$.add($|previousAct.timestamp - p.timestamp|$)
        if $!targetFound$ then
            $violated := true$
        $pendingActivations_{trace}$.removeAll()
    if $pendingActivations_{trace}.size = 0$ && $activated$ && $!violated$ then
        $validTracesNumber++$
$support := validTracesNumber/log.size$
Output: $support$, the support of the candidate constraint; $timeDistances$
```

constraints, we iterate each trace in the log in both directions so that each activation can have zero, one, or two possible targets. If there are no possible target, there is a violation in the trace. If there is only one possible target we evaluate the time distance between the activation and the corresponding target. If there are two possible targets, we consider the one with the smallest time distance from the considered activation.

3.2 Timed Alternate Response and Alternate Precedence

The algorithm presented in this section (Algorithm 2) can be used for the discovery of *timed alternate response* constraints. In the same way as illustrated for timed response and timed precedence constraints, also in this case a similar algorithm is able to discover *timed alternate precedence* constraints.

The inputs of this algorithm are an event log *log*, a candidate constraint with activation a and target t and a feasible correlation *corr*. The outputs of the algorithm are vector *timeDistances* containing the time distances between each activation of the candidate constraint and the corresponding targets, and the support of the candidate constraint *support*.

For each trace *trace* in *log*, $pendingActivations_{trace}$ is a vector containing the pending activations in *trace* for the candidate constraint under examination. $possibleTargets_{trace}$ is a vector containing the possible targets for the last pending activation encountered in *trace* and added to $pendingActivations_{trace}$. For each event *event* in *trace*, if the event name of *event* is equal to a, the constraint is activated in *trace* (line 10). In this case, the algorithm checks if there is one pending activation (*previousAct*) in $pendingActivations_{trace}$ and there is at least one possible corresponding target in $possibleTargets_{trace}$ for this activation (line 11). If this is the case, the algorithm checks if *previousAct* can be correlated to one of the possible targets in $possibleTargets_{trace}$ (through function $isValid$). If *previousAct* has a correlated target, then the time distance between *previousAct* and the corresponding target is added to *timeDistances* (line 17), otherwise a violation is detected (line 19).

If *event* is an activation, there is already one pending activation in $pendingActivations_{trace}$ and there are no possible corresponding targets for this activation ($possibleTargets_{trace}$ is empty), then a violation is detected (line 21) since an alternate response constraint does not allow two activations to occur one after another without any target in between (see Table 1).

Finally, if *event* is an activation and *event* is preceded by at least one possible target, then all the elements in $pendingActivations_{trace}$ are deleted (line 23). In all cases, when *event* is an activation, all elements in $possibleTargets_{trace}$ are deleted and *event* is added to $pendingActivations_{trace}$ (lines 24 and 25). If *event* is a target of the candidate constraint under examination, *event* is added to $possibleTargets_{trace}$ (line 27).

After that the last event in *trace* has been processed, the algorithm checks again if there is still one pending activation in $pendingActivations_{trace}$ with at least one possible corresponding target in $possibleTargets_{trace}$. If this is the case, the algorithm checks if the pending activation can be correlated to one of the possible targets. The candidate constraint is non-vacuously satisfied in *trace*, if, when all the events in *trace* have been processed, the boolean variable *activated* is true (*trace* contains at least one activation of the candidate constraint), $pendingActivations_{trace}$ is empty and the boolean variable *violated* is false.

3.3 Timed Chain Response and Chain Precedence

The algorithm presented in this section (Algorithm 3) can be used for the discovery of *timed chain response* constraints. A similar algorithm is able to discover *timed chain precedence* constraints.

The inputs of this algorithm are an event log *log*, a candidate constraint with activation a and target t and a feasible correlation *corr*. The outputs of the

Algorithm 3. Discovery algorithm for timed chain response.

