

UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Denizalp KAPISIZ
Rule Mining with RuM
Master's Thesis (30 ECTS)

Supervisor: Fabrizio Maria Maggi

Tartu 2019

Rule Mining with RuM

Abstract: Process Mining is a data science practice in Business Process Management (BPM). It is based on the analysis of process execution logs. Nowadays, ProM and Disco are two mainstream desktop applications for process mining. Disco can be used for procedural process discovery with intuitive user experience. However, it does not support declarative models. ProM gives support for a wealth of techniques for declarative process mining. However, its user interface is not so intuitive as the one of Disco. In the light of this, this Master's Thesis is focused on proposing RuM, an application for declarative process analysis with an intuitive user experience.

The application presents an environment for a set of process discovery, conformance checking and log generation methods based on declarative models. The methods can be configured with various options and the results can be post-processed to simplify and store them in the file system. The tool has been evaluated through a usability test involving 5 experts on declarative process analysis.

Keywords: Process mining, Declarative process models, Process discovery, Conformance checking, Process analytics tool

CERCS: P170 - Computer Science, Numerical Analysis, Systems, Control

Reeglikaeve tööriistaga RuM

Lühikokkuvõte: Protsessikaeve on äriprotsesside juhtimise (BPM) valdkonda kuuluv andmeteaduse tehnika, mille lähtepunktiks on protsessi täitmise logide analüüsimine. Hetkel kasutatakse protsessikaeve jaoks põhiliselt kahte tööluarakendust: Disco ja ProM. Disco sobib protseduuriliseks protsesside tuvastamiseks ja on intuitiivse kasutusliidesega, aga ei toeta deklaratiivseid mudeleid. ProM toetab mitmeid erinevaid deklaratiivse protsessikaeve tehnikaid, aga on ebaintuitiivsema kasutusliidesega kui Disco. Antud magistritöö eesmärk on välja pakkuda uus deklaratiivsele protsessianalüütikale suunatud tööriist RuM, mis pakub ühtlasi intuitiivset kasutajakogemust.

RuM pakub sobiva keskkonna protsesside väljaselgitamise, vastavuskontrolli ja logide genereerimise meetodite jaoks. Kõik RuM poolt toetatud meetodid tuginevad deklaratiivsetel mudelitel ja on kasutaja poolt seadistatavad. Meetodite tulemusi on võimalik lihtsustamise eesmärgil järeltöödelda ja failisüsteemi salvestada. RuM tööluarakenduse hindamiseks on teostatud kasutatavuse testimine, milles osales viis deklaratiivse protsessianalüütika valdkonna ekspert.

Võtmesõnad: Protsessikaeve, deklaratiivse protsessikaeve mudelid, protsesside väljaselgitamine, vastavuskontroll, protsessianalüütika tööriist

CERCS: P170 - Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Table of Contents

1. Introduction.....	5
2. Background.....	6
2.1. Process Mining.....	6
2.1.1. Process Discovery.....	7
2.1.2. Conformance Checking.....	7
2.2. Process mining tools.....	8
2.2.1. Disco.....	8
2.2.2. ProM.....	9
2.2.3. Apromore.....	10
2.3. The DECLARE Modeling Language.....	10
2.3.1. Existence Templates.....	11
2.3.2. Relation Templates.....	11
2.3.3. Negative Relation Templates.....	13
2.4. Multi-Perspective Declare.....	14
3. Tool architecture.....	16
3.1. Controller architecture.....	17
3.2. Model architecture.....	17
3.3. View architecture.....	17
3.4. Technologies.....	17
4. Functional overview.....	18
4.1. Process Discovery.....	18
4.1.1. The Declare Miner method.....	18
4.1.2. The Minerful method.....	19
4.1.3. Declarative model views.....	20
4.1.3.1. Textual representation.....	20
4.1.3.2. Declare representation.....	20

4.1.3.3. Automaton representation.....	23
4.1.4. Discovery tab.....	24
4.2. Conformance Checking.....	26
4.2.1. Changes in the DECL format.....	27
4.2.2. The Declare Analyzer method.....	27
4.2.3. The Declare Replayer method.....	28
4.2.4. The DataAware Declare Replayer method.....	28
4.2.5. Conformance Checking tab.....	29
4.3. Log Generation.....	35
4.3.1. The Alloy Log Generator method.....	35
4.3.2. The Minerful Log Generator method.....	35
4.3.3. Log Generation tab.....	35
4.4. MP-Declare Editor tab.....	37
5. Evaluation.....	41
5.1. User tests.....	41
5.2. Performance.....	43
5.2.1. Process Discovery methods.....	44
5.2.2. Log Generation methods.....	44
5.2.3. Conformance Checking methods.....	45
6. Conclusion.....	48
7. References.....	49
License.....	51

1. Introduction

The revolutionary advancement in data storage and central processing unit has given birth to data science practices in Business Process Management (BPM). A way to conduct these practices is to collect the outcomes of the processes and execute data mining and machine learning techniques on them [1]. This group of studies is known as process mining.

In process mining, there are three main branches: process discovery, conformance checking and process enhancement. Process discovery consists in constructing a model describing the process starting from executions of the process. Conformance checking consists in comparing an existing process model with process executions to identify discrepancies between them. Process enhancement consists in improving an existing process model with information coming from process executions.

Sometimes to test process mining techniques it is possible to simulate a process model thus producing an event log. In this context, an event log refers to a collection of the executions of the process. Log generation techniques can produce event logs with different characteristics that can be used to test process mining techniques under different conditions.

Process mining techniques are implemented in some desktop applications. The most prominent one is ProM [2]. It enables the users to work with a wealth of process mining algorithms. Another desktop application is Disco [3]. It introduces a lightweight framework only for process discovery with procedural models. Compared to ProM, its user interface is more intuitive in that it is easier to use and has a more user-friendly representation for the results. However, ProM provides more functionalities than Disco and, in particular, it supports declarative process mining, i.e., process mining based on declarative models. As a result, a desktop application supporting the declarative process mining functionalities available in ProM with a user interface similar to Disco can be developed.

In this thesis we aim at implementing RuM, a process mining tool that supports process discovery, conformance checking and log generation methods based on declarative models and with an intuitive user interface. The interface enables switching between the functionalities quickly and post-processing the output models with different options.

The thesis is organized as follows. Section 2 introduces the preliminaries including process mining, its branches and DECLARE. Section 3 introduces the architecture of the tool with benefited technologies. Section 4 presents an overview of the RuM functionalities. Section 5 discusses the evaluation of the tool. Section 6 wraps up the paper and highlights some future work.

2. Background

2.1 Process Mining

Process Mining can be defined as a group of algorithms to study a business process from its event logs [1]. An event log is a set of process execution traces and can be considered as a table with rows representing events, and columns including a trace identifier, an activity name, a timestamp and other event attributes. Events in a trace are ordered by timestamp.

Trace identifier	Name	Timestamp
Case No. 1	Activity 1	2019-02-02T09:00:00+02:00
Case No. 1	Activity 2	2019-02-02T10:00:00+02:00
Case No. 2	Activity 1	2019-02-03T09:00:00+02:00
Case No. 2	Activity 3	2019-02-03T11:00:00+02:00
Case No. 2	Activity 2	2019-02-03T11:30:00+02:00

Table 1 – Example log table

A log can be represented as an XES [4] (Extensible Event Stream) file or a MXML (Mining Extensible Markup Language) file [5]. In Figure 1, these representations are illustrated with the example log in Table 1.

```

<?xml version="1.0" encoding="UTF-8" ?>
<WorkflowLog>
  <Process id="Sample Process" description="A process">
    <ProcessInstance id="Case No. 1" description="First case">
      <WorkflowModelElement>Activity 1</WorkflowModelElement>
      <timestamp>2019-02-02T09:00:00+02:00</timestamp>
    </ProcessInstance>
    <ProcessInstance id="Case No. 2" description="Second case">
      <WorkflowModelElement>Activity 1</WorkflowModelElement>
      <timestamp>2019-02-02T10:00:00+02:00</timestamp>
    </ProcessInstance>
    <ProcessInstance id="Case No. 2" description="Second case">
      <WorkflowModelElement>Activity 3</WorkflowModelElement>
      <timestamp>2019-02-03T11:00:00+02:00</timestamp>
    </ProcessInstance>
    <ProcessInstance id="Case No. 2" description="Second case">
      <WorkflowModelElement>Activity 2</WorkflowModelElement>
      <timestamp>2019-02-03T11:30:00+02:00</timestamp>
    </ProcessInstance>
  </Process>
</WorkflowLog>
  
```

```

<?xml version="1.0" encoding="UTF-8" ?>
<log xes.version="1.0" xes.features="nested-attributes" openxes.version="1.0RC7" xmlns="http://www.xes-standard.org/">
  <trace>
    <string key="concept:name" value="Case No. 1"/>
    <event>
      <string key="concept:name" value="Activity 1"/>
      <date key="time:timestamp" value="2019-02-02T09:00:00+02:00"/>
    </event>
    <event>
      <string key="concept:name" value="Activity 2"/>
      <date key="time:timestamp" value="2019-02-02T10:00:00+02:00"/>
    </event>
  </trace>
  <trace>
    <string key="concept:name" value="Case No. 2"/>
    <event>
      <string key="concept:name" value="Activity 1"/>
      <date key="time:timestamp" value="2019-02-03T09:00:00+02:00"/>
    </event>
    <event>
      <string key="concept:name" value="Activity 3"/>
      <date key="time:timestamp" value="2019-02-03T11:00:00+02:00"/>
    </event>
    <event>
      <string key="concept:name" value="Activity 2"/>
      <date key="time:timestamp" value="2019-02-03T11:30:00+02:00"/>
    </event>
  </trace>
</log>
  
```

Figure 1 – MXML (left) and XES (right) format for example log in Table 1

2.1.1 Process Discovery

Process discovery is a branch of process mining including techniques for building a process model from an event log without any a priori knowledge. The model can be a procedural or declarative [6]. A procedural model includes all possible sequences found in the traces. Each trace in the log must fit one of the sequences. A declarative model includes a list of constraints over activities. The constraints must not be violated by any trace in the log.

Example 1: The constraints in Figure 2 can be discovered from the log in Table 1.

Constraints:

- 1- A trace must begin with Activity 1.
- 2- A trace must finish with Activity 2.
- 3- In a trace, before Activity 3 starts, Activity 1 must be completed.

Figure 2 – A declarative model discovered from the log in Table 1

2.1.2 Conformance Checking

Conformance checking is a branch of process mining that evaluates if there are discrepancies between an event log and an existing process model.

Example 2: Consider the log in Table 1 and the following declarative model

Constraints:

- 1. Before Activity 2 starts, Activity 3 must be completed.
- 2. After Activity 1 is completed, Activity 2 must be completed in the next two hours.

The first constraint is violated in the first trace and is fulfilled in the second trace. The second constraint is fulfilled in both traces. If we remove the first constraint from the model, then the model reflects the log better than before.

2.2 Process mining tools

2.2.1 Disco

Disco [3] is a lightweight desktop application to run a process discovery algorithm resulting in a procedural model. Figure 3 illustrates a screenshot after an event log is imported. The main panel is reserved for the representation of the model. The right panel is used to post-process the model such as displaying different metrics and filtering out some events and paths. The application is composed of three tabs: Map, Statistics and Cases. The user can find statistics about traces and events in the log in the Statistics tab. The Cases tab gives detailed information about each trace in the log.

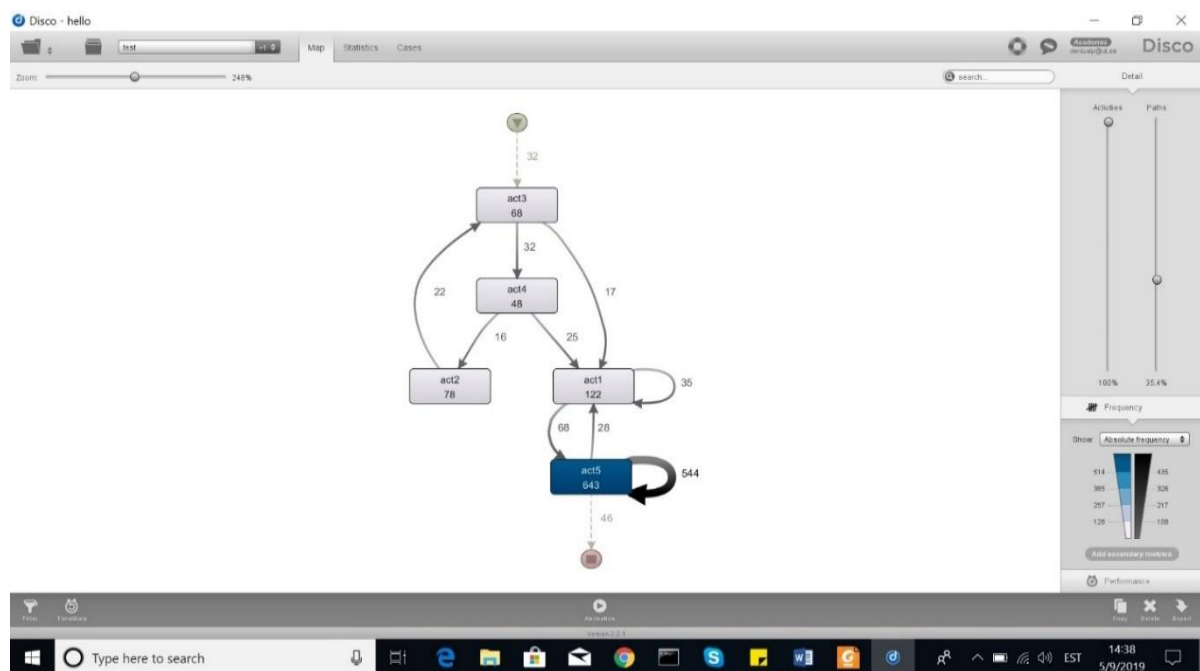


Figure 3 – Disco [3]

2.2.2 ProM

ProM is a desktop application supporting a variety of process mining algorithms as plugins [2]. Figure 4 shows the main panel of the application. The application is composed of three tabs: one for providing the inputs, one for selecting the plugins and one for showing the outputs. In the second tab, it is possible to see the available algorithms. After selecting one, the required input(s) are displayed. They can be either imported from the file system in the first tab or be generated using other algorithms. In the third tab, it is possible to visualize the outputs.

