Discovery of Multi-Perspective Declarative Process Models

Stefan Schönig[1], Claudio Di Ciccio[1], Fabrizio M. Maggi[2], and Jan Mendling[1]

[1] Vienna University of Economics and Business, Austria
{stefan.schoenig,claudio.di.ciccio,jan.mendling}@wu.ac.at
[2] University of Tartu, Estonia
f.m.maggi@ut.ee

Abstract. Process discovery is one of the main branches of process mining that allows the user to build a process model representing the process behavior as recorded in the logs. Standard process discovery techniques produce as output a procedural process model (e.g., a Petri net). Recently, several approaches have been developed to derive declarative process models from logs and have been proven to be more suitable to analyze processes working in environments that are less stable and predictable. However, a large part of these techniques are focused on the analysis of the control flow perspective of a business process. Therefore, one of the challenges still open in this field is the development of techniques for the analysis of business processes also from other perspectives, like data, time, and resources. In this paper, we present a full-fledged approach for the discovery of multi-perspective declarative process models from event logs that allows the user to discover declarative models taking into consideration all the information an event log can provide. The approach has been implemented and experimented in real-life case studies.

Keywords: Process Mining, Process Discovery, Multi-Perspective Process Model, Declarative Process Model, Declare

1 Introduction

Process mining, and specifically process discovery, is driven by the ambition to understand how a process is truly executed, why certain activities are executed and under which circumstances. It aims at constructing a process model from an event log consisting of traces, such that each trace corresponds to one execution of the process. Each event in a trace consists as a minimum of an event class (i.e., the activity to which the event corresponds) and generally a timestamp. In some cases, other information may be available such as the originator of the event (i.e., the performer of the activity) as well as data produced by the event in the form of attribute-value pairs. Discovery is of particular value for processes that offer various options to execute them. Those processes are often referred to as flexible, adaptive, unstructured or knowledge-intense. Often, procedural process models resulting from discovery are colloquially called Spaghetti models due to their complex structure [1]. Therefore, discovered process models can be represented as a set of declarative constraints for directly representing the causality of the behavior [25].

The benefits of declarative languages such as Declare [24], DPIL [32] or DCR Graphs [12] have been emphasized in the literature. It is also well known that behavior is typically intertwined with dependencies upon value ranges of data

parameters and resource characteristics [27,15]. Therefore, Declare has been extended towards Multi-Perspective Declare (MP-Declare) [5]. However, state-of-the-art mining tools such as MINERful [8,9] and DeclareMiner [19,16] do not support MP-Declare at this moment.

In this paper, we address this problem by proposing a mining technique for discovering MP-Declare models. We show that the discovery of MP-Declare allows for the acquisition of knowledge that goes beyond the classical declarative mining, which is focused only on the behavioral perspective in the vast majority of cases. Furthermore, we present the first foundational categorization of the conditions that are posed on declarative constraints with a special focus on how these categories are reflected into discovery metrics. We implemented our approach starting from the SQL-based process mining approach described in [29], relying on RXES, a standardized architecture for storing event log data in relational databases [11]. The approach has been validated with several real-life event logs provided by a large academic hospital, by five Dutch municipalities and by an Italian local police office for managing fines for road traffic violations.

The paper is structured as follows. Section 2 presents a typical discovery problem that we tackle with our research, and the notions both of Declare and MP-Declare modeling. Section 3 defines the framework we propose to delineate the boundaries of the process discovery task. Section 4 describes the approach developed on top of SQL. Section 5 presents the evaluation of our technique with 3 real-life cases. Section 6 discusses related work before Section 7 that concludes the paper.

2 Research Background

In this section, we first illustrate the research problem that we are addressing. We then summarize concepts of Declare and MP-Declare.

2.1 Research Problem

Declarative constraints are strong in representing the permissible behavior of business processes. Modeling languages like Declare [2] describe a set of *constraints* that must be satisfied throughout the process execution. Constraints, in turn, are based on *templates*. Templates are patterns that define parameterized classes of properties, and constraints are their concrete instantiations. Their semantics can be formalized using formal logics such as Linear Temporal Logic over finite traces (LTL_f) [23].

A central shortcoming of languages like Declare is the fact that templates are not directly capable of expressing the connection between the behavior and other perspectives of the process. Consider the example of a loan application process. The process analyst would be interested to learn about constraints such as the following:

1. Activation conditions: When a loan was requested and *account balance* $>$ *4,000 EUR*, the loan was subsequently granted in 95% of the cases.
2. Correlation conditions: When a loan was requested, the loan was subsequently granted and *amount requested* $=$ *amount granted* in 95% of the cases.
3. Target conditions: When a loan was requested, the loan was subsequently granted in 95% of the cases by a specific member of the financial board.

Table 1: Semantics for Declare templates in LTL_f.

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

4. Temporal conditions: When a loan was requested, the loan was subsequently granted *within the next 30 days* in 95% of the cases.

Standard Declare only supports constraints like the ones shown in Table 1. Here, the **F**, **X**, **G**, and **U** LTL_f future operators have the following meanings: formula $\mathbf{F}\psi_1$ means that ψ_1 holds sometime in the future, $\mathbf{X}\psi_1$ means that ψ_1 holds in the next position, $\mathbf{G}\psi_1$ says that ψ_1 holds forever in the future, and, lastly, $\psi_1 \mathbf{U} \psi_2$ means that sometime in the future ψ_2 will hold and until that moment ψ_1 holds (with ψ_1 and ψ_2 LTL_f formulas). The **O**, **Y** and **S** LTL_f past operators have the following meaning: $\mathbf{O}\psi_1$ means that ψ_1 holds sometime in the past, $\mathbf{Y}\psi_1$ means that ψ_1 holds in the previous position, and $\psi_1 \mathbf{S} \psi_2$ means that ψ_1 has held sometime in the past and since that moment ψ_2 holds. Consider, for example, the *response* constraint $\mathbf{G}(A \rightarrow \mathbf{F}B)$. It indicates that if A *occurs*, B must eventually *follow*. Therefore, this constraint is fully satisfied in 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 occurrence of A is not followed by a B. In $\mathbf{t}_2$, it is *vacuously satisfied* [13,4], i.e., in a trivial way, because A never occurs.

