Compliance Monitoring of Multi-Perspective Declarative Process Models

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

Marco Montali
Free University of Bolzano
Bolzano, Italy
montali@inf.unibz.it

Ubaier Bhat
University of Tartu
Tartu, Estonia
bhat@ut.ee

***Abstract*—Checking the compliance of a business process execution with respect to a set of regulations is an important issue in several settings. A common way of representing the expected behavior of a process is to describe it as a set of business constraints. Through monitoring facilities, it is possible to continuously determine the state of constraints on the current process execution, and to promptly detect violations at runtime. A plethora of studies has demonstrated that in several settings business constraints can be formalized in terms of temporal logic rules. However, in most of the existing works, the process behavior is mainly modeled in terms of control-flow rules, neglecting other equally important perspectives like data or time. In this paper, we overcome this limitation by presenting a novel monitoring approach based on MP-Declare, a multi-perspective version of the declarative process modeling language Declare. The approach has been implemented in the process mining tool ProM and has been experimented using artificial and real-life event logs.**

I. Introduction

One very important functionality that any process-aware information system should be able to support is *compliance monitoring* [1], i.e., the ability to verify at runtime whether the actual flow of work is compliant with the intended business process model. A common way of representing monitoring requirements that capture the expected behavior of a process, is by using declarative, business constraints. A plethora of studies has demonstrated that, in several settings, business constraints can be formalized in terms of temporal logic rules. Within this paradigm, the *Declare* constraint-based process modeling language [2], [3], [4] has been introduced as a front-end language to specify business constraints based on Linear Temporal Logic (LTL) over finite traces [5], [6]. The advantage of this approach is that the corresponding automata-theoretic characterization of this logic can be exploited to provide advanced monitoring facilities to continuously determine the state of constraints based on the events dynamically generated during the monitored process execution [7], [8], and to detect violations at the earliest moment possible by identifying conflicting constraints [9]. The disadvantage is that such constraints only account for the process control-flow, disregarding other important aspects such as event data and corresponding conditions, as well as temporal conditions.

Previous works have defined an extension of Declare incorporating these additional perspectives into the constraint language *MP-Declare* (*Multi-Perspective Declare*) [10]. However, from the foundational perspective, when monitoring MP-Declare constraints, the main issue is that early detecting violations that cannot be ascribed to a single constraint, but instead arise from the interplay between the current trace and multiple, conflicting constraints, is in general an undecidable problem. This comes from the fact that capturing data and control-flow perspectives requires to resort to first-order temporal logics. In [11], a data-aware variant of Declare is formalized and monitored using a reactive version of the Event Calculus [12]. The approach only monitors single constraints against the monitored trace, and is consequently unable to detect conflicts. In [13], the automata-theoretic approach used for Declare is extended to handle data conditions, by lazily binding copies of each constraint automaton when new event data are received. However, the approach does not deal with temporal conditions and works under the assumption that the domains of the data values are fixed and finite.

In this paper, we provide a completely new approach for monitoring MP-Declare, including sev-

2325-6362/19/$31.00 ©2019 IEEE
DOI 10.1109/EDOC.2019.00027

eral data types and corresponding conditions (including numerical data types), as well as temporal conditions. Specifically, our approach is grounded on Integer Linear Programming (ILP) [14]. For each MP-Declare template, we define ad-hoc, ILP-based procedures to monitor constraints and to detect conflicts involving constraints based on such template. The time complexity of each such procedures depends on the considered template, but is upper bounded in the worst case by a quadratic function.

The resulting framework has been implemented in the well-established process mining tool ProM (see http://www.promtools.org/). We experimentally assess its validity both on artificial and real-life event logs.

The rest of the paper is structured as follows. In Section II, we give some background notions needed to understand the paper. In Section III, we show how we can monitor individual MP-Declare constraints. In Section IV, we introduce our ILP-based approach to detect conflicting constraints. Section V shows how the presented approach works on synthetic and real life processes. Section VI analyzes the limitations of the proposed approach. Section VII discusses related work, while Section VIII concludes the paper and spells out directions for future work.

II. Preliminaries

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

A. Declare

Declare is a declarative process modeling language introduced in [15]. A Declare model consists of a set of constraints applied to (atomic) activities. Constraints, in turn, are based on templates. Templates are abstract parameterized patterns, and constraints are their concrete instantiations on real activities. Templates have a user-friendly graphical representation understandable to the user. Their semantics can be formalized using different logics [5], the main one being LTL for finite traces. Each constraint inherits the graphical representation and semantics from its template. The major benefit of using templates is that analysts do not have to be aware of the underlying logic-based formalization to understand the models. They work with the graphical representation of templates, while the underlying formulas remain hidden. Table I reports the main Declare templates, their graphical representation and their semantics. The reader can refer to [15] for a full description of the language.

TEMPLATE	NOTATION	DESCRIPTION
existence(1,A) existence(2,A) ... existence(n,A)	$n..*$ A	A has to occur at least once A has to occur at least twice ... A has to occur at least n times
absence(1,A) absence(2,A) ... absence(n,A)	$0..n$ A	A can never happen A can happen at most once ... A can happen at most n-1 times
exactly(1,A) exactly(2,A) ... exactly(n,A)	n A	A has to occur exactly once A has to occur exactly twice ... A has to occur exactly n times
init(A)	$init$ A	Each instance has to start with the execution of A
resp. existence(A,B)	A — B	If A occurs, B must occur as well
response(A,B)	A → B	If A occurs, B must eventually follow
precedence(A,B)	A ← B	B can occur only if A has occurred before
alternate response(A,B)	A ⇒ B	If A occurs, B must eventually follow, without any other A in between
alternate precedence(A,B)	A ⇐ B	B can occur only if A has occurred before, without any other B in between
chain response(A,B)	A ⇒ B	If A occurs, B must occur next
chain precedence(A,B)	A ⇐ B	B can occur only immediately after A
not resp. existence(A,B)	A ∦ B	If A occurs, B cannot occur
not response(A,B)	A ↛ B	If A occurs, B cannot eventually follow
not precedence(A,B)	A ↚ B	A cannot occur before B
not chain response(A,B)	A ⇏ B	If A occurs, B cannot occur next
not chain precedence(A,B)	A ⇍ B	A cannot occur immediately before B

Table I
Graphical notation and semantics of Declare templates.