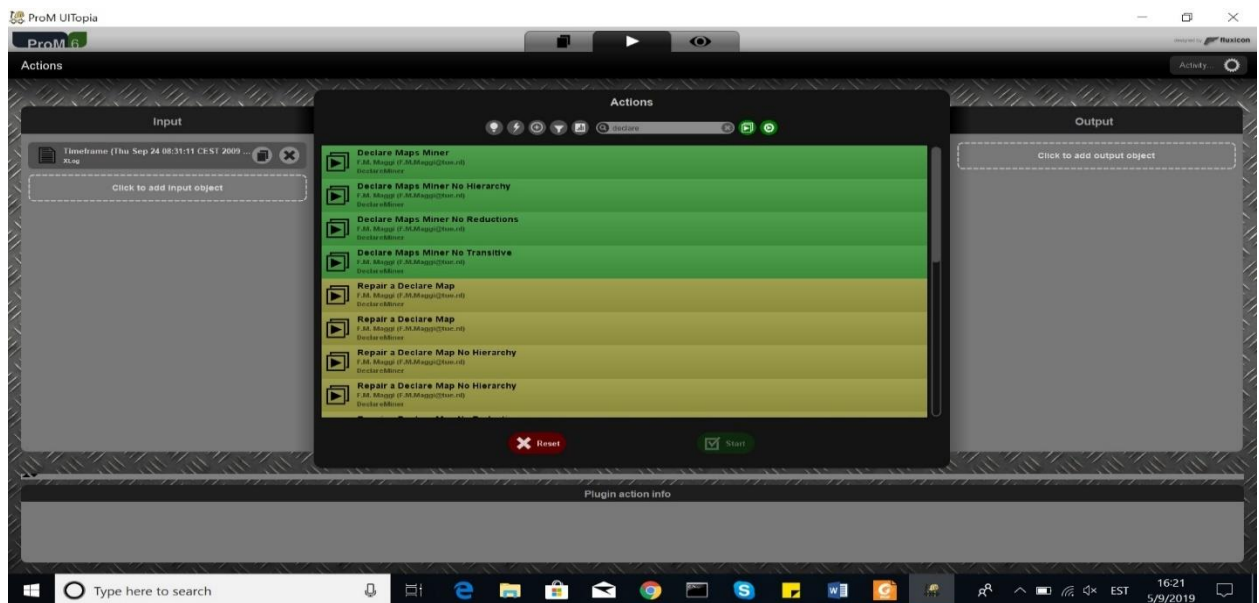


Figure 4 – ProM [2]

2.2.3 Apromore

Apromore is a web portal supporting a variety of process analytics operations such as discovering BPMN models and checking process models with respect to Declare constraints [23]. Figure 5 shows six menus: File, Discover, Analyze, Redesign, Implement and Monitor. File is used to create, edit, import and export input files. The remaining menus contain different process mining methods. After selecting a method, it is possible to choose an input either from the Folders available in the left panel of the application or from the file system. Then, the user can choose the settings and execute the method. It is also possible to store the output objects with different formats in the file system.

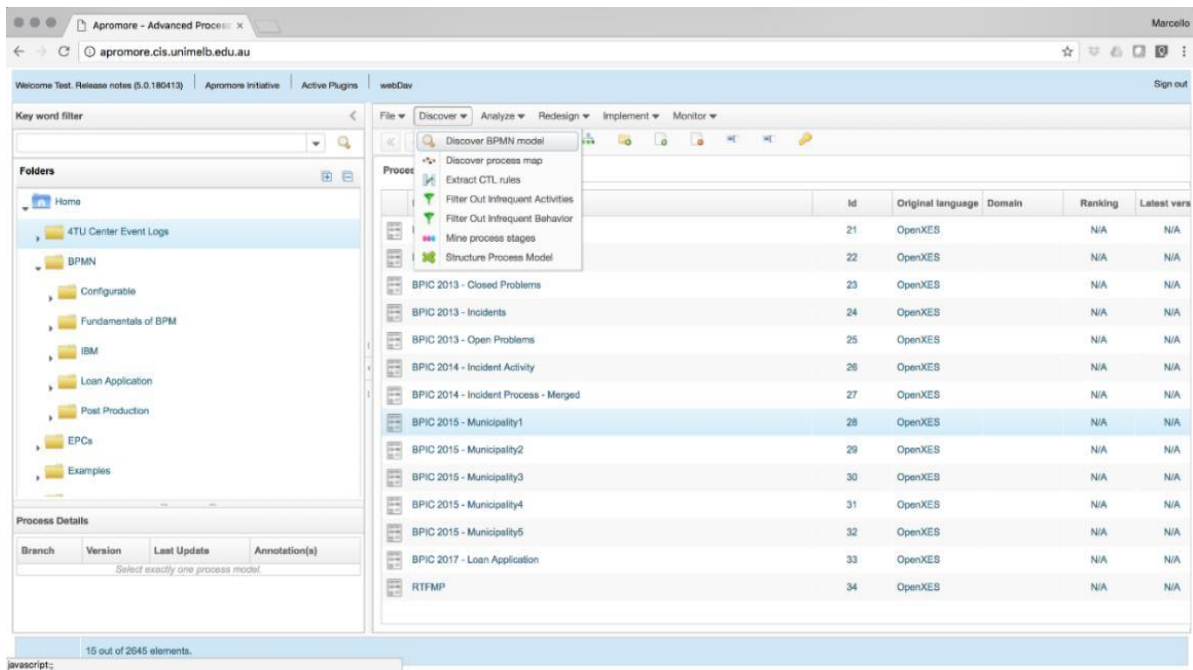


Figure 5 – Apromore [23]

2.3 The DECLARE Modeling Language

A declarative model can be represented using the DECLARE modeling language [7]. This language is composed of templates over activities. Each template represents a type of constraint on the number of occurrences or the position of activities in a trace. If a trace activates a constraint and obeys its rule, then we say that the trace satisfies the constraint. If it does not obey the rule, then we say that the trace violates the constraint. However, the trace may not activate a constraint at all. In this case, it is said that the trace vacuously satisfies the constraint [8].

The DECLARE templates are divided into three main groups: Existence, Relation, and Negative Relation.

2.3.1 Existence Templates

These templates constrain a single activity. There are five templates in this category: INIT, END, EXISTENCE, ABSENCE and EXACTLY. Table 2 presents each existence template with its description and graphical representation. The involved activity is the activation for this type of constraints.

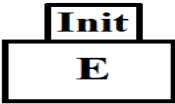
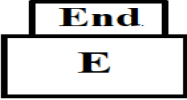
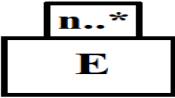
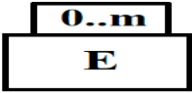
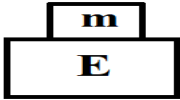
Name	Description	Representation
INIT(E)	The trace must start with E	
END(E)	The trace must finish with E	
EXISTENCE(n, E)	E must occur in the trace at least n times	
ABSENCE(m, E)	E must occur in the trace at most (m-1) times	
EXACTLY(m, E)	E must occur in trace exactly m times	

Table 2 – Existence templates

2.3.2 Relation Templates

A Relation template involves two activities. These templates are described in Table 3. Here, the underlined activity is the activation and the other activity is the target. If both activities are underlined, then they are both activation and target. The hierarchy (based on implication) of Relation templates is shown in Figure 6.

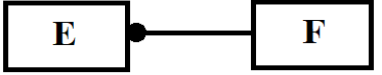



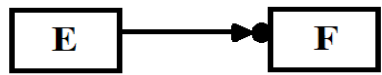


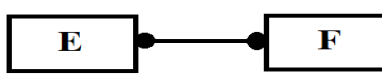
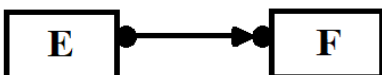


Name	Description	Representation
RESPONDED EXISTENCE(<u>E</u> , F)	If E occurs, then F occurs as well	
RESPONSE(<u>E</u> , F)	If E occurs, then F occurs after E	
ALTERNATE RESPONSE(<u>E</u> , F)	If E occurs, then F occurs afterwards before E recurs	
CHAIN RESPONSE(<u>E</u> , F)	If E occurs, then F immediately occurs afterwards	
PRECEDENCE(E, <u>F</u>)	F occurs if it is preceded by E	
ALTERNATE PRECEDENCE(E, <u>F</u>)	F occurs if it is preceded by E and no other F recurs in between	
CHAIN PRECEDENCE(E, <u>F</u>)	F occurs if it is immediately preceded by E	
CO-EXISTENCE(<u>E</u> , <u>F</u>)	If E occurs then F occurs and vice versa	
SUCCESSION(<u>E</u> , <u>F</u>)	F must occur after E and E must occur before F	
ALTERNATE SUCCESSION(<u>E</u> , <u>F</u>)	F must occur after E and E must occur before F and they alternate each other	
CHAIN SUCCESSION(<u>E</u> , <u>F</u>)	F must occur immediately after E and E must occur immediately before F	

Table 3 – Relation templates

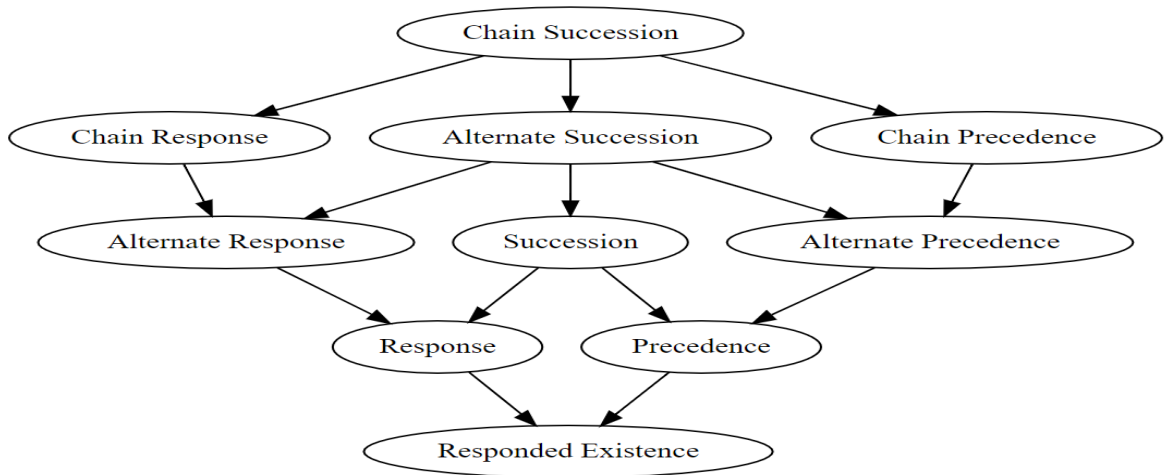


Figure 6 – Hierarchy among Relation templates

2.3.3 Negative Relation Templates

Name	Description	Representation
NOT CO-EXISTENCE(<u>E</u> , <u>F</u>)	E and F never occur together	
NOT SUCCESSION(<u>E</u> , <u>F</u>)	F never follows E	
NOT CHAIN SUCCESSION(<u>E</u> , <u>E</u>)	F cannot occur immediately after E and E cannot occur immediately before F	
NOT CHAIN RESPONSE(<u>E</u> , F)	If E occurs then F does not occur immediately after E	
NOT CHAIN PRECEDENCE(E, <u>F</u>)	F occurs if it is not immediately preceded by E	
NOT RESPONSE(<u>E</u> , F)	If E occurs then F does not occur after E	
NOT PRECEDENCE(E, <u>F</u>)	F occurs if it is not preceded by E	
NOT RESPONDED EXISTENCE(<u>E</u> , F)	If E occurs then F does not occur	

Table 5 – Negative Relation templates

2.4 Multi-Perspective Declare

Multi-Perspective Declare (MP-Declare) is an extension of DECLARE where conditions on data can be specified in addition to conditions on control-flow. These conditions can be of 3 types: activation conditions, correlation conditions and temporal conditions [9]. An activation condition is a condition on the data attached to the activation of a constraint. A correlation condition is a condition either on the data attached to the target of a constraint or on the data attached to the target and on the data attached to the activation together. A temporal condition is used to define a time distance between the target and the activation. For Existence templates, this distance is between the first activity in the trace and the activation.

Trace identifier	Name	Type	Cost	Timestamp
1	Activity1	T1	100	2019-02-02T09:00:00
1	Activity2	T2	200	2019-02-02T09:30:00
1	Activity3	T1	50	2019-02-02T10:30:00

Table 6 – Example log table for MP-Declare

Example 3:

`Response(Activity1, Activity2)[A.Cost > 75][T.Cost > 150 and T.Type != A.Type][[]]`

In Example 3, the first squared brackets is used to specify the activation condition, the second one is used to specify the correlation condition and the third one is used for specifying the temporal condition. "A.Cost" refers to the Cost attribute of Activity1 because the activation (A) is Activity1. "T.Type" refers to the Type attribute of Activity2 because the target (T) is Activity 2.