An *activation activity* of a constraint in a trace is an activity whose execution imposes, because of that constraint, some obligations on the execution of other activities (target activities) in the same trace (see Table 1). For example, A is an activation activity 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. An activation of a constraint leads to a *fulfillment* or to a *violation*. Consider, again, $\mathbf{G}(A \rightarrow \mathbf{F}B)$. In trace $\mathbf{t}_1$, the constraint is activated and fulfilled twice, whereas, in trace $\mathbf{t}_3$, it is activated and fulfilled only once. In trace $\mathbf{t}_4$, it is activated twice and the second activation leads to a violation (B does not occur subsequently).

2.2 Multi-Perspective Declare

The importance of more complex constraints that integrate activation, correlation, target and temporal dependencies has been emphasized by prior research and has led to the definition of a multi-perspective version of Declare [5]. Table 2

Table 2: Semantics for MP-Declare constraints in LTL_f.

Template	**LTL_f Semantics**
existence	$\top \rightarrow \mathbf{F}(e(A) \wedge \varphi_a(\boldsymbol{x})) \vee \mathbf{O}(e(A) \wedge \varphi_a(\boldsymbol{x}))$
responded existence	$\mathbf{G}((A \wedge \varphi_a(\boldsymbol{x})) \rightarrow (\mathbf{O}(B \wedge \varphi_c(\boldsymbol{x}, \boldsymbol{y}) \wedge \varphi_t(\boldsymbol{y})) \vee \mathbf{F}(B \wedge \varphi_c(\boldsymbol{x}, \boldsymbol{y}) \wedge \varphi_t(\boldsymbol{y}))))$
response	$\mathbf{G}((A \wedge \varphi_a(\boldsymbol{x})) \rightarrow \mathbf{F}(B \wedge \varphi_c(\boldsymbol{x}, \boldsymbol{y}) \wedge \varphi_t(\boldsymbol{y})))$
alternate response	$\mathbf{G}((A \wedge \varphi_a(\boldsymbol{x})) \rightarrow \mathbf{X}(\neg(A \wedge \varphi_a(\boldsymbol{x}))\mathbf{U}(B \wedge \varphi_c(\boldsymbol{x}, \boldsymbol{y}) \wedge \varphi_t(\boldsymbol{y})))$
chain response	$\mathbf{G}((A \wedge \varphi_a(\boldsymbol{x})) \rightarrow \mathbf{X}(B \wedge \varphi_c(\boldsymbol{x}, \boldsymbol{y}) \wedge \varphi_t(\boldsymbol{y})))$
precedence	$\mathbf{G}((B \wedge \varphi_a(\boldsymbol{x})) \rightarrow \mathbf{O}(A \wedge \varphi_c(\boldsymbol{x}, \boldsymbol{y}) \wedge \varphi_t(\boldsymbol{y})))$
alternate precedence	$\mathbf{G}((B \wedge \varphi_a(\boldsymbol{x})) \rightarrow \mathbf{Y}(\neg(B \wedge \varphi_a(\boldsymbol{x}))\mathbf{S}(A \wedge \varphi_c(\boldsymbol{x}, \boldsymbol{y}) \wedge \varphi_t(\boldsymbol{y})))$
chain precedence	$\mathbf{G}((B \wedge \varphi_a(\boldsymbol{x})) \rightarrow \mathbf{Y}(A \wedge \varphi_c(\boldsymbol{x}, \boldsymbol{y}) \wedge \varphi_t(\boldsymbol{y})))$
not responded existence	$\mathbf{G}((A \wedge \varphi_a(\boldsymbol{x})) \rightarrow \neg(\mathbf{O}(B \wedge \varphi_c(\boldsymbol{x}, \boldsymbol{y}) \wedge \varphi_t(\boldsymbol{y})) \vee \mathbf{F}(B \wedge \varphi_c(\boldsymbol{x}, \boldsymbol{y}) \wedge \varphi_t(\boldsymbol{y}))))$
not response	$\mathbf{G}((A \wedge \varphi_a(\boldsymbol{x})) \rightarrow \neg\mathbf{F}(B \wedge \varphi_c(\boldsymbol{x}, \boldsymbol{y}) \wedge \varphi_t(\boldsymbol{y})))$
not precedence	$\mathbf{G}((B \wedge \varphi_a(\boldsymbol{x})) \rightarrow \neg\mathbf{O}(A \wedge \varphi_c(\boldsymbol{x}, \boldsymbol{y}) \wedge \varphi_t(\boldsymbol{y})))$
not chain response	$\mathbf{G}((A \wedge \varphi_a(\boldsymbol{x})) \rightarrow \neg\mathbf{X}(B \wedge \varphi_c(\boldsymbol{x}, \boldsymbol{y}) \wedge \varphi_t(\boldsymbol{y})))$
not chain precedence	$\mathbf{G}((B \wedge \varphi_a(\boldsymbol{x})) \rightarrow \neg\mathbf{Y}(A \wedge \varphi_c(\boldsymbol{x}, \boldsymbol{y}) \wedge \varphi_t(\boldsymbol{y})))$

shows the semantics of Multi-Perspective Declare (MP-Declare) formally defined using LTL_f.

This semantics build on the notion of *payload* of an event. Consider again the loan request example. Henceforth, we write $e(\textit{credit check})$ to identify the occurrence of an event, in order to distinguish it from the activity name (*credit check*) when it is not clear from the context. At the time of *credit check*, i.e., when the timestamp $\tau^e_{\text{credit check}}$ elapses, the attributes *Req.ID*, *Resource*, *Applicant*, *AgeOfApplicant*, and *Debt* have the values *20160202*, *FinancialBoardU001*, *John*, *40*, and *10,000*, respectively. We refer to $p^e_{\text{credit check}} = (\textit{20160202}, \textit{FinancialBoardU001}, \textit{John}, \textit{40}, \textit{10,000})$ as its payload. To denote the projection of the payload $p^e_A = (x_1, \ldots, x_n)$ over attributes $x_1, \ldots, x_m$ with $m \leqslant n$, we use the shorthand notation $p^e_A[x_1, \ldots, x_m]$. In the example, $p^e_{\text{credit check}}[\textit{Req.ID}]$ is $(\textit{20160202})$, and $p^e_{\text{credit check}}[\textit{Applicant}, \textit{AgeOfApplicant}]$ is $(\textit{John}, \textit{40})$.