An *activation* of a constraint in a trace is an event whose occurrence imposes, because of that constraint, some obligations on another event (the rule *target*) in the same trace. For example, for the *response* constraint between A and B, A is an activation, because the execution of A forces B to be executed eventually. Event B is a target. An activation of a constraint can be a *fulfillment* or a *violation* for that constraint. When a trace is perfectly compliant with respect to a constraint, every activation of the constraint in the trace leads to a fulfillment. Consider, again, the *response* constraint between A and B. In trace $\langle A, A, B, C\rangle$, the constraint is activated and fulfilled twice, whereas, in trace $\langle A, B, C, B\rangle$, the same constraint is activated and fulfilled only once. On the other hand, when a trace is not compliant with respect to a constraint, an activation of the constraint

in the trace can lead to a fulfillment but also to a violation (at least one activation leads to a violation). In trace $\langle A, B, A, C\rangle$, for example, the response constraint is activated twice, but the first activation leads to a fulfillment (eventually B occurs) and the second activation leads to a violation (B does not occur subsequently).

B. MP-Declare

The Declare templates mentioned in the previous section only capture constraints related to control-flow. MP-Declare, whose semantics was first introduced in [10], takes into consideration also data and time perspectives. These perspectives are captured through three types of conditions, *activation condition*, *correlation condition* and *temporal condition*, specified in a constraint with the format [Activation] [Correlation] [Temporal]. Data which relates to the activation of a constraint is specified with the format $A.data$. Similarly, data related to the target is specified with the format $T.data$.

1) Activation conditions: These conditions are used to specify when a constraint is considered to be active. For example, constraint $absence(AbandonShip)$ without data conditions means that event $AbandonShip$ should never occur. However, if we add an activation condition to it as follows:

$$absence(AbandonShip)$$
$$[A.rank = \text{“}captain\text{”}]$$

this will imply that an event $AbandonShip$ can occur, but not if rank is equal to captain.

2) Correlation conditions: Correlation conditions are conditions that should be valid when the target occurs and can involve data related to both activation and target so that they can be used to specify correlations between two events. For example, the constraint:

$$response(PaymentReceived, DispatchOrder)$$
$$[A.pendingBalance = 0][T.id = A.id]$$

will only be activated if event $PaymentReceived$ occurs with $pendingBalance = 0$. If this happens, then it requires $DispatchOrder$ to eventually occur with id equal to the id of $PaymentReceived$.

3) Temporal conditions: Temporal conditions are used to specify time distances between two events. The format for specifying temporal conditions is: [$upperBound$,$lowerBound$,$unit$], where $unit$ can be s for seconds, m for minutes, h for hours and d for days. For example, the rule:

Step	Condition	Range	Has Solution?
0	init	$m < x < M$	true
1	$x < 10$	$m < x < 10$	true
2	$x > 0$	$0 < x < 10$	true
3	$x > 5$	$5 < x < 10$	true
4	$x < 100$	$5 < x < 10$	true
5	$x = 9$	$x = 9$	true
6	$x < 8$	*no solutions*	false

Table II
AN EXAMPLE OF AN ILP PROBLEM

$$response(PaymentReceived, DispatchOrder)$$
$$[A.pendingBalance = 0][T.id = A.id][0, 5, m]$$

is interpreted as: after that $PaymentReceived$ occurs with $pendingBalance = 0$, $DispatchOrder$ must eventually occur with id equal to the id of $PaymentReceived$ and within 5 minutes after the occurrence of $PaymentReceived$.

C. Integer Linear Programming

Integer Linear Programming (ILP) [14] is a method to achieve the optimal solution in a mathematical model in which requirements are represented in the form of linear relationships. Consider the example in Table II. Our aim is to find a real number x when we are giving certain conditions. In the first step, we have to define our default maximum and minimum possible values. At this stage, we can say that the solution for x is between m and M, i.e., $m < x < M$. If we need a first condition to be valid stating that x should be less than 10, our solution for x will be $m < x < 10$. Then, if we assume that x should be greater than 0, our solution for x will become $0 < x < 10$. If x should be greater than 5, our solution for x will become $5 < x < 10$. We can keep adding conditions to x which can reduce the range for x and bring us closer to its actual value. However, not all the new conditions will change the range for x. For example, if we say that x should be < 100 our range for x will not change because, in order to satisfy the previous conditions, x should already be less than 100. We can also have conditions that fix the value of x. Fore example, suppose that x should be equal to 9. Now, the minimum possible solution for x is 9 and the maximum possible solution for x is 9. If we add conditions that contradict the previous ones, we will no longer have a solution for x. For example, if we say that x must be less than 8, this condition will contradict our previous set of conditions.