```
Input: $log$, the event log to be processed; $(a, t)$, activation and target of a candidate constraint; $corr$ a
       selected correlation for linking each activation with the corresponding target
1  define vector $timeDistances$ containing the time distances between each activation and the corresponding
   targets
2  $validTracesNumber := 0$
3  foreach $trace \in log$ do
4      define vector $pendingActivations_{trace}$ containing events corresponding to pending activations in trace
       $trace$
5      $violated := false$
6      $activated := false$
7      foreach $event \in trace$ do
8          if $pendingActivations_{trace}.size = 1$ then
9              $p := element \in pendingActivations_{trace}$
10             if $event.name = t$ then
11                 if $isValid(corr, p, event)$ then
12                     $timeDistances.\text{add}(|p.timestamp - event.timestamp|)$
13                 else
14                     $violated := true$
15             else
16                 $violated := true$
17             $pendingActivations_{trace}.\text{removeAll}()$
18         if $event.name = a$ then
19             $pendingActivations_{trace}.\text{add}(event)$
20             $activated := true$
21     if $pendingActivations_{trace}.size = 0$ && $activated$ && $!violated$ then
22         $validTracesNumber++$
23 $support := validTracesNumber/log.size$
Output: $support$, the support of the candidate constraint; $timeDistances$
```

algorithm are vector *timeDistances* containing the time distances between each activation of the candidate constraint and the corresponding targets, and the support of the candidate constraint *support*.

For each trace *trace* in *log*, $pendingActivations_{trace}$ is a vector containing the pending activations in *trace* for the candidate constraint under examination. For each event *event* in *trace*, if *event* is an activation, it is added to $pendingActivations_{trace}$ and the constraint is activated in *trace* (lines 19 and 20). If $pendingActivations_{trace}$ contains an element, and *event* is a target event, then the algorithm checks if the pending activation in $pendingActivations_{trace}$ can be correlated to *event* through correlation *corr* (the algorithm checks if this is the case through function $isValid$). If the two events can be correlated, the time distance between the pending activation and the corresponding target *event* is added to *timeDistances*, otherwise a violation is detected (line 14). A violation is detected also if $pendingActivations_{trace}$ contains an activation and the current event *event* is not a target event (line 16) since according to the semantics of the chain response, a target should immediately follow an activation. The candidate constraint is non-vacuously satisfied in *trace*, if, when all the events in *trace* have been processed, the boolean variable *activated* is true, $pendingActivations_{trace}$ is empty and the boolean variable *violated* is false.

3.4 Time Intervals Identification

In this section, we describe how the time distances generated by the discovery algorithm described so far can be used to identify the time interval $[t_1, t_2]$ characterizing the discovered metric temporal constraint. Suppose that d_{min} and d_{max}

are the minimum and the maximum time distances between an activation and the corresponding target of the candidate constraint under examination. Then, if the candidate constraint is a positive relation constraint, we simply derive that $t_1 = d_{min}$ and $t_2 = d_{max}$.

Also in the case of a negative relation constraint, if an activation occurs at time t all the possible targets for that activations occur at a time point belonging to the interval $[t+d_{min}, t+d_{max}]$. However, in case of a negative relation constraint, the constraint is valid outside this interval. Therefore, we can associate two intervals to the candidate constraint (thus discovering two constraints from it), i.e., $[0, d_{min}[$ and $]d_{max}, \infty[$.

To deal with cases in which logs contain noise, it is also possible to choose the boundaries of the time intervals by removing outliers from the discovered set of time distances. This can easily be done by removing time distances that deviate more than a given threshold from the average.

4 Validation

The discovery algorithms presented in this paper have been implemented in a ProM plug-in. The plug-in takes an event log and a set of correlations (one for each candidate constraint) as input and produces a Declare map consisting of a set of Timed Declare constraints. We have conducted our validation by applying the implemented plug-in to the real-life event logs provided for the BPI challenges 2011 and 2012. In Sections 4.1 and 4.2, we present the two case studies.

4.1 A Case Study Based on the BPI Challenge 2012

The first case study we discuss is based on the application of the proposed approach to the event log provided for the BPI challenge 2012 [8] and taken from a Dutch financial institute. The event log pertains to an application process for personal loans or overdrafts. It contains 262,200 events distributed across 36 event classes (i.e., each event can be associated to one of 36 different possible event names) and includes 13,087 cases. The amount requested by the customer is indicated in the case attribute *AMOUNT_REQ*. In addition, the log contains the standard XES attributes for events: *concept:name*, *lifecycle:transition*, *time:timestamp*, and *org:resource*.[3] The event log merges three intertwined sub processes. Therefore, in each case, events belonging to different sub processes can occur. These events should be correlated with each other.

