UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Anti Alman

A Desktop Application for Advanced Business Rule Mining

Master's Thesis (30 ECTS)

Supervisor(s): Fabrizio Maria Maggi

Tartu 2020

Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr. Fabrizio Maggi who provided guidance and colossal support during the writing of this thesis and continued to support me even after the thesis went long over the deadline. Additionally, I would like to express my sincere gratitude to Dr. Alexander Nolte who provided invaluable guidance with the user evaluation of RuM.

I would also like to thank my colleagues at Quretec OÜ who were both understanding and supportive of my decision to fully dedicate myself to finishing my Master's Thesis.

Last but not least, I would like to thank my family and friends for being supportive throughout my university studies.

A Desktop Application for Advanced Business Rule Mining

Abstract:

Process mining is one of the research disciplines belonging to the field of Business Process Management (BPM). The central idea of process mining is to use real process execution logs in order to discover, model, and improve business processes. There are multiple approaches to modeling processes with the most prevalent being procedural models. However, procedural models can be difficult to use in cases where the process is less structured and has a high number of different branches and exceptions. In these cases, it may be better to use declarative models, because declarative models do not aim to model the end-to-end processes step by step, but they constrain the behaviour of the process using rules thus allowing for more variability in the process model.

There are multiple applications available for working with procedural models. For example, Disco and Apromore, both of which have a highly polished user interface and are relatively easy to use. However, there are currently no comparable applications for working with declarative models.

This thesis builds on the Master's Thesis of D. Kapisiz in order to develop an already existing application, RuM, into an accessible and easy to use process mining application. While RuM itself already has most of the needed functionality, the user interface of RuM is not well polished and does not have an appealing look in general. In this Master's Thesis we will completely redesign and reimplement the user interface of RuM while also making technical changes in order to enable its continued development. The new user interface has been thoroughly evaluated by conducting a user evaluation involving 4 experts of declarative models and 4 experts of business process mining in general. The main findings of the user evaluation will be presented as a part of this thesis.

Keywords:

Process mining, Declarative process models, Declare, Process discovery, Conformance checking, Process analytics tool, User Interface Design

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

Töölauarakendus edasijõudnud ärireeglite kaeveks

Lühikokkuvõte:

Protsessikaeve on üks äriprotsesside juhtimise valdkonda (BPM) kuuluvatest uurimistöö distsipliinidest. Protsessikaeve keskne idee on kasutada reaalsete protsesside täitmisel tekkivaid logisid äriprotsesside tuvastamiseks, modelleerimiseks ja parendamiseks. Protsesside modelleerimiseks on mitmeid lahendusi, milledest kõige levinum on protseduuriliste mudelite koostamine. Protseduuriliste mudelite kasutamine võib samas osutuda keeruliseks kui tegemist on vähem struktureeritud protsessiga, mis sisaldab palju erinevaid harusid ja erandeid. Sellistel juhtudel võib olla kasulikum koostada deklaratiivseid mudeleid, mis ei ürita kogu protsessi algusest lõpuni samm haaval modelleerida vaid seavad protsessi käitumisele kitsendusi ja lubavad protsessi mudelis rohkem varieeruvust.

Protseduuriliste mudelitega töötamiseks on hetkel mitmeid lihtsalt kättesaadavaid rakendusi. Näiteks Disco ja Apromore, milledest mõlema kasutusliides on hästi läbimõeldud ja lihtsasti kasutatav. Samas hetkel puudub nendega sarnane rakendus, mis sobiks deklaratiivsete mudelitega töötamiseks.

Käesolev magistritöö arendab edasi D. Kapisiz magistritöö raames loodud rakendust RuM eesmärgiga arendada ligipääsetav ja lihtsasti kasutatav protsessikaeve rakendus. Kuigi RuM sisaldab juba enamust vajalikust funktsionaalsusest, siis RuM-i kasutusliides ei ole lihvitud ja ei ole visuaalselt atraktiivne. Käesoleva magistritöö raames asendatakse kogu olemasolev kasutusliides täielikult ja lisaks tehakse mõningaid tehnilisemaid muudatusi eesmärgiga tagada RuM-i jätkuva arendamise võimekus. Uue kasutajaliidese hindamiseks viiakse läbi põhjalik kasutatavuse test nelja deklaratiivsete mudelite eksperdiga ja nelja protsessikaeve eksperdiga. Kasutatavuse testi põhilised leiud esitatakse käesoleva magistritöö osana.

Võtmesõnad:

Protsessikaeve, deklaratiivse protsessikaeve mudelid, Declare, protsesside väljaselgitamine, vastavuskontroll, protsessianalüütika tööriist, kasutusliidese disain

CERCS: P170 - Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimis-teooria)

Table of Contents

1 Introduction 7
2 Background 9
2.1 Overview of Process Mining 9
2.1.1 Process Discovery 9
2.1.2 Conformance Checking 10
2.1.3 Process Enhancement 10
2.1.4 Log Generation 10
2.2 Process Mining Applications 11
2.2.1 Disco 11
2.2.2 ProM 12
2.2.3 Apromore 12
2.2.4 MINERfulGUI 14
2.2.5 RuM (2019 version) 15
2.3 Declare Modeling Language 16
2.3.1 Main Concepts of Declare 16
2.3.2 Unary Templates 17
2.3.3 Positive Binary Templates 18
2.3.4 Negative Binary Templates 20
2.3.5 Constraint Conditions 22
2.3.6 *decl* File Format 22
3 Application Architecture 24
3.1 Used Technologies 24
3.2 Software Design 24
3.3 Dependency management 25
3.4 Application packaging 26
4 User Interface Design and Components 27
4.1 Application main layout 27
4.2 Application styles 28
4.3 Slide-in Panels 28
4.4 Feedback messages 30
4.5 Editable tables 30
5 Functional Overview 32
5.1 Discovery 32
5.1.1 Available methods 33

5.1.2 Result views 33
5.2 Conformance Checking 35
5.2.1 Available methods 36
5.2.2 Result views 36
5.3 MP-Declare Editor 38
5.3.1 Voice Input 38
5.3.2 Entering Activities and Attributes Manually 40
5.3.3 Entering Constraints Manually 41
5.3.4 Model Visualizations 41
5.4 Log generation 42
5.4.1 Available methods 42
5.4.2 Result view 43
6 User Evaluation 44
6.1 Study Methodology 44
6.2 Survey Overview and Results 45
6.3 Results of Usability Tests 46
6.3.1 Base User Interface Elements 47
6.3.2 Model Views 48
6.3.3 Discovery 49
6.3.4 Conformance Checking 50
6.3.5 MP-Declare Editor 51
6.3.6 Log Generation 51
6.3.7 Terminology issues 52
6.3.8 General impressions 52
7 Conclusion and Future Work 53
References 54
Appendix 57
I. User Evaluation Introductory Email 57
II. User Evaluation Consent Form 58
III. User Evaluation Task List 60
IV. Additional Files 62
V. License 63

1 Introduction

Business Process Management (BPM) has become an integral part of how companies organize their workflows starting from the higher levels of management as recommended in ISO 9000, ISO 9001 Quality Management Principles (especially principles 4, 5 and 6) [1] to modeling and optimizing lower level processes through the use of various process mining techniques [2, 3].

Process mining is the part of BPM which is focused on the analysis of business processes based on event logs containing information about process executions. Process mining techniques are usually divided into three main branches which are process discovery, conformance checking, and process enhancement. Process discovery is used to generate a model of the process based on the event log of the process. Conformance checking is used to compare an event log to a process model with the aim of finding discrepancies between the event log and the model. Process enhancement is used to modify the model based on the information gained from the event log.

In addition to the above mentioned three branches, it is also possible to use a process model to generate an artificial event log of the process. This can be useful for testing out new process mining algorithms or gaining a better understanding of how the process executions may look based on the model.

Process models can be divided into at least two types. The most common type is procedural models, which aim to describe end-to-end processes and allow only for activities that are explicitly triggered through control-flow [4]. However, modeling step by step the entire control-flow can be undesirable in some cases. For example, if the process is less structured and has a high number of different branches and exceptions the model could become quickly unreadable. In these cases, it may be a better choice to use declarative process models that model the process as a set of rules that the process should follow.

This thesis focuses on making some of the existing declarative process mining techniques more accessible to a wider range of specialists, by providing an easy to use application based on the declarative modeling language Declare [5, 6].

We aim to achieve this goal by fully redesigning and reimplementing the user interface of an already existing process mining application RuM (2019)[1] [7, 8]. The stylistic direction of the new user interface of RuM is loosely based on another existing application MINERfulGUI [9].

In addition to changing the user interface some architectural changes were made to RuM with the aim of making further development easier and improving the overall code quality. Some functionality changes were also made. However, adding further functionality was not the focus of this thesis.

A user evaluation with 4 Declare experts and 4 BPM experts was conducted to evaluate the final resulting application. Each test in the evaluation consisted of a one hour long recorded session. In each test, participants were required to use RuM based on the same list of tasks and to answer a survey about their experience. The methodology and the results of the study are described as a part of this thesis together with suggestions for further improvements based on the results of the study.

[1] We use the name "RuM (2019)" in order to distinguish the version developed by Denizalp Kapisiz from the version described in the rest of this thesis.

The thesis is organized as follows. Chapter 2 introduces the background of the thesis including a short overview of the process mining field, an overview of some of the existing process mining applications and an overview of the Declare language. Chapter 3 introduces the architecture of RuM and the technologies used in developing RuM. Chapter 4 introduces the main layout of the new user interface and describes some of the general elements used throughout the application. Chapter 5 gives an overview of the functionality of RuM. Chapter 6 describes the user evaluation methodology and provides an overview of the evaluation results. Section 7 provides an overview of the thesis and highlights some of the next steps in the development of RuM.

2 Background

This chapter starts with an overview of process mining branches and short descriptions of their potential uses. The chapter continues with a short overview of some of the other applications used for process mining. The chapter ends with an overview of the Declare modeling language which is the basis for process modeling in RuM.