Response

template.opening()

```
1 do nothing
```

template.fulfillment$(e, pending, fullfilments, T, \sigma_a, \sigma_c, \sigma_t)$

```
1 if π_activity(e) ∈ T then
2     foreach act ∈ pending do
3         if
            verify(σ_c, act, e) and verify(σ_t, act, e)
          then
4             pending ← pending\{act}
5             fulfillments ←
                fulillments ∪ {act}
```

template.violation()

```
1 do nothing
```

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

```
1 if π_activity(e) ∈ A and verify(σ_a, e) then
2     pendng ← pending ∪ {act}
```

tmplate.closing$(pending,\ violations)$

```
1 foreach act ∈ pending do
2     pending ← pending\{act}
3     violations ← violations ∪ {act}
```

where: $e = current\ event$
$A = non\ empty\ set\ of\ activations$
$T = non\ empty\ set\ of\ targets$
$violations = set\ of\ violations$
$fulfillments = set\ of\ fulfillments$
$pending = set\ of\ pending$
$\sigma_a = activation\ condition$
$\sigma_c = corellation\ condition$
$\sigma_t = time\ condition$

Table III
FRAMEWORK OPERATIONS FOR RESPONSE

III. COMPLIANCE MONITORING OF INDIVIDUAL MP-DECLARE CONSTRAINTS

We now illustrate how it is possible to verify if a process execution trace is compliant with an individual MP-Declare constraint. Our framework used for compliance monitoring takes as input a trace and a constraint, and is composed of the operations:

- opening: this method is called once per trace, before starting the analysis of the first event of the trace;
- fulfillments: this method is called for each event of the trace and is supposed to return the set of fulfillments of the constraint that have been observed so far;
- violations: this method is called for each event of the trace and is supposed to return the set of violations of the constraint that have been observed so far;
- activations: this method is called for each event of the trace and is supposed to update the set of activations of the constraint that have been observed so far;
- closing: this procedure is called once per trace, after all the events have been analyzed.

This set of operations can be instantiated for different Declare templates. For example, for the response template the operations are instantiated as in Table III:

- opening: not used;
- fulfillments: this procedure checks whether the input event refers to a target. If this is the case, then all pending activations that can be correlated to this target (in case the time and the correlation conditions are satisfied) become fulfillments;
- violations: not used;
- activations: the activation procedure checks whether the input event refers to an activation of the constraint and the activation condition is satisfied (in this case the event has to be added to the set of pending activations);
- closing: all pending activations that do not have a corresponding target when the entire trace has been processed become violations.

The current state of a trace with respect to a constraint is described using a four valued semantics. These values are *possibly satisfied*, *possibly violated*, *permanently satisfied* or *permanently violated*. These semantic values can be interpreted as follows:

- **Possibly satisfied**: means that the constraint is currently satisfied, but might be violated in the future.
- **Possibly violated**: means that constraint is currently violated, but might be satisfied in the future.
- **Permanently satisfied**: means that the constraint is permanently satisfied and can no longer become violated in the future.
- **Permanently violated**: means that the constraint is permanently violated and can no longer become satisfied in the future.

The semantic value for the current state of a trace depends on the type of constraint. Table IV

Template	Poss.viol	Poss.sat	Viol	Sat
response responded existence co-existence	$!done \wedge (p > 0)$	$!done \wedge (p = 0)$	$done \wedge (p > 0)$	$done \wedge (p = 0)$
not response not chain response precedence not precedence absence absence2 absence3 chain precedence not chain precedence alternate precedence	-	$!done \wedge (v = 0)$	$v > 0$	$done \wedge (v = 0)$
init	-	-	$v > 0$	$f > 0$
existence existence2 existence3	$!done \wedge (a < card)$	-	$done \wedge (a < card)$	$a >= card$
exactly1 exactly2	$!done \wedge (a < card)$	$!done \wedge (a = card)$	$(a > card) \vee (done \wedge (a < card))$	$done \wedge (a = card)$
not responded existence not succession not chain succession not co-existence	-	$!done \wedge (v = 0)$	$v > 0$	$done \wedge (v = 0)$
succession chain succession alternate succession chain response alternate response	$!done \wedge (v = 0) \wedge (p > 0)$	$!done \wedge (v = 0) \wedge (p = 0)$	$(v > 0) \vee (done \wedge (p > 0))$	$done \wedge (v = 0) \wedge (p = 0)$
choice	$!done \wedge (a = 0)$	-	$done \wedge (a = 0)$	$a > 0$
exclusive choice	$!done \wedge (a = 0)$	$!done \wedge (v = 0) \wedge (a > 0)$	$(v > 0) \vee (done \wedge (a = 0))$	$done \wedge (v = 0) \wedge (a > 0)$

Table IV

CRITERIA FOR SEMANTIC VALUES IDENTIFICATION. $v = |violations|$ IN THE CURRENT PREFIX, $f = |fulfillments|$ IN THE CURRENT PREFIX, $p = |pending|$ IN THE CURRENT PREFIX, a IS THE TOTAL NUMBER OF ACTIVATIONS IN THE CURRENT PREFIX, $card$ IS THE CARDINALITY OF AN EXISTENCE TEMPLATE (1 FOR EXISTENCE, EXACTLY1; 2 FOR EXISTENCE2, EXACTLY2; 3 FOR EXISTENCE3), $done$ CHECKS IF THE LAST EVENT OF THE CURRENT TRACE HAS BEEN EXECUTED.