Fig. 1 shows an excerpt of the table that ProM generates and that contains the correlations discovered from the log for each candidate constraint. In the table, for each candidate constraint (identified through its Declare template, its activation A and its target T), a list of correlations is specified. For each correlation, its support and degree of disambiguation is indicated. As well as the correlation support, the *degree of disambiguation* is a metric that helps the user

[3] XES (eXtensible Event Stream) is an XML-based standard for event logs proposed by the IEEE Task Force on Process Mining (`www.xes-standard.org`).

in understanding how good a correlation is. In particular, for each correlation, its degree of disambiguation is defined as the ratio between the number of ambiguous activations that can be disambiguated with the considered correlation and the total number of ambiguous activations.

In this example, for each candidate constraint, we choose the correlation "A.org:resource = T.org:resource" because, in all cases, this correlation has a high support and a good degree of disambiguation. Therefore, we assume that activation and target of a candidate constraint are correlated if and only if the corresponding activities are executed by the same resource. Then, we discovered *timed response constraints* with a minimum support of 30%.[4]

The plug-in allows the user to choose the time granularity to express the boundaries of the discovered time intervals. In a first attempt, we have chosen *days* as time granularity. The discovered map is shown in Fig. 2. From this map it is evident that *days* is not the best time granularity since, in many cases, with this granularity, the boundaries of the discovered time intervals are equal to 0. Therefore, we tried to choose a finer granularity (*hours*). In Fig. 3, we can see the result. The granularity of *hours* is clearly more suitable for this example. In the discovered map, we can see that, for example, the maximum delay between activation *W_Completeren aanvraag-complete* and the corresponding target *W_Nabellen offertes-start* is of 719.52 hours (around 1 month). In some cases, the two activities are executed in a time span smaller than 1 hour (the delay between the two activities can be 0).

4.2 A Case Study Based on the BPI Challenge 2011

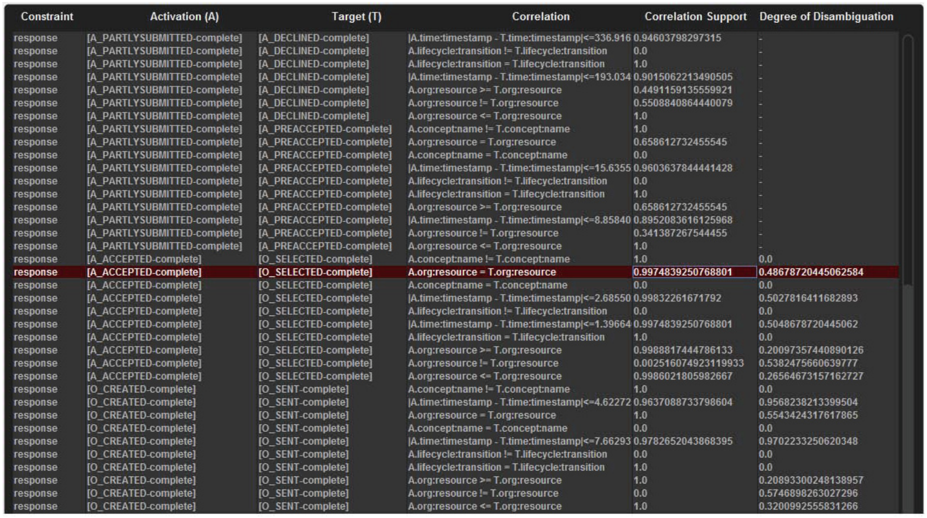
The second case study we present here is based on the application of the proposed approach to the BPI challenge 2011 event log [1] pertaining to the treatment of patients diagnosed with cancer in a large Dutch academic hospital. The event log contains 1,143 cases and 150,291 events distributed across 623 event classes. Each case in this event log is related to a different patient. We suppose, in this case, that all the events in a single case are correlated to each other and, for this reason, we adopt a conservative approach in the discovery of Timed Declare constraints. Our plug-in supports this when the user does not specify any correlation for a candidate constraint.

We discovered from this log *timed responded existence constraints* with a minimum support of 60%. The discovered map is shown in Fig. 4. The time granularity chosen for this map is *days*. The map shows that, for example, the maximum delay between activation *administrative fee - the first pol* and the corresponding target *First outpatient consultation* is of 1,127 days (more than 3 years). In some cases, the two activities are executed in the same day (the delay can be 0).