This example can be interpreted as follows: If Activity1 occurs with a cost greater than 75, then Activity2 with a different Type than Activity 1 and Cost greater than 150 should eventually occur.

In [10], a file format called DECL was introduced for MP-Declare models. The model in Example 3 can be represented in a DECL file as shown in Figure 7.

```
activity Activity1
bind Activity1: Type, Cost
activity Activity2
bind Activity2: Type, Cost
Type: T1, T2
Cost: integer between 10 and 300
Response[Activity1, Activity2] |A.Cost > 75 |B.Cost > 150 and different Type
```

Figure 7 – The DECL file for the model in Example 3

3. Tool architecture

In this thesis we present RuM, a desktop application for Rule Mining providing techniques for process mining based on declarative process models expressed in MP-Declare. The tool was developed using the model-view-controller paradigm [17]. In this pattern, there are three groups of objects: model, view and controller. In the context of our tool, the controller objects handle the functionalities with the model and view objects. The model objects are used to represent the outcomes coming from the controller objects. The view objects are created to visualize the model objects and to obtain the necessary inputs such as the input log and MP-Declare models from users. The architecture of the tool is shown in Figure 8.

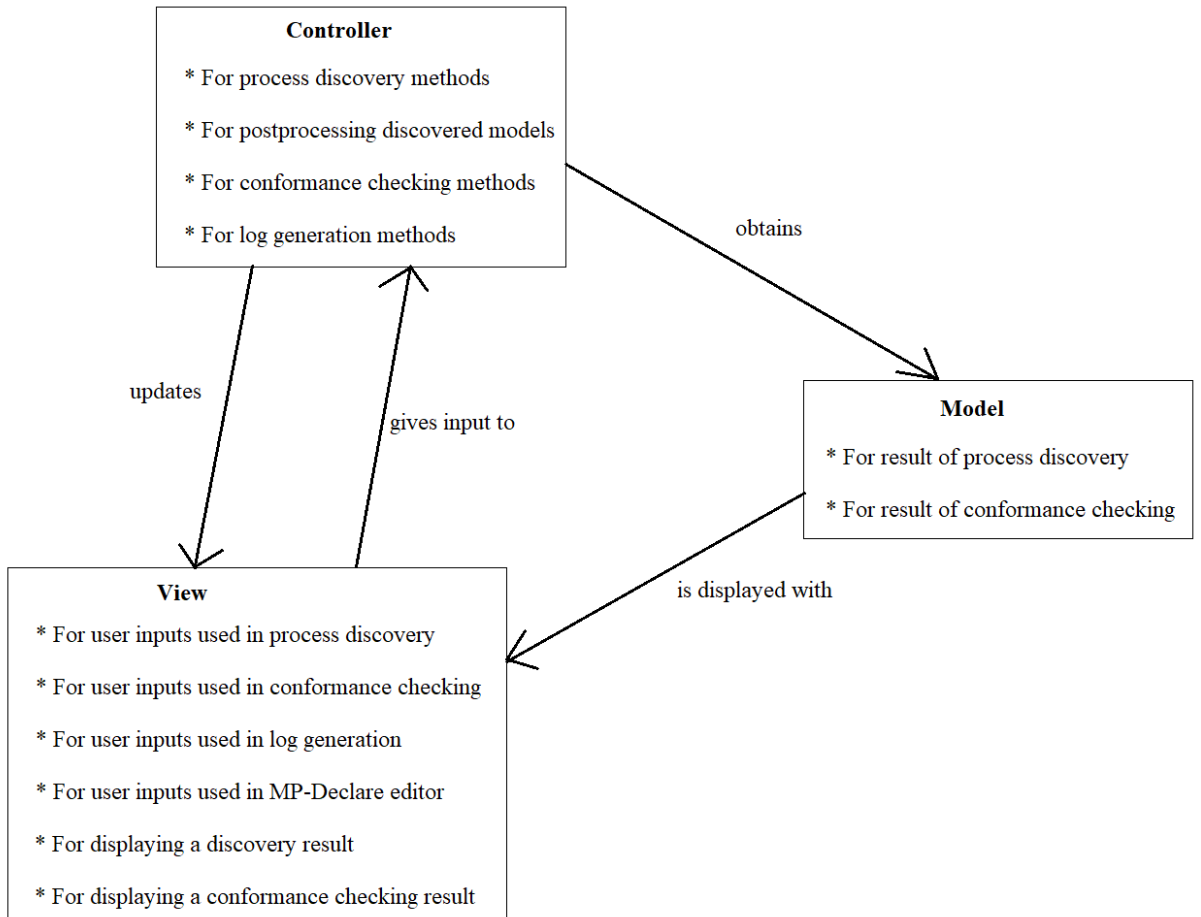


Figure 8 – The architecture of the tool

3.1 Controller architecture

Controllers are used to integrate process discovery, conformance checking and log generation in the tool. For each method involved in these areas, a controller is designed to prepare the inputs coming from UI and to obtain the result object. For process discovery, the result object is a declarative model. For conformance checking, it is an alignment model. For log generation, it is a log file in XES format.

After obtaining a declarative model, another controller is used to post-process the model, including to filter out activities and constraints, and to save the model in the file system or in the tool workspace.

3.2 Model architecture

Models are used to store the results of process discovery and conformance checking.

3.3 View architecture

Views are used to interact with the user for process discovery, conformance checking, log generation and MP-Declare model editing. Views are also responsible for the visualization of the models.

3.4 Technologies

The tool was developed in Java 8¹ because the existing libraries for Process Mining were implemented with Java 1-8. Secondly, we made use of the JavaFX 8² framework. This framework was used to design the view objects in the tool because it has browser support (i.e., the capability of rendering HTML pages) and it enables to build a user interface with JavaFX Scene Builder 2.0³ using FXML⁴ files.

The browser support of JavaFX 8 was used in the views to represent the Declare model. In order to use Graphviz in the tool, VizJS [18], a JavaScript library that can generate a graph in SVG⁵ format from DOT file by using Graphviz, was integrated. The SVG output can be displayed in a HTML page. Therefore, we could represent the Declare and the Automaton views.

¹<https://docs.oracle.com/javase/8/>

²<https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm#JFXST784>

³<https://docs.oracle.com/javase/8/scene-builder-2/get-started-tutorial/overview.htm#JSBGS164>

⁴https://docs.oracle.com/javase/8/javafx/fxml-tutorial/why_use_fxml.htm#BABIECHG

⁵https://www.w3schools.com/graphics/svg_intro.asp

4. Functional overview

In this thesis, a set of process discovery, conformance checking and log generation techniques based on declarative models were integrated into a desktop application. The tool offers a more user-friendly integrated environment for these methods than ProM. This section describes the methods with some definitions and examples, and their implementation in the tool. The source code of the tool is available in a public GitHub repository: <https://github.com/alpdenizz/RuleMiningTool>.

The tool's main panel has four tabs: Discovery, Conformance Checking, Log Generation and MP-Declare Editor. The first tab is used to discover a declarative model from an event log. The second tab is used for conformance checking starting from an MP-Declare model and an event log. The third tab is used to generate an event log from an MP-Declare Model. The last tab is used to create or update an MP-Declare model. The main panel of the tool is shown in Figure 9.

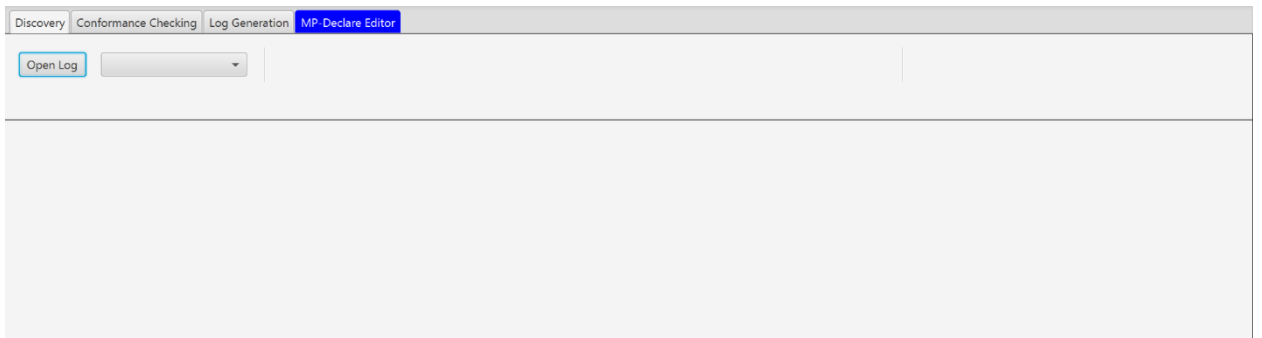


Figure 9 – The tool's main panel

4.1 Process Discovery

4.1.1 The Declare Miner method

The Declare Miner method [6] accepts an event log as a plain or gzipped XES or MXML file, a selection of Declare templates, a minimum constraint support, a vacuity detection option and a pruning option. The result model contains only the selected templates and constraints with support greater than or equal to the minimum constraint support.

Example 4: Let $M: \{ \text{Succession}(A, B), \text{Precedence}(B, C) \}$ be the obtained constraints. Suppose that only the Response template is selected for the result. Then the result would contain only $\text{Response}(A, B)$ because Succession implies Response by hierarchy.

Definition 1 (Constraint support in Declare Miner): It is the percentage of traces where the constraint is fulfilled. If vacuity detection is enabled, constraint vacuously satisfied in a trace are considered as violated.

Example 5: Let $L: \{ \langle A, B, C, A, D \rangle, \langle C, D, E, C, A \rangle, \langle A, D, C, B \rangle, \langle D, C, B, C \rangle \}$ be an example log and $\text{Response}(B, C)$ an example constraint. If vacuity detection is disabled, then the constraint is satisfied in traces 1, 2 and 4; hence, its support is 75%. Otherwise, trace 2 is removed from calculation, resulting in 50% support.

The pruning option is a method to simplify the obtained models. There are four pruning options: All Reductions, Hierarchy-based, Transitive Closure and None. The last option does not post-process the discovered model. The third option removes the constraints that can be derived as transitive closure of a set of other constraints. The second option removes the constraints that are implied by others based on the constraint hierarchy. The first option applies the second and the third. This procedure is illustrated in Example 6.

Example 6: Let $M: \{ \text{Response}(A, B), \text{Response}(B, C), \text{Response}(A, C), \text{Succession}(A, B) \}$ be the discovered model. Since $\text{Succession}(A, B)$ also means $\text{Response}(A, B)$, the latter can be removed by hierarchy. $\text{Response}(A, C)$ is the transitive closure of $\text{Response}(A, B)$ and $\text{Response}(B, C)$; therefore it can be removed because redundant. As a result, the model can be simplified as $\{ \text{Succession}(A, B), \text{Response}(B, C) \}$.

4.1.2 The Minerful method

The Minerful method [11] accepts an event log as a plain or gzipped XES or MXML file, a selection of Declare templates, and a minimum constraint support. After the method derives the discovered constraints, they are automatically post-processed by support and hierarchy. Constraints with support lower than the minimum constraint support are deleted. Constraints are also pruned using a Hierarchy-based approach as in Example 6. The support in this method is considered differently than in the Declare Miner method.

Definition 2 (Constraint support in Minerful): The constraint support is a percentage of constraint's activations that do not lead to a violation. However, this is valid for binary constraints. For the others, the support is considered as in Definition 1.

Example 7: Using the same log from Example 5 and constraint $\text{Response}(A, B)$, the constraint is activated two times and satisfied once in trace 1, activated once in trace 2 and not satisfied, and activated and satisfied once in trace 3. Therefore, its support is 50%.

4.1.3 Declarative model views

A declarative model derived from a process discovery method can be displayed with three options: Textual, Declare and Automaton.

4.1.3.1 Textual representation

The first option is to write each activity with its support and each constraint description with its support.

Definition 3 (Activity support): It is the percentage of traces where the activity occurs at least once.

Example 8: Let $L: \{ \langle A, B, C, B \rangle, \langle B, C, A, C \rangle, \langle C, C, B, A \rangle \}$ a log and $M: \{ \text{Precedence}(B, A), \text{Response}(B, C) \text{ and } \text{Init}(A) \}$ be the model discovered from the log. Then its Textual view would be the following:

Activities:

1. A exists in 100% of the traces.
2. B exists in 100% of the traces.
3. C exists in 100% of the traces.

Constraints:

1. In 66.67% of the traces, If C occurs then B eventually occurs before C.
2. In 33.33% of the traces, If B occurs then C eventually occurs after B.
3. In 33.33% of the traces, A is the first activity.

4.1.3.2 Declare representation

The Declare option displays the model with the graphical representation of Declare templates. For example, the model in Example 8 is represented in the Declare view as in Figure 10.

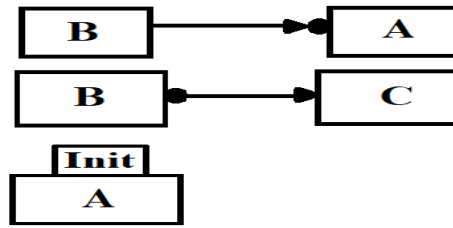


Figure 10 – Declare view for the model in Example 8

In Figure 10, activities A and B are duplicated. It can be claimed that the more constraints the model contains, the more complex the Declare view is since it includes several duplications. In order to have a better representation, we internalized the Declare view as a directed graph.