In Table 2, we use a shorthand notation for n-ples of attributes x_i, namely $\boldsymbol{x}$. Referring to the formal specification of constraints in LTL_f (cf. Tables 1 and 2), we call *activation* ϕ_a the sub-formula that lies on the left-hand side of the implication $\rightarrow$ operator, whereas the *target* ϕ_t is the formula that lies on its right-hand side. Templates in MP-Declare extend standard Declare with additional conditions on attributes: given events $e(A)$ and $e(B)$ with payloads $p^e_A = (x_1, \ldots, x_n)$ and $p^e_B = (y_1, \ldots, y_n)$, we define the *activation condition* φ_a, the *correlation condition* φ_c, and the *target condition* φ_t. The activation condition is part of the activation ϕ_a, whilst the correlation and target conditions are part of the target ϕ_t, according to their respective time of evaluation.

The *activation* condition is a statement that must be valid when the activation occurs. In the case of the response template, the activation condition has the form $\varphi_a(x_1, \ldots, x_n)$, meaning that the proposition φ_a over $(x_1, \ldots, x_n)$ must hold true. For example, to express that whenever *credit check* is executed and *Debt* is $<$ *20,000*, then eventually *grant* follows, we write: $\mathbf{G}((e(\textit{credit check}) \wedge p^e_{\text{credit check}}[\textit{Debt}] < \textit{20,000}) \rightarrow \mathbf{F}(e(\textit{grant})))$. In this example, activation ϕ_a consists of a statement about the occurrence of an event $(e(\textit{credit check}))$ and of a condition over an attribute of such event $(\varphi_a = p^e_{\text{credit check}}[\textit{Debt}] < \textit{20,000})$. In case *credit check* is executed but *Debt*

is $\geqslant$ *20,000*, the constraint is not activated. Target ϕ_t remains in the form of a standard Declare definition, because it specifies only the occurrence of the target event ($e(\textit{grant})$).

The *correlation* condition is a statement that must be valid when the target occurs, and relates the values of the attributes in the payloads both of the activation and the target event. It has the form $\varphi_c(x_1, \ldots, x_m, y_1, \ldots, y_m)$ with $m \leqslant n$, where φ_c is a propositional formula on the variables both of the payload of $e(A)$ and the payload of $e(B)$. For instance, whenever *credit check* is executed, then eventually *grant* must follow and the *Req.ID* attribute value associated with $e(\textit{credit check})$ must be the same as for $e(\textit{grant})$. We write: $\mathbf{G}((e(\textit{credit check}) \rightarrow \mathbf{F}(e(\textit{grant}) \wedge p^e_{\text{credit check}}[\textit{Req.ID}] = p^e_{\text{grant}}[\textit{Req.ID}]))$. In the example, target ϕ_t is the conjunction of $e(\textit{grant})$, specifying the occurrence of the event, and $p^e_{\text{credit check}}[\textit{Req.ID}] = p^e_{\text{grant}}[\textit{Req.ID}]$, correlating the attribute values of activation and target events. The activation remains defined as in the form of a standard Declare constraint.

Target conditions exert limitations on the values of the attributes that are registered at the moment wherein the target activity occurs. It has the form $\varphi_t(y_1, \ldots, y_m)$ with $m \leqslant n$, where φ_t is a propositional formula involving variables in the payload of $e(B)$. As an example, when activity *credit check* is performed, then eventually *grant* is executed and the *Resource* associated with $e(\textit{grant})$ must be *FinancialBoardU001*. We write $\mathbf{G}((e(\textit{credit check}) \rightarrow \mathbf{F}(e(\textit{grant}) \wedge p^e_{\text{grant}}[\textit{Resource}] = \textit{FinancialBoardU001}))$. As before, activation ϕ_a only consists of a statement about the occurrence of an event ($e(\textit{credit check})$), as for standard Declare. Target ϕ_t specifies what the value of an attribute of the target event ($p^e_{\text{grant}}[\textit{Resource}] = \textit{FinancialBoardU001}$) is, when it occurs ($e(\textit{grant})$). As shown in Table 2, declarative templates like *existence* have an activation which is meant to be always satisfied ($\phi_a = \top$). Therefore, only the target is meant to be enriched with target conditions.

In MP-Declare, also a *temporal* condition can be specified through an interval ($I = [\tau_0, \tau_1)$) indicating the minimum and the maximum temporal distance allowed between the occurrence of the activation and the occurrence of the corresponding target. It plays a fundamental role process modeling through constraints, thus we consider it as a first-class citizen in the categorization of conditions in MP-Declare. However, it falls in the category of correlation conditions, as it is based on the comparison of values associated to both activation and target events. In the light of Table 2, for example, the *response* constraint with a temporal condition indicates that, if the *credit check* occurs at time $\tau^e_{\textit{credit check}}$, *grant* must occur at some point $\tau^e_{\textit{grant}} \in [\tau^e_A + 1\text{day}, \tau^e_A + 7\text{days})$, hence $\mathbf{G}((e(\textit{credit check}) \rightarrow \mathbf{F}(e(\textit{grant}) \wedge \tau^e_{\text{credit check}} + 1\text{day} \leqslant \tau^e_{\text{grant}} < \tau^e_{\text{credit check}} + 7\text{days}))$.

Until now, no mining approach that can fully support MP-Declare is available.

3 Multi-Perspective Declare Discovery Framework

In this section, we describe our proposed framework for the discovery of MP-Declare models. In particular, we introduce the requirements and discuss how constraints are distinguished between the ones that are fulfilled and the ones that are not fulfilled throughout the log. An implementation of the framework is described in Section 4.