Template	Condition
response	when $p > 0$, for each pending activation, an ILP problem is instantiated using the correlation condition. When the activation becomes fulfilled, the corresponding ILP problem is deleted.
not response	when it is activated, the negation of the correlation condition is added to all active ILP problems.
precedence	can never be in conflict.
not precedence	can never be in conflict.
init	can never be in conflict.
absence	the negation of the activation condition is always added to all active ILP problems.
existence	when $f < 1$, an ILP problem is instantiated using the activation condition. When the constraint becomes fulfilled, the corresponding ILP problem is deleted.
responded existence	when $p > 0$, for each pending activation, an ILP problem is instantiated using the correlation condition. When the activation becomes fulfilled, the corresponding ILP problem is deleted.
not responded existence	when it is activated, the negation of the correlation condition is added to all active ILP problems.
chain response	when there is a pending activation, an ILP problem is instantiated using the correlation condition. When the pending activation becomes fulfilled, the corresponding ILP problem is deleted.
not chain response	can only be in conflict with a chain response with the same parameters in case both are activated.
chain precedence	can never be in conflict.
not chain precedence	can never be in conflict.
alternate response	when $p > 0$, for each pending activation, an ILP problem is instantiated using the correlation condition. When the activation becomes fulfilled, the corresponding ILP problem is deleted.
alternate precedence	can never be in conflict.

Table V

CRITERIA FOR ILP PROBLEM INSTANTIATION AND UPDATE

shows the criteria to determine the state of a trace for each constraint type.

IV. EARLY DETECTION OF CONFLICTING CONSTRAINTS

Consider constraints $absence(B)$ and $response(A, B)$ without data. The absence constraint says that activity B should never occur in a trace. The response constraint says that if A occurs then B must eventually occur in the trace. As soon as A occurs, these constraints can still be individually satisfied, but they are *conflicting* meaning that only one of them can be fulfilled at the end of the process execution. If B occurs in the trace after A, then the absence constraint will be violated and, if B does not occur, then the response constraint will be violated.

Table VI shows the same constraints but now with data conditions. These constraints are no longer in conflict when the response constraint

id	Constraint	Activation Activity	Target Activity	Activation Condition	Correlation Condition	Time Condition
1	absence	B	-	$B.x = 3$	-	-
2	response	A	B	$A.x = 1$	$B.x = 4$	-

Table VI
EXAMPLE 1

Step	Condition	Range	Has Solution?
0	init	$m < B.x < M$	true
1	$B.x = 4$	$B.x = 4$	true
2	$B.x! = 3$	$B.x = 4$	true

Table VII
ILP PROBLEM FOR EXAMPLE 1

is activated. Indeed, once activated, the response constraint requires B to occur with $B.x = 4$, while the absence constraint forbids the execution of B with $B.x = 3$. Both the conditions:

$$(B.x = 4) \wedge !(B.x = 3)$$

can be satisfied at the same time and there is no conflict anymore. To check if two constraints are conflicting, every time an activation of one of the two constraint occurs, conditions like the ones mentioned above are used to update an ILP problem as explained in Table V. If one of the instantiated ILP problems does not have solutions, then the involved constraints are in conflict. For example, in this case, when A with $A.x = 1$ occurs, the ILP problem in Table VII is instantiated. Since this ILP problem has a solution, the constraints represented in it are not in conflict.

Table VIII shows the same constraints, but this time the conditions have been changed. The response constraint has a correlation condition that requires the execution of B with $B.x = 3$ once activated. However, the absence constraint forbids the execution of B with $B.x = 3$. The ILP problem instantiated starting from these conditions (shown in Table IX) has no solutions and, therefore, the constraints are in conflict.

In the previous example, the correlation condition of $response(A, B)$ only depends on data related to B. However, this condition can also specify a correlation with the activation. In the example in Table X, the occurrence of a conflict depends on the value of $A.x$. Unless $A.x = 3$, we do not have any conflict.

Consider now the example in Table XI. Here, we have more than two constraints. However, in this specific case, we can still detect conflicts using ILP. Assume that A occurs with $A.x = 2$. This activates constraint 3 and triggers the ILP

id	Constraint	Activation Activity	Target Activity	Activation Condition	Correlation Condition	Time Condition
1	absence	B	-	$B.x = 3$	-	-
2	response	A	B	$A.x = 1$	$B.x = 3$	-

Table VIII
EXAMPLE 2

Step	Condition	Range	Has Solution?
0	init	$m < B.x < M$	true
1	$B.x = 3$	$B.x = 3$	true
2	$B.x! = 3$	*no solutions*	false

Table IX
ILP PROBLEM FOR EXAMPLE 2

id	Constraint	Activation Activity	Target Activity	Activation Condition	Correlation Condition	Time Condition
1	absence	B	-	$B.x = 3$	-	-
2	response	A	B	$A.x > 0$	$B.x = A.x$	-

Table X
EXAMPLE 3

id	Constraint	Activation Activity	Target Activity	Activation Condition	Correlation Condition	Time Condition
1	absence	C	-	$C.x < 10$	-	-
2	response	B	C	$B.x > 5$	$C.x < B.x$	-
3	response	A	B	$A.x > 0$	$B.x = A.x$	-

Table XI
EXAMPLE 4

Step	Condition	Range	Has Solution?
0	init	$m < B.x < M$	true
1	$B.x = 2$	$B.x = 2$	true

Table XII
ILP PROBLEM FOR EXAMPLE 4 WHEN A OCCURS WITH $A.x = 2$

Step	Condition	Range	Has Solution?
0	init	$m < B.x < M$	true
1	$B.x = 6$	$B.x = 6$	true

