UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Ubaier Ahmad Bhat

Runtime Monitoring of Data-Aware business rules with Integer Linear Programming

Master's Thesis (30 ECTS)

Supervisor: Fabrizio Maggi, PhD

Tartu 2016

Acknowledgements

I would like to thank Dr. Fabrizio Maggi for his tremendous support and guidance for this thesis. I would also like to express my gratitude to all staff member of our faculty who always helped me and guided me at every step in past two years.

I would also like to thank my friends and colleagues at Mobi Labs ÖU, without their support and flexibility it would have been very difficult to work and study at the same time.

I would like to express my gratitude to IT Academy for their scholarships and support.

Finally, I would like to thank by family and friends back in Kashmir for their support and patience.

Runtime Monitoring of Data-Aware business rules with Integer Linear Programming

Abstract:

Runtime Compliance Monitoring is vital building block in the Business Process Management lifecycle, in timely detection of non-compliance as well as provision of responsive and proactive countermeasures. In particular, it is linked to operational decision support, which aims at extending the application of process mining techniques to on-line, running process instances, so that deviations can be detected and it is possible to recommend what to do next and predict what will happen in the future instance execution.

In this thesis, we focus on Runtime Compliance Monitoring of data-aware business rules. In particular, we use Integer Linear Programming (ILP) for early detection of violations that occur from interplay of two or more constraints. An operational support provider has been implemented as part of process mining framework ProM and the approach has been validated using synthetic and real life logs.

Keywords: Process Mining, Runtime Compliance Monitoring, Data-Aware, Integer Linear Programming

CERCS: P170

Käitusaegse seire andmeteadlikud ärireeglid koos lineaarse planeerimisega

Lühikokkuvõte:

Käitusaegne seire (*Runtime Compliance Monitoring*) on oluline osa äriprotsesside halduse elutsüklis, mittevastavuse õigeaegses avastamises, samuti vastumeetmete korraldamises ja ennetamises. Täpsemalt on see seotud operatiivse otsuse toega, mille eesmärgiks on laiendada protsessikaeve tehnikat sidusrežiimis, käitada protsessi isendeid nii, et kõrvalekaldeid on võimalik avastada, ning on võimalik soovitada, mida võiks järgmiseks teha, ning samuti ennustada, mis hakkab juhtuma tulevaste juhtumite täitmisel. Antud magistritöö keskendub käitusaegse seire andmeteadlikele ärireeglitele. Töös kasutatakse varajaste rikkumiste tuvastamiseks lineaarset täisarvulist planeerimist (Integer Linear Programming (ILP)), mida rakendatakse kahe või enama kitsenduse koosmõjul. Töökorras toepakkujas on rakendatud protsessikaeve raamistikku ProM ja meetod on valideeritud kasutades sünteetilisi ja reaalseid logisid.

Võtmesõnad: protsessikaeve, käitusaegne seire, andmeteadlikkus, lineaarne täisarvuline planeerimine

CERCS: P170

Contents

List of Figures

List of Tables

Chapter 1
Introduction

There is an urgent demand for developing Information Systems in order to fully support business processes of companies, institutions and organizations in general. The rapidly changing markets impose frequent modification and updates to the business processes, leading to a constant decrease, in terms of time span, to the life-cycle of a business process definition [5].

In this context, one very important functionality that any process aware Information System should be able to support is compliance monitoring. Compliance monitoring is the ability to verify whether the actual flow of work is compliant with the intended business process model. Process models can be imperative (such as Petri Nets [20] or BPMN [21]) or declarative (Declare [1]). Most suitable approach to model fast changing, unpredictable processes is to use declarative modelling. These allow a modeller to design several possible execution paths as a compact set of business rules/constraints. A modeller can only focus on more interesting rules and any process execution that does not contradict these rules is allowed. [5].

In this thesis we have developed a technique and a tool to analyse complex data-aware constraints at runtime. In this context, we use sequence analysis for checking individual constraints and support early detection of violations using Integer Linear Programming (ILP). In particular, the technique will be able to identify violations of single constraints in isolation but also violations that derive from the interplay of two or more constraints. The technique is implemented as a Client-Server application that will take an event log as a real-time feed from an Information System, process the data for compliance monitoring

and instantaneously post the results in a user friendly format. The results have been verified using analysis of real and synthetic event-logs. The solution is implemented in the process mining tool ProM making it available for other researchers and industry experts.

1.1 Thesis outline

1.1.1 Related Work & Background

Chapters 2 and 3 focus on the literature review and a background of tools and techniques used in this thesis.

1.1.2 Run time verification of individual data-aware declare constraints

In Chapter 4, we will discuss our approach for run time verification of individual data-aware Declare constraints with sequence analysis.

1.1.3 Early detection of violations determined by interplay of two or more constraints

In Chapter 5, we will discuss our approach for early detection of violations determined by interplay of two or more constraints using Integer Linear Programming.

1.1.4 Implementation

In Chapter 6, we present details about the implementation.

1.1.5 Validation and Verification

In Chapter 7, we present results for verification and validation of our approach.

1.1.6 Conclusion and Future Work

In Chapter 8, we describe the outcome of this thesis and what can be done in the future.

In this thesis we address the following questions:

- Can sequence analysis be used to monitor the compliance of a business process with respect to complex business rules on control flow and data?
- Can Integer Linear Programming be used for early detection of violations?
- Is the proposed approach applicable to real-life case studies?

Chapter 2
Related Work

Compliance monitoring is a "branch of process mining for verifying whether the observed behaviour of a process, as recorded in a event log, is conformant with a given set of business rules which are provided in the form of process model"[26]. Sometimes the terms *conformance checking* is used for compliance monitoring. Even thought there is no clear distinction in these two terms conformance checking is mostly used for post-mortem analysis where as compliance monitoring is used for runtime analysis [12].

There are many techniques being developed to perform compliance monitoring. Two key components of compliance monitoring are:

- **Process Model**: Type of process models which can be **imperative/procedural** (such as Petri Nets [20] or BPMN [21]) or **declarative** (such as Declare [1], MP-Declare).
- **Perspective**: This can be either **Single Perspective** (i.e. looking only at control flow) or **Multi-perspective/Data-aware** (i.e. looking at control flow as well as data such as temporal constraints, resource allocation, work distribution, quality of service, etc. . .). For example, let us consider a process consisting of activities ***a, b, c, and d***. A model ***"abcd"*** describes the sequence in which the activities should take place. A control flow based conformance check will only evaluate whether the activities occur in this order or not. Any other perspective like "Who performed the activity?"or "What was the time between two activities?"or any other data related to these activities will not be evaluated.

Data-aware conformance means looking at both control as well as data flow. For example in a hospital scenario a multi-perspective conformance rule might indicate that a certain medical test has to be performed before a particular treatment can be given to a patient. In addition to this, there are also data-flow constraints on what the results of test should be and what the time limit between the test and treatment is.

Work done in field of conformance checking can be categorised as follows.

2.1 Procedural conformance checking without data

A bulk of work is available for conformance checking using Procedural models and looking only at Control flow and ignoring any data. These works are mostly based on replaying the log on a model and measure conformance by comparing an event stream generated by the model and an event stream that is derived from the execution [7, 11, 25]. In alignment-based approaches conformance checking is performed by aligning both the modelled behaviour and the behaviour observed in the log [4].

2.2 Procedural conformance checking with data

Conformance checking can be made much more reliable by taking a data-aware approach. [8] provides an approach of alignment-based conformance checking for procedural models.

2.3 Declarative conformance checking without data

As mentioned earlier declarative models are better for modelling processes in unpredictable environments. As with procedural models the initial work done on conformance checking for declarative models has focused mostly on control flow. [6] describes

APPROACH	CMF 1 time	CMF 2 data	CMF 3 resources	CMF 4 non-atomic	CMF 5 lifecycle	CMF 6 multi-instance	CMF 7 reactive mgmt	CMF 8 pro-active mgmt	CMF 9 root cause	CMF 10 compl. degree
Superv. Control Theory [17]	+/−	−	+	+	+	−	−	+	−	−
ECE Rules [31]	+	+/−	+	+	−	−	+	−	+/−	+
BPath (Sebahi) [42]	+	+	+	+	+/−	+	+	−	−	+/−
Gomez et al. [34]	+	−	−	+	n.a.	+/−	+	+	−	−
Giblin et al. [33]	+	n.a.	n.a.	n.a.	n.a.	n.a.	+	n.a.	n.a.	n.a.
Narendra et al. [40]	−	+	+	n.a.	−	+	+	−	−	+
Thullner et al. [41]	+	n.a.	n.a.	n.a.	n.a.	n.a.	+	−	−	n.a.
MONPOLY [26,27]	+	+	+	+/−	+/−	+	+	−	−	−
Halle et al. [24]	+/−	+	+/−	n.a.	n.a.	n.a.	+	n.a.	n.a.	n.a.
Dynamo [21–23]	+	+	+/−	+	n.a.	+	+	−	−	+/−
Namiri et al. [18]	+/−	+	+	+	−	+	+	−	−	−
MobuconEC [39]	+	+	+	+	+	+	+	−	−	+/−
Mobucon LTL [36–38]	+/−	−	−	+	−	−	+	+	+	+/−
SeaFlows [35]	+/−	+/−	+/−	+	+	+	+	+	+	+/−

Caption: + supported, + implementation publicly available, +/− partly supported, − not supported, n.a. cannot be assessed.

Figure 2.1: Classification of monitoring approaches [12].

an approach of conformance checking for declarative models.

2.4 Declarative conformance checking with data

An evolution of these approaches is to look at declarative models with data. [5] provides the basis for implementation of multi-perspective conformance using Metric First Order Temporal Logic(MFOTL).

2.5 Runtime compliance monitoring

The ability to monitor conformance can be crucial for any business or organisation. [17] provides a starting point looking into use of Linear Temporal Logic (LTL) for runtime compliance monitoring of Control flow. Mobucon LTL [13, 18, 15] have already been implemented in ProM and can be used to provide Control Flow based compliance monitoring. Mobocon LTL does perform early detection of violations but without any data related constraints. Mobocon EC [19] be used for compliance monitoring with respect to control flow and time related constraints. It does not provide any early detection of violations. In theory Mobocon EC can also be extended to be data-aware however this is

yet to be implemented. Figure 2.1 shows comparison of different compliance monitoring tools [12].

One of the open challenges in the context of compliance monitoring with declarative models is capability of supporting data-aware compliance monitoring at runtime. In this thesis we aim to provide a practical solution for this challenge.

Chapter 3
Background

In this chapter, we present the fundamental concepts required to understand the rest of the thesis.

3.1 Process Mining and Event logs

The main concept behind process mining is to discover, monitor and improve processes by extracting knowledge from data that is available in Information Systems [26].

Data for process mining comes in form of event logs which have been standardized into different formats. Until 2010 Mining eXtensible Markup Language(MXML) was standard format for event logs. Since 2010 eXtendible Event Stream(XES) as become the successor of MXML. [10, 26].

In XES each event refers to an activity (i.e, a well defined step in some process which belongs to a particular case or process instance). The events belonging to a trace (or a case) are ordered with respect to their execution times. There, a trace can be viewed as a sequence of events. Event logs can also store additional information about events such as the timestamp of the event, the resource (i.e. device/department/person executing the activity), or any data elements recorded with the event. In XES, data elements can be event attributes, i.e. data produced by the activities of a business process and trace attributes, particularly data which are associated to a whole process instance [5, 10].

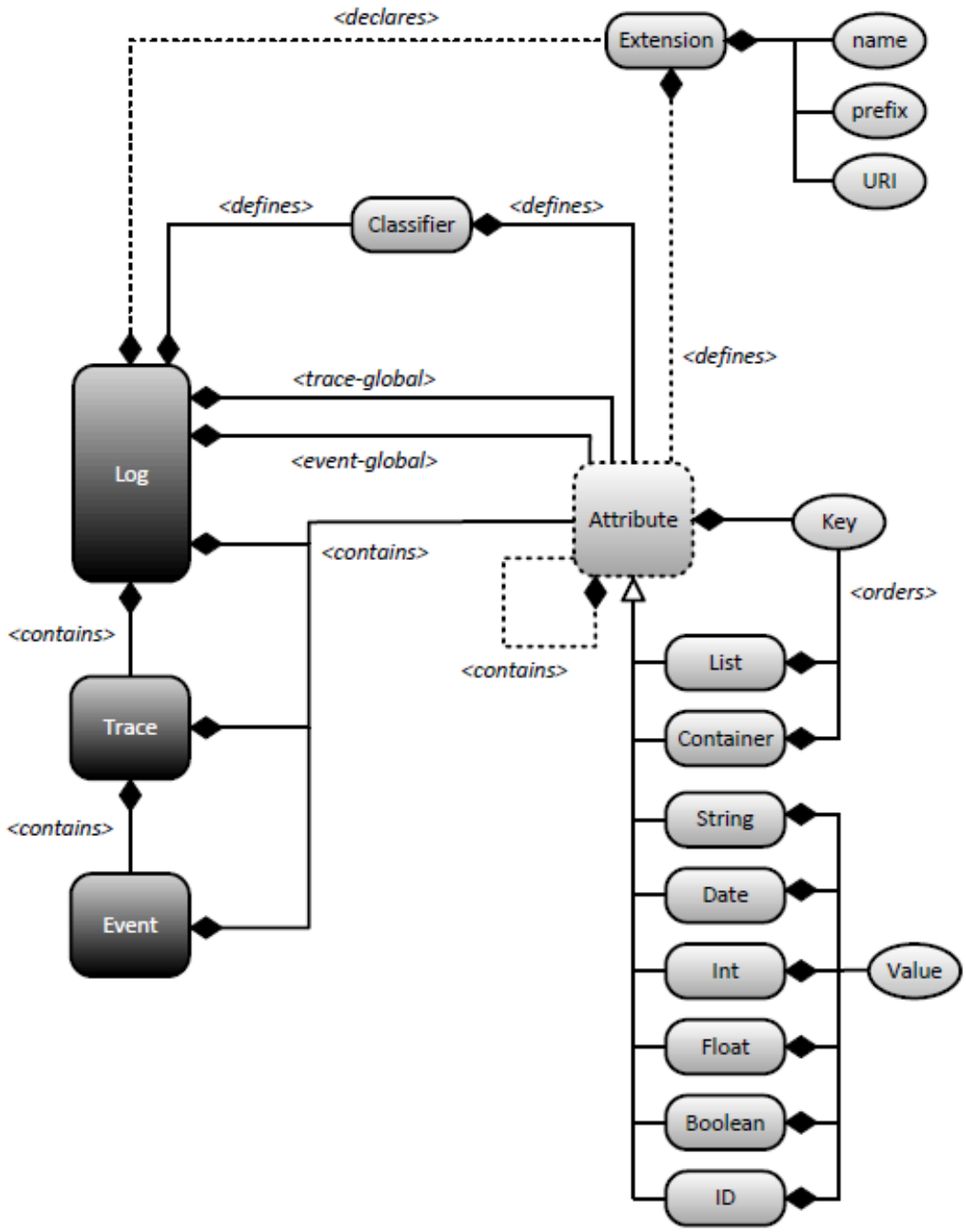


Figure 3.1: The UML 2.0 class diagram for the complete meta-model for the XES standard [10]