3.1 Requirements for the Discovery of Multi-Perspective Declare Constraints

The requirements presented in this paper concern the discovery of MP-Declare constraints like the ones introduced in Section 2.2. In particular, the requirements describe different types of multi-perspective conditions that can be discovered from a log and used to specify valid MP-Declare constraints. In line with the semantics introduced in Section 2.2, the conditions that can be discovered are activation, correlation, target, and time conditions.

Activation Conditions. An activation condition can be used for two different purposes, i.e., to build *discriminative constraints* or to build *descriptive constraints*. Suppose that, for a given standard Declare constraint, in an event log, there are both activations corresponding to fulfillments and activations corresponding to violations. The payloads of fulfillments and violations can be used as positive and negative examples to train a classifier that solves the following classification problem: "What is the (activation) condition to be specified on the payload of an activation of a constraint to guarantee that that activation corresponds to a fulfillment for that constraint?". In this case, the activation condition is a condition that is only valid in the positive cases and not in the negative cases (or vice versa) and is used to *discriminate* between fulfillments and violations for a given constraint. For example, consider the response constraint between *loan request* and *grant*. Suppose that when attribute *Amount* associated to $e(\textit{loan request})$ is lower than *100,000*, $e(\textit{loan request})$ is eventually followed by $e(\textit{grant})$, and when attribute *Amount* associated to $e(\textit{loan request})$ is greater than or equal to *100,000* $e(\textit{loan request})$ is not eventually followed by $e(\textit{grant})$. In such a case, the activation condition $p^e_{\textit{grant}}[\textit{Amount}] < \textit{100,000}$ discriminates between fulfillments and violations for the given response constraint. This is the type of constraints that is possible to discover with the approach presented in [18].

Nevertheless, activation conditions can also be *descriptive*. For example, it is possible to find the distribution (or the average) of the values of each attribute connected to the fulfillments of a constraint, regardless of their values when the constraint is violated. Notice that in all the examples mentioned so far, activation conditions consist of a binary proposition between a variable and a constant. These are the conditions we deal with in this paper. However, in general, these conditions can be more complex, because they can involve 2 or more variables.

Target and Correlation Conditions. Positive constraints, corresponding to the templates in rows 2–8 in Tables 1 and 2, are characterized by the fact that a fulfillment has always a correlated target and a violation never has a correlated target. In contrast, for negative constraints, a fulfillment never has a correlated target and a violation has always a correlated target. Therefore, target and correlation conditions can only be defined for positive constraints in case of fulfillment, whereas for negative constraints a correlation/target condition can only be defined in case of violation. For this reason, target and correlation conditions cannot discriminate between fulfillments and violations and can only be descriptive. Note that, for negative constraints, we talk about "negative correlations," i.e., conditions that should disconnect a forbidden target from a possible corresponding activation.

Complex correlation conditions can be discovered from an event log, i.e., every relation involving variables belonging to the payload of the activation and the target of a constraint. Here, we focus on relations between homologous attributes of activations and targets. For example, in the precedence constraint specifying that activity *check report* must be preceded by *write report*, it can be the case that the resource associated to $e(\textit{check report})$ is in 95% of the cases different from the one associated to $e(\textit{write report})$ and in 5% of the cases is the same. Note that we are here connecting homologous attributes, i.e., the resource associated to the activation and the same attribute associated to the target of the precedence constraint.

Time Conditions. Finally, time conditions relate to the time distance between the activation and corresponding targets. For example, for the response constraint between *make diagnosis* and *surgery*, the time distance between these two activities can be between 7 days and 14 days in 30% of the cases, between 15 days and 30 days in 60% of the cases, and higher than 30 days in 10% of the cases.

To summarize, the requirements we identify for the discovery of MP-Declare are:

1. discovering discriminative activation conditions;
2. discovering descriptive activation conditions;
3. discovering (descriptive) target and correlation conditions;
4. discovering time conditions.

3.2 Support and Confidence.

In this subsection, we describe the metrics that we use to discriminate those constraints that are fulfilled in the majority of cases, from those that are rarely satisfied, namely support and confidence. We consider two notions of support already defined in the literature, namely the event-based support [9] and the trace-based support [19]. The former is meant to be used for all constraints wherein both activation and target do not correspond to $\top$. For all the others, we use the second notion of support.

We denote the set of *events* in a *trace* $\mathbf{t}$ of an event log L that fulfill an LTL_f formula[3] ψ as $\models^e_{\mathbf{t}}(\psi)$. The set of all the *events* in *log* L that fulfill ψ are denoted as $\models^e_L(\psi)$. All the *traces* in *log* L consisting only of events that fulfill ψ are indicated as $\models^{\mathbf{t}}_L(\psi)$. Given a constraint Ξ comprising activation ϕ_a and target ϕ_t, we formally define the event-based support $\mathcal{S}^e_L$ and the trace-based support $\mathcal{S}^{\mathbf{t}}_L$ as follows:

$$\mathcal{S}^e_L = \frac{\sum_{i=1}^{|L|} \left|\models^e_{\mathbf{t}_i}(\Xi)\right|}{\left|\models^e_L(\phi_a)\right|} \qquad (1) \qquad\qquad \mathcal{S}^{\mathbf{t}}_L = \frac{\left|\models^{\mathbf{t}}_L(\Xi)\right|}{|L|} \qquad (2)$$

The confidence metric scales the support by the fraction of traces in the log wherein the activation condition is satisfied. According to the adopted notion of support, we have that:

(i) $\mathcal{C}^e_L = \mathcal{S}^e_L \times \left|\models^e_L(\phi_a)\right| / |L|$, and *(ii)* $\mathcal{C}^{\mathbf{t}}_L = \mathcal{S}^{\mathbf{t}}_L \times \left|\models^e_L(\phi_a)\right| / |L|$.