5 Conclusion

The recent contributions in the context of declarative process discovery do not take the time dimension into consideration. In this paper, we present algorithms

[4] The selected support threshold is low because the average support of the timed response constraints discovered from this log is low.



Constraint	Activation (A)	Target (T)	Correlation	Correlation Support	Degree of Disambiguation
response	[A_PARTLYSUBMITTED-complete]	[A_DECLINED-complete]	|A.time:timestamp - T.time:timestamp|<=336.916	0.94603798297315	-
response	[A_PARTLYSUBMITTED-complete]	[A_DECLINED-complete]	A.lifecycle:transition != T.lifecycle:transition	0.0	-
response	[A_PARTLYSUBMITTED-complete]	[A_DECLINED-complete]	A.lifecycle:transition = T.lifecycle:transition	1.0	-
response	[A_PARTLYSUBMITTED-complete]	[A_DECLINED-complete]	|A.time:timestamp - T.time:timestamp|<=193.034	0.9015062213490505	-
response	[A_PARTLYSUBMITTED-complete]	[A_DECLINED-complete]	A.org:resource >= T.org:resource	0.4491159135559921	-
response	[A_PARTLYSUBMITTED-complete]	[A_DECLINED-complete]	A.org:resource != T.org:resource	0.5508840864440079	-
response	[A_PARTLYSUBMITTED-complete]	[A_DECLINED-complete]	A.org:resource <= T.org:resource	1.0	-
response	[A_PARTLYSUBMITTED-complete]	[A_PREACCEPTED-complete]	A.concept:name != T.concept:name	1.0	-
response	[A_PARTLYSUBMITTED-complete]	[A_PREACCEPTED-complete]	A.org:resource = T.org:resource	0.658612732455545	-
response	[A_PARTLYSUBMITTED-complete]	[A_PREACCEPTED-complete]	A.concept:name = T.concept:name	0.0	-
response	[A_PARTLYSUBMITTED-complete]	[A_PREACCEPTED-complete]	|A.time:timestamp - T.time:timestamp|<=15.6355	0.9603637844441428	-
response	[A_PARTLYSUBMITTED-complete]	[A_PREACCEPTED-complete]	A.lifecycle:transition != T.lifecycle:transition	0.0	-
response	[A_PARTLYSUBMITTED-complete]	[A_PREACCEPTED-complete]	A.lifecycle:transition = T.lifecycle:transition	1.0	-
response	[A_PARTLYSUBMITTED-complete]	[A_PREACCEPTED-complete]	A.org:resource >= T.org:resource	0.658612732455545	-
response	[A_PARTLYSUBMITTED-complete]	[A_PREACCEPTED-complete]	|A.time:timestamp - T.time:timestamp|<=8.85840	0.8952083616125968	-