3.2 Declarative Modelling

Declare is a process modelling language which was proposed by van der Aalst and Pesic in ([24, 23, 1]). In Declare instead of modelling the whole process by specifying flow of activities we specify relationships between different activities using specific constraints or templates. This makes Declare models "open" i.e. anything that is not specified in the model is considered acceptable. This is different for procedural languages which are considered to be "closed" i.e anything that is not specified in the model is considered forbidden. Declare therefore gives more flexibility to the designers who can focus on the most important business rules. This makes Declare very suitable for complex execution environments [5].

3.2.1 Declare templates

Declare templates can be grouped into four categories: existence, relation, negative relations and choice.

1. Existence

This is a group of unary constraint. By unary we mean that these constraints are applicable to only a single activity. This group has three main type of constraints absence, existence and exactly.

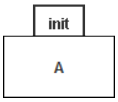
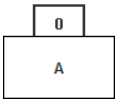
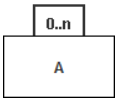
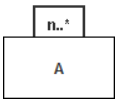
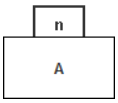
Template name	Symbol	Description
$init(A)$		A must be at start of each process instance.
$absence(A)$		A should never occur.
$absence(2, A)$ $absence(3, A)$ ... $absence(n, A)$		A should occur at most n times.
$existence(n, A)$		A should occur at least n times
$exactly(n, A)$		A should occur exactly n times

Table 3.1: Existence templates

Table 3.1 on 11 shows the symbols and description for each constraint. As the names suggest these templates are used to state whether an activity should take place or not.

2. Relation

Relation constraints describe relationships between two activities. Table 3.2 shows the list of relation templates.

Relation templates can are either ordered i.e. the activities should occur in a certain order or un-ordered i.e. activities can occur in any order

3. Negative

Negative constraints forbid the execution of a particular activities. These constrains like relative templates are either ordered or un-ordered.

Table 3.3 shows the list of negative constraints.

Template name	Symbol	Description
responded existence(A, B)		If A occurs then B must occur (before or in future).
$co - existence(A, B)$		If A occurs then B must occur (before or in future) and vis-versa
response(A, B)		If A occurs then B must eventually occur.
precedence(A, B)		If B occurs then A must have occurred in past.
succession(A, B)		After every A there has to be at least one B and B has to be preceded by A. B can happen only after A had occurred.
alternate response(A, B)		If A occurs then B must eventually occur without repetition in between.
alternate precedence(A, B)		If B occurs then A must have occurred in past without repetition in between.
alternate succession(A, B)		After each A is executed at least one B is executed. Another A can be executed again only after the first B. And B cannot occur before A. After it occurs, it can not happen before the next A again.
chain response(A, B)		If A occurs then B must occur immediately after A
chain precedence(A, B)		If B occurs then A must have occurred immediately before B
chain succession(A, B)		A and B can occur only next to each other.

Table 3.2: Relation templates

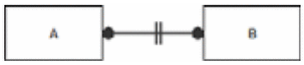
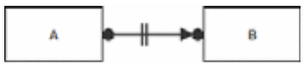
Template name	Symbol	Description
not respondedexistence(A, B) *not* $co - existence(A, B)$		Only one of the two tasks A or B can be executed, but not both.
not response(A, B) *not precedence*(A, B) *not succession*(A, B)		Before B there cannot be A and after A there cannot be B.
not chain response(A, B) *not chain precedence*(A, B) *not chain succession*(A, B)		A and B can never be executed next to each other where A if executed first and B second.

Table 3.3: Negative templates

4. Choice

In Choice templates one of the two activities must occur. Table 3.4 shows the symbols and descriptions of choice templates.

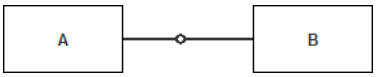
Template name	Symbol	Description
$choice(A, B)$		At least one from A and B has to be executed.
$exclusive\ choice(A, B)$		A or B has to occur but not both.

Table 3.4: Choice templates

Further details on Declare constraints an be found in [16, 22]

3.2.2 Declare with data

The Declare templates mentioned in the previous section only capture constraints related to cardinality and control flow. We can also add data related constraint to a Declare model. There are three types of data conditions that can be added. These are specified in the following order. [Activation] [Correlation] [Temporal]

Data which relates to the activation activity is specified in $A.data$ format. Similarly conditions related to a target activity are specified in $T.data$ format.

1. Activation

These conditions are used to specify when a template is considered to be active. For example without data the condition $absence(AbandonShip)$ that event abandon ship should never occur. However if we add data condition to it

$$absence(AbandonShip)$$
$$[A.rank == "captain"]$$

This will imply that an event abandon ship can be executed but not if the rank is equal to captain.

2. Correlation

Correlation conditions are used to specify data relationships between two activities. This means that correlation conditions do not apply to Existence templates as they are all unary and only contain one condition. Let us consider the example:

$$response(PaymentRecieved, DispachOrder)$$
$$[A.pendingBalance == 0][T.id == A.id]$$

This constraint will only be activated if activity $PaymentRecieved$ occurs with data $pendingBalance == 0$. If this happens then we require $DispachOrder$ to eventually occur with data id which must be equal to id of $PaymentRecieved$.

3. Temporal

Temporal conditions are used to specify time between two activities.

Format for specifying temporal conditions is: 0,$value$,$unit$ where unit can be s for seconds, m for minutes, h for hours and d for days. Let us take the following example.

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

This rule can be interpreted as: after $PaymentRecieved$ occurs with data $pendingBalance = 0$, $DispachOrder$ must occur with data id equal to id of $PaymentRecieved$ within $5minutes$.

3.2.3 Design tools

Declare models can be designed using Declare designer (see figure 3.2) which provides a GUI for rapidly designing Declare models. We can also use ProM plugins like Simple Declare designer and Simple Declare editor for designing and modifying Declare models.

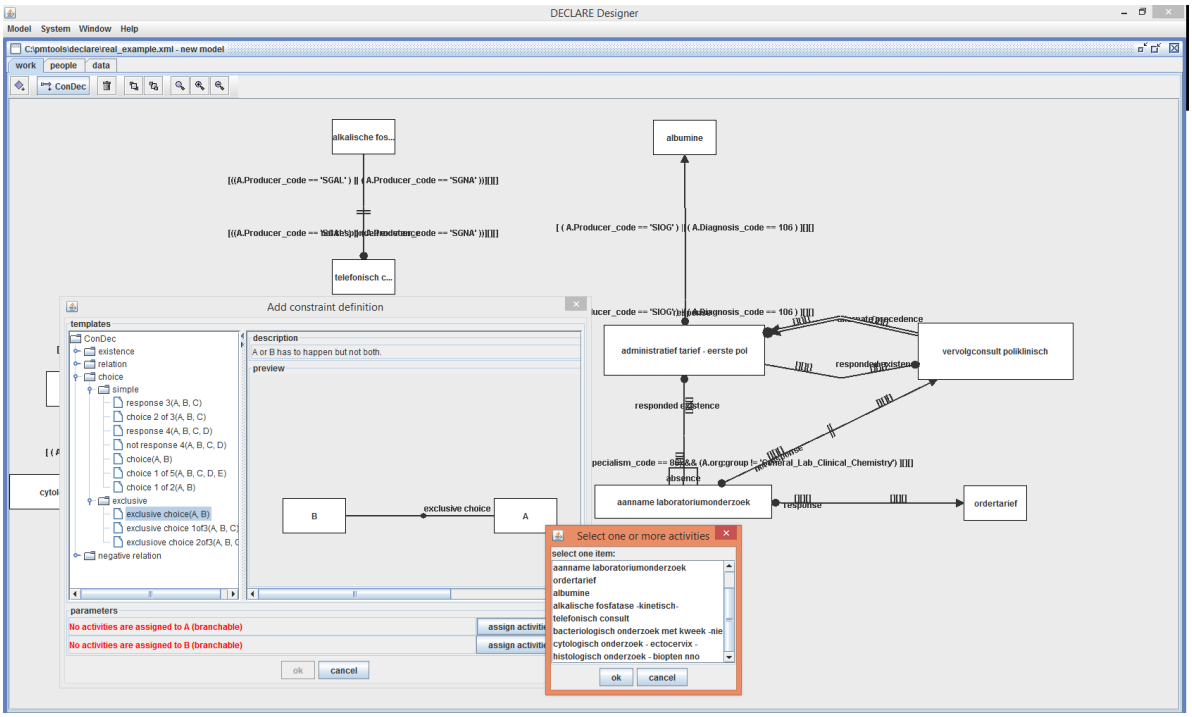


Figure 3.2: Declare designer

3.3 Integer Linear Programming

Integer Linear Programming(ILP) or Linear programming is a method to achieve the optimal solution in a mathematical model in which requirements are represented in the form of linear relationships.

Let us take the following example. Our aim is to find a real number x when we are giving certain conditions. Before going ahead we will have to define our default maximum and minimum possible values. This is important because otherwise we will have infinite

Step	condition	Range	Has Solution?
0	init	$m \geq x \geq M$	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 solution*	false

Table 3.5: Example find solution for single variable using linear programming

possibilities. Let us see choose very large negative number m and a very large positive number M.

At this stage we can say that the solution for x is between m and M i.e $m \geq x \geq M$.

Now let us conceder first condition which states that x should be less than ten. So now our solution for x will be $m \geq x < 10$

x should be greater than 0 so our solution for x will now become $0 < x < 10$.

x should be greater than 5 so our solution for x will now become $5 < x < 10$.

We can keep on adding conditions to x which can reduce the range for x and bring us closer to its actual value. However not all 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 previous conditions x should already be less than 10.

We can also have condition that fix the value of x. Let us say x should be equal to 9. Now minimum possible solution for x is 9 and maximum possible solution for x is 9.

If we add any more conditions at this point which are different from the previous condition 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.

Table 3.5 shows steps and conditions in our previous example. Figure 3.3 shows how the range for x will change with respect to each newly added condition

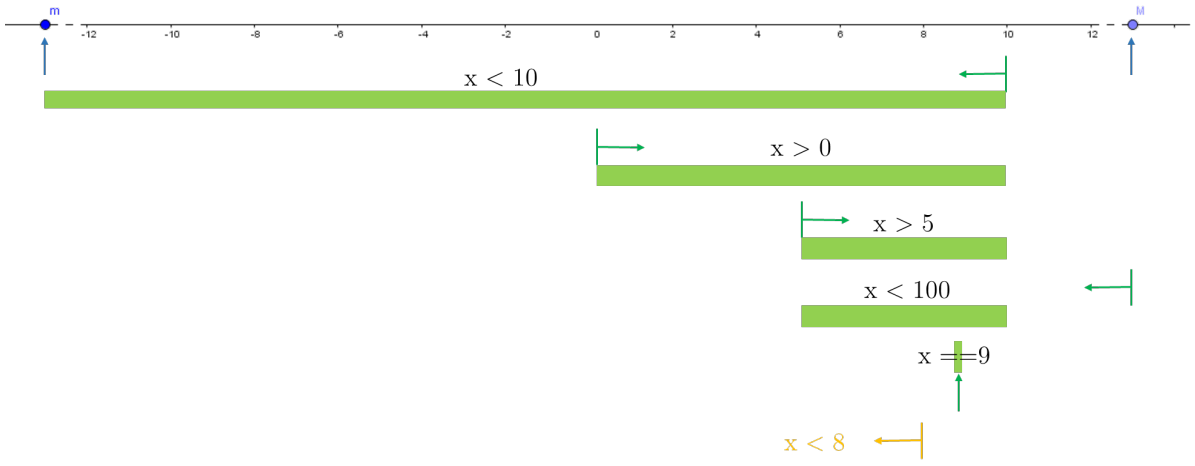


Figure 3.3: ILP example

We can also use ILP to find optimal solutions for linear expressions of form $x_1 + x_2 + .. + x_n$.

In this thesis we will use the ability for ILPs to find a solution as well as not finding a solution for a set of given conditions. This has been explained in up coming chapters.