2.1 Overview of Process Mining

Process mining refers to the field of intelligent process execution analysis based on information about the process executions recorded in event logs. This information can stem from a variety of different systems such as Business Process Management Suites (BPMSs), Enterprise resource planning systems (ERPs) or ticketing systems [10].

Process mining is usually divided into three main branches which are process discovery, conformance checking and process enhancement.

All the above mentioned process mining branches assume the existence of an event log that contains information about the real life executions of a process. The event log is assumed to consist of events (activities that were performed) in a chronological order and each event is assumed to belong to a single trace (an instance of a process execution)[2]. Process mining is first and foremost based on facts about process executions as recorded in the event logs [11]. One of the most common standards for event logs and the one used in RuM is eXtensible Event Stream (XES) [12].

In addition to the three main branches we will also describe log generation which is included as an additional functionality in RuM.

2.1.1 Process Discovery

Process discovery is the most prominent process mining technique. It typically aims to generate a model of the process based only on the data found in the event log while making minimal assumptions about the properties of the log and of the resulting process model.

No a-priori knowledge about the process is used, because this knowledge may not exist or may be inaccurate when compared to the real life execution of the process. This also means that the process discovery algorithms are independent of the domain they are used in. Or, in other words, the same process discovery algorithm can be used in any domain as long as the events of a process execution are recorded in a standardized way.

Process discovery is not limited to modeling the sequence in which individual activities are performed (control-flow discovery). Process mining can also be used to discover organizational models, business rules, policies, etc. And, in addition to the control-flow, process models can also include other aspects of the process like, for example, the time perspective or other data perspectives.

The focus of RuM is on discovery of declarative models which do not attempt to model the end-to-end processes, but instead to model concrete rules that the process adheres to (constraints) using the process modeling language Declare.

[2] Some papers and applications use the term “case” instead of the term “trace” to refer to an instance of a process execution. Throughout this thesis and RuM we use the term “trace” as defined in the eXtensible Event Stream (XES) standard.

2.1.2 Conformance Checking

Conformance checking is used to compare the modeled behavior to the observed behavior of the process as recorded in an event log. The model may be created by hand or may be the result of process discovery. The event log is usually based on real process executions however an artificially generated log can also be used.

The main aim of conformance checking is to diagnose deviations between the model and the event log. Information about these deviations can be used in a variety of ways such as a starting point for model enhancement, to judge the quality of a discovered model, to identify deviating traces and what they have in common, etc.

RuM provides two conformance checking methods. The first evaluates conformance through counting the number of constraint activations, violations and fulfilments in the event log [13]. The second replays the entire event log on the provided model and produces an alignment between each trace in the log and the constraints defined in the model. It is also possible to use RuM for detecting events that have different attributes[3] than allowed by the model.

2.1.3 Process Enhancement

Process enhancement is used to modify existing process models based on the information gained from analysing previous process executions. For example, by changing the order in which activities are intended to be performed or by removing unnecessary activities from the process model.

There are two main types of enhancement. The first type is repair which focuses on modifying the model so that it better reflects the behavior recorded in the log. In the context of declarative models this can be achieved by adding, removing or modifying the constraints found in the model. The second type is extension, which aims to add new perspectives to the model. In the context of declarative models, this can be achieved by adding new data conditions to the constraints already defined in the model.

RuM currently does not provide automated methods for process enhancement. However, manual process enhancement is supported through a model editor in RuM, which supports Multi-Perspective Declare, an extended version of Declare supporting the definition of constraints with conditions over data attributes and timestamps [14].

2.1.4 Log Generation

In addition to the three main branches of process mining it is also possible to generate artificial logs based on pre-existing models. These logs can be helpful for testing new process mining techniques or to gain a better understanding of how the process executions may look based on the model of the process.

RuM provides two different log generation methods, one of which is also capable of generating event attributes if attribute definitions and their bindings to activities are defined in the model.

[3] Some parts of the RuM user interface use the term “payloads” to refer to attributes. Throughout this thesis we prefer the term “attributes”. However the term “payloads” may be used in some cases when describing the user interface. Both terms are considered interchangeable throughout this thesis.

2.2 Process Mining Applications

In this chapter, we give an overview of some of the process mining applications available. For each application we mention some of its strengths and weaknesses in comparison to the new version of RuM.

Based on the overview of the existing process mining applications, we believe there is room and need for a desktop application that is easy to use and focuses on declarative models. We aim to position RuM as the starting point for people wishing to use declarative models for their process mining needs or who wish to start exploring the capabilities of declarative models.

2.2.1 Disco

Disco [15] is a relatively lightweight and well known process mining application that can be used for discovering procedural process models, exploring the event log (both general statistics and specific traces) and animating process executions on the discovered log. Disco uses its own discovery algorithm called Disco Miner which emphasises process discovery speed [16].

The main design goals for Disco were usability, fidelity and performance which has resulted in an application that is easy to use and requires very little prior knowledge of process mining techniques. The main view of Disco after opening a small event log is shown in Figure 1. It is worth pointing out that no other actions besides opening the event log were performed to get the initial results shown in the figure.

The main drawback of Disco in comparison to RuM is the fact that it does not support declarative process models. Disco also does not support conformance checking, log generation or model editing capabilities.

Disco can be considered one of the main inspirations for creating RuM due to its accessibility and streamlined interface.

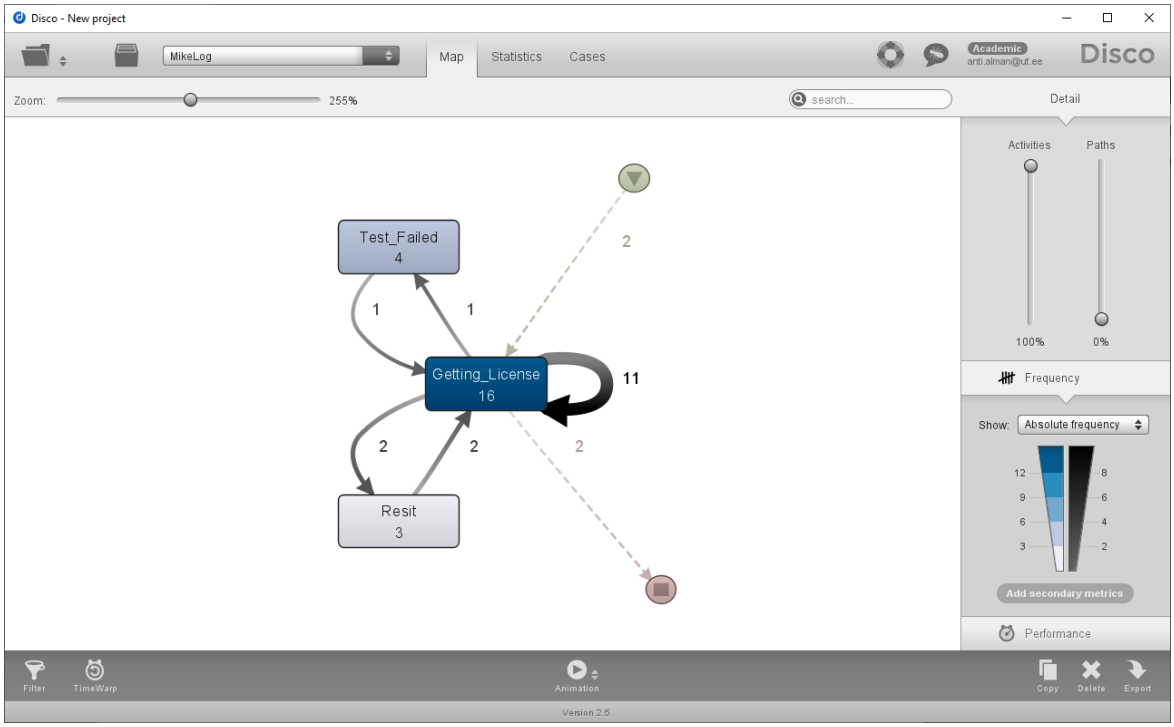


Figure 1. Main results panel of Disco after opening a small event log.

2.2.2 ProM

ProM [17] is an open source desktop application that supports a wide variety of process mining techniques in the form of plug-ins. Plug-ins have been created for all branches of process mining including plugins for declarative models using the Declare language. The package manager for ProM 6.9 lists over 200 different plugins at the time of writing this thesis.

One of the main strengths of ProM is its plug-ins based architecture [18] combined with the open nature of ProM development. All researchers and developers are invited to contribute by implementing new plug-ins. This has turned ProM into an ideal application for developing, testing and distributing new process mining algorithms, but it has also made ProM an invaluable tool for process mining professionals.

The user interface of ProM (shown in Figure 2) while powerful can be considered also as one of the drawbacks of ProM. For a first time user it is quite difficult to understand the logic of the application and what the various buttons do. This is further amplified by the user interface differences between the various available plug-ins and the occasional stability issues.

While ProM provides most of the functionality that RuM provides (and a lot more), RuM aims to be a more accessible alternative for both process mining experts and novices who want to access declarative process mining functionalities.