Definition 4 (Directed graph): It is a set of objects connected with each other. The objects are also called nodes and the connections are also called edges. A connection involves two nodes. An edge is denoted with a name and the order of nodes. For example, in Figure 11, the nodes are A and B, and the edge is Edge1(A, B). "Edge1" is the identifier and "(A, B)" shows the order.

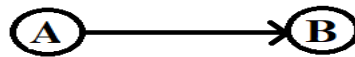


Figure 11 – A sample directed graph

In order to represent a discovered model as a directed graph, the nodes and edges must be extracted from the model. The nodes are the activities involved in the model. The edges are the constraints. In a constraint, the template name is the name of the edge and the order of activities is the order of nodes. An edge is drawn in the style derived from its template name. For example, Response(A, B) is drawn differently from Precedence(A, B). The constraints with one activity are represented by a rectangle over the activity node.

Example 9: For the model in Example 8, the involved activities are A, B and C; hence, the nodes are A, B and C. There are two edges: Precedence(B, A) and Response(B, C). Additionally, activity A has a Init constraint. Therefore the graph would be as in Figure 12.

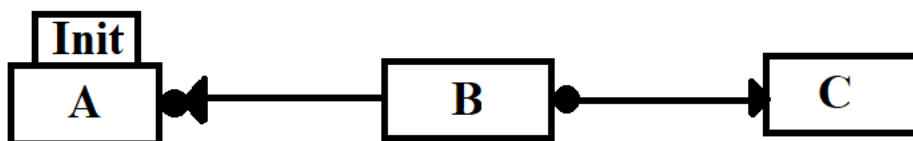


Figure 12 – A better Declare view for the model in Example 5

In this context, an algorithm was required to visualize a directed graphs in the tool. In order to choose it, we determined three criteria, namely, the layout, the input and the configuration. According to the first criterion, the output should include as few as possible crossed edges. Additionally, the edges should be as similar as to Declare template graphical representations. The second criterion means that the input preparation for the algorithm should be as easy as possible. For the third criterion, the algorithm should be configured as easily as possible. In the end, we decided to use Graphviz [12].

Graphviz enables drawing a directed graph using a command-line interface. It takes as input a DOT file, a language defined by Graphviz for graphs. In the file, it is possible to configure the nodes and the edges. Additionally, it is possible to define some properties for the graph such as the layout, the positioning and the size.

In order to represent a discovered model with Graphviz, we defined a procedure to create a DOT file from the model. For this, we decided to display the Existence templates of an activity in one node as horizontally divided rectangles, where the activity name and support were placed in the bottom. For the constraints involving two activities, we designed the edges as similar as possible to the original graphical representations and placed its support as the label of the edge. After that, we extracted the activities and the constraints from the model, and inserted the corresponding representations into the DOT file.

Example 10: The graph in Figure 12 can be represented with Graphviz as shown in Figure 13:



Figure 13 – DOT representation and graph for Example 9

In the DOT representation, "digraph" refers to a directed graph. The first four lines are about the general settings for the graph, the nodes and the edges. The next three lines are the nodes in the model. The last two lines are the edges in the model. The reader can find more about the DOT language in [13].

4.1.3.3 Automaton representation

The Automaton option for an output model displays it as a finite-state machine. An example of a Declare model and its Automaton view is shown in Figure 14.

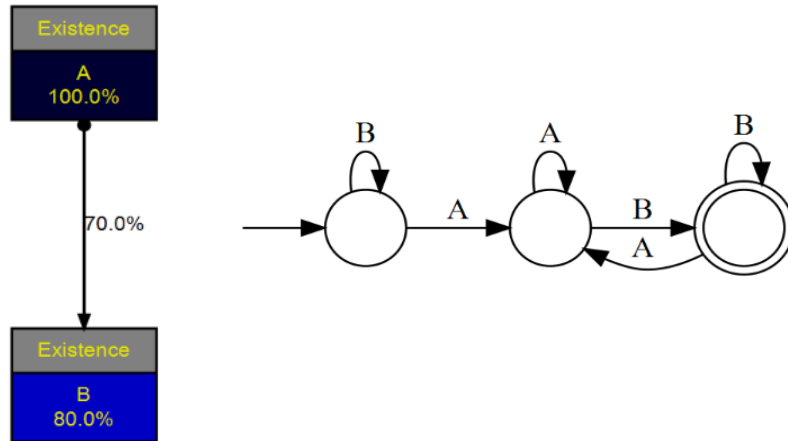


Figure 14 – A Declare model (left) and its Automaton view (left)

Example 11: Let $T1: \langle B, A, B \rangle$ and $T2: \langle A, B, A \rangle$ be two example traces. Using the automaton in Figure 14, we can check if the traces are satisfying the constraints. In the automaton, there are three states: the leftmost is the starting state. The rightmost is a final state and the other states are non-final states. If we execute a trace on the automaton and we reach a final state at the end, then the trace satisfies the constraints; otherwise, it violates the constraints.

For trace $T1$, we are in the starting state at the beginning. After executing activity B , we stay in the same state. After executing A , we are in the middle state. After executing B , we are in the final state; therefore, trace $T1$ is a valid trace. For trace $T2$, after executing A , we are in the middle state. After executing B , we are in the final state. After executing A again, we are in the middle state that is non-final; hence, trace $T2$ is not a valid trace for the Declare model.

4.1.4 Discovery tab

The discovery tab of RuM is shown in Figure 15. Here, a log file in plain or gzip compressed XES and MXML formats can be opened using the "Open Log" button. Then, the log file is automatically processed using the Declare Miner method. In the end, a declarative model is obtained and displayed using the Declare view in the main panel. Figure 15 shows an example of a process discovery result.

It is possible to zoom the view using the slider "Zoom". In the right panel, there are two sliders for filtering the model based on constraint support and activity support. They can be set to a higher value to remove constraints and activities with lower support from the output model. The output model view can be changed using the choices below the label "View model as". The choices include Declare, Textual and Automaton. The buttons "Export" and "Save" are used to export the model to the file system and save the model in the workspace of the tool. The "Export" button is used to export the output model to the file system as a DECL file if the model view is Declare, as a TXT file if the model view is Textual and as a DOT file if the model view is Automaton. The "Save" button is used to save the model in the workspace of the tool so that it can be used as input for other functionalities.

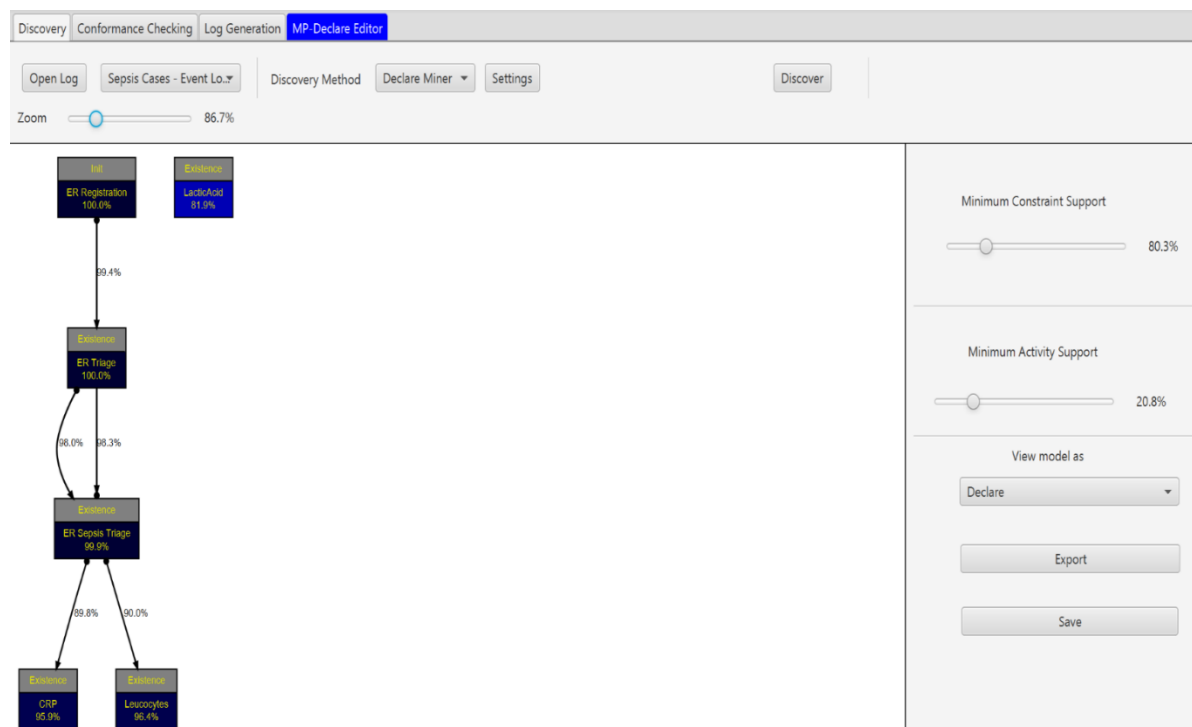


Figure 15 – An example process discovery result

A process discovery can be configured using the "Settings" button. The settings include a Declare templates list, a slider to set the minimum constraint support and some additional settings, if the method is the Declare Miner, for enabling vacuity detection and different types of pruning.

The settings panel for the Declare Miner method is shown in Figure 16. Here, the button "Close" is used to leave settings panel and return to the main panel. The button "Restore Default" is used to return to the default settings. The list "Declare Templates" can be multi-selected to choose the Declare templates to be used for discovery. The minimum support slider is "Minimum Trace Support" for the Declare Miner method and "Minimum Event Support" for the Minerful method to indicate the different types of support used in these methods. The choice "Vacuity Detection" enables vacuity detection. The selection "Pruning" is used to filter out redundant constraints. The process discovery can be started with the selected log file and configuration by clicking the "Discover" button. After the model is obtained, if the "Settings" button is clicked, the settings panel shows the most recent configuration.

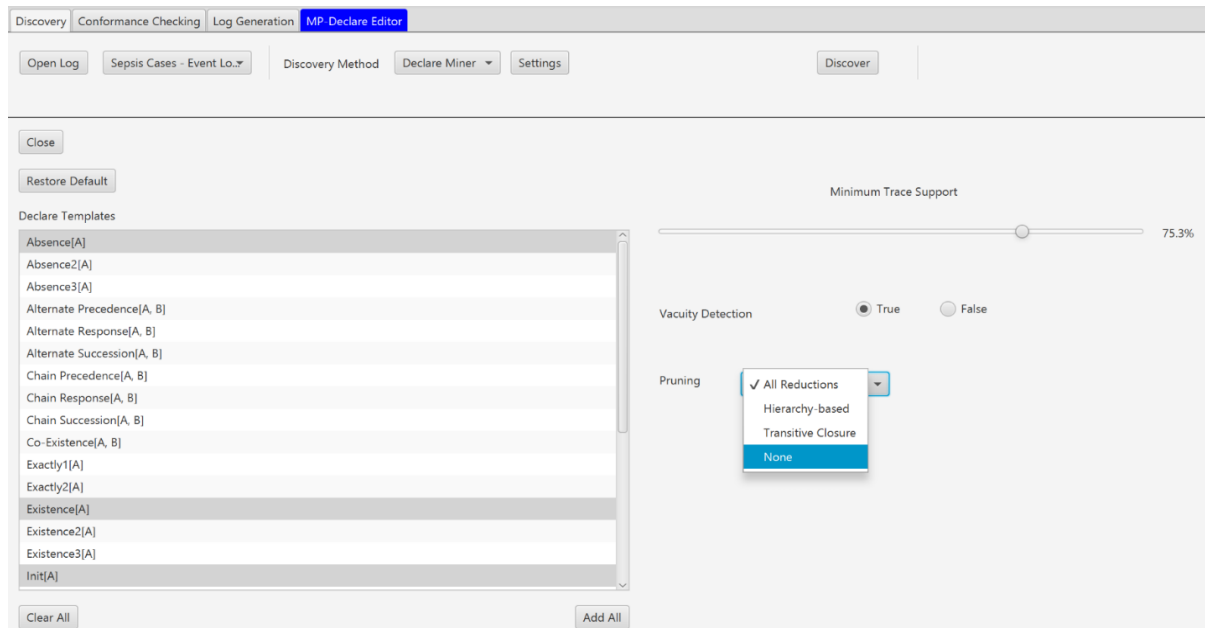


Figure 16 – Settings screen for the Declare Miner method

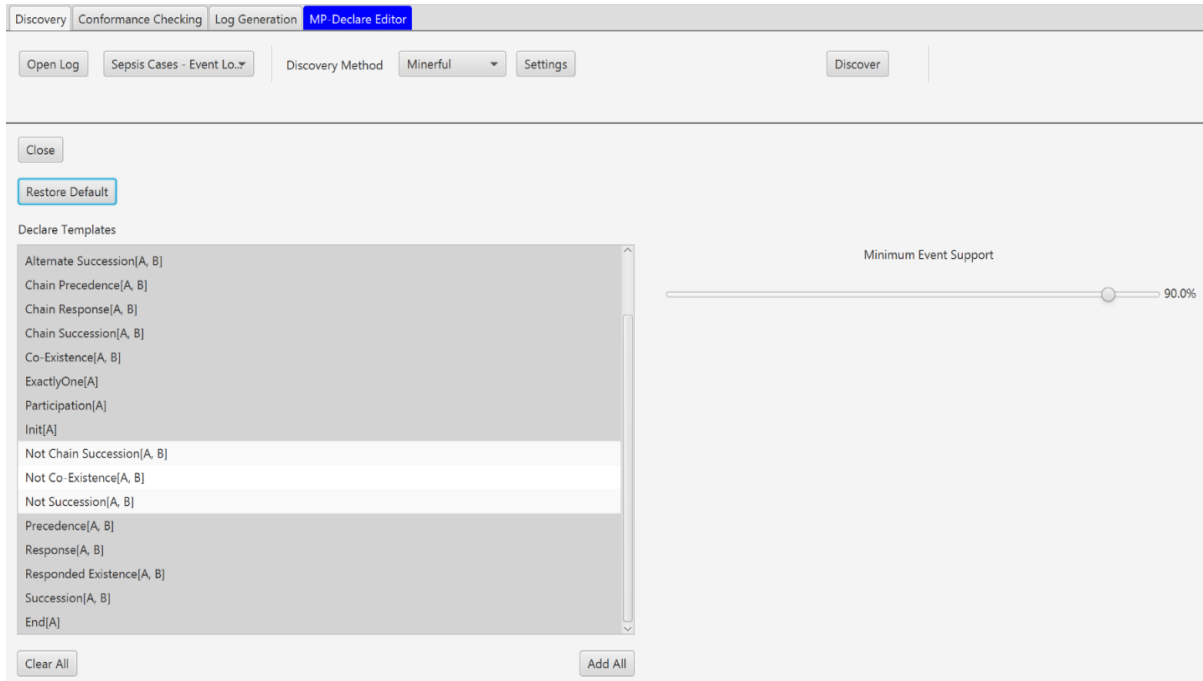


Figure 17 – Settings screen for the Minerful method

4.2 Conformance Checking

Conformance checking requires two inputs: an MP-Declare model as a DECL file and an event log as a plain or gzipped XES or MXML file. There are three methods for conformance checking: Declare Analyzer [9], Data-Aware Declare Replayer [14] and Declare Replayer [15].