Chapter 4

Run time verification of individual data-aware Declare rules

4.1 Internal working of Declare Analyzer

We are going to use the analysis engine (templates) from the Declare Analyzer (which is a plug-in in ProM for offline conformance checking), as a black box to perform sequence analysis of each event. In this section we will describe some of the internal workings of Declare Analyzer as explained in [5] so that we can have a better understanding of our approach and implementation.

Main component of Declare Analyzer is the CheckLogConformance method which is reported in Algorithm 1. This algorithm requires a Declare Model and an event log.

Algorithm 1: CheckLogConformance from Burattin [5] Algorithm 1

Input: Model: a Declare model
Log: log of events
Output: A set of fulfilling and violating traces/constrants

```
1 Let fullfill and viol be maps that, given a trace and a constraint, return the set of
  fulfilling and violating events
2 foreach trace ∈ Log do
3     foreach constr ∈ Model do
4         viol, fullfill ← CheckTraceConformance(trace,constr)
5         viol[][] ← viol
6         fullfill[][] ← fullfill
7 return viol, fulfill
```

The described algorithms *CheckTraceConformance* can be seen as a "framework" used for conformance checking with respect to different Declare templates.

Each template has its own algorithm for the following operations.

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

Let us consider the template for response 4.1. The operations described for sequence analysis are used in the following way:

- opening: not used;
- fulfilments: this procedure checks whether the item 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) becomes fulfilments.
- violations: not used;
- activations: the activation procedure checks whether the input event refers to an activation of the constraint and the activation condition σ_a 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.

Response

template.opening()

```
1 do nothing
```

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

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

$template.violation(e, trace, pending, violations, T, \sigma_c, \sigma_t)$

```
1 do nothing
```

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

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

$tmplate.closing(pending,\ fulfillments,\ violations)$

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

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

Table 4.1: Algorithms for Response from Declare Analyzer as described in [5] Table 3

4.2 How the sequence analysis are invoked in on-line settings

In previous section we briefly looked at how Declare templates are analysed in Declare Analyzer in a off-line setting. In the Declare Analyzer we input the whole event log and the Declare model at once and then perform the analysis and visualize the results. In an on-line setting we do not have the whole log and have to process each event separately as it is being streamed.

For each model we setup all templates in the given model. Once all the events are completed we make the last event as done. This flag is used to set the permanent state for the particular constraint.

Table 4.2 shows how the algorithm described in previous section has been adapted for runtime analysis. *state* in this algorithm is based on *Four valued semantics* described in next section.

$getState(viol, fulfill, pending, activations)$ method uses the criteria described listed in Table 4.3.

4.3 Four valued semantics

The current state of a trace with respect to a constraint can be described using a four valued semantics. These values are ***possibly satisfied***, ***possibly violated***,***permanently satisfied*** or ***permanently violated***. A trace will acquire any one of these states only once the activation condition related to particular constraint has been fulfilled at least once. We have discussed how other online monitoring tools are using similar semantics in the Literature review. These semantic values can be interpreted using as follows:

- **Possibly satisfied**: This means that the constraint has been activated and is

Invoking sequence analysis in online setting

$os.accept(session, model)$

1 do nothing

$os.simple(session, trace)$

Input: Session: a Declare model
event: log of events
Output: A set of fulfilling and violating traces/constrants

1 Let fulfill and viol be maps that, given a trace and a constraint, return the set of fulfilling and violating events **foreach** $constr \in Model$ **do**
2 $\quad viol, full \leftarrow CheckTraceConformance(trace, constr)$
3 $\quad viol[][] \leftarrow viol$
4 $\quad fulfill[][] \leftarrow fullfill$
5 $\quad state \leftarrow getState(viol, fulfill, pending, activations)$
6 **return** state;

Table 4.2: Algorithm for invoking analyser in runtime setting

currently satisfied. However there is a possibility that the constraint might be violated in the future.

- **Possibly violated**: The constraint has been activated and is currently violated. However the trace can still recover in the future i.e. it can be satisfied.
- **Permanently satisfied**: This means that the constraint has been permanently satisfied and can no longer be violated in the future.
- **Permanently violated**: This means that the constraint has been permanently violated and can no longer recover in the future.

The semantic value for the current state of a trace can depend on the type of constraint. Table 4.3 on page 23 shows the semantic criterion for each constraint type.

- violations v : number of violations in current template

- fulfillments f : total number of fulfillments in current template
- pending activations p : total number of pending activations in current template
- activations a : total number of activations of the current template.
- limit $limit$: for come cases like absence, existence and exactly.

Template	**Poss.voil**	**Poss.sat**	**Voil**	**Sat**
response	$p > 0$	$p == 0$	*	*
not response not chain response precedence not precedence absence absence2 absence3 chain precedence not chain precedence alternate precedence	-	$v == 0$	$* \vee v > 0$	*
init strong init	-	-	$* \vee v > 0$	$* \vee f > 0$
existence existence2 existence3	$f < limit$	-	*	$* \vee f >= limit$
exactly1 exactly2	$v == 0 \wedge f < limit$	$v == 0 \wedge f == limit$	$* \vee v > 0$	*
responded existence	$f == a$	$f < a$	*	*
not responded existence not succession not chain succession not co-existence	-	$v == 0$	$* \vee v > 0$	*
succession chain succession co-existence alternate succession	$v == 0 \wedge f < a$	$v == 0 \wedge f == a$	$* \vee v > 0$	*
chain response alternate response	$v == 0 \wedge p > 0$	$v == 0 \wedge p == 0$	$* \vee v > 0$	*
choice	-	-	$* \vee f == 0$	$* \vee f > 0$
exclusive choice	-	-	$* \vee v > 0$	$* \vee v == 0$

*** Poss.viol at end of trace will become Viol and Poss.sat will become Sat**

Table 4.3: Criterion for semantic values

- done $*$: This is the default case to choose between permanently satisfied and permanently violated after we receive the last event. If the state of the process after analysis of the last event is Possibly satisfied or Permanently satisfied final sate will become Permanently satisfied else it will become Permanently violated.

Algorithm 2: Semantics for Response

Input: pendingActivations: From Declare Analyzer analysis
Result: state

```
1  if pendingActivations > 0 then
2  |  state = possiblyViolated;
3  else
4  |  state = possiblySatisfied;
5  if done then
6  |  if state == possiblySatisfied then
7  |  |  state = permanentlySatisfied
8  |  else
9  |  |  state = permanentlyVoilated
10 return state
```

Algorithm 3: Semantics for Not Response/Not Chain Response

```
1  if voilations > 0 then
2  |  state = permanentlyVoilated;
3  else
4  |  state = possiblySatisfied;
5  if done then
6  |  if state == possiblySatisfied then
7  |  |  state = permanentlySatisfied
8  |  else
9  |  |  state = permanentlyVoilated
```

4.4 Compliance degree of a single case (healthiness)

Each constraint in a given model has a weight which can be used to indicate how important the particular constraints is compared to other constraints in the model. The weight along with current state of a case is used to calculate the compliance degree or healthiness.

We use the semantic value to indicate whether the compliance degree will go up or down. The compliance degree is normalized to a maximum value of 1.

If the constraint is permanently violated the health is reduced to 0. If the possibly violated, health is reduced by 50% see Table 4.4

$$h = \frac{\sum_{i=1}^{n} w_i \times s_i}{\sum_{i=1}^{n} w_i}$$

where: h = health

i = index of constraint

w = weight of constraint

s = score of current state

n = total number of constraints

id	**State**	**score**
1	Permanently Violated	0
2	Possibly Violated	0.5
3	Possibly Satisfied	1
4	Permanently Satisfied	1

Table 4.4: Example of conflicting constraints

Chapter 5

Early detection of violations determined by interplay of two or more constraints

The previous section dealt with individual constraint, however it is not enough as in a multi-constraint model individual constraints will might contradict each other. For this we use concept of conflicting sets which introduces a new global semantic value to indicate relationship between individual constraints.

5.1 Early detection of violation in Simple case

Consider the constraint $absence(B)$ and $response(A, B)$. Let us first consider the case without any data. In the absence constraint we are saying that activity B should never occur. But in response we say that if A occurs then B must eventually occur. As individual constraints these are fine but as soon as A occurs only one of the two constraints can be fulfilled. If B occurs then absence will be violated and if B does not occur then response will be violated. Therefore we can call these two constraints as conflicting constraints.

Table 5.1 shows the same constraints but now with data conditions. Now even though we have the same constraint; now we have conditions. This means that these constraints are no longer in conflict. Once activated $response(A, B)$ requires $B.x == 4$ to occur while $absence(B)$ requires $B.x == 3$ should not occur. Both these conditions:

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

can be satisfied at the same time. Therefore there is no conflict when A occurs.

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 5.1: Example of conflicting constraints

Table 5.2 shows the same constraints but this time the conditions have been changed. $response(A, B)$ requires $B.x == 3$ once activated however $absence(B)$ requires $B.x == 3$. Both these conditions put together provide the following obligation:

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

As we can see that both these conditions can no longer be satisfied at the same time and therefore we have detected a conflict as only one of the two constraints can be satisfied when A occurs.

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 5.2: Example of non conflicting constraints

In the previous example the correlation condition of $response(A, B)$ only depends on value of B however this condition can also be specified in relation with the Activation activity. Table 5.2 shows updated conditions. Now where the two constraints are in conflict or not will depend on the value of $A.x$. Unless $A.x = 3$ we do not have any conflict.

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 5.3: Example of non conflicting constraints

We use linear programming to detect whether these conflicts can take place or not. We have touched upon how linear programming works in Chapter 3. In case of detecting conflicts we treat each constraint condition as a part of a set of simultaneous equations (system of equations) $c \in P$. If we find a solution then we say that there is no conflict otherwise there is a conflict which means that one of the two constraints will definitely be violated at the end of the trace.

Algorithm 4 shows steps required to find conflict for each template. Note that only activated templates can be updated and used for conflict detection. Not all templates can be used for conflict detection.

Table 5.4 shows the criteria for considering a template activated. In our current implementation we do not consider choice based templates and templates which require counting like absence2, existence2 and succession.

Existence and Absence are always added to the problem set. Also conditions for all Existence and Negative templates are permanently added to the problem set once added.

Algorithm 4: Finding conflict

Input: activated
Output: hasSolution

```
1 if activate then
2     update(P)
3     min, max ← solve(P)
4     return min ∈ R ∧ max ∈ R
5 else
6     return true;
```

In the above example we have only shown simple conditions based on single variables. However we are able to use complex conditions based on multiple variables and logical operations. e.g $(A.x > 5) \vee ((A.y == 10) \wedge (A.user =' Jhon')$.

5.2 How to deal with indirect obligations

In the previous section we looked at how conflicts can be detected in advance using linear programming. Previous approach is based on finding direct conflicts which are related to correlation activity of templates activated by the current event.

1. $absence(B)$

2. $response(A, B)$

Once template 2 is activated we are able to check for conflicts reated to B by updating our problem set which already has the condition for absence.

However, we can improve this approach to detect conflicts which can be triggered by another activity. Let us look at the following example without data

1. $absence(C)$

2. $response(B, C)$

3. $response(A, B)$

In this example hen A occurs B must eventually occur because of template 3 $response(A, B)$. Assuming that this condition will be satisfied in the future i.e B will occur we can say that template 2 $response(B, C)$ will also eventually be activated. This will also make obligatory on C to occur eventually. But template 1 $absence(C)$ makes it obligatory that C should never occur. Therefore we can say that occurrence of A to cause of conflict in these templates.

Now let us look at the same example with data conditions in table 5.5.

In this example it is very difficult to see whether or not a conflict will take place. Let us assume A occurred with data $A.x = 2$. This means that template 3 $response(A, B)$ will be activated and update the problem. Problem set for B only contains one equation now $B.x = 2$ which has a solution $S1 \leftarrow (\{B.x_{min} = 2\ ,\ B.x_{max} = 2\})$. Now let us see if we can activate any other template using these values of B. Template 1 only depends on C therefore it can't be activated by B. Template 2 has B as activation activity B. To figure out whether this template can be activated or not we an use Integer Linear Programming to find the minimum and maximum values required for fullfilling the activation condition $B.x > 5$. In this case the solution $S2 \leftarrow (\{B.x_{min} = 6\ ,\ B.x_{max} = \infty\})$. Since the minimum value from the previous result is lower than that required for activation ($S1 \notin S2$) we can't say that template 2 will be activated. This will mean that we will not be able to detect any conflicts at this stage.

Let us take another occurrence of A with data $A.x = 6$. This time solution of problem set related to B is $S1 \leftarrow (\{B.x_{min} = 6\ ,\ B.x_{max} = 6\})$. Solution for activation condition for template 2 is still the same $S2 \leftarrow (\{B.x_{min} = 6\ ,\ B.x_{max} = \infty\})$. This time since minimums in S1 are $\geq$ then minimums in S2 and maximums in S1 are $\leq$ maximums in S2 i.e ($S1 \in S2$). We can say that template 2 will definitely be activated.

This means that we can update our problem set for C which till now only contained $\neg(C.x < 10)$. Updated problem set will become $\neg(C.x < 10) \wedge (C.x < B.x) \wedge (B.x > 5) \wedge (B.x == 6)$ and we have a conflict.

Using the same process we can recursively go deeper to find conflict caused by chained triggers. For example in Table 5.6 activity A with $A.x = 6$ can activate all other templates.