$\mathcal{S}^{\mathbf{t}}_L$ counts the number of events that fulfill the constraint in every trace and sums

[3] We recall that a propositional formula is an LTL_f formula.

such numbers up along the log. In the example of Section 2.1, the four occurrences of A fulfill *response*(A, B), out of which 2 occur in $\mathbf{t}_1$, 1 in $\mathbf{t}_3$ and 1 in $\mathbf{t}_4$. Thereupon, it scales the number of events fulfilling the constraint by the number of events that fulfill the activation only. In the example, the five occurrences of A satisfy the activation. Therefore, the event-based support of *response*(A, B) is equal to 4/5, namely 0.8. Its confidence amounts to $4/5 \times 3/4 = 0.6$, because A occurs in 3 traces over 4. $\mathcal{S}_L^{\mathbf{t}}$ counts instead the number of traces that fulfill the constraint. In the example, $\mathbf{t}_1$, $\mathbf{t}_3$ and $\mathbf{t}_3$ fulfill *existence*(A). Thereafter, such quantity is scaled by the number of traces in the log, which are four in the example. Thus, the trace-based support of *existence*(A) is 3/4, i.e., 0.75. In the next section, we show how these notions apply to MP-Declare.

4 Multi-Perspective Declare Discovery with SQL

Our proposed discovery framework has been implemented using the SQL-based process discovery approach described in [29] because of its versatility towards customization. The approach has been adopted for the realization of a proof-of-concept software module and relies on the use of RXES. RXES is a standardized architecture for storing event log data in relational databases introduced in [11]. The RXES architecture uses a database to store the event log where traces and events are represented by tables with identifiers. RXES provides a full implementation of all OpenXES interfaces using the database as a backend. In [29], it has been shown that it is possible to discover commonly used process constraints by means of conventional SQL queries. Queries can be tailored to arbitrary aspects of a process, e.g., control flow, data attributes, and organizational issues.

4.1 Declarative Process Discovery with SQL

First, we describe the general functionality of SQL-based process discovery. The following query represents the basic structure of an SQL-query that discovers all constraints instantiation of the standard template *Response* with two thresholds *minSupp* and *minConf*. Here, subqueries are marked with brackets.

```
SELECT  'Response', A, B, [Support], [Confidence]
FROM Log l1, Log l2, [ActivityCombinations] c
WHERE l1.Activity = c.A AND l2.Activity = c.B AND
      l2.ID IN(SELECT TOP 1 ID
               FROM [Log] l2
               WHERE b.Activity = c.B AND l2.case = l1.Trace AND
                     l2.Time > l1.Time
               ORDER BY Time ASC)

GROUP BY c.A, c.B
HAVING [Support] > minSupp AND [Confidence] > minConf
```

The SQL expression for calculating the *support* of *response* constraints is given as:

```
COUNT(*) / (SELECT COUNT(ID) FROM Log WHERE Activity = A)
```

The query tests if at least one occurrence of activity B exists that follows the currently observed occurrence of A. In case the logical EXISTS term in the WHERE clause evaluates to true, the currently observed tuple corresponds to a fulfillment of the constraint. The resulting set of tuples represents all the fulfillments of the *response* template.

Event ID	Case ID	Activity Name	Timestamp	x_1	x_2	...	x_n
1	1	a	2015-11-06 15:31:00	id_1	3	...	5
2	1	b	2015-11-06 15:35:00	id_1	2	...	4
3	1	c	2015-11-06 15:37:00	id_2	3	...	4
4	2	b	2015-11-06 16:22:00	id_2	4	...	4
5	2	c	2015-11-06 16:45:00	id_2	3	...	4
...							

Table 3: Event log excerpt stored in a denormalized relational database table.

4.2 The Multi-Perspective Case

Consider the event log excerpt given in Table 3. In addition to the columns for *Event ID*, *Case ID*, *Activity Name* and *Timestamp* the table contains n columns for different data attributes x_1, x_2,..., x_n. SQL queries like the *response* query can be enhanced to comprise data attributes as well. For example, the *MP-Response* query below discovers all the *response* constraints for each value combination of the involved data attributes x_1, x_2,..., x_n. Therefore, the GROUP BY and the SELECT clause additionally contain the list of event parameters. Each query can be adjusted to the analyst's needs, i.e., additional constraint activation, target or correlation conditions like $l1.x_1 = l2.x_1$ or $l1.x_2 > l2.x_2$ can be added to the WHERE clause of the query. Note, that $l1$ and $l2$ respectively refer to the events assigned to the first and the second parameter of the *response* template. Consequently, the result set provides a fine-grained resolution of the constraints that hold for certain activities specifying information about the data perspective, e.g., by providing the distribution or the average of the values of the considered data attributes when a fulfillment of the constraint occurs.

```
SELECT  'MP-Response', A, B, l1.x1, ...,  l1.xn, [Support], [Confidence]
FROM Log l1, Log l2, [ActivityCombinations] c
WHERE l1.Activity = c.A AND l2.Activity = c.B AND
      l2.ID IN(SELECT TOP 1 ID
               FROM [Log] l2
               WHERE b.Activity = c.B AND l2.Trace = l1.Trace AND
                     l2.Time > l1.Time
               ORDER BY Time ASC)

GROUP BY c.A, c.B, l1.x1, ...,  l1.xn
HAVING [Support] > minSupp AND [Confidence] > minConf
```

The subquery to compute the *support* value implements the event-based *support* definition in Eq. 1 as described in Section 3. The subquery is given by:

```
COUNT(*) / (SELECT COUNT(ID)
            FROM Log
            WHERE Activity = A AND Log.x1 = l1.x1 AND ... AND Log.xn = l1.xn)
```

Similar to the *MP-Response* query also other templates can be discovered with SQL queries considering the data perspective. The following *MP-Existence* query, e.g., discovers the values of the data attributes when a certain activity is performed.

```
SELECT  'MP-Existence', A, l1.x1, ...,  l1.xn, [Support], [Confidence]
FROM Log l1, [ActivityCombinations] c
WHERE l1.Activity = c.A
GROUP BY c.A, c.B, l1.x1, ..., l1.xn
HAVING [Support] > minSupp AND [Confidence] > minConf
```

Here, the *support* value is computed with the subquery below. SQL queries for other MP-Declare constraints can be formulated in a similar way.

```
COUNT(Distinct Instance) /
(SELECT COUNT(*) FROM (SELECT Trace FROM Log GROUP BY Trace))
```

5 Evaluation

In order to assess our approach, we have applied it on several well-known benchmarks in the process mining field. The evaluation shows that important information would be most likely neglected if perspectives other than the pure behavioral one were not taken into account.