In conformance checking, an activity in the input MP-Declare model should match an activity in the input log by its name. There can be some unmatched activities in the model or in the log. Usually, activities in the log also have the field "lifecycle:transition", a set of values for an activity denoting its status such as "start" and "complete". We consider the name of an activity in the input log as the combination of activity name, a separator "-" and the lifecycle transition. Activities in the model that do not have the lifecycle transition specified are considered as having "complete" as transition value. In order to have a match between an event in the log and an activity in the model, its name and transition values must be equal.

```

<event>
  <int key="x" value="1"/>
  <string key="y" value="Lino"/>
  <string key="concept:name" value="A"/>
  <string key="lifecycle:transition" value="start"/>
  <date key="time:timestamp" value="2019-06-21T22:11:19.733+03:00"/>
</event>
<event>
  <int key="x" value="25"/>
  <string key="y" value="Gino"/>
  <string key="concept:name" value="B"/>
  <string key="lifecycle:transition" value="complete"/>
  <date key="time:timestamp" value="2019-06-21T22:12:14.733+03:00"/>
</event>
<event>
  <int key="x" value="30"/>
  <string key="y" value="Gino"/>
  <string key="concept:name" value="C"/>
  <string key="lifecycle:transition" value="complete"/>
  <date key="time:timestamp" value="2019-06-21T22:28:20.733+03:00"/>
</event>

```

Figure 18 – An example trace with 3 events

Example 12: let $M: \{ \text{Response}(\text{A-start}, \text{B-complete}), \text{Precedence}(\text{B}, \text{C}) \}$ be a Declare model. From Figure 18, the events in the log are $\{ \text{A-start}, \text{B-complete}, \text{C-complete} \}$. The activities in the model are $\{ \text{A-start}, \text{B-complete}, \text{B}, \text{C} \}$. As a result, we have the mapping $\{ \text{A-start} \Rightarrow \text{A-start}, \text{B-complete} \Rightarrow \text{B-complete}, \text{B} \Rightarrow \text{B-complete}, \text{C} \Rightarrow \text{C-complete} \}$. Therefore, the first constraint is satisfied because after event A-start, there is B-complete in the trace. The second constraint is satisfied because before C-complete, there is B-complete in the trace.

4.2.1 Changes in the DECL format

We changed the DECL file format from [10] to be able to insert temporal conditions. In particular, we inserted an additional pipe "|" to the end of a constraint. After that pipe, a temporal condition can be written.

4.2.2 The Declare Analyzer method

In the Declare Analyzer method, the output model includes the total number of activations, fulfillments and violations of each constraint in the input MP-Declare model in each trace of the log. Moreover, it allows the user to select a specific trace and see fulfillments and violations of each constraint in the input MP-Declare model in that specific trace.

4.2.3 The Declare Replayer method

In the Declare Replayer method, the output model includes the alignments of the constraints in the input MP-Declare model with each trace in the input log. In an alignment, each activity is assigned as either "Move in Log", "Move in Model" or "Move in Log and Model" [15]. "Move in Log" states that the activity should be deleted from the trace to make it compliant with the MP-Declare model. "Move in Model" indicates that the activity should be inserted in the trace. "Move in Log and Model" denotes that the activity does not trigger a violation in the input model.

Example 13: Let $\langle A, B, C, B, A \rangle$ be an example trace and $\text{Response}(A, C)$ be a constraint. The constraint is violated in the trace. In order to eliminate the violation, two actions can be made: delete activity "A" in the end or insert activity "C". The first four activities are moves in log and model. With the first action, the last activity would be a move in log. With the second action, the fifth activity would be a move in log and model, and the last activity would be a move in model.

In this method, there can be many options to eliminate violations. In order to choose the optimal one, a control flow cost model can be defined. According to this model, each activity in the log or in the model has a cost for a move in log or move in model. The higher the cost for an action, the less likely it will be done. Considering Example 13, if the cost of move in log for activity "A" were less than the cost of move in model for activity "C", then only the former would be done due to having a lower cost.

This method ignores the data conditions in the input MP-Declare model. In the next section, we introduce a Declare Replayer "Data-Aware" where alignments are performed taking into consideration the data perspective also.

4.2.4 The Data-Aware Declare Replayer method

The Data-Aware Declare Replayer method does not ignore the activation conditions in the input model. It introduces an adjustment to reduce the violations in the input log: "Move in Log and Model with different data". This means to change the corresponding attribute value of an activity by either inserting a new value for the attribute or deleting the attribute from the activity.

Example 14: Let N be an attribute for the activities in the trace in Example 13. Suppose that trace is: <A(N: 10), B(N: 20), C(N: 30), B(N: 40), A(N: 50)>. Let us define an activation condition for the constraint in Example 13: Response(A, C) |A.N > 45. In order to eliminate the violation, three actions can be made: setting the attribute value of the last "A" to 45, removing the last "A" from the trace or inserting a "C" to end of the trace. The first action is a "Move in Log and Model with different data" in the alignment, the second is "Move in Log", and the third is "Move in Model". The remaining activities are "Move in Log and Model with same data" in the alignment.

In this method, in addition to the control flow cost model, a data cost model can be defined. According to this model, each activity that exists both in the log and the model has a cost of changing an attribute value or of removing the attribute. The higher the cost for an action, the less likely it will be done. Considering Example 14, if the cost of changing attribute N for activity "A" was less than the cost of inserting "C" to the end and of the cost of removing the last "A", then the former would be done due to lower cost.

For the constraints without data condition in the input model, the Data-Aware Declare Replayer works same as the Declare Replayer. It has also "Move in Log", "Move in Model" and "Move in Log and Model with same data" in the alignment. The last action covers the activities that do not cause a violation in the constraints. The control flow cost model can also be used in the Data-Aware Declare Replayer. Its output model includes the alignment of the constraints in the input MP-Declare model with each trace in the input log.

4.2.5 Conformance Checking tab

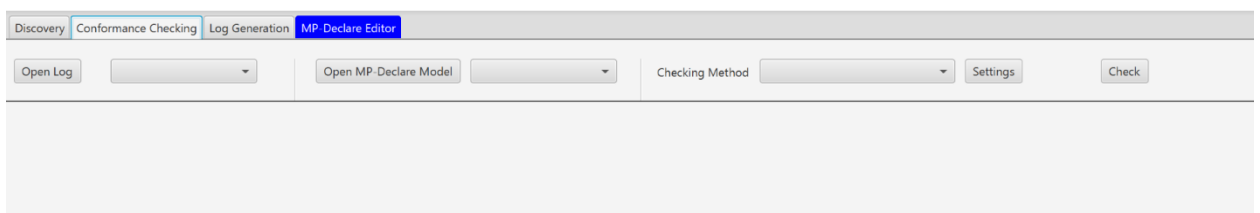
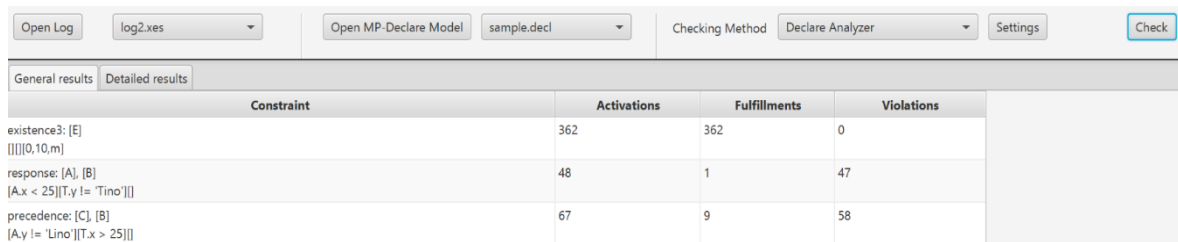


Figure 19 – Initial panel for Conformance Checking tab

In Figure 19, the button "Open Log" can be used to open a log. The button "Open MP-Declare Model" can be used to open an MP-Declare model in DECL format. After a log and a model are opened, two checking methods are available, the Declare Analyzer and

the Data-Aware Declare Analyzer. If the opened model does not contain any data condition, then the Declare Replayer is also available. After selecting a checking method, conformance checking can be started by clicking the "Check" button.



Constraint	Activations	Fulfillments	Violations
existence3: [E] [[[]][0,10,m]	362	362	0
response: [A], [B] [A.x < 25][T.y != 'Tino']	48	1	47
precedence: [C], [B] [A.y != 'Lino'] [T.x > 25]	67	9	58

Figure 20 – Two tabbed result panel for the Declare Analyzer method

In Figure 20, the two tabbed result panel for the Declare Analyzer method is shown. In the "General results" tab, the total number of activations, fulfillments and violations are listed for each constraint in the input MP-Declare model.

In Figure 21, the "Detailed results" tab is shown. In this panel, there are three sections: trace list, constraint list and trace view. The first section is used to select a trace from the input log. A trace is represented using the trace identifier and the total number of activations, fulfillments and violations of the constraints in the MP-Declare model in the trace. The second section is used to select a constraint from the input MP-Declare model. After a trace and a constraint is selected, fulfillments and violations of the constraint in the trace can be analyzed in the trace view. There is a legend above the trace view pointing that the events of the trace are displayed with a red background if the event is a violation for the constraint and with a green background if the event is a fulfillment. The attributes of an event in a trace are shown by passing with the mouse over the event. It is also possible to sort the traces by number of activations, number of fulfillments, number of violations and in alphabetical order.