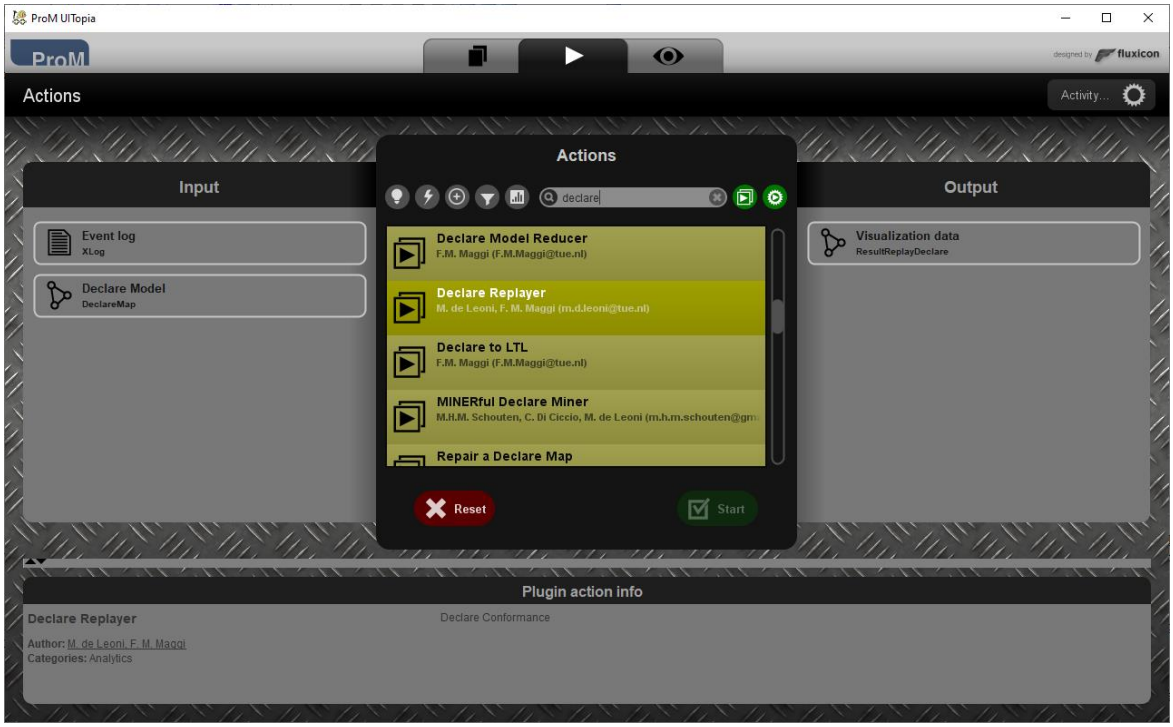


Figure 2. Actions view of ProM with Declare Replayer plugin selected.

2.2.3 Apromore

Apromore [19] is a web-based process analytics platform supporting the full spectrum of process mining functionality.

There are two editions available. The first one is a free to use community edition which includes the core platform and all the experimental plugins developed by the open-source

community. The second one is a pay to use enterprise edition which includes performance optimizations, feature-richer plugins, add-ons and connectors on top of the Apromore core platform.

The source code for the community version of Apromore is publicly available and can be used to set up and fully configure Apromore based on the user's needs. The community edition is also available as a pre-compiled image that runs on Docker[4].

The user interface of Apromore is quite straightforward and relatively easy to get started with. An example of the user interface after opening a simple event log is shown in Figure 3. It is worth pointing out that no other actions besides uploading and opening the event log were performed to get the initial results shown in the figure.

Apromore has two main drawbacks with respect to RuM. The first one is that the initial setup of the community edition of Apromore can be too complicated for the average user. The pre-compiled Docker image is of great help here, but Docker itself may be too difficult to use for non-developers. The second is that Apromore has only limited support for working with declarative models[5].

While additional support for working with declarative models could be added to Apromore, we assume that it is easier for the end user to get started with a simpler and more focused desktop application.

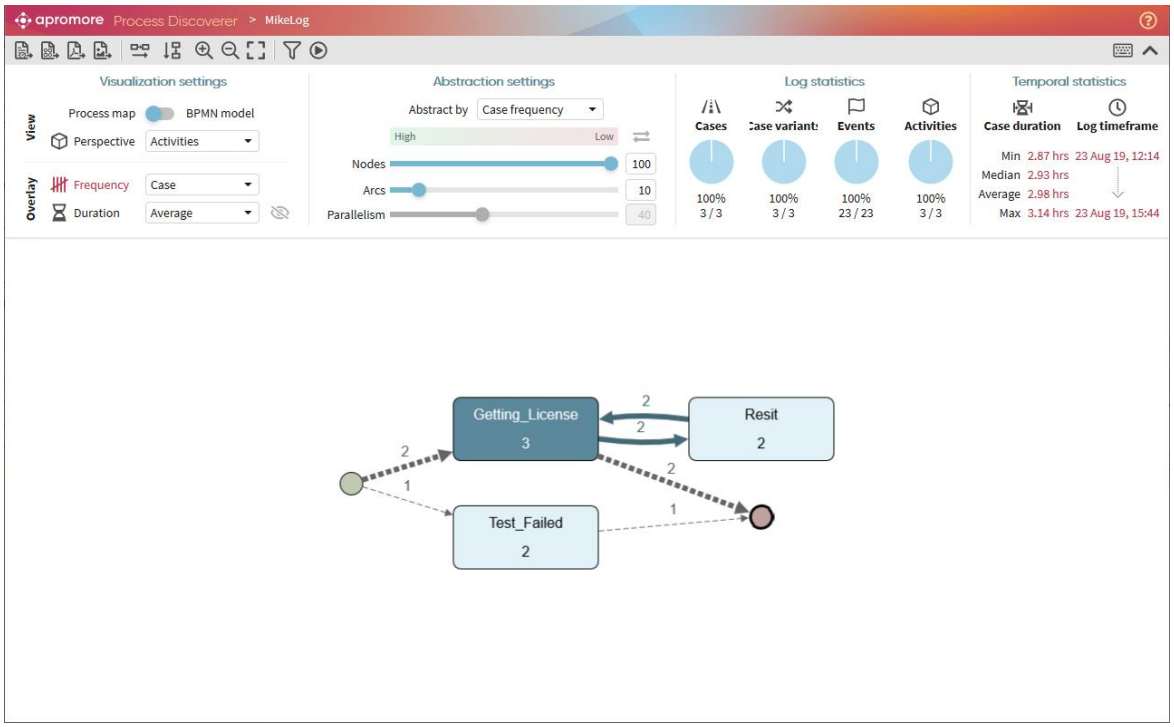


Figure 3. Process Discoverer view of Apromore after opening a small event log.

[4] https://www.docker.com/
[5] https://apromore.org/documentation/features/compliance_declare/

2.2.4 MINERfulGUI

MINERfulGUI [9] is a desktop application that focuses on automated discovery of declarative business process constraints. The core process mining functionality of MINERfulGUI is provided by the MINERful algorithm [20]. The application provides functionality for automatic process discovery, creating, modifying and simplifying a process model, performing a basic fitness check and generating a log based on a preexisting declarative process model. The application is currently under development.

The main strengths of MINERfulGUI are its visual design, intuitive user interface (shown in Figure 4) and the fact that it is possible to have multiple models and event logs open at the same time. It also introduces a novel way of visualizing Declare constraints which is easier to read when compared to the official notation, especially when it comes to unary constraints (constraints that apply only for a single activity).

The main drawback of MINERfulGUI in comparison with RuM is its focus on a single process mining algorithm. We wanted RuM to be able to utilize multiple different process mining algorithms while also presenting a unified view of the results regardless of the algorithm used.

The user interface design of RuM takes significant inspiration from MINERfulGUI when it comes to colors, layout of the application and basic user interface elements such as buttons, menus, separators etc. However, there are major changes in how most of the views are structured and presented. For example, the user interface and functionality of conformance checking and model editing are completely different in RuM.

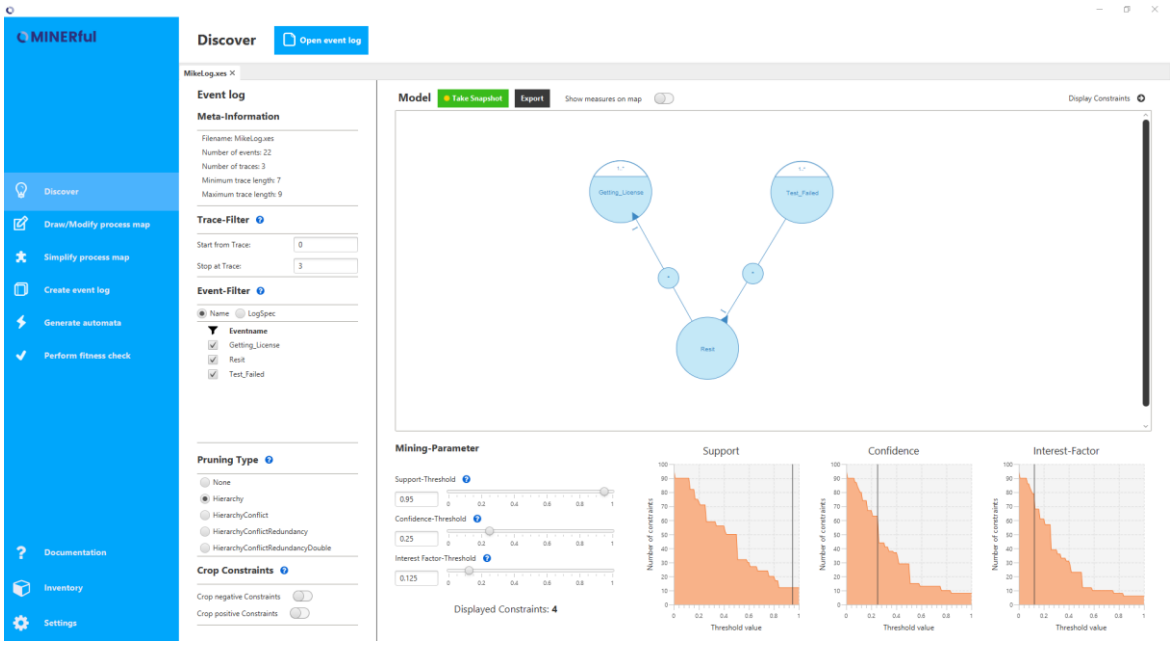


Figure 4. Discover view of MINERfulGUI after opening a small event log.

2.2.5 RuM (2019 version)

RuM (2019 version) [7, 8] is the initial attempt to create a comprehensive desktop application with an intuitive user interface and a focus on declarative models. The application was developed by Denizalp Kapisiz in 2019 as a part of his Master's Thesis.

RuM (2019 version) supports all three branches of process mining in the context of declarative models and log generation. The main contribution in RuM (2019 version) is integrating multiple different process mining algorithms into a single application and creating a new model editor based on the Declare language. However, the relatively large scope of the provided functionality meant sacrificing usability and user interface design.