Table XIII
ILP PROBLEM FOR EXAMPLE 4 WHEN A OCCURS WITH $A.x = 6$

Step	Condition	Range	Has Solution?
0	init	$m < B.x < M$	true
1	$B.x = 6$	$B.x = 6$	true
2	$C.x < 6$	$B.x = 6, C.x < 6$	true
3	$\neg(C.x < 10)$	*no solutions*	false

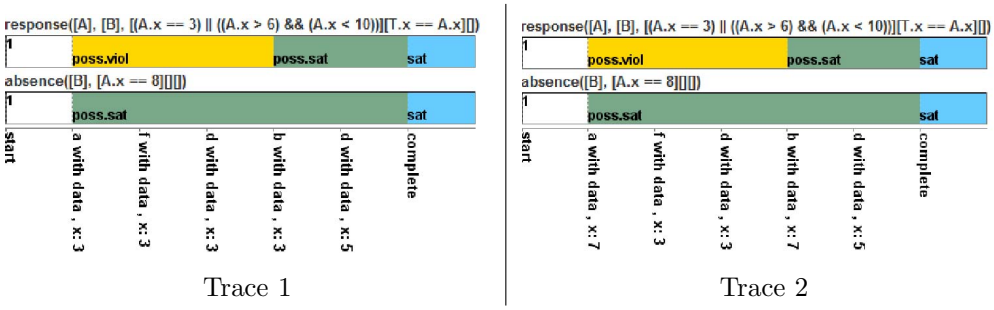
Table XIV
UPDATED ILP PROBLEM FOR EXAMPLE 4 WHEN A OCCURS WITH $A.x = 6$

problem shown in Table XII, which has a solution so that there is no conflict. Note that this is true because the execution of B with $B.x = 2$ does not activate any other constraint.

Suppose now that A occurs with $A.x = 6$. This triggers the ILP problem shown in Table XIII.

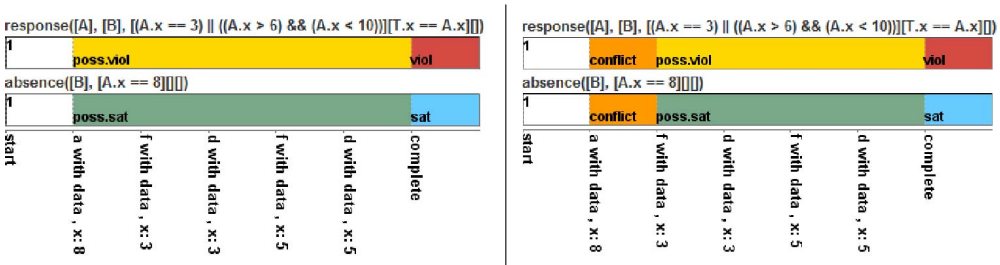
id	Constraint	Activation Activity	Target Activity	Activation Condition	Correlation Condition	Time Condition
1	*response*	A	B	$(A.x = 3) \vee ((A.x > 6) \wedge (A.x < 10))$	$B.x = A.x$	–
2	*absence*	B	–	$B.x = 8$	–	–

Table XV
MP-Declare model 1



Trace 1 Trace 2

Figure 1. Compliance monitoring output for MP-Declare model 1



Conflict detection *Off* Conflict detection *On*

Figure 2. Compliance monitoring output for MP-Declare model 1, Trace 3

id	Description	Template	A	T	Activation
R1	If "administratief tarief - eerste pol "occurs in a trace, it is always preceded by "vervolgconsult poliklinisch "and between "administratief tarief - eerste pol "and "vervolgconsult poliklinisch "you cannot find another "administratief tarief - eerste pol ";	alternate precedence	vervolgconsult_poliklinisch	administratief_tarief__eerste_pol	
R2	If "administratief tarief - eerste pol "or "vervolgconsult poliklinisch "occur in a trace, they always coexist;	responded existence	vervolgconsult_poliklinisch	administratief_tarief__eerste_pol	
R3	If "aanname laboratoriumonderzoek "occurs in a trace, it is always followed eventually by "ordertarief "and vice versa if "ordertarief "occurs, it is always preceded by "aanname laboratoriumonderzoek ";	response	aanname_laboratoriumonderzoek	ordertarief	
R4	If "administratief tarief - eerste pol "or "aanname laboratoriumonderzoek "occur in a trace, they always coexist;	responded existence	administratief_tarief__eerste_pol	aanname_laboratoriumonderzoek	
R5	If "aanname laboratoriumonderzoek "occurs in a trace, it is never followed by "vervolgconsult poliklinisch ";	not response	aanname_laboratoriumonderzoek	vervolgconsult_poliklinisch	
R12	If "administratief tarief - eerste pol "occurs in a trace and the condition (over case and event attributes) "(Age $\leq$ 70 && Producer code = SIOG) || (Diagnosis = Maligne neoplasma cervix uteri && Diagnosis code=106)) "holds, then "administratief tarief - eerste pol "is followed eventually by "albumine ";	response	administratief_tarief__eerste_pol	albumine	((A.Age <= 70) && (A.Producer_code = 'SIOG')) || ((A.Diagnosis = 'Maligne_neoplasma' cervix uteri) && (A.Diagnosis_code = 106))
R13	If "telefonisch consult "occurs in a trace and the condition (over case and event attributes) "(Treatment code=101) && (Producer code=SGAL || Producer code=SGNA) "holds, then "alkalische fosfatase -kinetisch- "does not occur in the same trace.	not responded existence	telefonisch_consult	alkalische_fosfatase__kinetisch_	(Treatment_code = 101) && ((A.Producer_code = 'SGAL') || (A.Producer_code = 'SGNA'))
R14	If event attribute "Section "is equal to "Section 4 "and event attribute "Specialism code "is equal to "86 ", the activity is executed by "org:group=General Lab Clinical Chemistry ";	absence	aanname_laboratoriumonderzoek	-	(A.Section = 'Section_4') && (A.Specialism_code = 86) && (A.org:group != 'General_Lab_Clinical_Chemistry')
R15	"bacteriologisch onderzoek met kweek -nie "is always executed by "org:group=Medical Microbiology ";	absence	bacteriologisc_ onderzoek_met_kweek__nie	-	(A.org:group != 'Medical_Microbiology')
R16	"cytologisch onderzoek - ectocervix - "and "histologisch onderzoek - biopten nno "are always executed by "org:group=Pathology ";	absence absence	histologisch_onderzoek__biopten_nno cytologisch_onderzoek__ectocervix__	-	(A.org:group != 'Pathology') (A.org:group != 'Pathology')