response	[A_PARTLYSUBMITTED-complete]	[A_PREACCEPTED-complete]	A.org:resource != T.org:resource	0.341387267544455	-
response	[A_PARTLYSUBMITTED-complete]	[A_PREACCEPTED-complete]	A.org:resource <= T.org:resource	1.0	-
response	[A_ACCEPTED-complete]	[O_SELECTED-complete]	A.concept:name != T.concept:name	1.0	0.0
response	[A_ACCEPTED-complete]	[O_SELECTED-complete]	A.org:resource = T.org:resource	0.9974839250768801	0.48678720445062584
response	[A_ACCEPTED-complete]	[O_SELECTED-complete]	A.concept:name = T.concept:name	0.0	0.0
response	[A_ACCEPTED-complete]	[O_SELECTED-complete]	|A.time:timestamp - T.time:timestamp|<=2.68550	0.99832261671792	0.5027816411682893
response	[A_ACCEPTED-complete]	[O_SELECTED-complete]	A.lifecycle:transition != T.lifecycle:transition	0.0	0.0
response	[A_ACCEPTED-complete]	[O_SELECTED-complete]	|A.time:timestamp - T.time:timestamp|<=1.39664	0.9974839250768801	0.5048678720445062
response	[A_ACCEPTED-complete]	[O_SELECTED-complete]	A.lifecycle:transition = T.lifecycle:transition	1.0	0.0
response	[A_ACCEPTED-complete]	[O_SELECTED-complete]	A.org:resource >= T.org:resource	0.9988817444786133	0.20097357440890126
response	[A_ACCEPTED-complete]	[O_SELECTED-complete]	A.org:resource != T.org:resource	0.002516074923119933	0.5382475660639777
response	[A_ACCEPTED-complete]	[O_SELECTED-complete]	A.org:resource <= T.org:resource	0.9986021805982667	0.2656467315716272
response	[O_CREATED-complete]	[O_SENT-complete]	A.concept:name != T.concept:name	1.0	0.0
response	[O_CREATED-complete]	[O_SENT-complete]	|A.time:timestamp - T.time:timestamp|<=4.62272	0.9637088733798604	0.9568238213399504
response	[O_CREATED-complete]	[O_SENT-complete]	A.org:resource = T.org:resource	1.0	0.5543424317617865
response	[O_CREATED-complete]	[O_SENT-complete]	A.concept:name = T.concept:name	0.0	0.0
response	[O_CREATED-complete]	[O_SENT-complete]	|A.time:timestamp - T.time:timestamp|<=7.66293	0.9782652043868395	0.9702233250620348
response	[O_CREATED-complete]	[O_SENT-complete]	A.lifecycle:transition != T.lifecycle:transition	0.0	0.0
response	[O_CREATED-complete]	[O_SENT-complete]	A.lifecycle:transition = T.lifecycle:transition	1.0	0.0