The user interface of RuM (2019 version) is in some cases quite cumbersome to use, for example in the model editor or for changing the parameters of discovery. Visually it relies entirely on the basic style defined by the user interface framework, which while functional is not visually appealing. An example of the user interface is shown in Figure 5.

In this thesis, we aim to fix many of the shortcomings of the user interface of RuM (2019 version). We have reused some parts of the implementation, but the user interface has been completely redesigned and much of the application logic was also reimplemented.

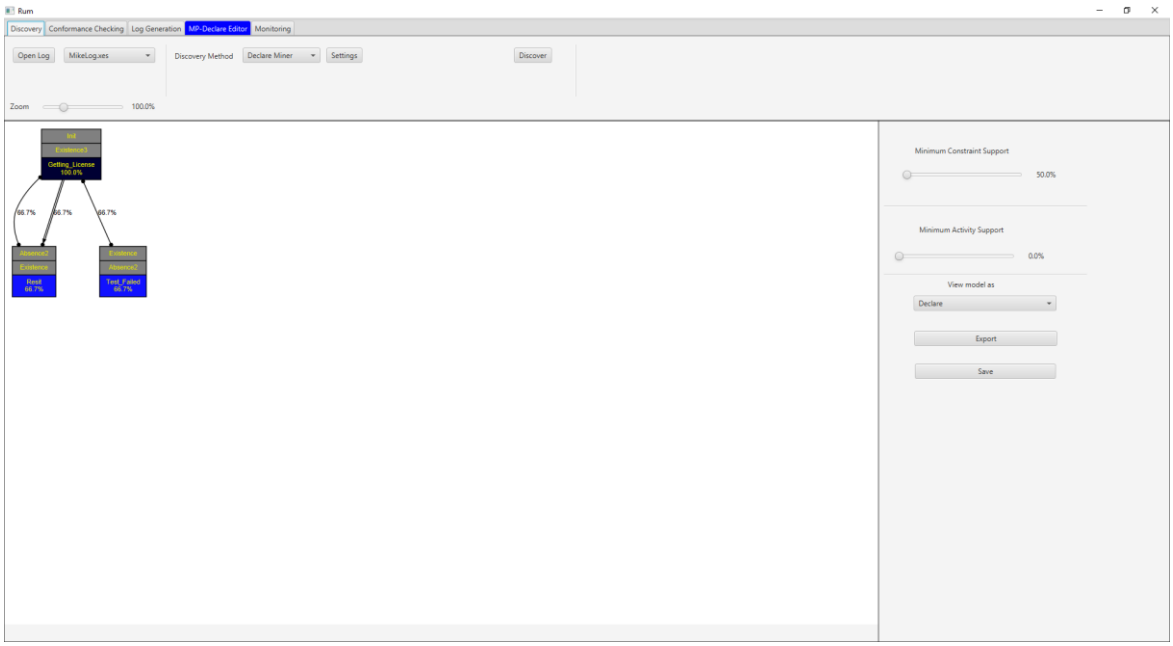


Figure 5. Discovery view of RuM (2019) after opening a small event log and rediscovering the model with 50% constraint support.

2.3 Declare Modeling Language

This chapter starts with an overview of the main concepts of the Declare language. The chapter continues with a description of the constraint templates and conditions used in RuM. The chapter ends with a short description of the decl file format that is used for import and export process models in RuM.

2.3.1 Main Concepts of Declare

Declare is a modeling language that uses a constraint-based declarative approach to define a loosely-structured process model [21]. The language is grounded in temporal logic, but no knowledge of temporal logic is required to use the language effectively.

The aim of the language and of declarative models in general is to describe a process in such a way that all the important aspects of the process are defined (such as activity A occurs immediately after activity B) while not requiring the entire process with all of its details to be modeled.

The main building blocks of the language are constraints. Each constraint consists of a template (a constraint type) and a reference to one or two activities depending on the template. Templates are divided into three groups: Unary, Positive Binary and Negative Binary. As the names of the groups suggest a unary template refers to a single activity while both positive and negative templates refer to two activities.

A template basically defines the semantic meaning of the constraint and activities in the constraint define the activities to which this meaning applies. For example, if the constraint template is “Exactly1” and the activity is A then this means the activity A should be performed exactly once during a single process execution.

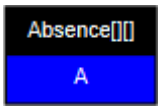
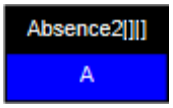
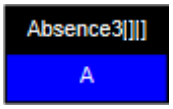
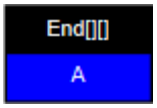
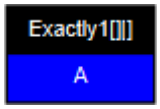
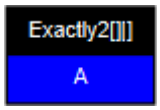
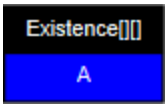
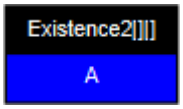
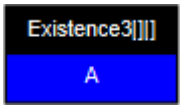
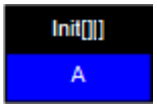
When working with Declare constraints it is important to understand three main concepts which are constraint activation, constraint fulfilment and constraint violation. For each constraint there is at least one activity that is considered the activation of the constraint. If the activation occurs during the process execution then the corresponding constraint is considered to be activated. The occurrence of an activation triggers some obligations on the occurrence of another activity (the target). For example, for the Response constraint having A as activation and B as target, the execution of A forces B to be executed eventually after A.

When a constraint is activated then it must be either fulfilled or violated by the end of the process execution. An activated constraint will be fulfilled when the condition defined by the constraint is satisfied, otherwise the constraint will be violated. If a constraint is not activated during the process execution then the constraint is considered to be vacuously satisfied [22].

2.3.2 Unary Templates

All unary templates refer to a single activity. A unary template is used either to define the cardinality of an activity in the process or to define where in the process the activity should occur. Table 1 presents all unary templates available in RuM[6].

Table 1. Unary templates used in RuM.

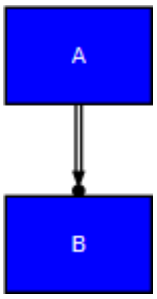
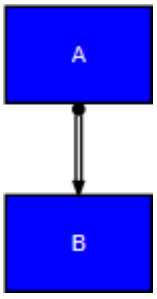
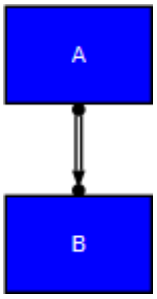
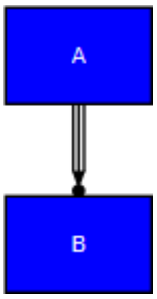
Name	Description	Representation in RuM
Absence[A]	Activity A does not occur	
Absence2[A]	Activity A occurs at most once	
Absence3[A]	Activity A occurs at most twice	
End[A]	Activity A must occur as the last event	
Exactly1[A]	Activity A occurs exactly once	
Exactly2[A]	Activity A occurs exactly twice	
Existence[A]	Activity A occurs at least once	
Existence2[A]	Activity A occurs at least two times	
Existence3[A]	Activity A occurs at least three times	
Init[A]	Activity A must occur as the first event	

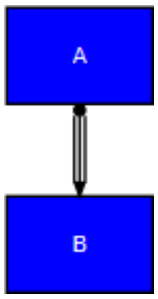
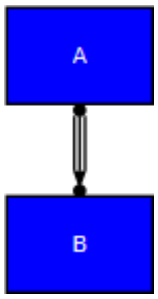
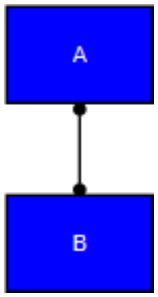
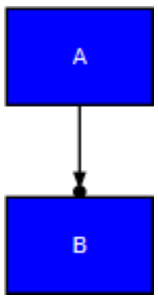
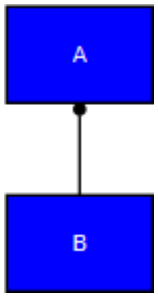
[6] The list of available templates is further limited based on the specific process mining algorithm used. This applies also to positive and negative binary templates.

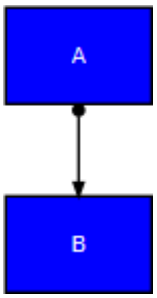
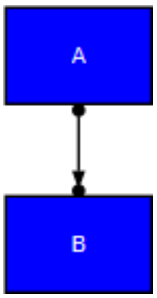
2.3.3 Positive Binary Templates

All positive binary templates refer to two activities. A positive binary template is used to define a "positive" relation between the two activities. Positive relation in this context means that if the activation occurs, then the target will also occur in some specific relation to the activation. Table 2 presents all positive binary templates available in RuM.

Table 2. Positive binary templates used in RuM.