Table XVI
Rules and corresponding Declare Constraints

This time the solution of the ILP problem requires the occurrence of B with $B.x = 6$. Therefore, we have to update our ILP problem as shown in Table XIV and we have a conflict. Using the same procedure, we can recursively find conflicts caused by chained triggers. However, as shown in Section VI, this approach cannot be applied to deal with conflicts involving more than two constraints in the general case.

V. Evaluation

Our implementation (available at https://bitbucket.org/ubaierbhat/onlinedeclareanalyzer/overview) has been developed as an operational support provider [16], [17], [18] of the process mining tool ProM. For solving ILP problems, we use the Java based LP solver provided in [19].

In Figures 1 and 2, the interface of the tool is shown. Each box shows how the state of a constraint varies every time a new event is processed. The events in the trace under analysis are shown on the horizontal axis. To test the tool, we first used a synthetic log and the model shown in Table XV, using a log containing three traces with different data values.

In Trace 1 (shown in Figure 1, left), A occurs with $A.x = 3$. This means that the response constraint becomes possibly violated (indicated in yellow). The absence constraint is possibly satisfied (indicated in green). Therefore, at this stage we cannot say whether any or all the rules will be permanently satisfied or violated at the end of the trace. As can be seen in the figure, both rules become permanently satisfied at the end of the trace (indicated in blue) because eventually B occurs with $B.x = 3$.

In Trace 2 (shown in Figure 1, right), A occurs with $A.x = 7$. Again, this means that the response constraint becomes possibly violated. This triggers the ILP problem $(B.x = 7) \wedge !(B.x = 8)$ so that we cannot say whether any or all the rules will be permanently satisfied or violated. Since B with $B.x = 7$ eventually occurs, both rules become satisfied at the end of the trace.

In Trace 3 (shown in Figure 2), A occurs with $A.x = 8$. The response constraint becomes possibly violated, but, this time, this triggers the ILP problem $(B.x = 8) \wedge !(B.x = 8)$ so that a conflict is raised. At the end of the trace, indeed, one of the two constraints (the response constraint in this case) becomes violated (indicated in red). The figure shows the states of the monitored constraints when the mechanism for conflict detection is disabled (left) and enabled (right). When the conflict detection is enabled, a conflicting state is indicated in orange.

id	Constraint	Activation Activity	Target Activity	Activation Condition	Correlation Condition	Time Condition
1	*response*	A	B	$A.x = 3$	$B.x = A.x$	–
2	*precedence*	C	B	$B.x = 3$	$C.x = B.x$	–
3	*response*	C	D	$C.x = 3$	$D.x = C.x$	–
4	*not response*	D	B	$D.x = 3$	$B.x = D.x$	–

Table XVII
Counterexample 1

id	Constraint	Activation Activity	Target Activity	Activation Condition	Correlation Condition	Time Condition
1	*response*	A	B	–	$B.x = A.x$	–
2	*response*	C	B	–	$B.x = C.x$	–
3	*chain response*	B	D	–	$D.x = B.x$	–
4	*not response*	D	B	–	$B.x \neq D.x$	–

Table XVIII
Counterexample 2

A demo of the tool based on a real-life event log is available at https://www.youtube.com/watch?v=-X9b77cTB-U. The event log used was provided for the BPI Challenge 2011 [20] and records the treatment of patients diagnosed with cancer from a large Dutch hospital. The MP-Declare constraints used are rules R1–R5 and R12–R16 described in [1] and shown in Table XVI. This example shows that our monitoring framework can deal with any type of data. Integers and reals are natively handled by the LP solver [19]. We deal with strings by mapping them to integers.

VI. Limitations

The presented approach can be used to check conflicts between two constraints. When the constraints become more than two, in the general case, it is not possible to apply the proposed approach for conflict detection. We give here two examples that show cases in which the approach for finding conflicts caused by chained triggers introduced in Section IV cannot be applied.

The first example is shown in Table XVII. When A occurs with $A.x = 3$, even if the occurrence of B with $B.x = 3$ is required by $response(A, B)$ and forbidden by $not\ response(D, B)$ (leading to a conflict in the set of constraints), there is a sequence $(\langle A, C, B, D\rangle)$ fulfilling all the constraints.

Another example is given in Table XVIII. In this example, when A occurs with $A.x = 1$ and then C occurs with $C.x = 2$, B has to occur twice, once with $B.x = 1$ and once with $B.x = 2$. We can decide to satisfy $response(A, B)$ first. In this case, B has to occur with $B.x = 1$, which imposes to execute D with $D.x = 1$ immediately next

(according to *chain response*(B, D)). This, in turn, triggers *not response*(D, B), which forbids to execute B with $B.x \neq 1$. Therefore, we have a conflict. In fact, *response*(C, B) still requires to execute B with $B.x = 2$. The second possibility is that we decide to satisfy *response*(C, B) first. In this case, B has to occur with $B.x = 2$, which imposes to execute D with $D.x = 2$ immediately next (according to *chain response*(B, D)). This, in turn, triggers *not response*(D, B), which forbids to execute B with $B.x \neq 2$. Also in this case we have a conflict since *response*(A, B) still requires to execute B with $B.x = 1$.