S.no	Constraint Template	Condition
1	*response*	$p > 0$
2	*notresponse*	$p > 0$
3	*precedence*	–
4	*notprecedence*	–
5	*init*	–
6	*absence*	–
7	*existence*	$f < 1$
8	*respondedexistence*	$f < a$
9	*notrespondedexistence*	$f > 0$
10	*chainresponse*	$p > 0$
11	*notchainresponse*	$p > 0$
12	*chainprecedence*	–
13	*notchainprecedence*	–
14	*alternateresponse*	$p > 0$
15	*alternateprecedence*	–

Table 5.4: Criterion for constraint considered activated for Conflict detection

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 5.5: Example of non conflicting constraints

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

Table 5.6: Example of non conflicting constraints

Chapter 6
Implementation

ProM (or Process Mining framework) is an open source framework for process mining algorithms. ProM is based on Java and provides a platform to developers of the process mining algorithms that is easy to use and easy to extend [3, 14]. ProM also provides a generic Operational Support (OS) environment that allows the tool to interact with external Information System at runtime. A workflow management system can send a stream of events to ProM OS service which is connected to **Operational Support providers**. These providers perform different types of analysis at runtime [9]. In this thesis, we use Prom OS for implementing our approach as a OS provider. We use a client server architecture as show in figure 6.1.

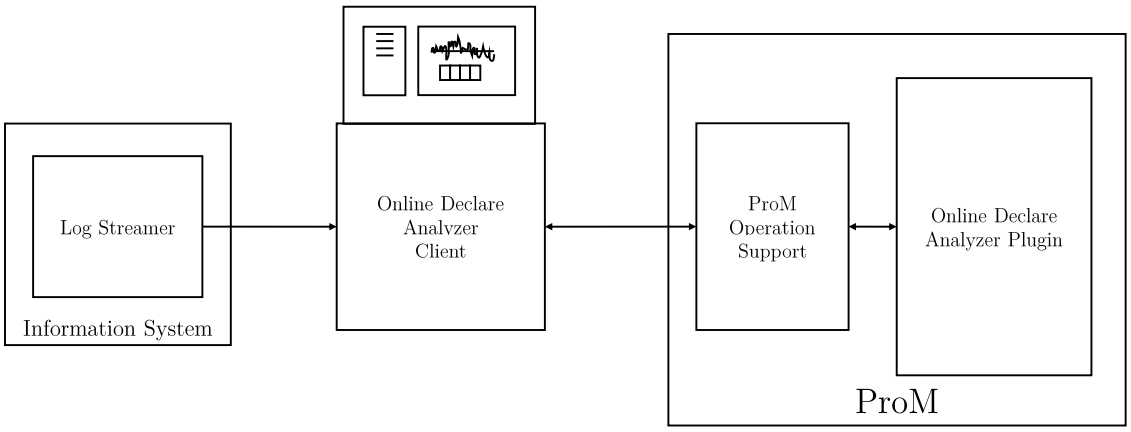


Figure 6.1: Architecture of implementation

As shown in the diagram have three main components: a log streamer, a client/visualizer and a server running ProM.

6.0.1 Online Declare Analyzer Plugin

Our implementation has been developed inside ProM as Online Declare Analyzer Plugin. After configuring Operation Support, this plugin connected to it. Operation Support creates a session for each new trace are received. This keeps each trace separate and we can process each trace and its conformance model independently. For performing Integer Linear Programming we use Java based LP solver [2].

6.1 Log Streamer

In absence of a real Information System connected with the client we use a Log streamer to simulate an Information System. The purpose of a log streamer is first to send out a model of business rules, followed a stream of by the events. For this implementation we have used a modified version of Log Replayer used in Mobucon LTL and adapted it to transmit the event along with data.

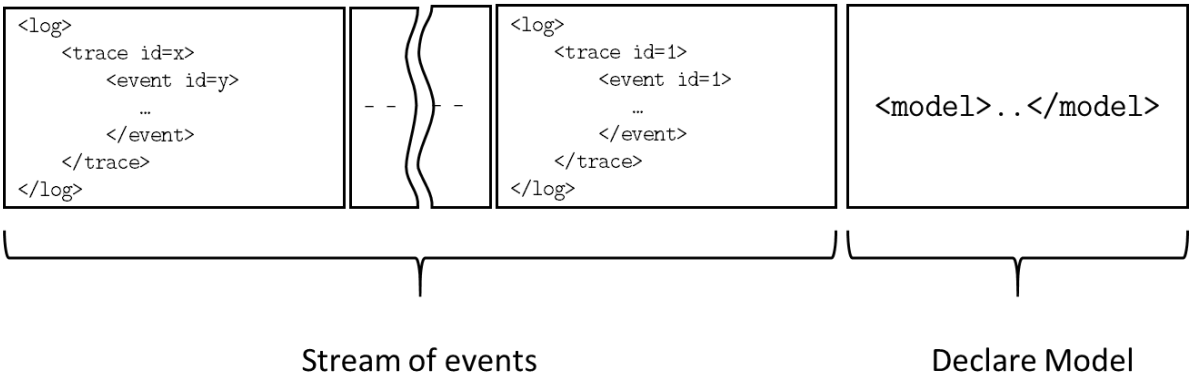


Figure 6.2: Data packet transmitted by Log streamer

The Log streamer starts to transmit data as soon as it is connected to Online Declare

Analyzer Client (ODAC). Figure 6.1 shows the data packets as they will be transmitted. The format used for sending a stream of events is XES.

6.2 Online Declare Analyzer Client

The Online Declare Analyzer Client (ODAC) acts as a server for incoming stream of events and business rules from an Information system/workflow system and as a client for the Operational Support system. ODAC forwards the model and events to the Online Declare Analyzer Server and then displays the analysis results in a user friendly format as they occur.

ODAC is a modified version of the Mobocon LTL Client. The client has been adapted to transmit events along with their data attributes. The visualizer has also been adapted to show event data.

Figure 6.2 shows the different components of ODAC.

1. Case selector: This component enables the user to select a particular case. This component also shows a ”Warning” message for cases which have a low compliance degree.

2. Event details: This component displays the details of each event as the are received after analysis. It displays the name of the activity along with any data and timestamps.

3. Constraint state: This component shows the state of a particular constraint in the case in a colour coded format which match with different values of fore valued semantics.

4. Constraints: This component shows the constraint details including name of the

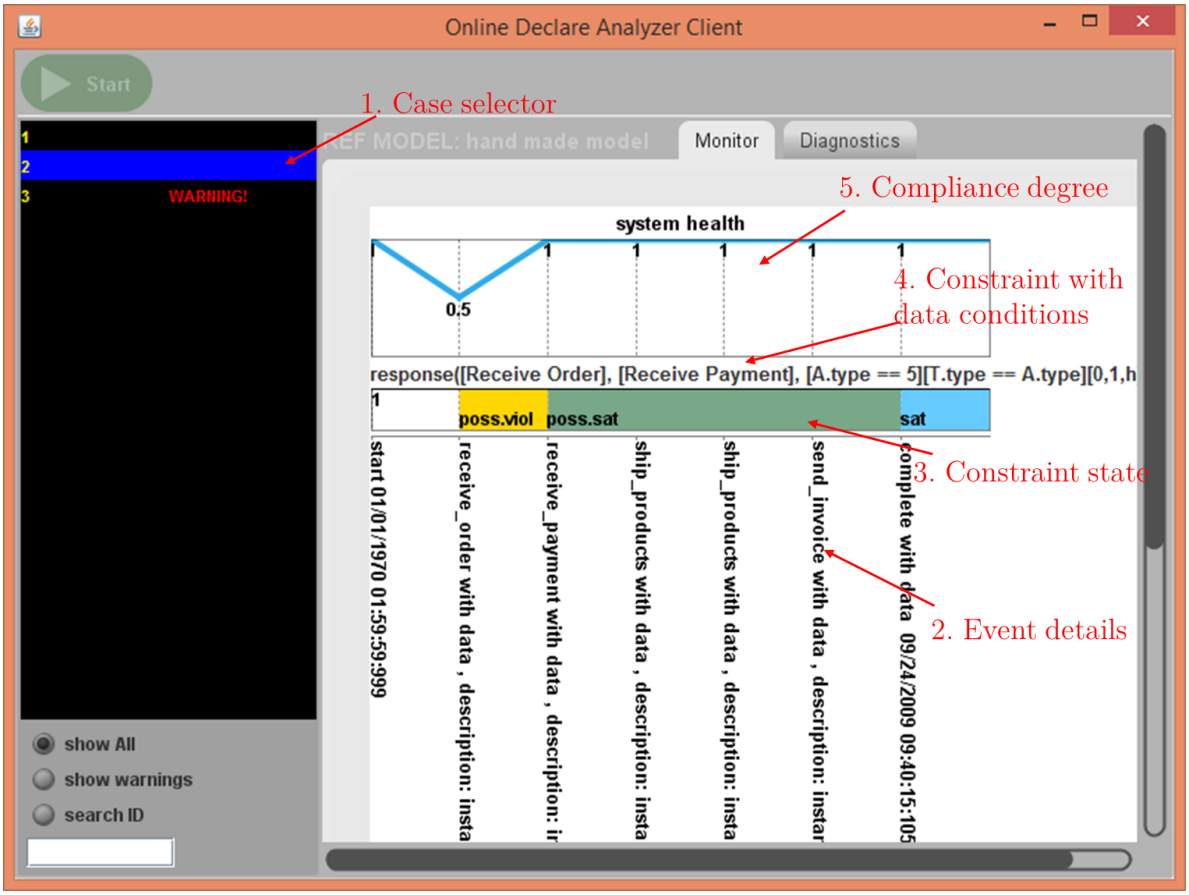


Figure 6.3: Online Declare Analyzer Client

constraint, activation and target activities, activation condition, correlation condition and any temporal condition.

5. Compliance degree: This component shows the compliance degree or health of the system in a graphical format.

Chapter 7
Verification & Validation

In order to verify our approach we are going to use different sets of event logs and business rules. We will start with simpler examples and increase the complexity as we go ahead.

7.1 Verification of individual data-aware Declare rules

First we tested single constraint using a synthetic log. Figure 7.1 shows the model used for testing.

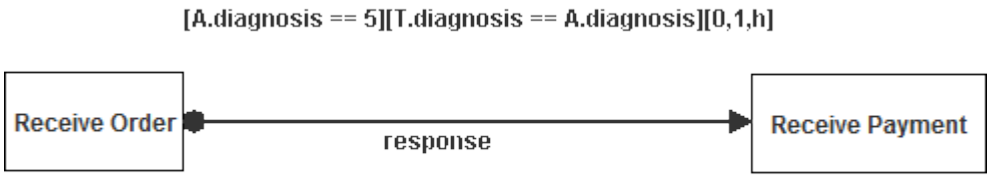


Figure 7.1: Graphical representation of single constraint model

id	Constraint	Activation Activity	Target Activity	Activation Condition	Correlation Condition	Time Condition
1	*response*	*A*	*B*	$(A.diagnosis == 5))$	$T.diagnosis == A.diagnosis$	$0, 1, h$

Table 7.1: Rules for model with single constraint

As we can see the model contains conditions for activation, correlation as well as time. Figure 7.2 shows the results for three traces. In the first trace, activation conditions $A.diagnosis == 5$ is never fulfilled and therefore the constraint is never activated and remains satisfied till the end. In trace 2 all the conditions (activation, correlation and time) are satisfied as *Receive Order* occurs with $diagnosis = 5$ followed by

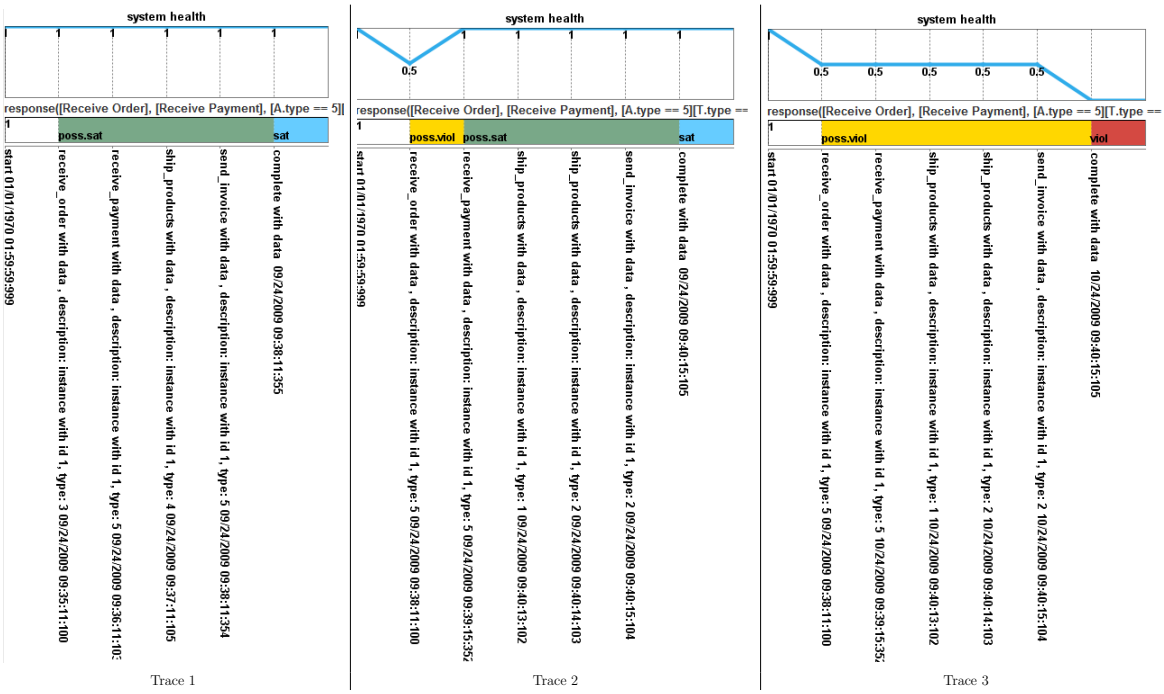


Figure 7.2: Output

Receive Payment with data *diagnosis* = 5 within an hour as specified in the model. In trace 3 *Receive Order* occurs with *diagnosis* = 5 followed by *Receive Payment* with data *diagnosis* = 5 however it occurs next day therefore violating the time conditions.