response	[O_CREATED-complete]	[O_SENT-complete]	A.org:resource >= T.org:resource	1.0	0.20893300248138957
response	[O_CREATED-complete]	[O_SENT-complete]	A.org:resource != T.org:resource	0.0	0.5746898263027296
response	[O_CREATED-complete]	[O_SENT-complete]	A.org:resource <= T.org:resource	1.0	0.3200992555831266

Fig. 1. Correlations selection

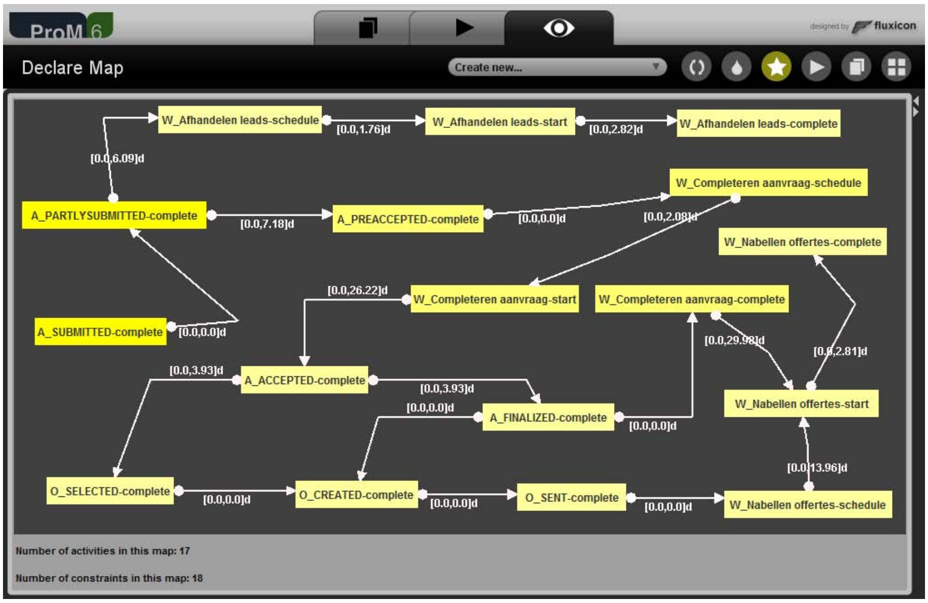
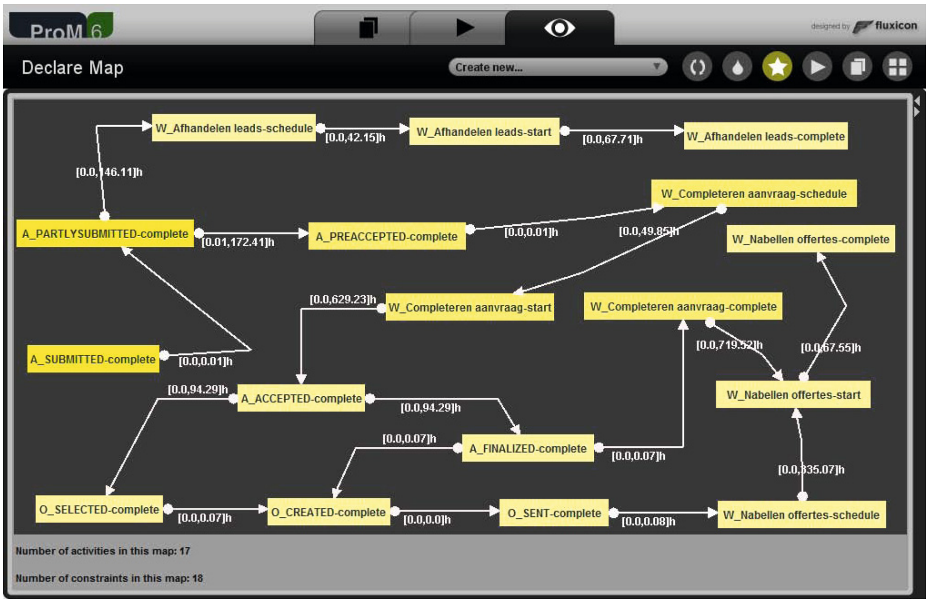


Fig. 2. Discovered *timed response constraints* from the log provided for the BPI Challenge 2012 (time granularity: *days*)

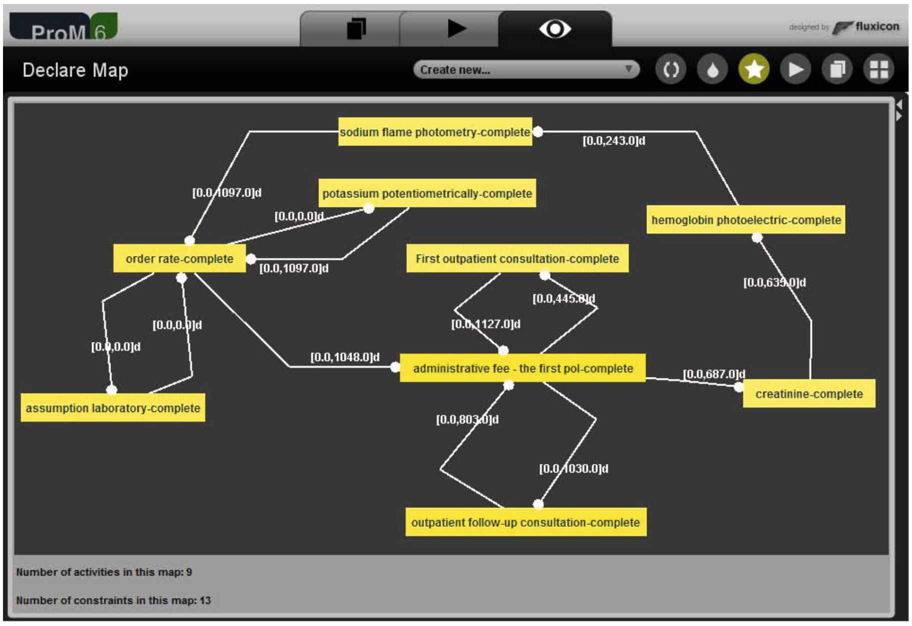
for discovering Timed Declare constraints from event logs. We use correlations to connect each activation of a constraint with the corresponding target thus improving the reliability of the discovered metric temporal constraints. The proposed algorithms can also be used for discovering negative relation constraints.



Number of activities in this map: 17

Number of constraints in this map: 18

Fig. 3. Discovered *timed response constraints* from the log provided for the BPI Challenge 2012 (time granularity: *hours*)


ProM 6
designed by fluxicon
Declare Map
Create new...
sodium flame photometry-complete
[0.0,243.0]d
[0.0,1097.0]d
potassium potentiometrically-complete
[0.0,0.0]d
hemoglobin photoelectric-complete
order rate-complete
[0.0,1097.0]d
First outpatient consultation-complete
[0.0,445.0]d
[0.0,635.0]d
[0.0,0.0]d
[0.0,1127.0]d
[0.0,0.0]d
[0.0,1048.0]d
administrative fee - the first pol-complete
[0.0,687.0]d
creatinine-complete
assumption laboratory-complete
[0.0,803.0]d
[0.0,1030.0]d
outpatient follow-up consultation-complete

Number of activities in this map: 9

Number of constraints in this map: 13

Fig. 4. Discovered *timed responded existence constraints* from the log provided for the BPI Challenge 2011 (time granularity: *days*)

To this aim, we have extended the notion of correlation support also to cover this group of constraints. Our evaluation using real-life logs demonstrates that the proposed approach is applicable in real-life settings.