VII. Related Work

In recent years, an increasing number of researchers are focusing on the analysis [10], [21], [22], [23], [24], [25] and execution [26], [27] of multi-perspective declarative models. In this context, several approaches have been presented for compliance monitoring of business processes with respect to a set of compliance rules. An in-depth analysis on the state of the art on compliance monitoring can be found in [1]. Here, we mention the main contributions in this field.

A compliance monitoring framework based on a reactive version [12] of the Event Calculus [28] is presented in [11]. Being based on the Event Calculus, it can easily accommodate explicit time constraints, as well as data-enriched events and corresponding data-aware conditions.

SeaFlows is a compliance checking framework that addresses design and runtime checking. It aims at encoding compliance states in an easily interpretable manner to provide advanced compliance diagnosis. The core concepts described in [29], [30] were implemented within the prototype named SeaFlows Toolset. With SeaFlows, compliance rules are modeled as *compliance rule graphs* (CRG).

In [31], the authors present an approach to combine data-related and control-flow related compliance rules that can be applied to business process models. An approach for compliance monitoring in business processes based on complex event processing is presented in [32].

With BPath [33], Sebahi proposes an approach for querying execution traces based on XPath. BPath implements a fragment of first-order hybrid logic and enables data-aware and resource-aware constraints.

[34] presents an approach based on Timed Propositional Temporal Logic for transforming high level regulatory constraints into REALM constraints that can be monitored during process runtime. The paper presents architectural considerations and an implementation of the transformation on the basis of a case study.

[35] provides an approach based on LTL-FO$^+$. LTL-FO$^+$ is a linear temporal logic augmented with full first-order quantification over the data inside a trace of XML messages that focuses on monitoring data-aware constraints over business processes. The data is part of the messages that are exchanged between the process activities.

MONPOLY [36], [37] is a runtime verification framework for security policies, specified in the logic MFOTL, a variant of LTL-FO with metric time. The high expressiveness of MFOTL allows one to express sophisticated compliance rules involving advanced temporal constraints, as well as data- and resource-related conditions.

None of the above mentioned approaches provides a full-fledged implementation of an approach for the compliance monitoring of individual MP-Declare constraints and their interplay. In [13], an approach similar to the one proposed in this paper is presented. However, this approach works under the assumption that the domains of the data values are fixed and finite. In our approach, we overcome this limitation, but, differently from [13], we cannot deal with conflicts involving more than two constraints in the general case.

VIII. Conclusion

In this paper, we have demonstrated that ILP can be used in a runtime setting to monitor business processes with complex rules on control-flow, data and time. We have shown that our approach can be used for early detection of conflicts of constraints. We were able to successfully validate our approach with synthetic and real-life logs.

For future work, we plan to better study the foundations of the presented approach, on the one hand by formally proving its correctness, and on the other to study its limitations, as well as the intrinsic limitations of the problem as such. We are currently working on a monitoring solution for MP-Declare implemented using the SAT technology that could be useful to overcome such limitations.

Acknowledgment

This research is partially supported by the Estonian Research Council Grant IUT20-55.

References

[1] L. T. Ly, F. M. Maggi, M. Montali, S. Rinderle-Ma, and W. M. P. van der Aalst, "Compliance monitoring in business processes: Functionalities, application, and tool-support," *Inf. Syst.*, vol. 54, pp. 209–234, 2015. 1, 8, 9

[2] W. M. P. van der Aalst, M. Pesic, and H. Schonenberg, "Declarative Workflows: Balancing Between Flexibility and Support," *Computer Science - R&D*, pp. 99–113, 2009. 1

[3] M. Westergaard and F. M. Maggi, "Declare: A tool suite for declarative workflow modeling and enactment," in *BPM (Demos)*, vol. 820. ceur-ws.org, 2011. 1

[4] M. Pesic, H. Schonenberg, and W. M. P. van der Aalst, "DECLARE: Full Support for Loosely-Structured Processes," in *EDOC 2007*, 2007, pp. 287–298. 1

[5] M. Montali, M. Pesic, W. M. P. van der Aalst, F. Chesani, P. Mello, and S. Storari, "Declarative Specification and Verification of Service Choreographies," *ACM Transactions on the Web*, vol. 4, no. 1, pp. 1–62, 2010. 1, 2

[6] G. De Giacomo, R. De Masellis, and M. Montali, "Reasoning on LTL on finite traces: Insensitivity to infiniteness," in *Proc. of the 28th AAAI Conference on Artificial Intelligence*, 2014, pp. 1027–1033. 1

[7] F. M. Maggi, M. Montali, M. Westergaard, and W. M. P. van der Aalst, "Monitoring business constraints with Linear Temporal Logic: An approach based on colored automata," in *BPM*, 2011, pp. 132–147. 1

[8] G. De Giacomo, R. De Masellis, M. Grasso, F. M. Maggi, and M. Montali, "Monitoring business metaconstraints based on LTL and LDL for finite traces," in *BPM*, 2014, pp. 1–17. 1

[9] F. M. Maggi, M. Westergaard, M. Montali, and W. M. P. van der Aalst, "Runtime verification of LTL-based declarative process models," in *Runtime Verification - RV 2011*, 2011, pp. 131–146. 1