Name	Description	Representation in RuM
Alternate Precedence[A, B]	Each time activity B occurs, it is preceded by activity A and no other activity B can recur in between Activation: B Target: A	
Alternate Response[A, B]	Each time activity A occurs, then activity B occurs afterwards before activity A recurs Activation: A Target: B	
Alternate Succession[A, B]	Activity A and activity B occur together if and only if the latter follows the former, and they alternate each other Activation: A and B Target: A and B	
Chain Precedence[A, B]	Each time activity B occurs, then activity A occurs immediately beforehand Activation: B Target: A	

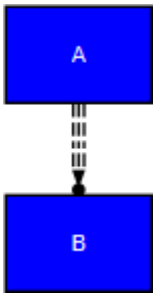
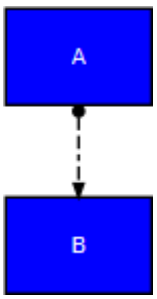
Chain Response[A, B]	Each time activity A occurs, then activity B occurs immediately afterwards Activation: A Target: B	
Chain Succession[A, B]	Activity A and activity B occur together if and only if the latter immediately follows the former Activation: A and B Target: A and B	
Co-Existence[A, B]	Activities A and B occur together Activation: A and B Target: A and B	
Precedence[A, B]	Activity B occurs if it is preceded by activity A Activation: B Target: A	
Responded Existence[A, B]	If activity A occurs then activity B occurs as well Activation: A Target: B	

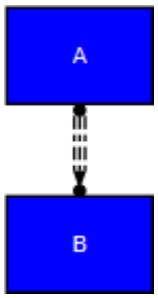
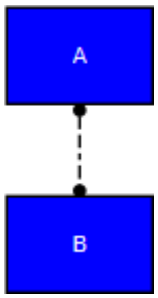
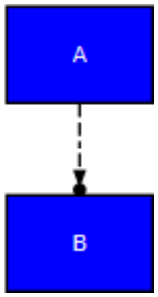
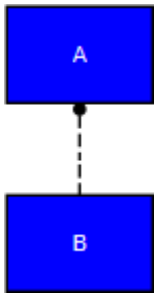
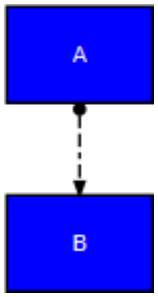
Response[A, B]	If activity A occurs then B occurs after activity A Activation: A Target: B	
Succession[A, B]	Activity A occurs if and only if it is followed by activity B Activation: A and B Target: A and B	

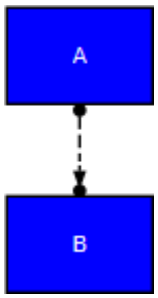
2.3.4 Negative Binary Templates

All negative binary templates refer to two activities. A negative binary template is used to define a “negative” relation between the two activities. Negative relation in this context means that if the activation occurs then this restricts the occurrence of the activity in some specific way. Table 3 presents all negative binary templates available in RuM.

Table 3. Negative binary templates used in RuM.

Name	Description	Representation in RuM
Not Chain Precedence[A, B]	Each time activity B occurs, then activity A does not occur immediately beforehand Activation: B Target: A	
Not Chain Response[A, B]	Each time activity A occurs, then activity B does not occur immediately afterwards Activation: A Target: B	

Not Chain Succession[A, B]	Activities A and B occur together if and only if the latter does not immediately follow the former Activation: A and B Target: A and B	
Not Co-Existence[A, B]	Activities A and B never occur together Activation: A and B Target: A and B	
Not Precedence[A, B]	Activity B occurs if it is not preceded by activity A Activation: B Target: A	
Not Responded Existence[A, B]	If activity A occurs then activity B does not occur Activation: A Target: B	
Not Response[A, B]	If activity A occurs then activity B does not occur after activity A Activation: A Target: B	

Not Succession[A, B]	Activity A can never occur before activity B Activation: A and B Target: A and B	

2.3.5 Constraint Conditions

In addition to templates and activities, a constraint may also define data conditions. Data conditions model the process based on the data gathered during a specific activity execution. These conditions are defined in an extension of Declare named Multi-Perspective Declare (MP-Declare) [23].

There are three types of conditions in MP-Declare: activation conditions, correlation conditions and temporal (time) conditions[7]. Activation condition defines a condition that must hold true in order for the constraint to be activated. Correlation condition defines a condition that must hold true in order for the constraint to be fulfilled (only applicable to binary constraints).

The meaning of time conditions depends on whether they are used in the context of a unary or a binary template. In case of unary templates, the time condition defines the minimum and maximum allowed time distance of the activity from the start of the trace. In case of binary constraints, the time condition defines the minimum and maximum time distance allowed between the target and the activation.

For example an activation condition may be defined as “A.Grade>4” which means that the constraint will be activated only if the attribute “Grade” of the activation activity is greater than 4.

A correlation condition may be defined, for example, as “same Driver” which means the constraint will be fulfilled only if the value of attribute Driver is the same for both the activation and the target.

A time condition may be defined, for example, as “1,5,d” which means an activity must occur after 1 day and before 5 days from the start of the process execution (in case of unary constraints) or the target must occur between 1 and 5 days after the occurrence of the activation.

2.3.6 *decl* File Format

RuM uses the decl file format for importing and exporting process models. This format was first introduced together with the Alloy Log Generator [24]. Initially the decl file format supported only activation and correlation conditions, however the format was extended during the development of RuM (2019) to also allow for time conditions to be defined.

The decl file format is basically a text file where each line defines an activity, an attribute, a binding between an activity and an attribute, or a constraint. An activity is defined by its

[7] Temporal conditions are referred to as “time conditions” in RuM and throughout the rest of this thesis. This helps users not familiar with Declare to better understand the meaning of this type of conditions.

name, an attribute is defined by its name, type and either by the possible value range (in case of integers and floats) or by possible values (in case of enumerative attributes). A binding is defined by linking an activity to the attributes that the activity should always be recorded with. A constraint is defined by the template name, related activities and optionally the constraint conditions. An example of a small declarative model defined in decl format is given in Figure 6.

```
activity Driving_Test
bind Driving_Test: Driver, Grade
activity Getting_License
bind Getting_License: Driver
Driver: John, Jane
Grade: integer between 1 and 5
Response[Driving_Test, Getting_License] |A.Grade>4 |same Driver |1,5,d
```

Figure 6. An example of a declarative model in decl format.

The response constraint in the model in Figure 6 specifies that when activity Driving_Test occurs with attribute Grade>4 then activity Getting_License should follow with the same value for attribute Driver and between 1 and 5 days.

3 Application Architecture

In this chapter, we introduce the architecture of RuM. The chapter starts with an overview of the main technologies used in developing RuM. The chapter continues with an overview of some of the main software design aspects of the application. The chapter concludes with a description of how the application is currently packaged and points out possible areas of improvement in this regard.

3.1 Used Technologies

Compared to RuM (2019) the current version was updated from Java 8[8] to Java 11[9]. This upgrade was mainly necessary because Java 8 is reaching the end of its lifecycle and public updates to it are expected to end in December 2020[10]. The secondary reason was to support other projects related to RuM which required a newer version of Java. Java 11 was chosen because it is the latest Long-Term-Support (LTS) release at the time of writing and therefore enables the continued development of RuM in the future.

Together with updating the required Java version it was also decided to upgrade the user interface framework from JavaFX 8[11] to JavaFX 11[12]. This was in part a logical upgrade to do while upgrading the Java version, but also allowed to take advantage of newer user interface elements and general API improvements of the framework.

A natural companion to JavaFX is Scene Builder which provides a visual user interface for designing application views. JavaFX Scene Builder 11.0.0[13] was used extensively throughout the development of the current version of RuM.

The current version of RuM uses the same approach for Declare model visualizations as the one used in RuM (2019). Visualizations rely on the built in browser support of JavaFX[14]. The built in browser is used in combination with VizJS to generate and display a graph in SVG[15] format from a DOT file by using Graphviz [25].

Some implementation changes were made. The main change was reusing the same WebView[16] object when the visualization is updated instead of creating a new one for each update. This increases the responsiveness of the user interface. Additionally, the base html pages for the different visualizations were merged into a single html page.

3.2 Software Design

The application is designed following the model-view-controller paradigm [26]. This paradigm divides an application into three conceptual parts: model, view and controller. The main aim of this division is to enforce a separation of concerns such that the model objects are responsible for the operations related to the application domain, the view objects are responsible for displaying the applications state and allowing the user to interact with the application and the controller objects are responsible for the user interactions with the model.

[8] https://docs.oracle.com/javase/8/
[9] https://docs.oracle.com/en/java/javase/11/
[10] https://www.oracle.com/java/technologies/java-se-support-roadmap.html
[11] https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm
[12] https://gluonhq.com/products/javafx/
[13] https://gluonhq.com/products/scene-builder/
[14] https://openjfx.io/javadoc/11/javafx.web/javafx/scene/web/WebEngine.html
[15] https://www.w3schools.com/graphics/svg_intro.asp
[16] https://openjfx.io/javadoc/11/javafx.web/javafx/scene/web/WebView.html

In the context of RuM the responsibilities of these three conceptual parts can be further described as follows:

- Model objects are responsible for
  - Holding the data contained in all the logs and models the user has either opened, created or modified,
  - Holding the input and output data of all the supported process mining tasks based on the user inputs and the task results,
  - Performing all supported process mining tasks that the user has requested (i.e. running the actual process mining algorithms),
  - Performing all preprocessing and postprocessing that may be needed for a specific process mining task.
- View objects are responsible for
  - Providing all of the user interface elements that the user can interact with (including input fields, buttons for navigation, buttons for different actions etc.),
  - Receiving all possible inputs from the user,
  - Displaying the state of the application (for example which user interface elements can be used, whether or not a process mining task is in progress, displaying error messages etc.),
  - Displaying and allowing the user to navigate the results of all supported process mining tasks.
- Controller objects are responsible for
  - Receiving, validating and processing all user inputs,
  - Handling the navigation and view changes in the user interface,
  - Loading the data contained in all the logs and models the user has either opened, created or modified,
  - Preparing and starting all supported process mining tasks that the user has requested,
  - Preparing the results of all supported process mining tasks for displaying them in the corresponding views.

In the context of JavaFX, it is also important to point out that JavaFX encourages the separation of view objects from the rest of the application by design. This is because it is encouraged (but not required) to define all the different application views and user interface components in FXML files[17]. This recommendation has been followed throughout RuM with only a few minor exceptions.

3.3 Dependency management

All the dependencies in RuM (2019) were managed by using a separate folder for jar files and manually adding these files to the build path of the application. This however was an error prone and quite time consuming process when developing the application further. For this reason, it was decided to start using Maven[18].

All the dependencies were replaced with versions from mvnrepository[19] where possible and the corresponding jars were deleted from the RuM source code repository. For dependencies

[17] https://docs.oracle.com/javase/8/javafx/fxml-tutorial/why_use_fxml.htm
[18] http://maven.apache.org/
[19] https://mvnrepository.com/

that are not available in mvnrepository a local repository was created instead[20] containing all the corresponding dependencies. This has the benefit that all dependencies are managed by maven and therefore the setup for new developers will be much easier.

3.4 Application packaging

RuM is currently packaged as a "fatjar" using the Apache Maven Shade Plugin[21]. This means that the application is compiled into a single runnable jar file that contains all the dependencies of the application. In addition, the compiled jar also contains all operating system specific JavaFX libraries. The benefit of this is that the same jar file can be used on Windows, Linux and Mac operating systems with the only requirement that Java 11 has to be installed.