We can also see from the results that the health of the system is also indicated correctly. In trace 1 it remains 100%. In trace 2 it drops to 50% as soon as the state becomes possibly violated but then recovers back to 100% once the conditions are satisfied. In trace 3 the health is never resumed as the constraint fails permanently at the end of the process.

7.2 Early detection of violations

7.2.1 Example 1

Our business rules are shown in Table 7.2 consists two constraints (see figure 7.3 for graphical representation) . In order to make it easier to understand the tables do not show the conditions in $A.$ and $T.$ format. This example shows approach described in section 5.1

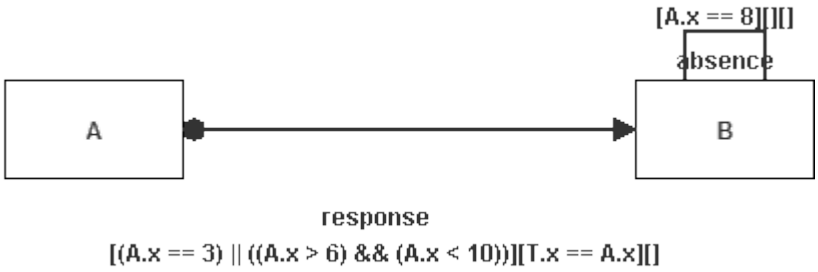


Figure 7.3: Graphical representation of example 1

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 7.2: Example 1 Rules

Our second rule requires absence of $B.x == 8$ i.e an event with activity B with data $x = 8$ should never occur. However if we look at the first rule we can see that if activity A occurs $x = 8$ then will require B should also occur with $x = 8$. We tested this model with a log of three traces with different data values.

In Trace 1 A occurs with data $x = 3$ this means that our first constraint rule is activated. So at this stage we can not say whether any or all of the rules will permanently satisfied or violated and we will have to wait for B to occur. But as we can see in figure 7.4 both rules are satisfied at the end of this trace because B occurs with $x = 3$.

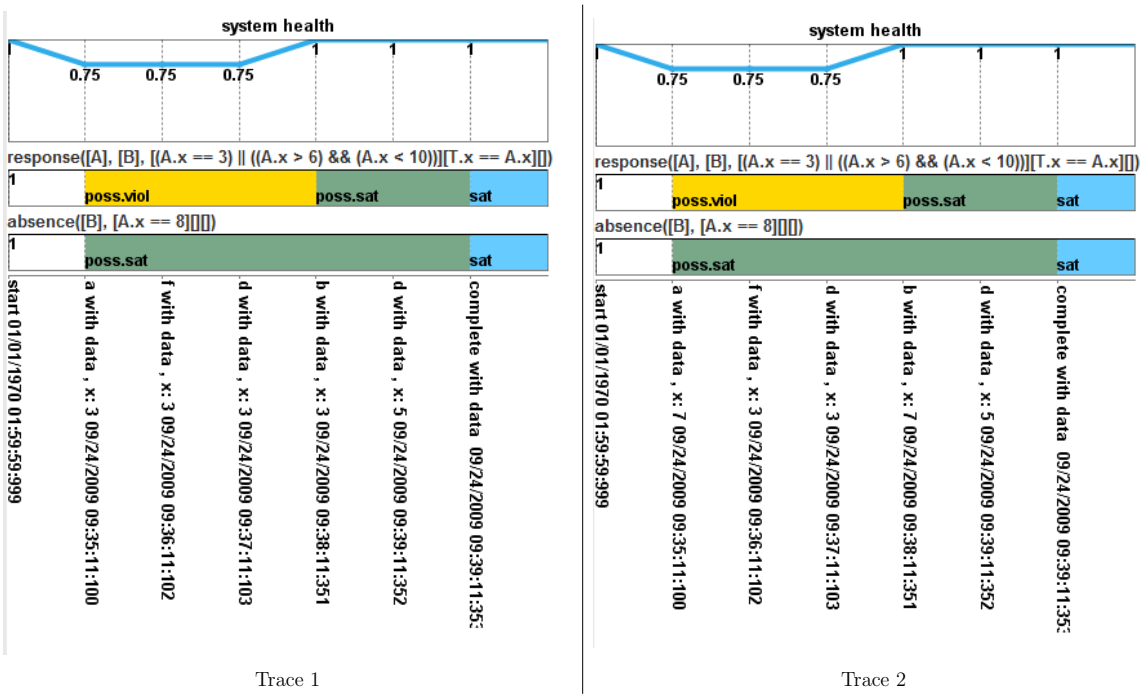
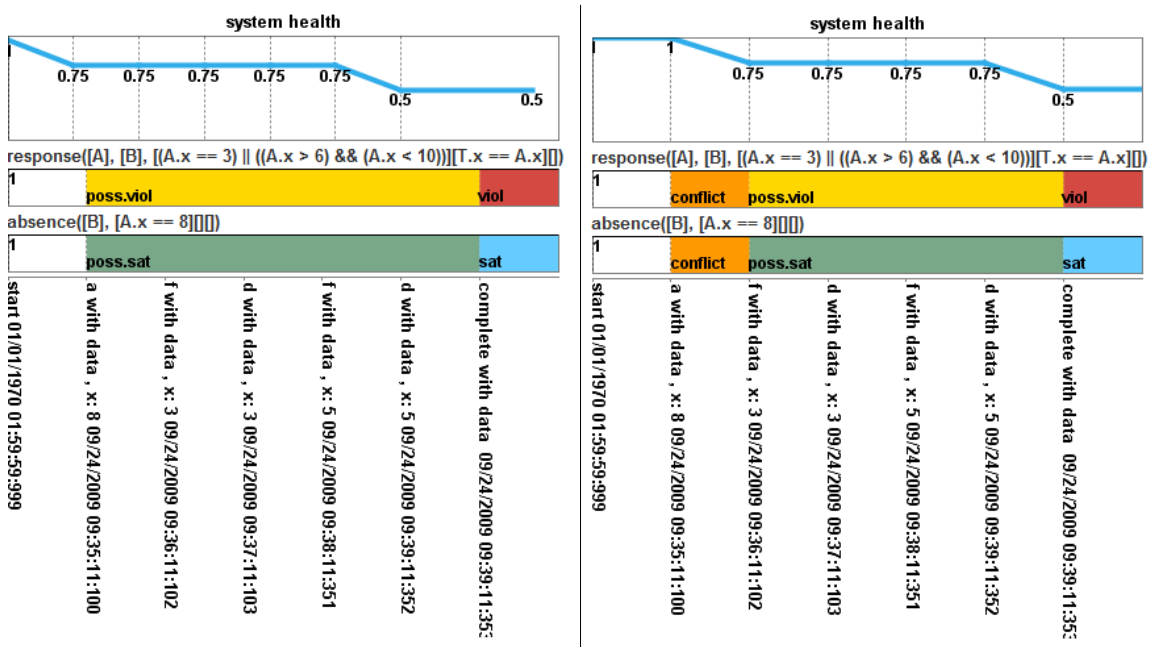


Figure 7.4: Compliance monitoring output for example 1

In Trace 2 A occurs with data $x = 7$ again this means that our first constraint rule is activated. This makes our problem set equal to $(B.x == 7) \wedge !(B.x == 8)$ So at this stage again we can not say whether any or all of the rules will permanently satisfied or violated and we will have to wait for B to occur. But as we can see in figure 7.4 both rules are satisfied at the end of this trace because B with $x = 7$ occurs.



Conflict detection turned *Off*

Conflict detection *On*

Figure 7.5: Compliance monitoring output for example 1 Trace 3

7.2.2 Example 2

Rules for this example are shown in Table 7.3 (Figure 7.6 shows the graphical representation).

This example is similar to our previous example in the sense that it will demonstrate the ability to detect conflict which occur in directly related rules. However it also demonstrates the ability of our implementation to handle strings and not just numeric data.

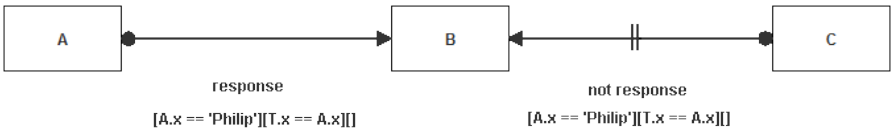


Figure 7.6: Graphical representation of example 2

id	Constraint	Activation Activity	Target Activity	Activation Condition	Correlation Condition	Time Condition
1	*response*	A	B	$A.x ==' Philip'$	$B.x == A.x$	–
2	*not response*	C	B	$C.x ==' Philip'$	$C.x == A.x$	–

Table 7.3: Example 2 Rules

As we can see in figure 7.7 in trace 1 since neither activity A occurs with data $x = Philip$ or activity C occurs with data $x = Philip$ both rules are satisfied.

In trace 2 when activity A occurs with $x = Philp$ at this point we do not detect any possible violation as related to other rules as no other rule is activated. When C occurs we detect a conflict i.e. now we can definitely say that at least one of the two rules will be violated. It is interesting to note here that without the indication of a conflict it would look like that the second rule is most likely to be violated as it has remained in the state of possible violation till this point. As we see at the end of the trace that when activity B occurs it is not the second by the first rule that is violated. We were able to detect this possibility of this happening early on in the process.

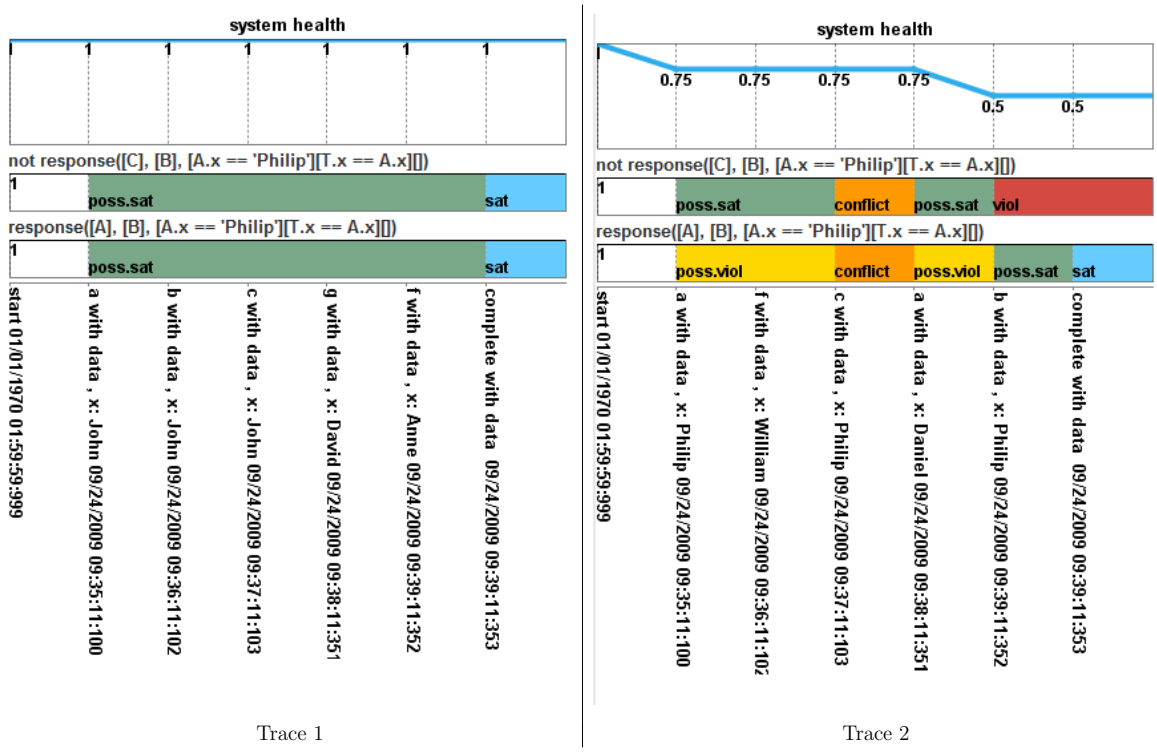


Figure 7.7: Compliance monitoring output for Example 2

Figure 7.8 shows the conflict in the model in conflict state. We are able to detect possible violation with respect to B was soon as A and C have occurred.

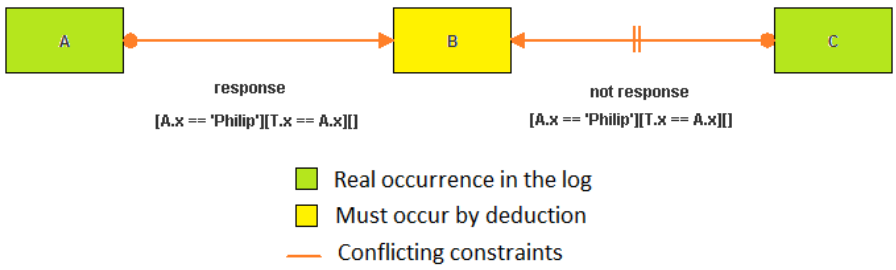


Figure 7.8: Graphical representation of example 2

7.2.3 *Example 3*

Rules are shown in Table 7.4 (Figure 7.9 shows the graphical representation).