[10] A. Burattin, F. M. Maggi, and A. Sperduti, "Conformance checking based on multi-perspective declarative process models," *Expert Syst. Appl.*, vol. 65, pp. 194–211, 2016. 1, 3, 9

[11] M. Montali, F. M. Maggi, F. Chesani, P. Mello, and W. M. P. van der Aalst, "Monitoring business constraints with the event calculus," *ACM Trans. on Intelligent Systems and Technology*, vol. 5, no. 1, p. 17, 2013. 1, 9

[12] F. Chesani, P. Mello, M. Montali, and P. Torroni, "A logic-based, reactive calculus of events," *Fundam. Inform.*, vol. 105, no. 1-2, pp. 135–161, 2010. 1, 9

[13] R. De Masellis, F. M. Maggi, and M. Montali, "Monitoring data-aware business constraints with finite state automata," in *ICSSP*, 2014, pp. 134–143. 1, 9

[14] A. Schrijver, *Theory of Linear and Integer Programming*. New York, NY, USA: John Wiley & Sons, Inc., 1986. 2, 3

[15] M. Pesic, "Constraint-Based Workflow Management Systems: Shifting Control to Users," Ph.D. dissertation, TU/e, 2008. 2

[16] F. M. Maggi, M. Montali, and W. M. P. van der Aalst, "An operational decision support framework for monitoring business constraints," in *FASE 2012*, 2012, pp. 146–162. 8

[17] M. Westergaard and F. M. Maggi, "Modeling and verification of a protocol for operational support using Coloured Petri nets," in *PETRI NETS 2011*, 2011, pp. 169–188. 8

[18] F. M. Maggi and M. Westergaard, "Designing software for operational decision support through Coloured Petri nets," *Enterprise IS*, vol. 11, no. 5, pp. 576–596, 2017. 8

[19] "Lp solver," http://lpsolve.sourceforge.net/5.5/. [Online]. Available: http://lpsolve.sourceforge.net/5.5/ 8

[20] 3TU Data Center, "BPI Challenge 2011," 2011, doi:10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54. 8

[21] F. M. Maggi, M. Dumas, L. García-Bañuelos, and M. Montali, "Discovering data-aware declarative process models from event logs," in *BPM*, 2013, pp. 81–96. 9

[22] R. P. J. C. Bose, F. M. Maggi, and W. M. P. van der Aalst, "Enhancing Declare maps based on event correlations," in *BPM*, 2013, pp. 97–112. 9

[23] S. Schönig, C. D. Ciccio, F. M. Maggi, and J. Mendling, "Discovery of multi-perspective declarative process models," in *ICSOC*, 2016, pp. 87–103. 9

[24] V. Leno, M. Dumas, and F. M. Maggi, "Correlating activation and target conditions in data-aware declarative process discovery," in *BPM*, 2018, pp. 176–193. 9

[25] C. Sturm, M. Fichtner, and S. Schönig, "Full support for efficiently mining multi-perspective declarative constraints from process logs," *Information*, vol. 10, no. 1, p. 29, 2019. 9

[26] M. Käppel, N. Schützenmeier, S. Schönig, L. Ackermann, and S. Jablonski, "Logic based look-ahead for the execution of multi-perspective declarative processes," in *BPMDS, EMMSAD*, 2019, pp. 53–68. 9

[27] L. Ackermann, S. Schönig, S. Petter, N. Schützenmeier, and S. Jablonski, "Execution of multi-perspective declarative process models," in *OTM Conferences (1)*, 2018, pp. 154–172. 9

[28] R. A. Kowalski and M. J. Sergot, "A logic-based calculus of events," *New Generation Computing*, vol. 4, no. 1, pp. 67–95, 1986. 9

[29] L. T. Ly, "SeaFlows – A compliance checking framework for supporting the process lifecycle," Ph.D. dissertation, University of Ulm, May 2013. 9

[30] L. T. Ly, S. Rinderle-Ma, D. Knuplesch, and P. Dadam, "Monitoring business process compliance using compliance rule graphs," in *OTM Conferences (1)*, 2011, pp. 82–99. 9

[31] D. Schleicher, S. Grohe, F. Leymann, P. Schneider, D. Schumm, and T. Wolf, "An approach to combine data-related and control-flow-related compliance rules," in *IEEE SOCA*, 2011, pp. 1–8. 9

[32] E. Mulo, U. Zdun, and S. Dustdar, "Domain-specific language for event-based compliance monitoring in process-driven SOAs," *Service Oriented Computing and Applications*, vol. 7, no. 1, pp. 59–73, 2013. 9

[33] S. Sebahi, "Business process compliance monitoring: a view-based approach," Ph.D. dissertation, LIRIS, 2012. 9

[34] C. Giblin, S. Müller, and B. Pfitzmann, "From regulatory policies to event monitoring rules: Towards model-driven compliance automation," Tech. Rep. Report RZ 3662, 2006. 9

[35] S. Hallé and R. Villemaire, "Runtime monitoring of message-based workflows with data," in *ECOC*, 2008, pp. 63–72. 9

[36] D. A. Basin, F. Klaedtke, S. Müller, and B. Pfitzmann, "Runtime monitoring of metric first-order temporal properties," in *FSTTCS*, vol. 2, 2008, pp. 49–60. 9

[37] D. A. Basin, M. Harvan, F. Klaedtke, and E. Zalinescu, "MONPOLY: Monitoring usage-control policies," in *RV*, vol. 7186, 2011, pp. 360–364. 9