The modularization principles introduced in Java 9[22] are currently not followed in RuM. In fact, the use of the Shade Plugin breaks any strong encapsulation that may be defined by the modules that RuM is using. While RuM does not rely on the strong encapsulation to be broken it was decided that during this stage of development it is not worth redesigning the application to follow the modularization principles of Java 9 and above. Such a redesign would require going through all and probably changing many of the dependencies of RuM while providing no extra functionality.

In the long term the packaging of RuM should be changed such that builds are platform specific (i.e. separate for each operating system) as this would reduce the size of the compiled application. It would also be helpful to package the application in such a way that Java does not need to be installed on the user's machine and thus the barrier of entry could be lowered further.

[20] https://stackoverflow.com/questions/4955635/how-to-add-local-jar-files-to-a-maven-project/28762617#28762617
[21] http://maven.apache.org/plugins/maven-shade-plugin/
[22] https://www.oracle.com/corporate/features/understanding-java-9-modules.html

4 User Interface Design and Components

In this chapter, we start by introducing the main layout of the new user interface of RuM. The chapter continues with a short description of how the application styles are handled. The chapter concludes with descriptions of some of the general user interface elements used throughout the application, such as tables, slide-in panels and feedback messages.

4.1 Application main layout

The main layout of RuM (shown in Figure 7) consists of the following components:

- A sidebar menu on the left that allows users to quickly navigate the different sections of the application. The sidebar is always visible but can be minimized so that only the icons of the sections are shown. The available sections are Discovery, Conformance Checking, Log Generation and MP-Declare Editor.
- A section header which contains the title of the section and the main button(s) to start working in that section. The header is always visible, except when the start page of the application is displayed. The header usually contains one button which allows to select the main file needed to work with the functionalities available in the corresponding section (either an event log or a model). In the case of MP-Declare Editor it is possible to start editing a new empty model or to import and change an existing one.
- A row of tabs where each tab name corresponds to one of the main files that the user has opened (and not yet closed) in that section. Currently there is no hard limit on the number of tabs.
- A section and tab specific work area which is used to set the parameters of the process mining tasks, to start the tasks and to explore the task results. The work area is usually organized in columns such that the left-most column is used for setting the parameters and the following columns are used for exploring the results.

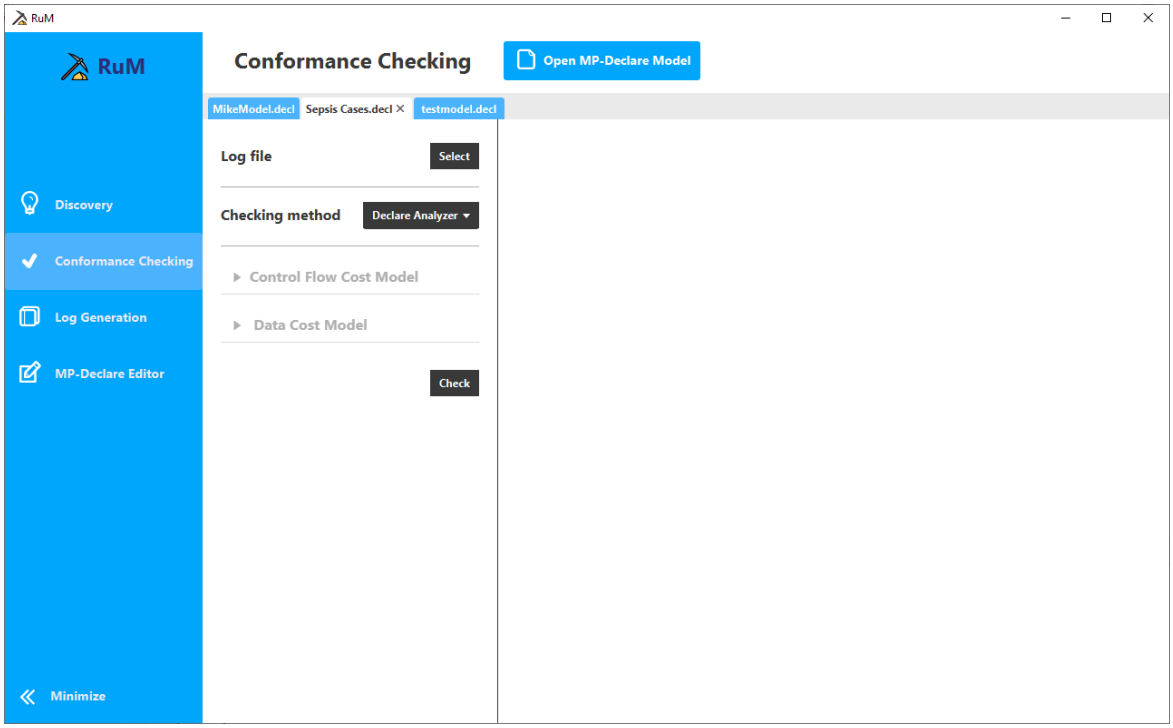


Figure 7. Conformance checking section of RuM with multiple tabs opened as an example of the application's main layout.

It is also important to point out that the application layout and logic is designed in such a way that the navigation is restricted as little as possible. The user is free to navigate different tabs and sections of the application even if a process mining task is running and multiple tasks can be running at the same time.

In all sections except for conformance checking it is also possible to export the results of that section. The export button is always located at the top-right of the work area. In the case of the Discovery and the MP-Declare Editor sections the exported file also depends on the result view that is currently selected. In the case of the Log Generation section the exported file will always be an event log in the XES format.

4.2 Application styles

JavaFX allows for the user interface to be styled using Cascading Style Sheets (CSS)[23] . This possibility has been heavily utilized in developing the new user interface of RuM.

Currently the styles are split into two stylesheets. One for the popup windows and one for the rest of the application. Currently there are about 150 custom style rules defined, some of which are overrides of the JavaFX default styles while others are rules specific to RuM. The use of inline styles has been generally avoided.

The stylesheets mostly follow CSS Guidelines Styleguide [27] with some exceptions such as using tabs instead of two spaces for indentation. Using tabs was decided because tabs are already used to indent Java code in RuM.

The colors of the user interface are largely based on MINERfulGUI. The main background color is white (#ffffff). The sidebar and header buttons use light blue (#00a6fb). Primary buttons and borders use black (#393939) and secondary buttons use green (#2ecc71). Primary and secondary buttons also have an "inverted" effect on hover where the colors of the button and the text are swapped (except for the button borders which remain unchanged on hover).

The application makes heavy use of borders to separate the different parts of the user interface visually. Two different colors are used for the borders. Primary borders use the same shade of black as the primary buttons (#393939) while secondary borders use grey (#aaaaaa).

4.3 Slide-in Panels

During the development of the new RuM user interface it became clear that some process mining tools require the possibility to set more parameters than can conveniently fit in a single view. An example of this is conformance checking where some methods allow for setting costs for all activities in the process model and in some cases also for setting costs for all attributes encountered in the event log.

One possibility to solve this would be to create separate configuration views and allowing the user to navigate to these configuration views by using tabs. This however would have introduced a second row of tabs into RuM (in addition to the tabs for opened files) and we decided against this approach.

The solution we came up with is the use of slide-in panels. A slide-in panel in the context of RuM means an extra configuration panel that slides into the view from the parameters column of the work area when the user presses the corresponding button. Each slide-in panel

[23] https://openjfx.io/javadoc/11/javafx.graphics/javafx/scene/doc-files/cssref.html

handles some specific functionality (for example setting the activity costs). An example of an opened slide-in panel can be seen in Figure 8.

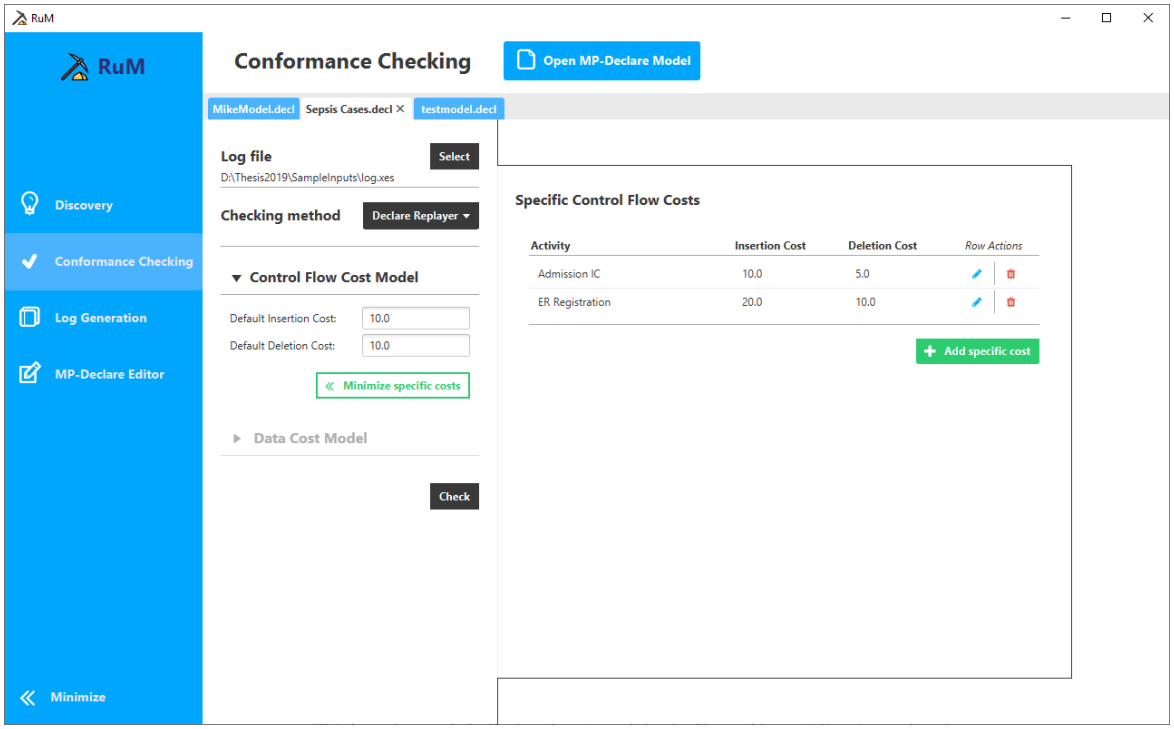


Figure 8. Opened slide-in panel in conformance checking for setting the activity (control flow) costs.