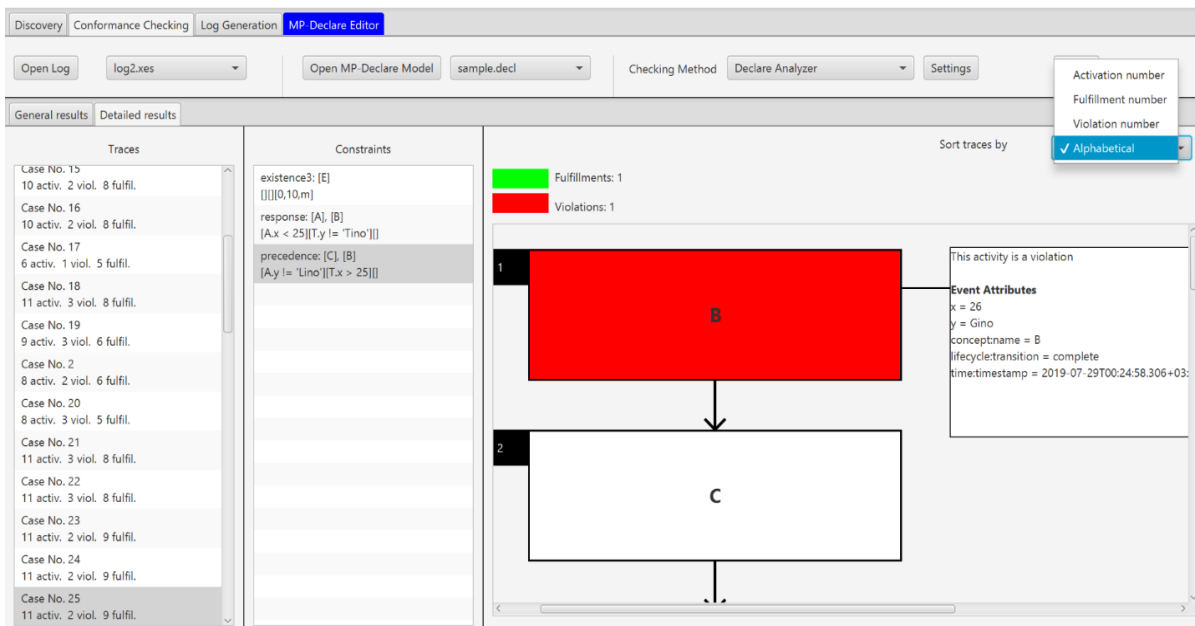


Figure 21 – "Detailed results" tab

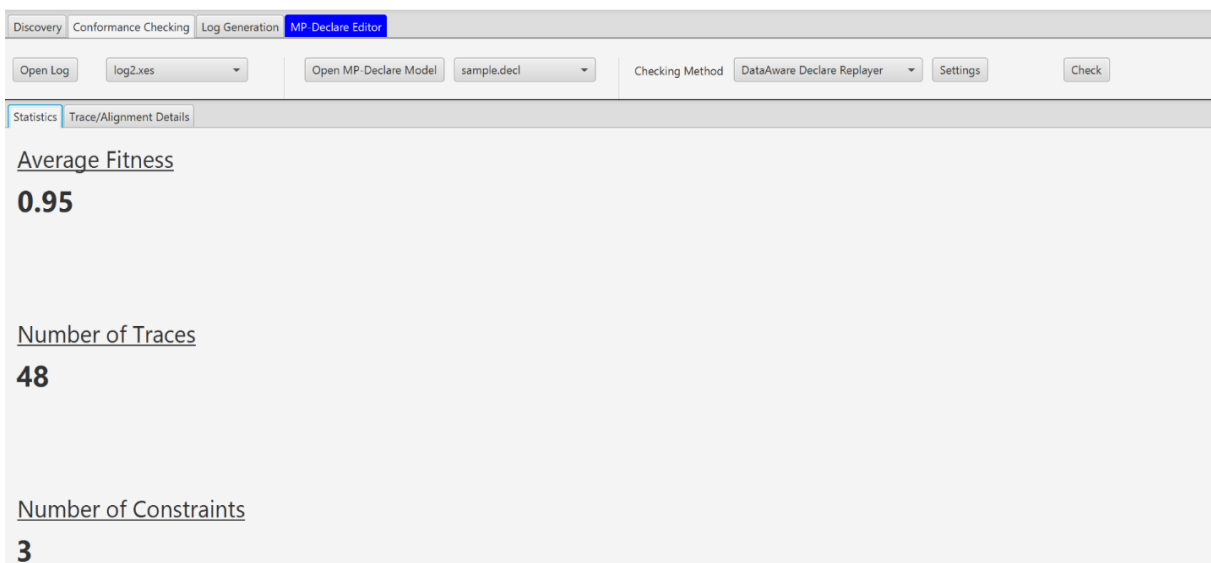


Figure 22 - Two tabbed result panel for the Data-Aware Declare Replayer method

In Figure 22, the two tabbed result panel for the Data-Aware Declare Replayer method is shown. In the "Statistics" tab, the average fitness, the number of traces in the input log and number of constraints in the input MP-Declare model are listed. The fitness of a trace is a value between 0 and 1 calculated as in [15]. The higher the value, the less violations are present in the trace. The average of fitness over all traces is reported under "Average Fitness".

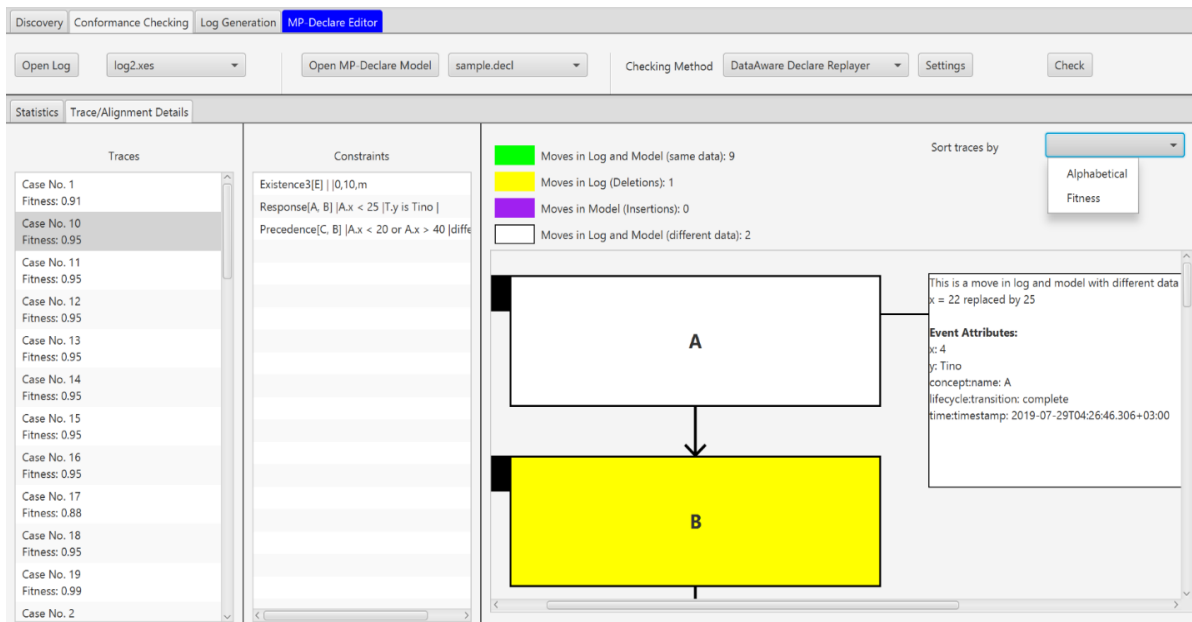


Figure 23 – "Trace/Alignment" tab

In Figure 23, the "Trace/Alignment" tab for the Data-Aware Declare Replayer method is shown. In this panel, there are three sections: traces, constraints and trace view. In the first section, the traces from the input log are listed with their identifiers and average fitness. In the second section, the constraints from the input MP-Declare model are displayed. When a trace is selected, its alignment with the input MP-Declare model is shown in the trace view. There is a legend above the trace view pointing out that events of the trace are displayed with a green background if the event is a move in log and model (same data), with a yellow background if the event is a move in log, with a purple background if the event is a move in model, and with a white background if the event is a move in log and model (different data). The attributes of an event in a trace are shown by passing with the mouse over the event. It is also possible to sort the traces by fitness and in alphabetical order.

For the Data-Aware Declare Replayer method, a control flow cost model and a data cost model can be configured by clicking the "Settings" button. Then, a two tabbed panel opens. The first tab can be used for setting the control flow cost model and the second tab can be used for setting the data cost model.

In Figure 24, the "Control Flow Cost Model" tab is shown. In order to define a deletion cost or an insertion cost for an activity, first, the activity should be selected from the menu under the label "Activity". Second, either "Move In Model (Insertion)" or "Move In Log (Deletion)" should be selected. After writing a float number under the label "Cost", the

cost can be inserted into the table by clicking "Add Matching". By default, there are two entries in the table with activity value "*" indicating that that cost applies to all activities. A selected entry can be edited or deleted by clicking the "Remove Matching" button; the configuration panel can be closed with the "Close" button. In this case, main panel is displayed. The configuration can set to the default one using the "Restore Default" button.

Control Flow Cost Model

Close

Restore Default

Activity: * Action: Move In Model (Insertion) Cost: 10.0

Activity: * Action: Move In Log (Deletion) Cost: 10.0

Add Matching

Remove Matching

Figure 24 – "Control Flow Cost Model" tab

Data Cost Model

Close

Restore Default

Activity: * Attribute: * Non-writing cost: 1.0 Faulty-value cost: 1.0

Add Matching

Remove Matching

Figure 25 – "Data Cost Model" tab

In Figure 25, the "Data Cost Model" tab is shown. In order to define both non-writing cost and faulty-value cost for an attribute of an activity, first, the activity should be selected from the menu under the label "Activity". Second, one of its attributes or "*" should be selected. After writing a float number for non-writing and faulty-value cost, the cost can be added to the table by clicking the "Add Matching" button. By default, there is one entry in the table with activity and attribute values "*" meaning that that cost is valid for all activities and attributes. A selected entry can be edited or deleted by clicking the "Remove Matching" button; the configuration panel can be closed with the "Close" button. In this case, main panel is displayed. The configuration can set to the default one using the "Restore Default" button.

After configuring the Data-Aware Declare Replayer method, it can be started by clicking the "Check" button. When the result is produced, it is possible to go to the most recent configuration by clicking "Settings" again; however, if either a new log file or a new model file is selected, then clicking "Settings" would open a default setting because the configuration is dependent on both the input log and the model.

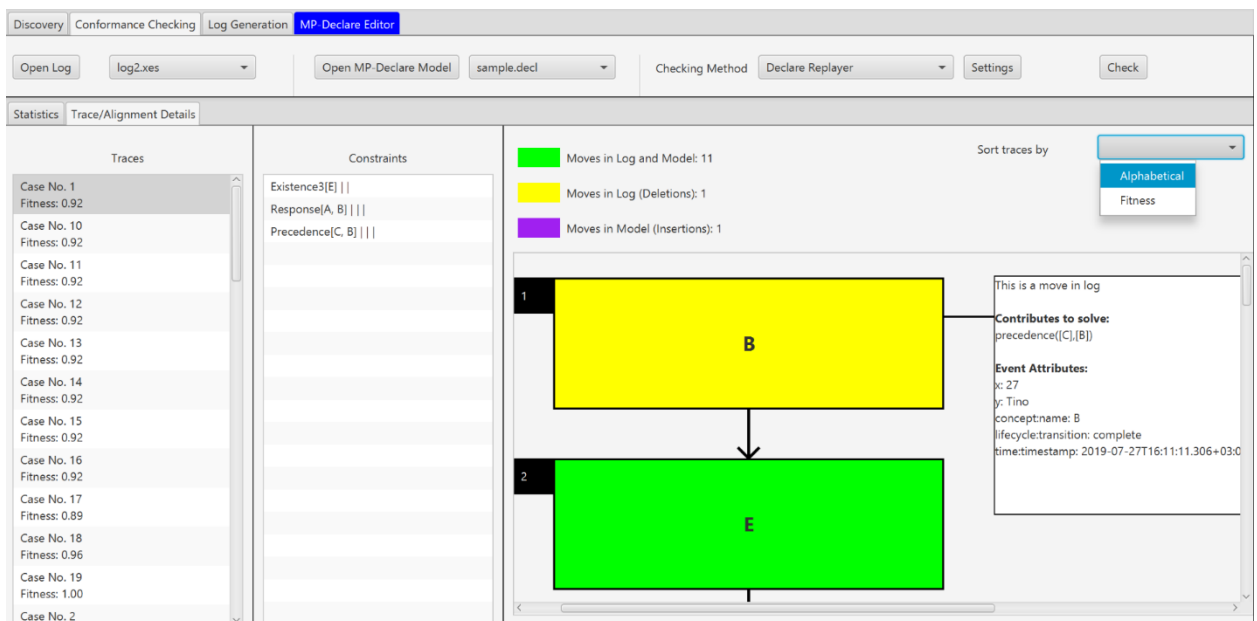


Figure 26 - "Trace/Alignment" tab for the Declare Replayer method

For the Declare Replayer method, a result is displayed with the two tabbed panel shown in Figure 26. The "Statistics" tab is the same as the one of the Data-Aware Declare Replayer method. In the "Trace/Alignment Details" tab, the difference is in the legend in

that it does not have "Move in Log and Model (different data)" and "Move in Log and Model (same data)" is written simply as " Move in Log and Model ".

4.3 Log Generation

Log generation requires as inputs an MP-Declare model as a DECL file. A log generation method should be configured by specifying a destination for the output log in the file system, the minimum length for traces, the maximum length and the number of traces in the output log. In the end, a log file in XES format is generated. RuM provides two methods for log generation: the Alloy Log Generator [10] and the Minerful Log Generator [16].

4.3.1 The Alloy Log Generator

This method generates a log using a SAT solver and an encoding of the input MP-Declare model in Alloy.

4.3.2 The Minerful Log Generator

This method cannot generate a log by using an MP-Declare model with data conditions. The method generates a log using automata derived from the input model.

4.3.3 Log Generation tab

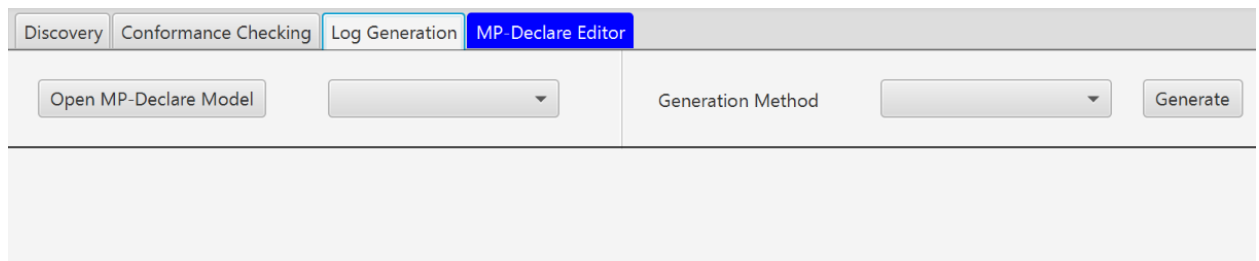


Figure 27 – Log Generation tab

Figure 27 presents the Log Generation tab of RuM. The button "Open MP-Declare Model" opens an MP-Declare model as a DECL file. After a model is opened, "Alloy Log Generator" is selected in the choices next to the label "Generation Method". Then, a configuration panel is displayed for the generation. Moreover, it is possible to see the constraints in the input model in the right panel. Figure 28 shows the configuration panel for the Alloy Log Generator.

Figure 28 – Configuration panel for the Alloy Log Generator

In Figure 28, the information needed for the generation of the log can be inserted in the left panel. First, a destination in the file system for the output log must be selected. For this purpose, the button "Select" can be used to pick a location and a name for the log. Second, the integer values for "Min Trace Length", "Max Trace Length" and "Number of Traces" must be defined into the text areas next to their labels.