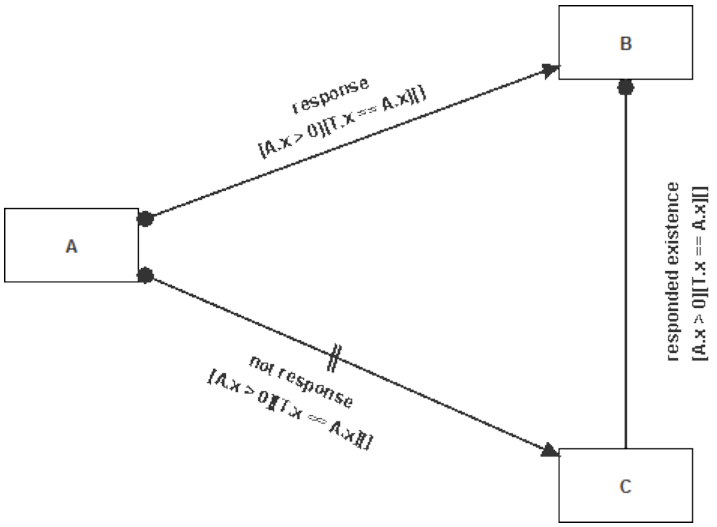


Figure 7.9: Graphical representation of example 3

id	Constraint	Activation Activity	Target Activity	Activation Condition	Correlation Condition	Time Condition
1	*responded existence*	B	C	$(B.x > 0)$	$C.x == B.x$	–
2	*not response*	A	C	$(A.x > 0)$	$C.x == A.x$	–
3	*response*	A	B	$(A.x > 0)$	$B.x == A.x$	–

Table 7.4: Example 3 Rules

As we can see that this time we have three rules. Two of them are rule 1 and rule 2 are directly related as they share a common target activity C. Rule 2 has negation associated C i.e C should not occur with the giving data condition where as Rule 1 says that C should occur if activated and with the given data conditions. At this point it is quite difficult to see a possibility of conflict between these rules as they do not share the same activation activity like the previous two examples.

Using this example we will be able to validate our approach which was described in section 5.2.

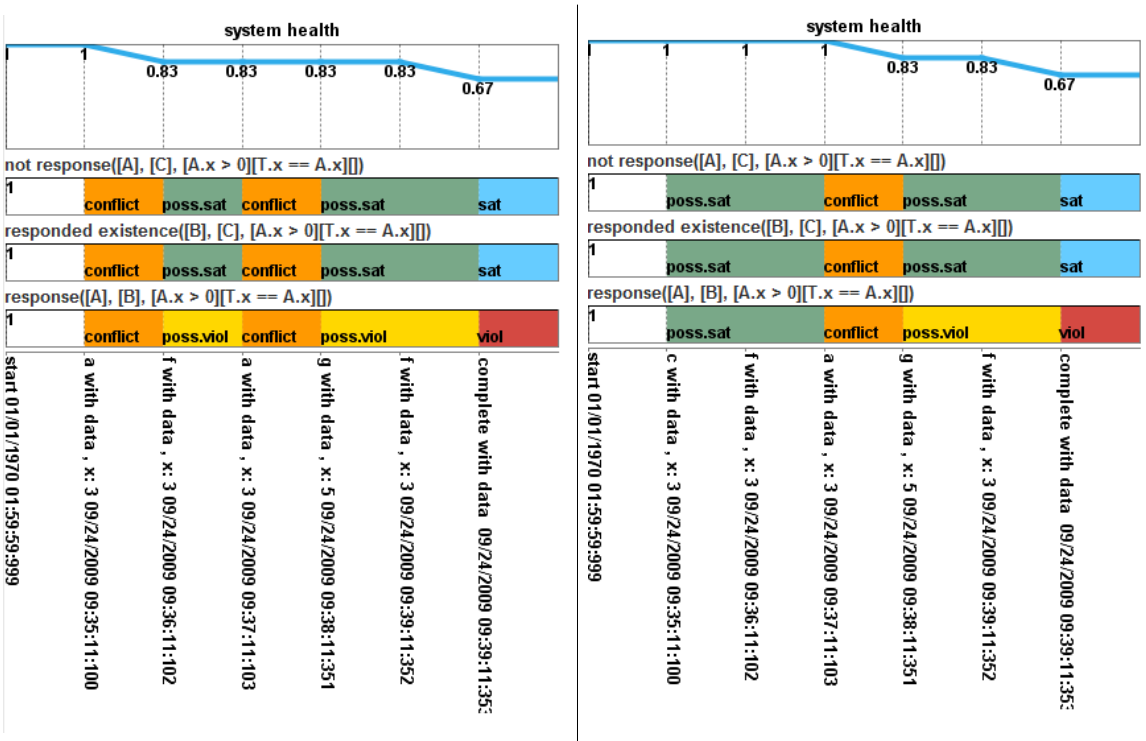


Figure 7.10: Compliance monitoring output for example 3

Figure 7.10 displays out put of trace 1 and trace 2. As we can seen as soon as A occurs satisfying the activation condition for rule 3 we are able to predict a future violation and we can definitely say one of the three rules will be permanently violated at the end of the process.

If we look at Figure 7.11 trace 3 we are able to see again that when A activates rule 1 and rule 3. We see all three rules in conflict. Interestingly when B occurs and activates rule three our prediction of violation is narrowed down to rule 1 and rule 2. As we see at the end rule 2 is violated at the end of the process. In trace 4 we can see that all the rules can be satisfied at the end of the process.

Figure 7.12 shows the conflict in the model in conflict state. It only takes occurrence of activity A to deduce that B also occur and the C must also occur resulting in a conflict.

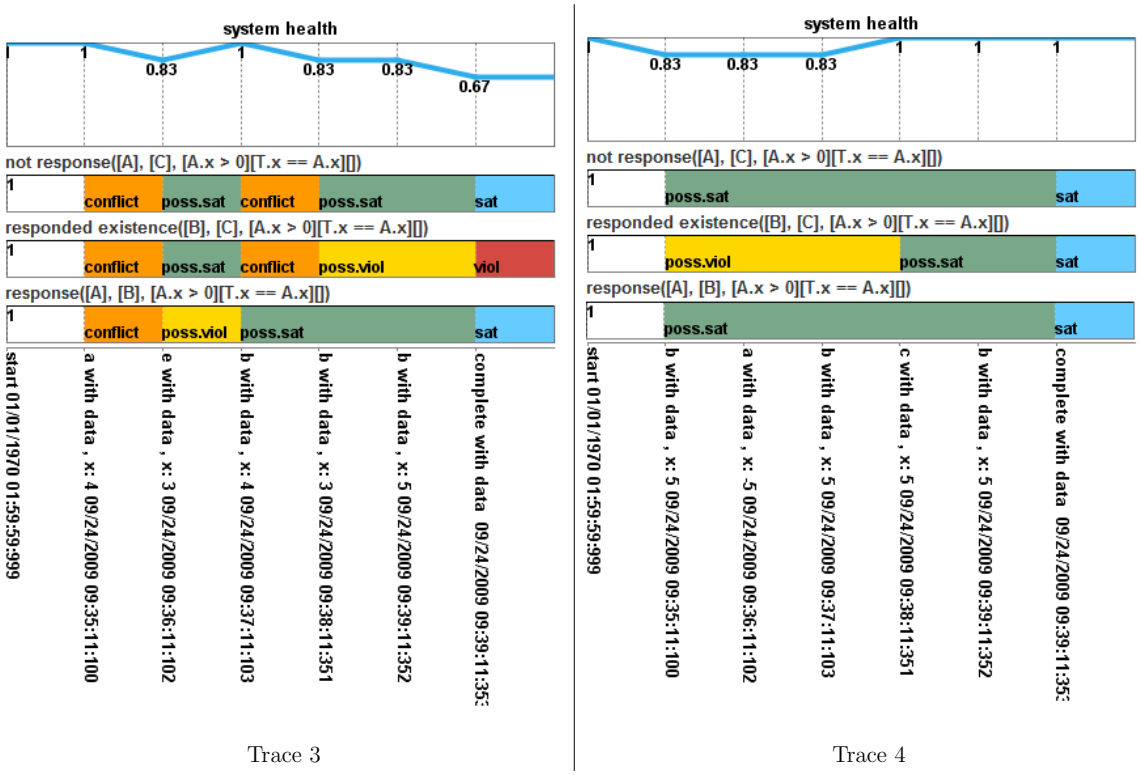


Figure 7.11: Compliance monitoring output for example 3

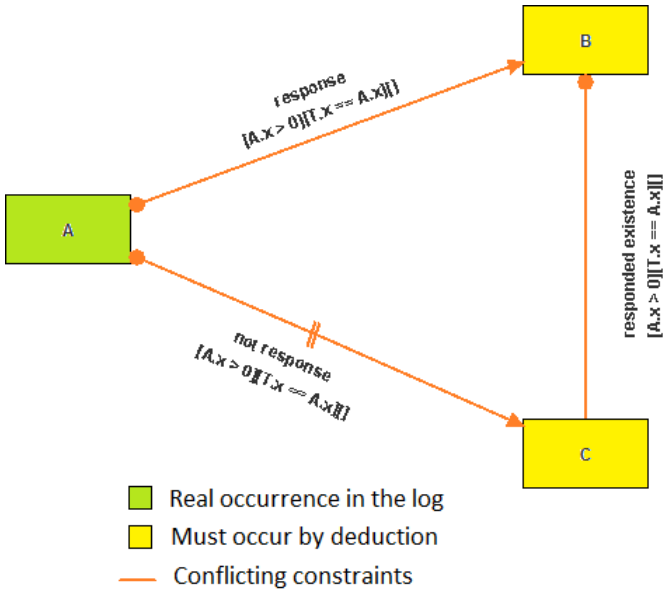


Figure 7.12: Graphical representation of example 3

7.2.4 Example 4

Our business rules are shown in Table 7.5 (Figure 7.13 shows the graphical representation).

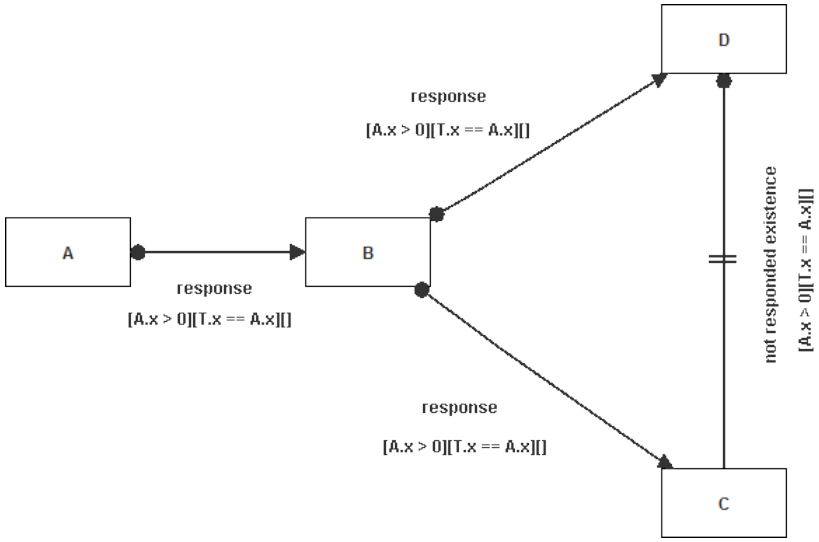


Figure 7.13: Graphical representation of example 4

id	Constraint	Activation Activity	Target Activity	Activation Condition	Correlation Condition	Time Condition
1	*not responded existence*	D	C	$(A.x > 0)$	$C.x == D.x$	–
2	*response*	A	B	$(A.x > 0)$	$B.x == A.x$	–
3	*response*	B	D	$(B.x > 0)$	$D.x == B.x$	–
4	*response*	B	C	$(B.x > 0)$	$C.x == B.x$	–

Table 7.5: Example 4 Rules

Figure 7.14 demonstrates that we were able to predict a conflicting a conflict in non related activities.

Figure 7.15 shows the conflict in the model in conflict state.

Constraints *not responded existence*(C, D) and *responce*(A, B) are activated when activities C and A occur. Using ILP we able to predicted that *responce*(B, C) and *responce*(B, D) can also be activated and are able to find the conflict.

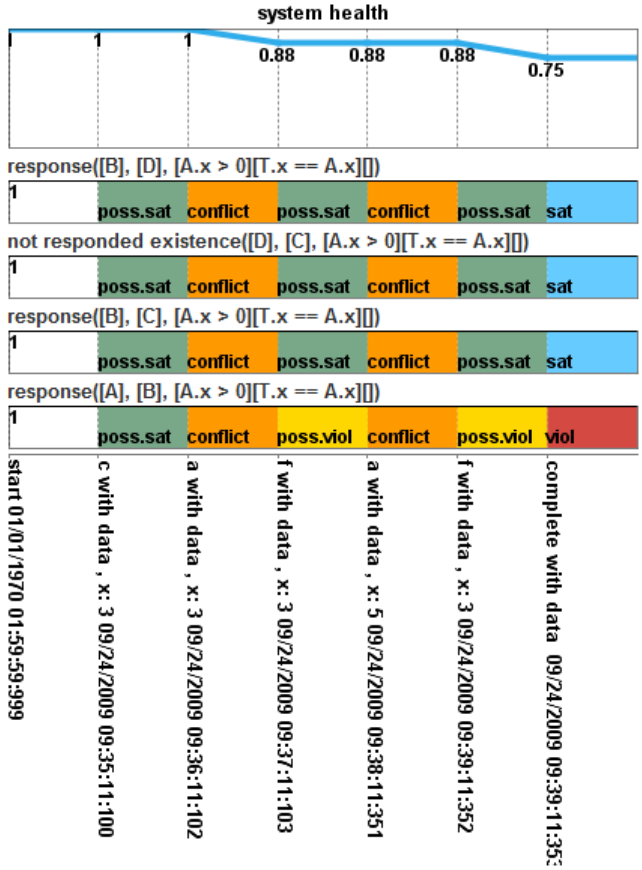


Figure 7.14: Compliance monitoring output for Example 4

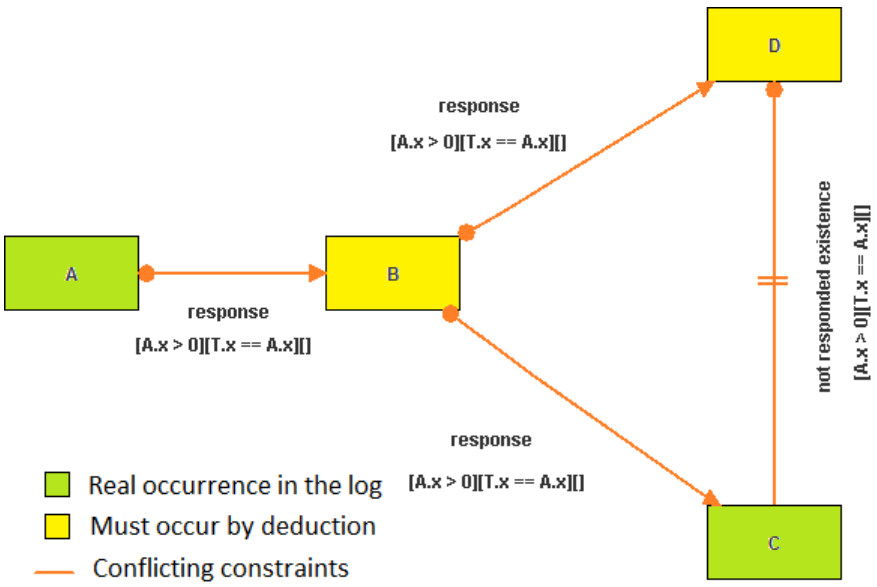


Figure 7.15: Graphical representation of example 4

7.2.5 Example 5

Our business rules are shown in Table 7.6 (Figure 7.16 shows the graphical representation).