Slide-in panels do not use any existing JavaFX components. Instead, they are created by using a StackPane[24]. When a slide-in panel is opened then a new user interface element is placed on top of the StackPane with an offset on the x axis such that the new element is initially not visible. The slide-in effect is achieved by creating and playing a TranslateTransition[25] animation which moves the new element into its final visible position.

An alternative would have been to use the HiddenSidesPane[26] component from ControlsFX library, but this component did not work correctly in Scene Builder and therefore was not compatible with the rest of the application.

[24] https://openjfx.io/javadoc/11/javafx.graphics/javafx/scene/layout/StackPane.html
[25] https://openjfx.io/javadoc/11/javafx.graphics/javafx/animation/TranslateTransition.html
[26] https://controlsfx.bitbucket.io/org/controlsfx/control/HiddenSidesPane.html

4.4 Feedback messages

There are three main types of feedback messages used in RuM: busy layers, alert layers and popup windows. Examples of all three can be seen in Figure 9.

Busy layers are used to notify the user that a task is in progress. The busy layer does not give any indication of how far the task has progressed because that information is not reported by most of the process mining algorithms used in RuM. Alert layers are used to give user feedback that does not require immediate attention. Alert layers are connected with a specific tab and do not block off the rest of the application. Popup windows are used for immediate feedback and require the user to respond before continuing to use the rest of the application.

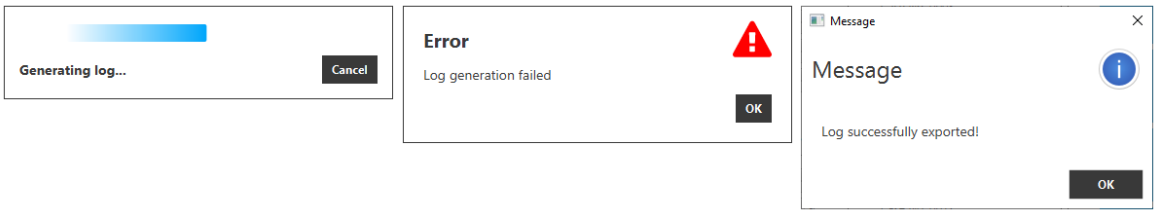


Figure 9. Examples of feedback messages in RuM. Message types in order from left to right: busy layer, alert layer, popup window.

The implementation for busy layers and alert layers is technically the same and very similar to how slide-in panels are implemented. When a layer needs to be displayed then it is placed on top of the corresponding StackPane and the rest of the StackPane content is disabled until the layer is removed. This approach allows for the rest of the application to remain usable even while the layer is visible.

The popup windows are implemented using a standard component of JavaFX named Alert[27].

4.5 Editable tables

There are a few data structures in RuM where using a table is the logical approach. In all these cases the data in the table must also be editable. During the development of RuM it became apparent that the builtin solution for editing table data in JavaFX is strictly "edit-by-cell" meaning that at most a single cell of a table can be in an editing state at any given time. This however causes problems in cases where row level validation is required (for example validating that target activity is not empty iff a binary template is selected).

The "edit-by-cell" restriction is not well documented, but it becomes apparent when looking at the source code of the TableView[28] component of JavaFX. There are a couple of workarounds proposed, but none of the solutions we found seemed suitable for RuM and therefore a custom solution was implemented.

We will not discuss this solution in detail here, but the main idea is to bypass the table editing capabilities in JavaFX and instead rely on the fact that basically any JavaFX component can be displayed in a table cell (including for example editable text fields). These components can be set to update automatically when the underlying data model changes (for

[27] https://openjfx.io/javadoc/11/javafx.controls/javafx/scene/control/Alert.html
[28] https://openjfx.io/javadoc/11/javafx.controls/javafx/scene/control/TableView.html

example, if some boolean attribute is set to true then show an editable text field, otherwise show a read-only label).

From the user's perspective a new row is almost always added in an editing state (the only exception being the voice input in the MP-Declare Editor). The user can fill cells of the row in any order and save or cancel the changes in the row using buttons located at the end of the row. When the changes in a row are cancelled then the row reverts to its last saved state or if there is no such state then the row will be deleted. If a row is not in its editing state, then editing can be started by either double-clicking the row or by using the pencil icon at the end of the row. It is also possible to remove a row that is not in its editing state by clicking on the corresponding button at the end of the row. An example of all three possible row states is given in Figure 10.

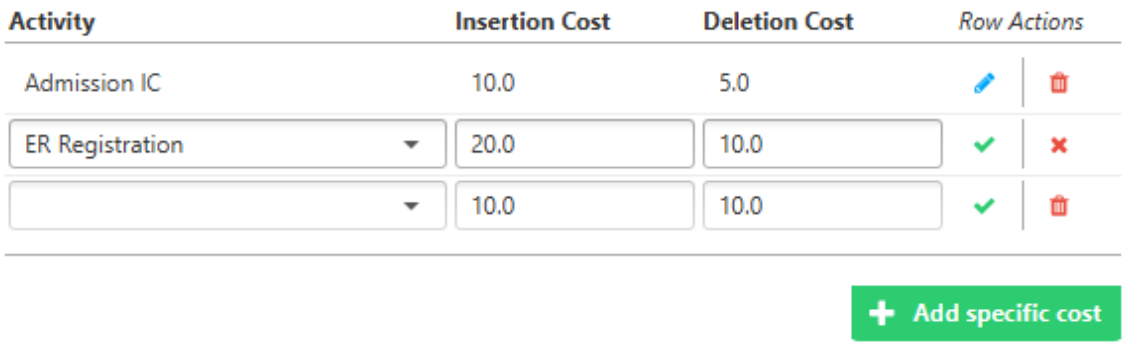


Figure 10. Three possible row states in RuM in order from top to bottom: saved row, previously saved row in its editing state, a new row in its editing state with default values.

Data validation for tables is done in three stages. Whenever a cell loses focus then the cell level validation is triggered which performs all validations that do not require information about any other cells in the table (for example checking the input format). If the user tries to save the row then a row-level validation is triggered, which performs all validation checks in the context of that single row (for example if the target activity of a binary constraint is selected). If the row is valid then the table level validation is triggered, which looks at the entire table to check if saving the current row does not cause a conflict with any other row (for example if the name of a new activity is unique). If all validation checks pass then the row is saved, otherwise the conflicting fields of the row will be highlighted with a red border.

5 Functional Overview

In this chapter we give a functional overview of the application in the following order: Discovery, Conformance Checking, MP-Declare Editor and Log Generation. For each section of the application we describe the user interface together with the main functionalities of that section. For each section of the application (except MP-Declare Editor) we also provide an overview of the available process mining methods and parameters.

The source code of the application is available in a public Bitbucket repository: https://bitbucket.org/doorless1634/thesis/commits/tag/Version_0.4.6. The compiled version of the application can be found in a publicly viewable Google Drive folder: https://drive.google.com/drive/folders/1qqGjGf1NX_ZhrXEF5CwEfPgVhx0pE60G?usp=sharing.

5.1 Discovery

The Discovery section of the application focuses on the process discovery branch of process mining. This section allows the user to open an event log and to use multiple different methods in order to discover a Declare model of the process.

Both input parameters and the results are presented in a single view. One slide-in panel is used to select the templates for the discovery task and multiple views of the results are provided. The functionality of the result views is the same regardless of the discovery method that was used. It is also possible to export the discovered model in the decl format, as a text file and as a dot graph. An example of the Discovery view is given in Figure 11.

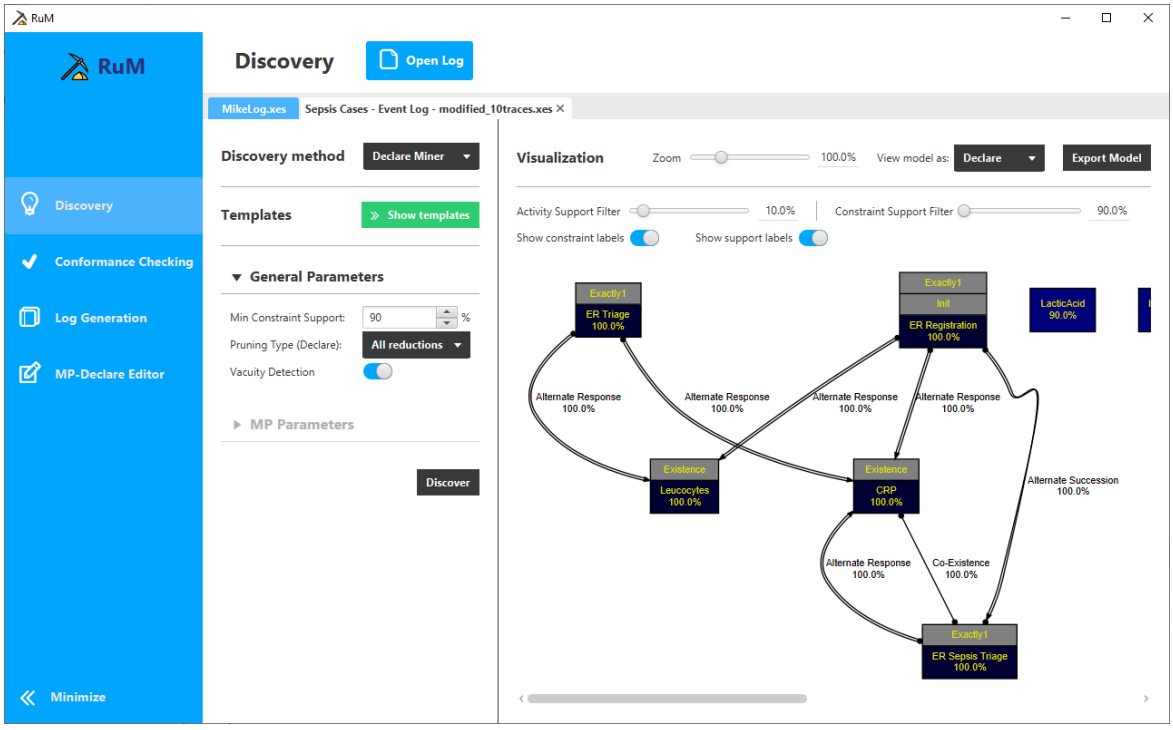


Figure 11. Discovery section showing discovery results with the default input parameters.