References

1. 3TU Data Center: BPI Challenge 2011 Event Log (2011), doi:10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54
2. van der Aalst, W., Pesic, M., Schonenberg, H.: Declarative Workflows: Balancing Between Flexibility and Support. Computer Science - R&D, 99–113 (2009)
3. Barros, A., Decker, G., Dumas, M., Weber, F.: Correlation Patterns in Service-Oriented Architectures. In: Dwyer, M.B., Lopes, A. (eds.) FASE 2007. LNCS, vol. 4422, pp. 245–259. Springer, Heidelberg (2007)
4. Bose, R.P.J.C., Maggi, F.M., van der Aalst, W.M.P.: Enhancing declare maps based on event correlations. In: Daniel, F., Wang, J., Weber, B. (eds.) BPM 2013. LNCS, vol. 8094, pp. 97–112. Springer, Heidelberg (2013)
5. Burattin, A., Maggi, F.M., van der Aalst, W.M.P., Sperduti, A.: Techniques for a Posteriori Analysis of Declarative Processes. In: EDOC, pp. 41–50 (2012)
6. Chesani, F., Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Exploiting Inductive Logic Programming Techniques for Declarative Process Mining. In: Jensen, K., van der Aalst, W.M.P. (eds.) ToPNoC II. LNCS, vol. 5460, pp. 278–295. Springer, Heidelberg (2009)
7. Di Ciccio, C., Mecella, M.: Mining constraints for artful processes. In: Abramowicz, W., Kriksciuniene, D., Sakalauskas, V. (eds.) BIS 2012. LNBIP, vol. 117, pp. 11–23. Springer, Heidelberg (2012)
8. van Dongen, B.: Bpi challenge 2012 (2012), `http://dx.doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f`
9. Kupferman, O., Vardi, M.Y.: Vacuity Detection in Temporal Model Checking. International Journal on Software Tools for Technology Transfer, 224–233 (2003)
10. Maggi, F.M., Bose, R.P.J.C., van der Aalst, W.M.P.: Efficient Discovery of Understandable Declarative Models from Event Logs. In: Ralyté, J., Franch, X., Brinkkemper, S., Wrycza, S. (eds.) CAiSE 2012. LNCS, vol. 7328, pp. 270–285. Springer, Heidelberg (2012)
11. Maggi, F.M., Mooij, A.J., van der Aalst, W.M.P.: User-Guided Discovery of Declarative Process Models. In: CIDM, pp. 192–199 (2011)
12. Montali, M.: Specification and Verification of Declarative Open Interaction Models: a Logic-Based Approach. Ph.D. thesis, University of Bologna (2009)
13. Pichler, P., Weber, B., Zugal, S., Pinggera, J., Mendling, J., Reijers, H.A.: Imperative Versus Declarative Process Modeling Languages: An Empirical Investigation. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM Workshops 2011, Part I. LNBIP, vol. 99, pp. 383–394. Springer, Heidelberg (2012)
14. Reijers, H.A., Slaats, T., Stahl, C.: Declarative modeling-an academic dream or the future for bpm? In: Daniel, F., Wang, J., Weber, B. (eds.) BPM 2013. LNCS, vol. 8094, pp. 307–322. Springer, Heidelberg (2013)
15. Thati, P., Roşu, G.: Monitoring algorithms for metric temporal logic specifications. Electron. Notes Theor. Comput. Sci. 113, 145–162 (2005)
16. Westergaard, M., Maggi, F.M.: Looking into the future: Using timed automata to provide a priori advice about timed declarative process models. In: Meersman, R., et al. (eds.) OTM 2012, Part I. LNCS, vol. 7565, pp. 250–267. Springer, Heidelberg (2012)