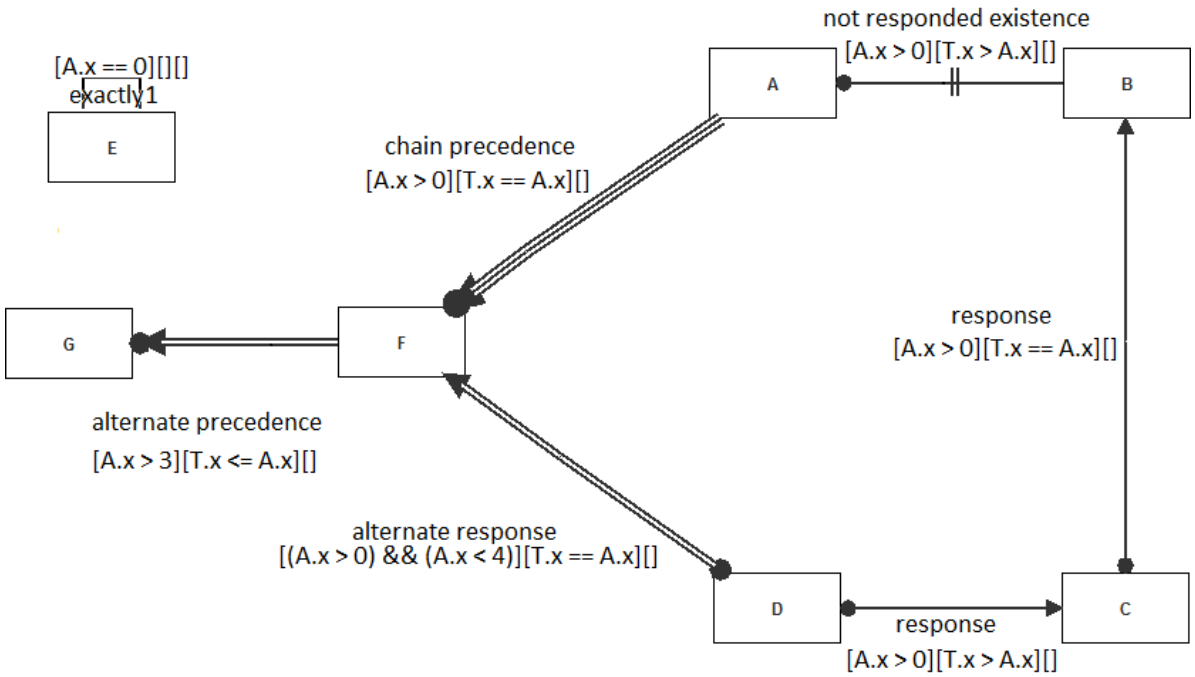


Figure 7.16: Graphical representation of example 5

id	Constraint	Activation Activity	Target Activity	Activation Condition	Correlation Condition	Time Condition
1	*not responded existence*	A	B	$(A.x > 0)$	$B.x == A.x$	–
2	*responded*	C	B	$(C.x > 0)$	$B.x == C.x$	–
3	*responded*	D	C	$(D.x > 0)$	$D.x == C.x$	–
4	*exactly*1	E	–	$(E.x == 0)$	–	–
5	*alternateprecedence*	F	G	$(F.x > 3)$	$G.x == F.x$	–
6	*chainprecedence*	A	F	$(A.x > 0)$	$F.x == A.x$	–
7	*alternateresponse*	D	G	$(D.x > 0) \wedge (D.x < 4)$	$G.x == D.x$	–

Table 7.6: Example 5 Rules

Finally 7.17 shows most complex case we are able to predict the possible violation on occurrence of D.

Figure 7.18 shows the conflict in the model in conflict state.

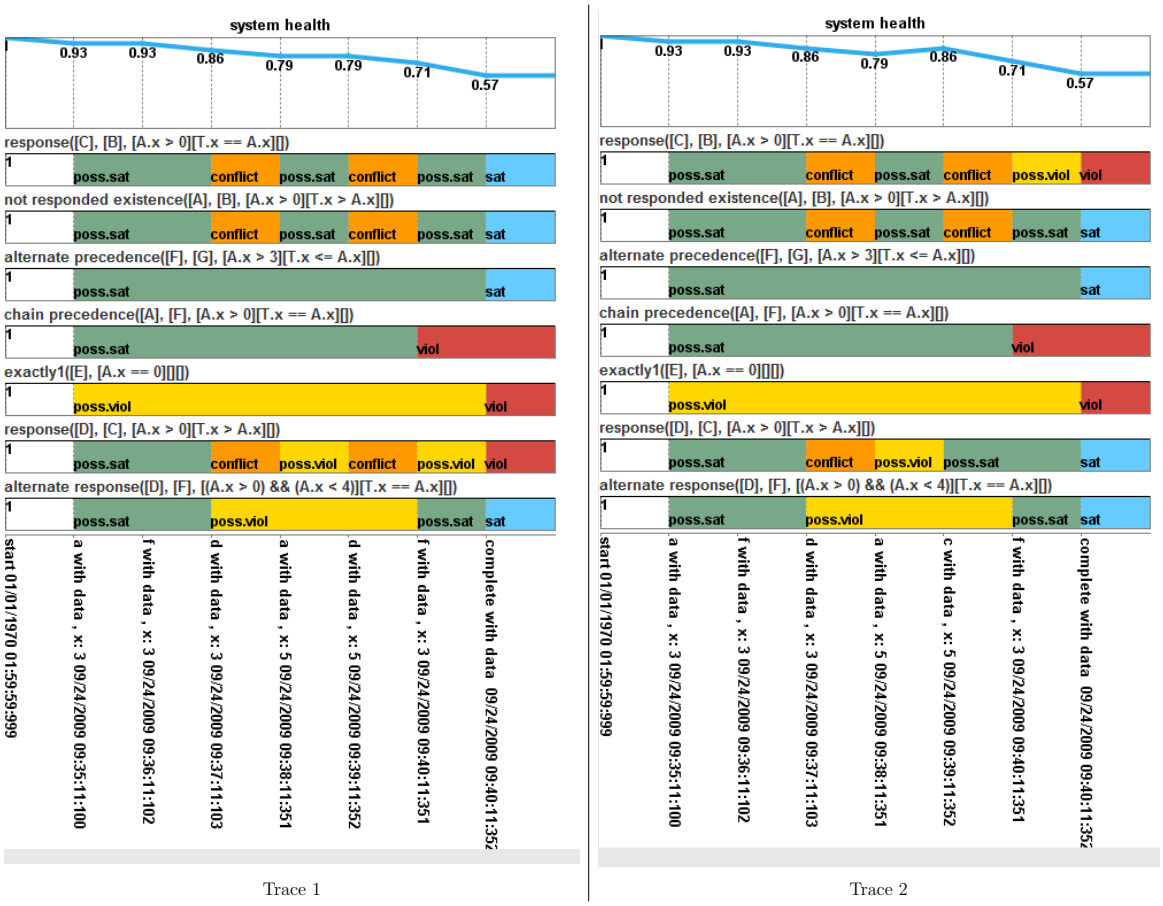


Figure 7.17: Compliance monitoring output for example 5

Constraints *not responded existence*(A, B) and *responce*(D, C) are activated when activities A and D occur. Using ILP we able to predicted that *responce*(C, B) can also be activated and are able to find the conflict.

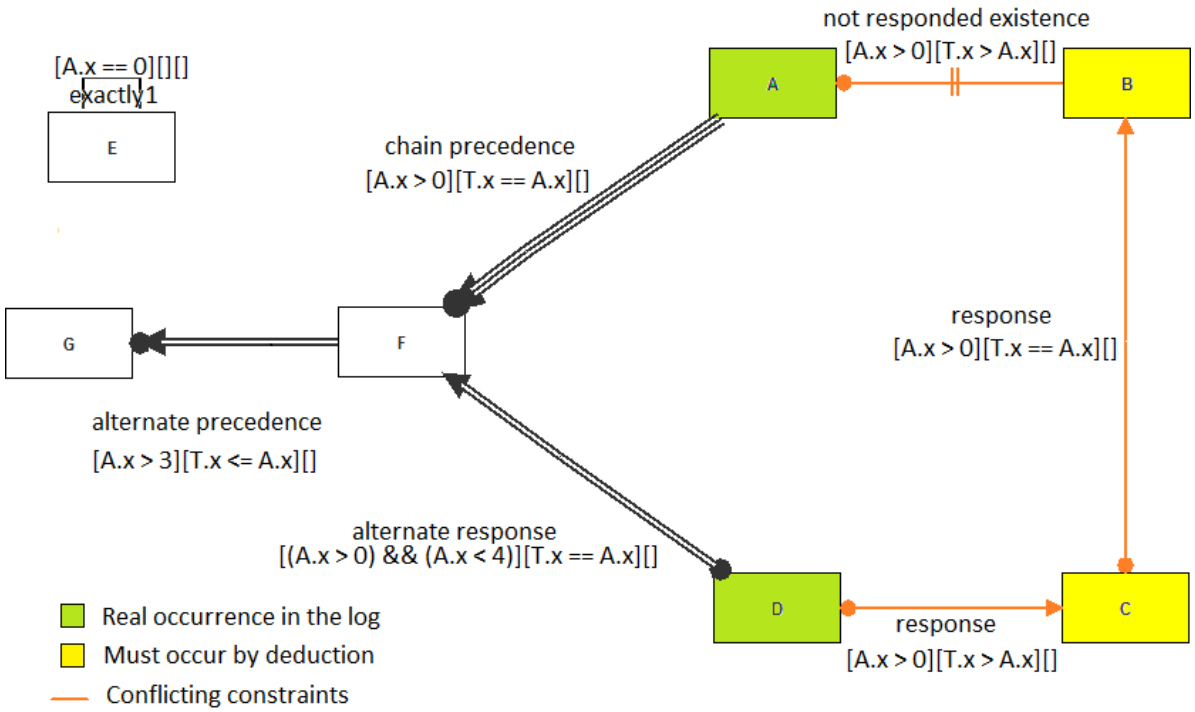


Figure 7.18: Graphical representation of example 5

7.2.6 *Example real world example*

A real world examples we have chosen to use the hospital log from BPI challenge 2011 and business rules as described in [23]. These rules had been extracted from the hospital logs.

We selected ten rules from the paper. The selected rules have been presented in Table 7.7. Rules R1 - R5 do not contain any data constraints. Respective Declare template for each rule is also shown in the table.

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 7.7: Rules and corresponding Declare Constraints

Figure 7.19 shows the final model which was used for analysis. Figure 7.20 shows the outcome for one of the traces. We did not find any conflict in any of the traces. This is because the rules have actually been extracted from the logs removing any possibility of conflicts. This also the reason for most of the rules being satisfied at the end of the process.