Third, "Vacuity Detection", "Generate Negative Traces" and "Even Length Distribution" can be enabled. If the first option is set to true, then all constraints in the input model will be activated at least once in each trace of the output log. If the second option is set to true, then all trace in the output log will violate at least one constraint of the input MP-Declare model. If the third option is set to true, the lengths of the traces of the output log will be evenly distributed between the minimum trace length and the maximum trace length. After the configuration is set, a log can be generated by clicking the "Generate" button.

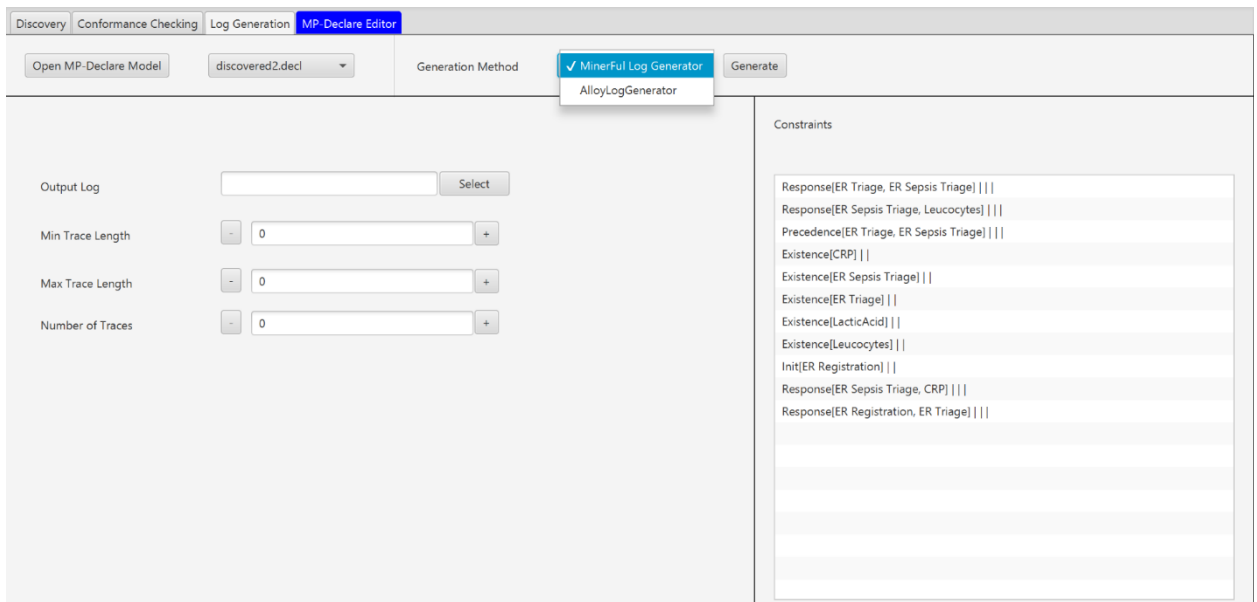


Figure 29 - A configuration screen for Minerful Log Generator

Figure 29 displays the configuration panel for the Minerful Log Generator. This method is available only if the input model does not contain any data condition in the constraints. The last three options available in Figure 28 are not available here. The remaining options are the same as in Figure 28. After the configuration is set, a log can be obtained by clicking the "Generate" button.

4.4 MP-Declare Editor tab

In this tab, an MP-Declare model as can be created or edited by adding new activities, by defining new data attributes, either enumerative or numeric, by inserting new constraints, or by updating existing constraints.

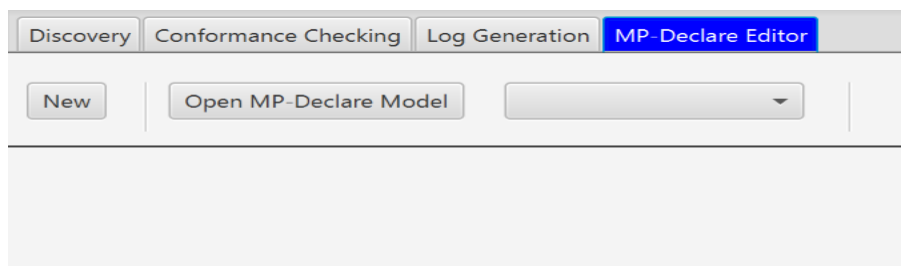


Figure 30 – The MP-Declare Editor tab

Figure 30 shows the MP-Declare Editor tab. The button "New" is used to create a new MP-Declare Model. The button "Open MP-Declare Model" opens an existing MP-Declare Model either in the file system or in the tool workspace.

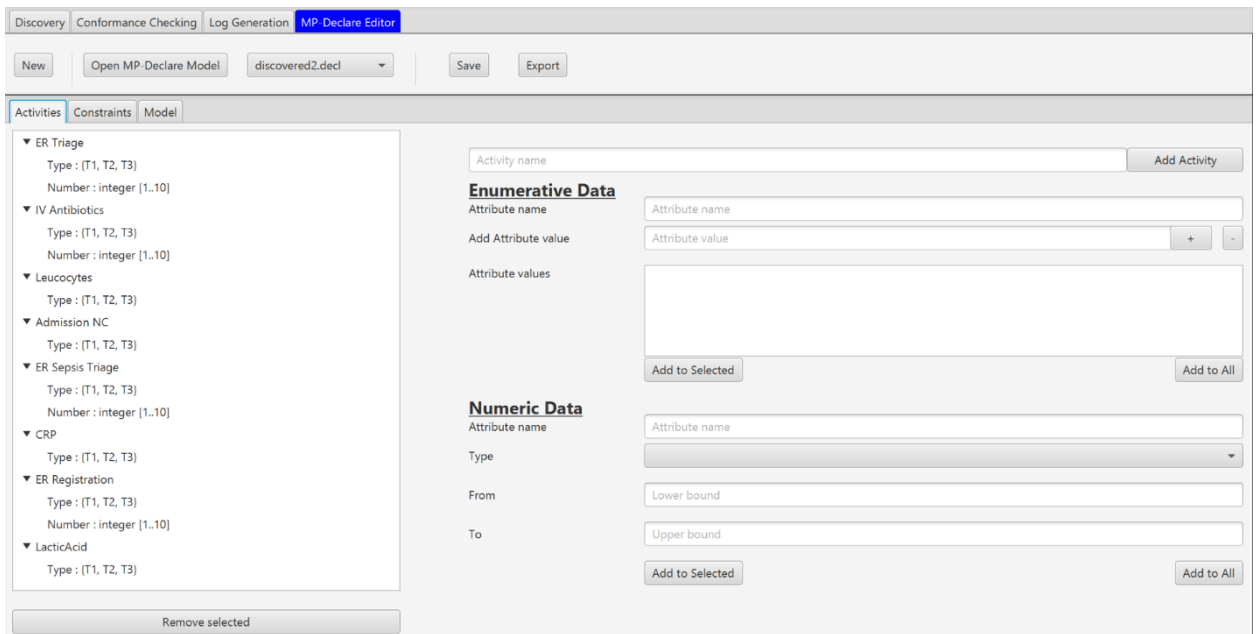


Figure 31 – Activities tab

Figure 31 shows the "Activites" tab of the editor. After opening an MP-Declare model, its activities and attributes are shown in the left panel. It is possible to delete an activity or an attribute by selecting it and clicking on the "Remove selected" button.

An activity can be added by writing a name in the area before the "Add Activity" button and clicking the button. For an activity, an enumerative attribute can be defined. The user needs to specify an attribute name and a set of attribute values. For this purpose, an attribute name must be written in the area next to the "Attribute name" label. At least one value for the attribute must be inserted. To do that, the value has to be written in the area next to the "Add Attribute value" label and the button "+" should be clicked. Then, the attribute value is placed in the list next to the "Attribute values" label. A defined value in the list can be deleted by selecting it and clicking the "-" button. In order to attach the attribute to an activity, the activity must be selected and the "Add to Selected" button must be clicked. In this case, more than one activities can be selected and the attribute is added to them. It is possible to add the attribute to all activities using the "Add to All" button. If an

activity has an attribute with the same name, then the existing attribute is overwritten and its occurrences in the other activities are replaced with the new definition.

In order to add a numeric attribute to an activity, an attribute name must be written in the area next to the "Attribute name" label. Then, a type, either integer or float, must be specified. Finally, its bounds must be defined. The lower bound must be written in the area next to the label "From" and the upper bound must be written to the area next to the label "To". In order to attach the attribute to an activity, the activity must be selected and the "Add to Selected" button must be clicked. In this case, more than one activities can be selected and the attribute is added to them. It is possible to add the attribute to all activities using the "Add to All" button. If an activity has an attribute with the same name, then the existing attribute is overwritten and its occurrences in the other activities are replaced with the new definition.

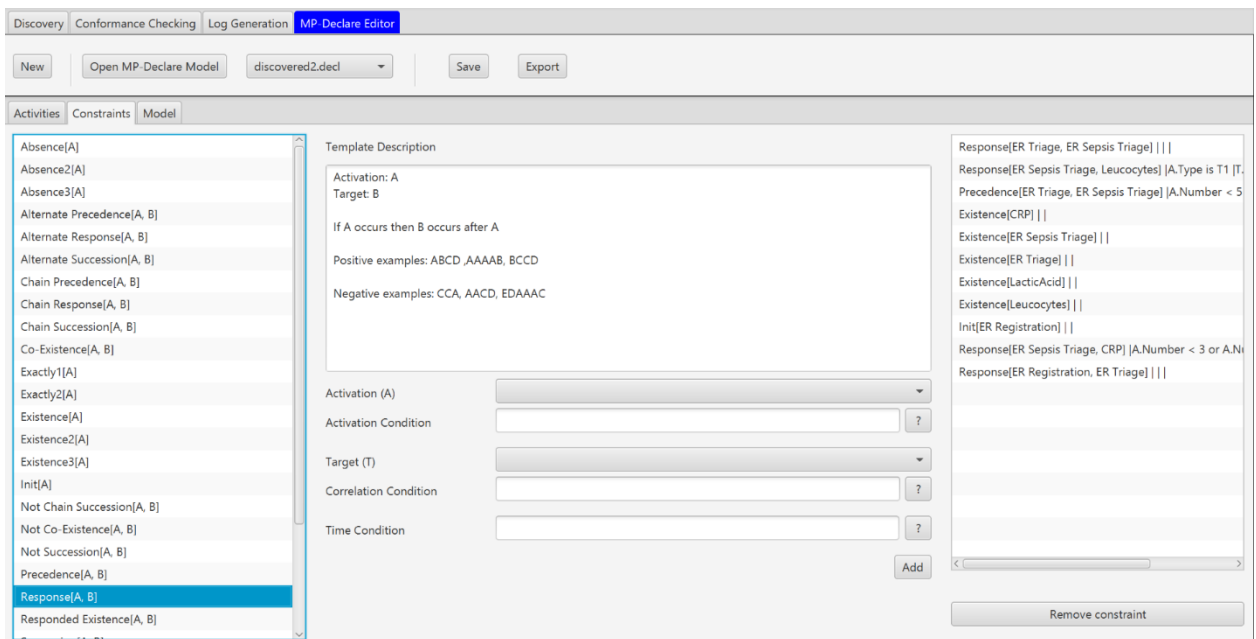


Figure 32 – Constraints tab screen

Figure 32 shows the "Constraints" tab of the editor. It is used to create or update the constraints in the MP-Declare model. Here, there are three sections. The first section contains the list of Declare templates that can be used to create a constraint. If a template is selected, then its description is displayed in the middle. The second section is used to select the activities and to write the data conditions for the constraints. After selecting a template

from left, the defined activities from "Activities" tab can be used to set activation and target of the constraint. In addition, an activation condition, a correlation condition and a time condition can be specified. The user can consult a tutorial by clicking "?" to learn how to write data conditions. The constraint can be added by clicking the "Add" button. The third section contains the list of the constraints contained in the model. After an entry is selected from there, its template description is displayed, its target and activation and its data conditions are shown. Then, the "Add" button becomes "Update" and it can be used to update the selected constraint. It is possible to delete a selected constraint by clicking the "Remove selected" button.



Figure 33 – Model tab

Figure 33 shows how the constraints in Figure 32 are represented in Declare. The view is different from the one in the Discovery tab because there is no support here. In addition, the model include the data conditions here. These conditions are represented in squared brackets including from left to right the activation, the correlation and the time condition attached to a constraint.

After an MP-Declare model is designed, the model can be exported to the file system by clicking the "Export" button. The model can also be saved in the tool workspace by clicking the "Save" button.

5. Evaluation

5.1 User tests

In order to evaluate the usability of our tool, we conducted a usability test. First, a task list [19] covering most of the available functionalities was designed. Second, a group of 5 experts on declarative process analysis was selected for the experiment. Third, we asked the experts to execute the tasks in the task list to estimate time and difficulty of the tasks.

At the beginning of each test, we briefly introduced the tool and the task list to the users. Then, they started the experiment and tried to complete the task list. At the end of the test, we interviewed the users about the tool and the tasks. The questions were on the overall experience, difficulty of the tasks and UI recommendations on the tool.