5.1 Activation Conditions: Road Traffic Fine Management Log

We first evaluated our approach for the discovery of activation conditions using the publicly available real-life event log of a Road Traffic Fine Management Process.[4] The event log records executions of the process enacted in an Italian local police office for managing fines for road traffic violations. It contains 150,370 traces and 561,470 events for 11 different activities. We first queried the event log for standard *response* constraints without considering data attributes. Using the thresholds *minSupp=0.7* and *minConf=0.3* we extracted five constraints. In order to discover data conditions, we exemplarily focus on the constraint $C =$ *response(add penalty, send for credit collection)*. After the discovery phase, it was found $\mathcal{S}^e_L(C) = 0.74$ and $\mathcal{C}^e_L(C) = 0.39$, i.e., in 74% of the cases where a penalty was given, the case was sent for credit collection.

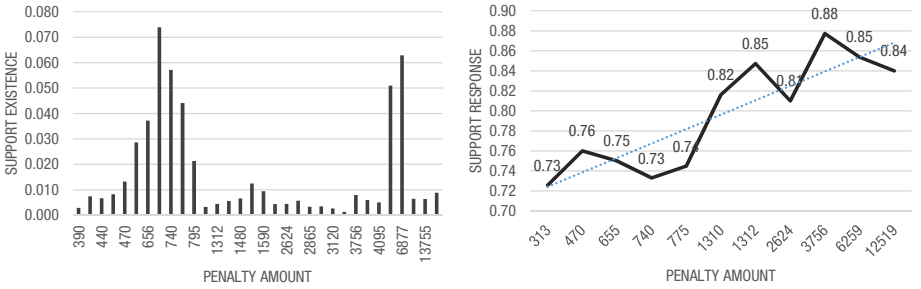
We then discovered *MP-Existence* and *MP-Response* constraints. In particular, we incorporate data in the form of the data attribute *Amount* that indicates the amount of money an accused person has to pay as a penalty. First, we mined the event log for *MP-Existence* constraints on the activity *add penalty*. The results (Fig.1a) show the support of the existence of the activity in correlation with the occurring values of the penalty amount. The distribution reveals that, in most of the cases, when *add penalty* was performed, the penalty amount had a value between 470 and 795. Furthermore, we discovered the influence of the penalty amount on the probability that the case is sent for credit collection by applying an *MP-Response* query for discovering activation conditions over the data attribute *Amount*. Fig. 1b shows that the support of *MP-Response* constraints between *add penalty* and *send for credit collection* on average increases with an increasing amount of the penalty, i.e, the higher the penalty amount is, the lower the probability that the fine is paid is.

5.2 Time Conditions: Building Permit Process in Municipalities

Next, we applied our approach to the event logs pertaining to an administrative process in five Dutch municipalities for evaluating the time differences between activations and correlated targets of a constraint. The different event log files[5] contain all building permit applications over a period of approximately four years.

[4] DOI: 10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5

[5] DOI: 10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1



(a) *MP-Existence(add penalty)* (b) *MP-Response(add penalty, send for credit collection)*

Fig. 1: Support values of *MP-Existence* and *MP-Response* constraints.

	Activity A	Activity B	TimeDiff [d]	Support	Confidence
MunA	Assessment Completed	Generating Decision	8	1	0.06
	Register Date Request	Phase Appl. Received	5	0.92	0.92
MunB	Assessment Completed	Generating Decision	18	1	0.90
	Register Date Request	Phase Appl. Received	25	1	1
MunC	Assessment Completed	Generating Decision	5	0.97	0.78
	Register Date Request	Phase Appl. Received	15	1	1
MunD	Assessment Completed	Generating Decision	6	1	0.06
	Register Date Request	Phase Appl. Received	3	0.8	0.82
MunE	Assessment Completed	Generating Decision	12	0.98	0.34
	Register Date Request	Phase Appl. Received	6	0.95	0.95

Table 4: *MP-Response* constraints discovered with average time differences.

The processes in the five municipalities are almost identical. The event log *MunA* contains 1,199 cases, *MunB* 832 cases, *MunC* 1,409 cases, *MunD* 1,053 cases and *MunE* 1,156 cases. For each event log, we executed an *MP-Response* query that discovers response constraints considering the time perspective and evaluating the time difference (with the granularity of days) between activation and target activities. Table 4 shows an excerpt of the results for each log, i.e., the constraints over activity pairs (*assessment of content completed, generating decision environmental permit*) and (*register submission date request, phase application received*). There are two conclusions that can be drawn from these results:

(i) The time between activation and target activities in the different event logs is significantly different. While for *MunA* and *MunD* the average time from the completion of the content assessment to the generation of the permit decision is only 8 and 6 days respectively, for *MunB* the difference is 18 days on average. A similar observation can be made for the time between the registration of the request date and the notice of application received. Here, the difference is on average even bigger between *MunB* (25 days) and *MunA* (5 days), *MunD* (3 days) and *MunE* (6 days).

(ii) There is a clear discrepancy between the constraint fulfillment (support) in case of big and small time differences between activation and target activities.

Activity A	Activity B	Support	Confidence
Calcium Speed Test	Receiving Laboratory Analysis	0.95	0.13
Chloride Speed Test	Receiving Laboratory Analysis	0.96	0.06
Bicarbonat Test	Receiving Laboratory Analysis	0.96	0.22
Phosphate Speed Test	Receiving Laboratory Analysis	0.96	0.03

Table 5: Standard *response* constraints for selected activities.

Activity A	Activity B	Resource(B)	Support	Confidence
Calcium Speed Test	Rec. Lab Analysis	Gen. Lab	0.91	0.12
Chloride Speed Test	Rec. Lab Analysis	Gen. Lab.	0.96	0.06
Bicarbonat Test	Rec. Lab Analysis	Gen. Lab.	0.96	0.22
Phosphate Speed Test	Rec. Lab Analysis	Gen. Lab.	0.96	0.03

Table 6: Target resource conditions extracted with *MP-Response*.