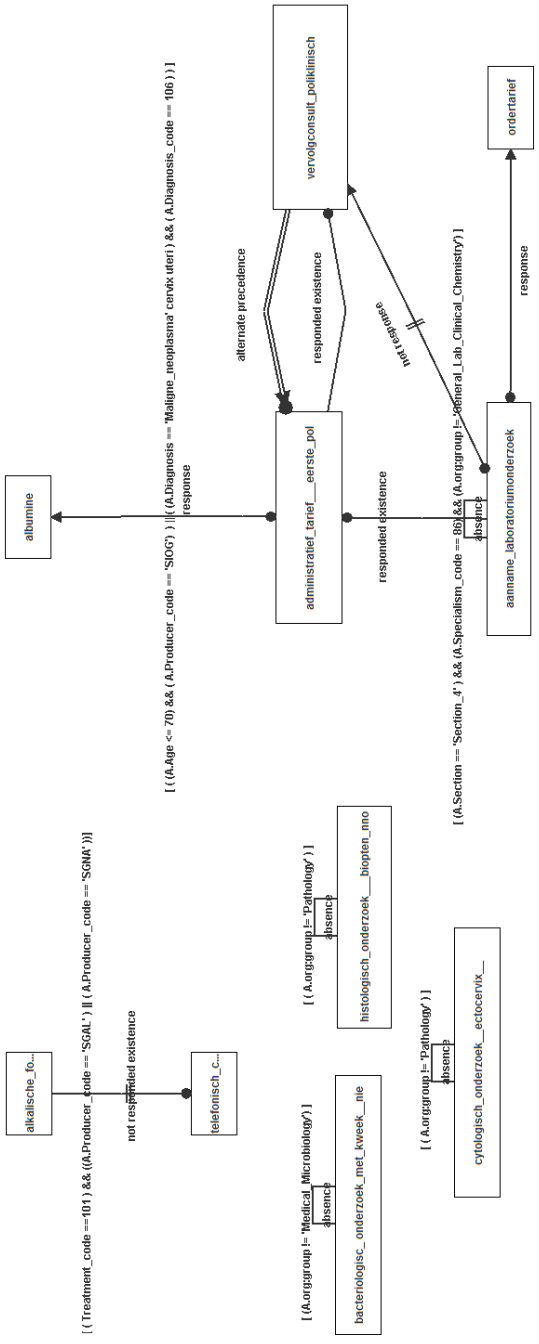


Figure 7.19: Declare model for hospital log

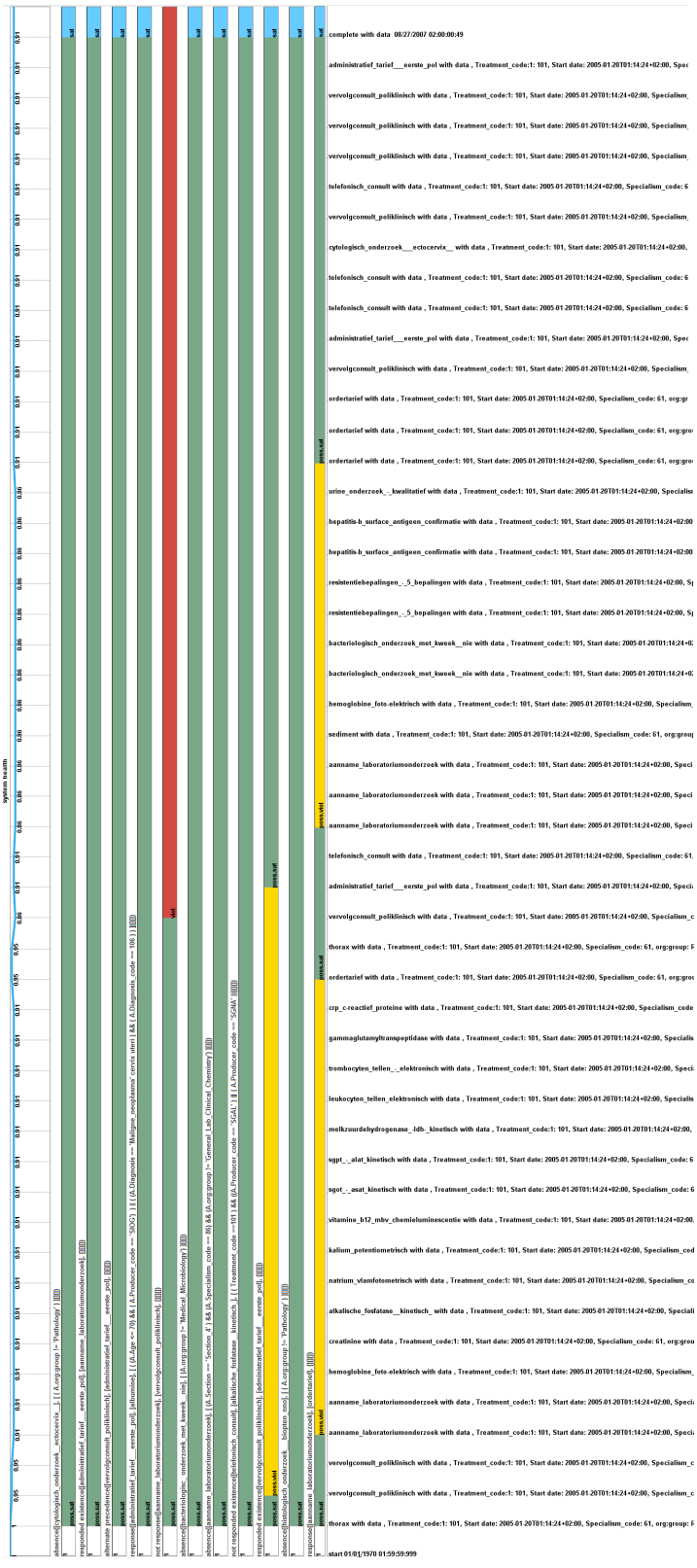


Figure 7.20: Results from runtime monitoring of hospital log

7.3 Performance

The processing time for each event depends on the complexity of the model with respect to number of business rules, complexity of the rules like And, Or operations, and how the activities are connected to each other in the rules. Time take to process each event for the real life hospital log on an average took less than 25 milliseconds. Figure 7.21 shows the time taken per event for one of the traces in hospital log.

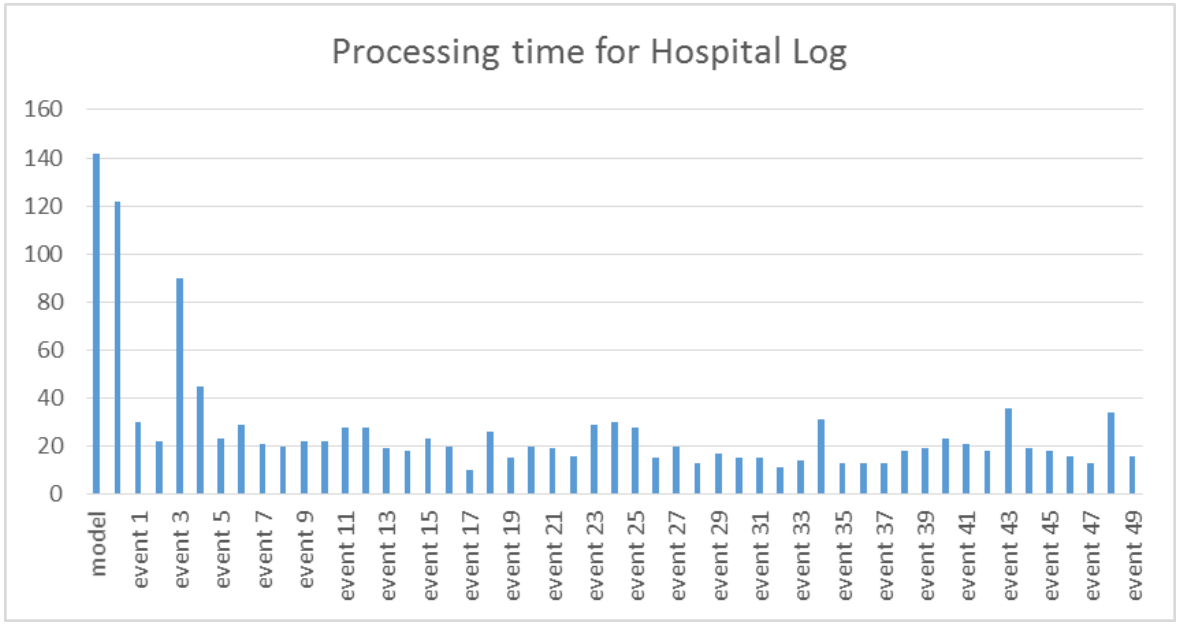


Figure 7.21: Processing time for each event in one Trace from Hospital Log

Chapter 8
Conclusion and Future Work

In the beginning of this thesis we wanted to address the following questions:

- Can sequence analysis be used to monitor the compliance of a business process with respect to complex business rules on control flow and data?
- Can Integer Linear Programming be used for early detection of violations?
- Is the proposed approach applicable to real-life case studies?

Through our approach and implementation we have demonstrated that sequence analysis can be used in a runtime setting to monitor business processes with complex business rules on control flow and data. We have demonstrated that we can use Integer Linear Programming for early detection of conflicts with business rules in a run-time environment. We were able to successfully validate our approach with synthetic as well as real world logs. We have also demonstrated that the implementation is applicable in the real world with respect to processing time and efficiency.

Future Work

In our implementation we did not consider time constraints for early detection of violations. However current implantation can easily be extended to include time constraints for early detection of violations.

Constraints that require counting like absense2, exactly1, succession etc. where out of scope of the current implementation for early detection violations. This can also be added in the future.

Another improvement that can be done to the current implementation is to design a better User Interface for visualization of results.

Bibliography

[1] Declare. https://www.win.tue.nl/declare/. 1, 4, 10

[2] Lp solver. http://lpsolve.sourceforge.net/5.5/. 33

[3] Prom. http://www.processmining.org/. 32

[4] Adriansyah, A., van Dongen, B. F., and van der Aalst, W. M. Conformance checking using cost-based fitness analysis. In *Enterprise Distributed Object Computing Conference (EDOC), 2011 15th IEEE International* (2011), IEEE, pp. 55–64. 5

[5] Burattin, A., Maggi, F. M., and Sperduti, A. Conformance checking based on multi-perspective declarative process models. *CoRR abs/1503.04957* (2015). viii, 1, 6, 8, 10, 18, 20

[6] Burattin, A., Maggi, F. M., Van der Aalst, W. M., and Sperduti, A. Techniques for a posteriori analysis of declarative processes. In *Enterprise Distributed Object Computing Conference (EDOC), 2012 IEEE 16th International* (2012), IEEE, pp. 41–50. 5

[7] Cook, J. E., and Wolf, A. L. Software process validation: quantitatively measuring the correspondence of a process to a model. *ACM Transactions on Software Engineering and Methodology (TOSEM) 8*, 2 (1999), 147–176. 5

[8] de Leoni, M., Munoz-Gama, J., Carmona, J., and Van der Aalst, W. M. Decomposing alignment-based conformance checking of data-aware process models. In *On the Move to Meaningful Internet Systems: OTM 2014 Conferences* (2014), Springer, pp. 3–20. 5

[9] Federici, M., Rizzi, W., Di Francescomarino, C., Dumas, M., Ghidini, C., Maggi, F. M., and Teinemaa, I. A prom operational support provider for predictive monitoring of business processes. 32

[10] Günther, C. Xes-standard. http://www.xes-standard.org. vii, 8, 9

[11] He, C., and Ma, C. Measuring behavioral correspondence to a timed concurrent model. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01)* (2001), IEEE Computer Society, p. 332. 5

[12] Ly, L. T., Maggi, F. M., Montali, M., Rinderle-Ma, S., and van der Aalst, W. M. P. Compliance monitoring in business processes: Functionalities, application, and tool-support. *Inf. Syst. 54* (2015), 209–234. vii, 4, 6, 7

[13] Maggi, F. M. Mobuconltl, prom plugin. https://svn.win.tue.nl/repos/prom/Packages/MoBuConLTL/Trunk. 6

[14] Maggi, F. M. Declarative process mining with the declare component of prom. In *BPM (Demos)* (2013). 32

[15] Maggi, F. M., Montali, M., Westergaard, M., and Van Der Aalst, W. M. Monitoring business constraints with linear temporal logic: An approach based on colored automata. In *Business Process Management*. Springer, 2011, pp. 132–147. 6

[16] Maggi, F. M., Mooij, A. J., and van der Aalst, W. M. P. User-guided discovery of declarative process models. In *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2011, part of the IEEE Symposium Series on Computational Intelligence 2011, April 11-15, 2011, Paris, France* (2011), pp. 192–199. 13

[17] Maggi, F. M., Westergaard, M., Montali, M., and van der Aalst, W. M. Runtime verification of ltl-based declarative process models. In *Runtime Verification* (2012), Springer, pp. 131–146. 6

[18] Maggi, F. M., Westergaard, M., Montali, M., and van der Aalst, W. M. P. Runtime verification of ltl-based declarative process models. In *Runtime Verification - Second International Conference, RV 2011, San Francisco, CA, USA, September 27-30, 2011, Revised Selected Papers* (2011), pp. 131–146. 6

[19] Montali, M., Maggi, F. M., Chesani, F., Mello, P., and van der Aalst, W. M. Monitoring business constraints with the event calculus. *ACM Transactions on Intelligent Systems and Technology (TIST) 5*, 1 (2013), 17. 6

[20] Murata, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE 77*, 4 (1989), 541–580. 1, 4

[21] OMG, O. Business process model and notation (bpmn) version 2.0. *Object Management Group* (2011). 1, 4

[22] Pesic, M. *Constraint-based workflow management systems: shifting control to users*. PhD thesis, Technische Universiteit Eindhoven, 2008. 13

[23] Pesic, M., Schonenberg, H., and Van der Aalst, W. M. Declare: Full support for loosely-structured processes. In *Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International* (2007), IEEE, pp. 287–287. 10, 51

[24] Pesic, M., and Van der Aalst, W. M. A declarative approach for flexible business processes management. In *Business Process Management Workshops* (2006), Springer, pp. 169–180. 10

[25] Rozinat, A., and van der Aalst, W. M. Conformance checking of processes based on monitoring real behavior. *Information Systems 33*, 1 (2008), 64–95. 5

[26] Van Der Aalst, W. *Process mining: discovery, conformance and enhancement of business processes*. Springer Science & Business Media, 2011. 4, 8

Non-exclusive licence to reproduce thesis and make thesis public

I, Ubaier Ahmad Bhat (date of birth: 17th of February 1985),

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

1.1 reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2 make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

Runtime Monitoring of Data-Aware business rules with Integer Linear Programming

supervised by Fabrizio Maggi. PhD.

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 19.05.2016