Generally, the users mentioned that the tool's user interface was instructional to complete the tasks. According to their experience, the "Discovery" tab was the most intuitive one; whereas the tasks related to "MP-Declare Editor" were the most complex. In addition, they think that the integration of declarative process analysis techniques in a single tool is an important advancement in this field. Also, the representation of the output models was clear and the tool responsive. On the other hand, some of the tasks were more complex to complete than others for all users. In the "MP-Declare Editor" tab, the hardest task for them was to update the constraints with data conditions. In the "Log Generation" tab, when the generation failed, the reason was not clear to the users. In the "Discovery" tab, they could not set the required values with the sliders easily.

The task list was divided into three groups: the first group was dedicated to process discovery, the second one was focused on model editing and log generation, and the third one was focused on conformance checking. According to the experiment results, the first group of tasks was the easiest to be followed for the users; the second group was the most difficult. This is also reflected in the fact that the second group of tasks was the most time-consuming whereas the first one was the least time-consuming. All in all for the users, the time to complete all the tasks was from 24 minutes to 32. Table 7 presents the time needed for executing each group of the tasks.

Task Group	Execution time
Process discovery	7-10 minutes
Model editing and Log generation	9-12 minutes
Conformance checking	8-10 minutes

Table 7 – Time spent on each group of tasks

The participants recommended to give the possibility to the user to insert manually the values specified with the sliders. In addition, they proposed that the "Activities" tab in the "MP-Declare Editor" could be redesigned to separate activity specification and data binding. Another mentioned point was to provide more information about the implemented methods in the user interface for non-expert users. Also, the participants suggested to define a button in the "MP-Declare Editor" to remove all data conditions from the constraints in an MP-Declare model. Finally, they suggested to disable the buttons when they are not connected to any action, like the configuration button for the Declare Analyzer.

In addition, the participants recommended to make the editor more strict so that the users would know the problems with the model and use it more safely in the other tabs. They suggested to provide a feature to be able to zoom the trace view in the "Conformance Checking" tab. Furthermore, a bug in the "Log Generation" tab was discovered since the generation could be started without providing any destination file. Another proposal was to enable multi-remove of activities, attributes and constraints in the editor. Also, it was suggested to support going from one input area to another in the panels with Tab.

Most participants suggested to better explain how to use operations, how to read the results and why errors occur in the tool and change the way the templates in the discovery configuration are selected (with a check-box instead of using multi-selection). Another suggested change was related to the detailed result view in the "Conformance checking" tab of the Declare Analyzer method. When a trace is selected without selecting a constraint, the overall results for that trace should be presented in the main screen and the user should can select a constraint. When a constraint is selected without selecting a trace, the overall results for that constraint should be displayed in the main screen and the user can select a trace.

When the user selects both a trace and a constraint, the results for that combination should be shown.

All in all, the usability test was useful to understand the perspective of process mining experts and the possible points of improvement of the tool. The feature of the tool most appreciated is the integration of process discovery, log generation and conformance checking methods with declarative models in only one application using a common format for Declare models (DECL). This allows using the output of one method as input of another method.

In addition, the user can display Declare models as Automaton and Textual views. Moreover, he/she can store the model as DECL, TXT and DOT file in the file system. The discovered Declare models can be filtered using different metrics. The tool can be used to design declarative models and update MP-Declare models (from DECL files) easily.

The users found the tool sufficiently intuitive, user-friendly and a valuable instrument to conduct process analysis through declarative process mining techniques. The tool can run on any operating system.

5.2 Performance

This section presents a performance overview of process discovery, conformance checking and log generation methods in the tool. For each method, the computation time is reported for different sizes of logs and models.

The tests were executed on HP Elitebook 840 G5 with the following configuration:

- **CPU:** 2.71 GHz Intel Core i5 4 core
- **RAM:** 16GB DDR4 2400MHz
- **OS:** Windows 10 Enterprise Edition (Version 10.0.18362.239)

While testing a case for a method, the tool was closed and reopened for the next case. Furthermore, before starting the tool, the computer's state was with CPU at most 10% and RAM at most 25%.

5.2.1 Process Discovery methods

In this part, the Declare Miner and the Minerful methods were analyzed using three logs: Sepsis Cases [20], BPIC 2012 [21] and BPIC 2017 [22].

For the Declare Miner method, only non-negative Declare templates were selected. The minimum constraint support was 100%. Pruning was set to "All Reductions" and vacuity detection was true. For the Minerful method, only non-negative Declare templates were selected and the minimum constraint support was 100%. Table 8 shows the results for the process discovery methods. It is evident that Minerful method outperforms the Declare Miner on large datasets; however, the difference is subtle on small datasets.

Used Log	Declare Miner method	Minerful method
<u>Sepsis Cases</u> Traces: 1050, Events: 15214	10.59 sec	4.46 sec
<u>BPIC 2012</u> Traces: 13087, Events: 262200	100.5 sec	30.58 sec
<u>BPIC 2017</u> Traces: 31509, Events: 1202267	452.6 sec	126.59 sec

Table 8 – Performance results for the process discovery methods

5.2.2 Log Generation methods

In this section, the Alloy Log Generator and the Minerful Log Generator were analyzed with three MP-Declare models. These models were constructed as follows. The number of activities was an even positive integer N . We inserted an Existence constraint for each activity, resulting in N existence constraints. Secondly, we added $N/2$ Response constraints. For the experiments, we used N equal to 8, 12 and 18 and named the models as Model-8, Model-12 and Model-18.

For example, Model-6 would have the constraints: {Existence(Activity1), Existence(Activity2), Existence(Activity3), Existence(Activity4), Existence(Activity5), Existence(Activity6), Response(Activity1, Activity6), Response(Activity2, Activity5), Response(Activity3, Activity4)}.

MP-Declare model	Alloy Log Gen.	Minerful Log Gen.
Model-8	17.08 sec	2.72 sec
Model-12	18.63 sec	6.05 sec
Model-18	28.48 sec	139.1 sec

Table 9 – Performance results for the log generation methods

Table 9 shows the time to generate a log containing 100 traces with minimum length 18 and maximum length 36 from Model-8, Model-12 and Model-18 with the log generation methods. For the Alloy Log Generator, vacuity detection option was set to true. According to the results, it can be seen that the Alloy Log Generator method is faster than the Minerful method as the number of possible traces increases; however, the Minerful method becomes faster than the Alloy Log Generator as the number of possible traces is lower.

5.2.3 Conformance Checking methods

In this part, the conformance checking methods were tested with respect to log and model sizes. We extended the models Model-8, Model-12 and Model-18 from previous section to Model-8wd, Model-12wd and Model-18wd, which means they are with data conditions. These models can be found in

https://github.com/alpdenizz/RuleMiningTool/tree/master/models_for_loggen

The first row of Table 10 can be interpreted as follows. An input log was generated with Alloy Log Generator from Model-8wd using number of traces: 100, minimum trace length: 8 and maximum trace length: 16. This log was checked with respect to a model containing only 4 Response constraints different from the ones in Model-8wd. For the Declare Replayer method, the data conditions in the input model were removed.

Log	Model	Declare Analyzer	Data-Aware Declare Replayer	Declare Replayer
Model-8wd Min Trace Len: 8 Max Trace Len: 16	Constraints: 4	1.94 sec	19.72 sec	4.15 sec
Model-12wd Min Trace Len: 12 Max Trace Len: 24	Constraints: 4	2.12 sec	37.85 sec	7.50 sec
Model-18wd Min Trace Len: 18 Max Trace Len: 36	Constraints: 4	2.37 sec	73.68 sec	13.66 sec

Table 10 – Execution times for the conformance checking methods w.r.t. trace length

Table 10 presents the required time for the conformance checking methods with respect to trace length. It is evident that the Declare Analyzer method is the fastest and the Data-Aware Declare Replayer is the slowest. Secondly, as the trace length increases, the increase in execution time for the Declare Analyzer and the Declare Replayer is subtle; it is more significant for the Data-Aware Declare Replayer. This was expected since the Declare Analyzer does not compute alignments (and for this reason it is the fastest method) and the Declare Replayer does not consider data (differently from the Data-Aware Declare Replayer).

Log	Model	Declare Analyzer	Data-Aware Declare Replayer	Declare Replayer
Model-18wd Min Trace Len: 18 Max Trace Len: 36	Constraints: 4	2.59 sec	56.37 sec	8.63 sec
Model-18wd Min Trace Len: 18 Max Trace Len: 36	Constraints: 6	2.54 sec	129.18 sec	39.73 sec
Model-18wd Min Trace Len: 18 Max Trace Len: 36	Constraints: 9	2.62 sec	503.92 sec	381.62 sec

Table 11 – Execution times for the conformance checking w.r.t. model size

Table 11 presents the required time for the conformance checking methods with respect to model size. For the Declare Analyzer method the model size influences the execution times much less than for the Declare Replayers.

6. Conclusion

This thesis presents a process mining tool providing a wealth of techniques based on declarative models. The users can discover a declarative model from an event log, generate a log from an MP-Declare model, do conformance checking comparing a log and an MP-Declare model and design an MP-Declare model. They can also post-process the results, filter them, and store them in the file system. Moreover, the integration of the different methods in only one tool allows the users to create a tool chain and use different methods in combination.

For the evaluation, a usability test was conducted with experts in declarative process analysis. The participants completed a task list in a think aloud session on Skype. Afterwards, we interviewed them on their overall experience, pros and cons, and recommendations for the tool.

In the future, the tool could be refactored to have more interfaces so that the integration of other declarative process analyses could be more feasible. It could be migrated to the latest JavaFX version with Spring framework to mitigate the compatibility and the memory leak risks. The results of the user evaluation also provide a lot of useful hints for improving the interface in the future.

7. References

- [1] W. M. P. van der Aalst. *Process mining: discovery, conformance and enhancement of business processes*. Springer Berlin Heidelberg, 2011.
- [2] "ProM website," [Online]. Available: <http://www.promtools.org>.
- [3] C. W. Günther & A. Rozinat. "Disco: Discover Your Processes" *BPM (Demos)*, 940, pp. 40-44, 2012.
- [4] H. Verbeek, J. Buijs, B. van Dongen, W. M. P. van der Aalst. "XES, XESame, and ProM 6". *Information Systems Evolution*. vol. 72, pp. 60–75, Springer, 2011.
- [5] B. van Dongen, W. M. P. van der Aalst. "A Meta Model for Process Mining Data". *EMOI-INTEROP*, 160, p. 30, 2005.
- [6] F. M. Maggi, C. Di Ciccio, C. Di Francescomarino & T. Kala. "Parallel algorithms for the automated discovery of declarative process models". *Information Systems*, pp. 136-152, 2018.
- [7] M. Pesic. "Constraint-based workflow management systems : shifting control to users". *Doctor of Philosophy, Department of Industrial Engineering & Innovation Sciences, Eindhoven*. DOI: 10.6100/IR638413
- [8] O. Kupferman & M. Y. Vardi. "Vacuity detection in temporal model checking". *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, pp. 82-98. Springer, Berlin, Heidelberg, September 1999.
- [9] A. Burattin, F. M. Maggi and A. Sperduti, "Conformance checking based on multiperspective declarative process models," *Expert Systems with Applications*, pp. 194-211, 2016.
- [10] V. Skydaniienko, "Data-aware Synthetic Log Generation for Declarative Process Models," *Masters Thesis, University of Tartu*, 2018.
- [11] C. Di Ciccio & M. Mecella. "On the discovery of declarative control flows for artful processes". *ACM Transactions on Management Information Systems (TMIS)*, 5(4), p. 24, 2015
- [12] J. Ellson, E.R. Gansner, E. Koutsofios, S.C. North and G. Woodhull. "Graphviz-open source graph drawing tools". *International Symposium on Graph Drawing*, pp. 483-484. Springer Berlin Heidelberg, 2001.

- [13] E. Gansner, E. Koutsofios and S.C. North. "Drawing graphs with dot". 2006. [Online] Available: <http://www.graphviz.org/pdf/dotguide.pdf>
- [14] C. Mawoko, "Aligning Data-Aware Declarative Process Models and Event Logs" *Masters Thesis, University of Tartu*, 2019.
- [15] M. de Leoni, F. M. Maggi and W. M. van der Aalst, "An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data," *Information Systems*, pp. 258-277, 2015.
- [16] C. Di Ciccio, M. L. Bernardi, M. Cimitile and F. M. Maggi, "Generating Event Logs Through the Simulation of Declare Models," *Lecture Notes in Business Information Processing*, pp. 20-36, 2015.
- [17] G.E. Krasner and S.T. Pope. "A description of the model-view-controller user interface paradigm in the smalltalk-80 system". *Journal of object oriented programming*, 1(3), pp. 26-49. 1988.
- [18] Viz.js "A hack to put Graphviz on the web" [Online] Available: <https://github.com/mdaines/viz.js>
- [19] The task list for usability test [Online] Available: https://drive.google.com/open?id=1_6PbATLepfYsQr_ljBdg9uYv6aRuh8nn
- [20] F (Felix) Mannhardt. *Sepsis Cases - Event Log*. en. 2016. DOI: 10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460. URL: <https://data.4tu.nl/repository/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460>.
- [21] B.F. Van Dongen. *BPI Challenge 2012*. nl. 2012. DOI: 10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f. URL: <https://data.4tu.nl/repository/uuid:3926db30-f712-4394-aebc-75976070e91f>.
- [22] B.F. Van Dongen. *BPI Challenge 2017*. en. 2017. DOI: 10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b. URL: <https://data.4tu.nl/repository/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b>.
- [23] "Apromore website," [Online]. Available: <https://apromore.org>.

License

Non-exclusive licence to reproduce thesis and make thesis public

I, Denizalp KAPISIZ,

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

reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

Rule Mining with RuM,

supervised by **Fabrizio Maria Maggi.**

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.
3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.
4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Denizalp KAPISIZ

14/08/2019