Consider the response constraints between the registration of the request date and the notice of application received. In those municipalities where the time difference between activation and target activity is high, i.e., *MunB* (25 days) and *MunC* (15 days), the constraint has been fulfilled in every case (*support=1*). For *MunA* (5 days, support=0.92), *MunD* (3 days, 0.8) and *MunE* (6 days, 0.95) on the other hand, the time differences are lower and the constraint has only been fulfilled in a considerably smaller amount of cases. A potential conclusion might be that a more thorough and systematic way of work leads to a higher degree of constraint satisfaction, i.e., more compliant process executions.

5.3 Target and Correlation Conditions: Hospital Log

Finally, we validated the approach with an event log[6] that records the treatment of patients diagnosed with cancer from a large Dutch hospital. The event log contains 1,143 cases and 150,291 events distributed across 623 activities.

We first queried the event log for standard *response* constraints without considering the data perspective. Then, we discovered conditions considering the the *Resource* attribute of the target activity (denoted as *Resource(B)*) using an *MP-Response* query. Finally, we discovered correlation conditions taking into consideration the resources of both activation (denoted as *Resource(A)*) and target activities by querying the log with an *MP-Response* query. All queries have been specified with the following thresholds: *minSupp=0.9* and *minConf=0.02*. We explain the results by means of four constraints referring to different blood test activities and the activity *receiving laboratory analysis*. Table 5 shows the results for standard *response*. After tests for *chloride, bicarbonate* and *phosphate* the *laboratory analysis results* have been received in 96% of all cases, while for *calcium* they have been received in 95% of the cases. Note, that these constraints do not consider the data perspective.

Let us now take into account the resources performing activities. Table 6 shows the target conditions for these constraints. The results reveal that after most of the considered blood tests the receipt of the analysis results has always been performed by *General Lab Clinical Chemistry*. This is highlighted by the

[6] DOI: `10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54`

Activity A	Activity B	Res(A)	Res(B)	Support	Confidence
Calcium Speed Test	Rec. Lab Analysis	Gen. Lab.	Gen. Lab.	0.91	0.12
Chloride Speed Test	Rec. Lab Analysis	Gen. Lab.	Gen. Lab.	0.96	0.06
Bicarbonat Test	Rec. Lab Analysis	Gen. Lab.	Gen. Lab.	0.96	0.22
Phosphate Speed Test	Rec. Lab Analysis	Gen. Lab.	Gen. Lab.	0.96	0.03

Table 7: Correlation resource conditions extracted with *MP-Response*.

Activity	Resource	Cases	Support	Confidence
Receiving Laboratory Analysis	General Lab Clinical Chemistry	797	0.697	0.697
	Medical Microbiology	315	0.276	0.276
	PharmacyLaboratory	5	0.004	0.004
	Special Lab Radiology	3	0.002	0.002
	Special Lab Nurosensory	2	0.001	0.001

Table 8: Resource-based *MP-Existence* constr. for *Receiving Laboratory Analysis*.

fact that the support values of the *MP-Response* constraints are identical to the standard *response* constraints, i.e., *support=0.96*. Only in case of *calcium* the support decreased to 0.91, which indicates that in this case also other resources performed the target activity. An even more specific result set is given in Table 7 that shows the correlation conditions for the constraints, i.e., the support values in case of identical resources for both activities. The results for the *MP-Response* query highlight that in most of the cases wherein the analysis results have been received after the blood tests, the performing resources of the two corresponding activities are identical and equal to *General Lab Clinical Chemistry*. For *calcium*, again, this fact only applies to 91% of the cases. In order to get an insight into the set of resources involved in activity *receiving laboratory analysis*, we applied an *MP-Existence* query. The results in Table 8 show a diverse set of resources performing this activity, which explains why the support is lower in case of *calcium*. The evaluation reported hitherto shows the range of disclosing previously unknown relationships between behavioral constraints and all the additional perspectives that can be analyzed using the information contained in an event log.

6 Related Work

Several approaches have been proposed in the literature for the discovery of declarative process models. In [19], the authors present an approach that allows the user to select from a set of predefined Declare templates the ones to be used for the discovery. Maggi et al. propose an evolution of this approach in [20] to improve performances. Other approaches to improve the performances of the discovery task are presented in [10,31]. Additionally, there are post-processing approaches that aim at simplifying the resulting Declare models in terms of redundancy elimination [21,9,7] and disambiguation [3].

The approaches proposed in [14,6] allow for the specification of rules that go beyond the traditional Declare templates. An approach similar to the SQL-based one used in this paper is presented in [26] and is based on temporal logic query checking. In [30], the authors define *Timed Declare*, an extension of Declare that relies on timed automata. In [17], an approach for analyzing event logs with

Timed Declare is proposed. The *DPILMiner* [28] exploits a discovery approach to incorporate the resource perspective and to mine for a set of predefined resource assignment constraints. In [22], the authors introduce for the first time a data-aware semantics for Declare and [18] first covered the data perspective in declarative process discovery, although this approach only allows for the discovery of *discriminative* activation conditions.

7 Conclusions

In this paper, we proposed a framework for the discovery of MP-Declare models. We implemented our approach using SQL queries tailored to analyze a process from different perspectives, e.g., control flow, data attributes as well as organizational and time perspectives. The approach has been validated with several real-life event logs provided by a large academic hospital, by five Dutch municipalities and by an Italian local police office for managing fines for road traffic violations. The application of our technique to these real-life process event logs revealed dependencies and correlations with additional parameters such as data values, time conditions and resource specifications.

The approach at hand serves as a building block for a variety of extensions in future work. For example, we plan to ease the interpretation of multi-perspective mining results by applying preprocessing methods to event logs and postprocessing methods to the discovered multi-perspective models. Furthermore, the full specification of a new, domain-independent and user-customizable SQL-based framework for mining MP-Declare constraints is in our plans for future research.

References

1. van der Aalst, W.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. van der Aalst, W., Pesic, M., Schonenberg, H.: Declarative Workflows: Balancing Between Flexibility and Support. Computer Science - R&D pp. 99–113 (2009)
3. Bose, J.C., Maggi, F.M., van der Aalst, W.: Enhancing Declare Maps Based on Event Correlations. In: Business Process Management. pp. 97–112 (2013)
4. Burattin, A., Maggi, F.M., van der Aalst, W.M., Sperduti, A.: Techniques for a Posteriori Analysis of Declarative Processes. In: EDOC. pp. 41–50. IEEE, Beijing (Sep 2012)
5. Burattin, A., Maggi, F.M., Sperduti, A.: Conformance checking based on multi-perspective declarative process models. CoRR abs/1503.04957 (2015)
6. Chesani, F., Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Exploiting Inductive Logic Programming Techniques for Declarative Process Mining. Transactions on Petri Nets and Other Models of Concurrency (ToPNoC), Special Issue on Concurrency in Process-Aware Information Systems 5460, 278–295 (2009)
7. Di Ciccio, C., Maggi, F.M., Montali, M., Mendling, J.: Ensuring model consistency in declarative process discovery. In: BPM. Lecture Notes in Computer Science, vol. 9253, pp. 144–159. Springer (2015)
8. Di Ciccio, C., Mecella, M.: A two-step fast algorithm for the automated discovery of declarative workflows. In: CIDM. pp. 135–142. IEEE (April 2013)
9. Di Ciccio, C., Mecella, M.: On the discovery of declarative control flows for artful processes. ACM TMIS 5(4), 24:1–24:37 (2015)
10. Di Ciccio, C., Schouten, M.H.M., de Leoni, M., Mendling, J.: Declarative process discovery with MINERful in ProM. In: BPM Demos. pp. 60–64 (2015)

11. van Dongen, B.F., Shabani, S.: Relational XES: data management for process mining. In: CAiSE Forum 2015. pp. 169–176 (2015)
12. Hildebrandt, T.T., Mukkamala, R.R., Slaats, T., Zanitti, F.: Contracts for cross-organizational workflows as timed dynamic condition response graphs. J. Log. Algebr. Program. 82(5-7), 164–185 (2013)
13. Kupferman, O., Vardi, M.Y.: Vacuity Detection in Temporal Model Checking. International Journal on Software Tools for Technology Transfer 4, 224–233 (2003)
14. Lamma, E., Mello, P., Riguzzi, F., Storari, S.: Applying inductive logic programming to process mining. In: Inductive Logic Programming, 17th International Conference, ILP 2007, Corvallis, OR, USA, June 19-21, 2007, Revised Selected Papers. pp. 132–146 (2007)
15. de Leoni, M., van der Aalst, W.M.P., Dees, M.: A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. Inf. Syst. 56, 235–257 (2016)
16. Maggi, F.M.: Declarative process mining with the declare component of prom. In: BPM Demo sessions 2013, 26-30, 2013 (2013)
17. Maggi, F.M.: Discovering metric temporal business constraints from event logs. In: BIR. Lecture Notes in Business Information Processing, vol. 194, pp. 261–275. Springer (2014)
18. Maggi, F.M., Dumas, M., García-Bañuelos, L., Montali, M.: Discovering data-aware declarative process models from event logs. In: BPM. pp. 81–96 (2013)
19. Maggi, F.M., Mooij, A., van der Aalst, W.: User-Guided Discovery of Declarative Process Models. In: CIDM. pp. 192–199 (2011)
20. Maggi, F., Bose, J., van der Aalst, W.: Efficient discovery of understandable declarative models from event logs. In: CAiSE. pp. 270–285 (2012)
21. Maggi, F., Bose, R., van der Aalst, W.: A knowledge-based integrated approach for discovering and repairing declare maps. In: CAiSE (2013)
22. Montali, M., Chesani, F., Mello, P., Maggi, F.M.: Towards data-aware constraints in declare. In: SAC. pp. 1391–1396. ACM (2013)
23. Montali, M., Pesic, M., van der Aalst, W.M.P., Chesani, F., Mello, P., Storari, S.: Declarative Specification and Verification of Service Choreographies. ACM Transactions on the Web 4(1) (2010)
24. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: Declare: Full support for loosely-structured processes. In: IEEE International EDOC Conference 2007. pp. 287–300 (2007)
25. Pichler, P., Weber, B., Zugal, S., Pinggera, J., Mendling, J., Reijers, H.A.: Imperative versus declarative process modeling languages: An empirical investigation. In: BPM Workshops. pp. 383–394 (2011)
26. Räim, M., Di Ciccio, C., Maggi, F.M., Mecella, M., Mendling, J.: Log-based understanding of business processes through temporal logic query checking. In: OTM Conferences. vol. 8841, pp. 75–92. Springer (2014)
27. Rozinat, A., Mans, R.S., Song, M., van der Aalst, W.M.P.: Discovering simulation models. Inf. Syst. 34(3), 305–327 (2009)
28. Schönig, S., Cabanillas, C., Jablonski, S., Mendling, J.: A Framework for Efficiently Mining the Organisational Perspective of Business Processes. Decision Support Systems (2016)
29. Schönig, S., Rogge-Solti, A., Cabanillas, C., Jablonski, S., Mendling, J.: Efficient and Customisable Declarative Process Mining with SQL. In: CAiSE (2016)
30. Westergaard, M., Maggi, F.M.: Looking into the future: Using timed automata to provide a priori advice about timed declarative process models. In: OTM. LNCS, vol. 7565, pp. 250–267. Springer (2012)
31. Westergaard, M., Stahl, C., Reijers, H.: UnconstrainedMiner: Efficient Discovery of Generalized Declarative Process Models. BPM CR, No. BPM-13-28 (2013)
32. Zeising, M., Schönig, S., Jablonski, S.: Towards a Common Platform for the Support of Routine and Agile Business Processes. In: Collaborative Computing: Networking, Applications and Worksharing (2014)