5.1.1 Available methods

There are four methods available for process discovery: Declare miner [28], Minerful [29], MP-Declare Miner [30] and MP-Minerful. The MP variants use the same base mining algorithm that is implied by the name, but with an added post processing step that aims to discover data conditions from the event log. Currently, the resulting data perspective is only shown in the textual view of the results.

With regards to MP-Minerful, it is important to note that this is not an already existing method, but a combination of Minerful and the same post processing step used in MP-Declare Miner. MP-Minerful was included because the output structures of Declare miner and Minerful were similar enough that the same post processing was possible to apply for both with very little extra effort.

The following general parameters are available for process mining:

- List of templates allows the user to define which templates have to be included in the final discovered model[29]. The templates that can be selected depend on the specific discovery model that is used.
- Minimum constraint support allows the user to set the percentage of traces that the constraint must be fulfilled in in order to be included in the output model.
- Pruning type allows the user to select which post processing method is used to remove redundant constraints from the discovered model[30]. The specific pruning types that can be selected depend on the selected discovery method.
- Vacuity detection can be used to detect and remove constraints that are vacuously satisfied. This parameter is available only for Declare Miner and MP-Declare miner.

5.1.2 Result views

The discovery results can be explored by using the following views: Declare, Textual and Automaton. All these views support filtering based on activity support and constraint support. The filters are applied on the fly without rediscovering the model. Each view shows the same process model but with a different visualization and an emphasis on different details.

Process discovery results are initially displayed in the Declare view by default. An example of the Declare view is already given in Figure 11. The main purpose of this view is to give a visual overview of the constraints in the discovered model. Each activity is represented as a single rectangle containing the activity name and support (percentage of traces in which the activity occurs). Unary constraints are displayed above the activity using the corresponding template name. Binary constraints are displayed as arcs between activities. Compared to RuM (2019) we have added binary constraint names and possibility to show or hide both the name and the support labels of the binary constraints.

The textual view is meant for users who are less familiar with the visual representation of Declare constraints. Textual view gives a description of the model using more natural sentences that aim to describe the meaning of each constraint. If the data perspective is included in the discovered model (MP-Declare Miner and MP-Minerful methods) then this

[29] Declare Miner and Minerful handle the input templates differently. Declare miner uses the input templates during the discovery process while Minerful first discovers the model using all supported templates and then filters out the constraints based on templates which are not in the input list.

[30] In the case of MP-Declare Miner and MP-Minerful, pruning is applied before discovering the data conditions.

will also be shown in the textual view. An example of the textual view is given in Figure 12.

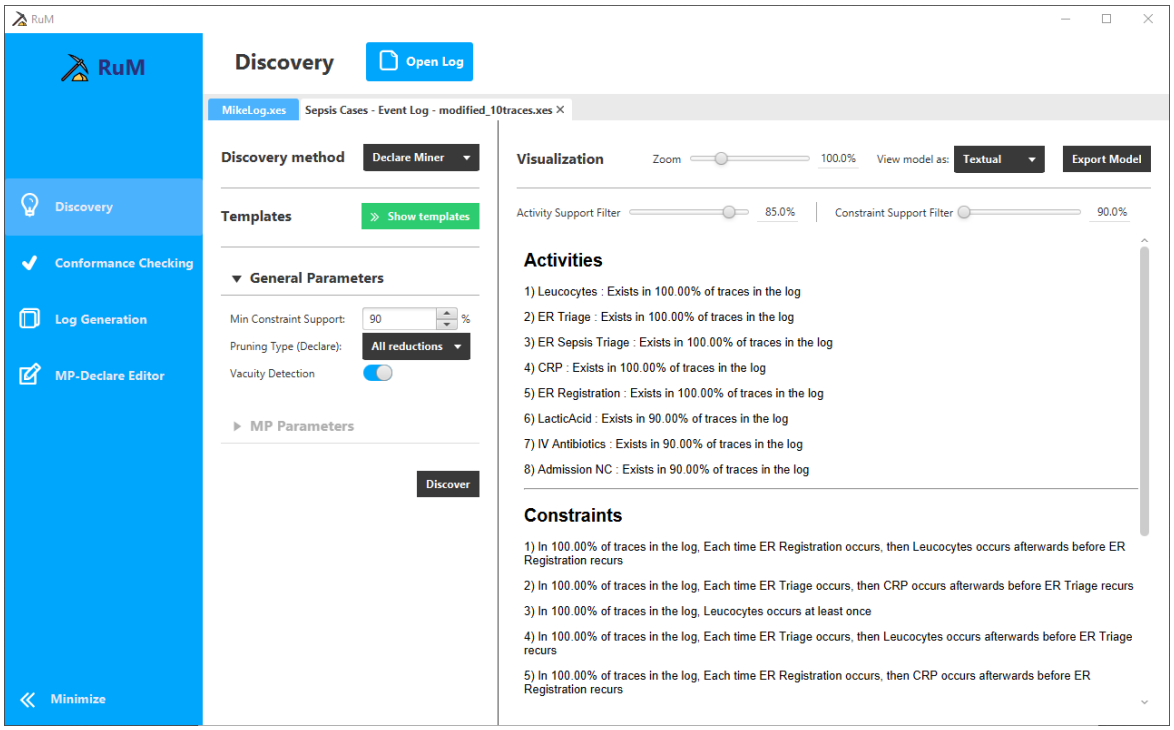


Figure 12. An example of the Textual view of a discovered process model.

The Automaton view displays the discovered process model as a finite-state machine. This view is meant for people who are either familiar with the formal semantics of Declare or with automatons in general. The automaton basically shows the states that a trace can go through in order to satisfy all the constraints contained in the model. If an activity specified as a label on an arc connecting two states in the automaton is encountered in the event log, then the corresponding state transition will happen in the automaton (taking into account the current state which the automaton is in). If the final state of the automaton upon reaching the end of the trace is also a final state of the automaton, then the trace satisfies the model.

Otherwise the trace contains one or multiple violations. An example of the Automaton view is given in Figure 13.

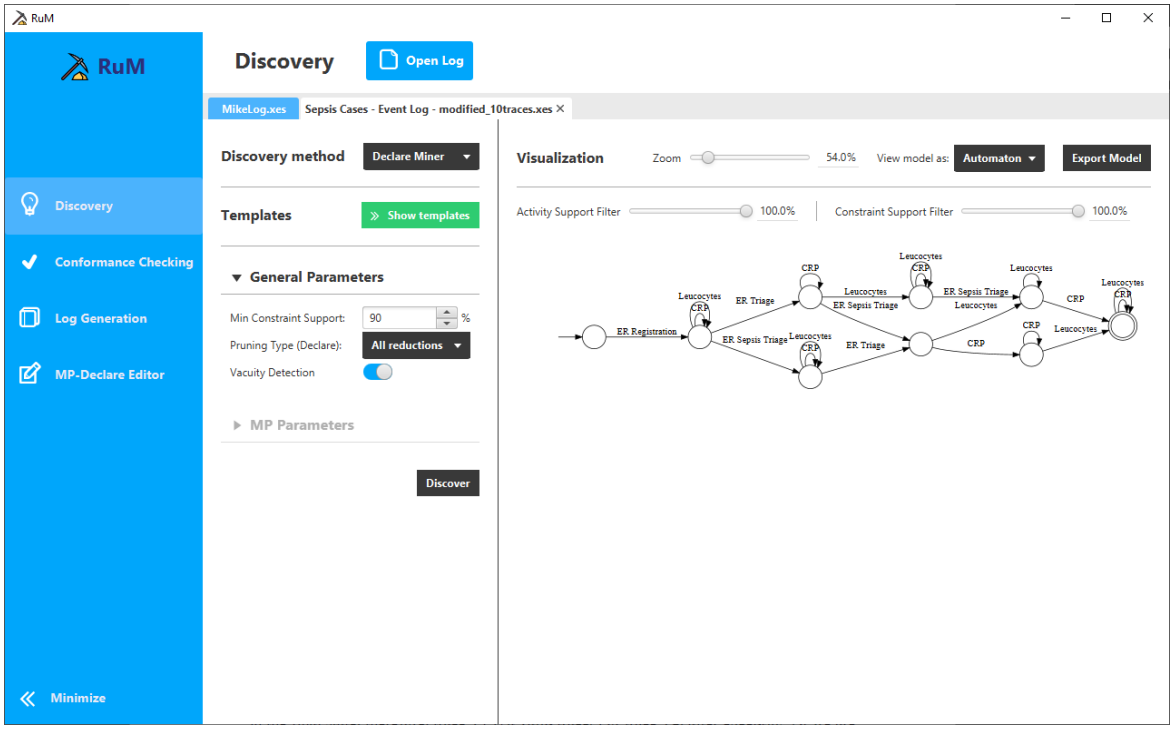


Figure 13. An example of the Automaton view of a discovered process model.

5.2 Conformance Checking

The Conformance Checking section of the application focuses on the conformance checking branch of process mining. This section allows the user to open a model and then to choose an event log in order to compare the two.

Both input parameters and the results are presented in a single view. Up to two slide-in panels are used depending on the selected conformance checking method. The structure of the results view is the same for all conformance checking methods however the specific details shown depend on the method (for example one method may report the number of

violations/fulfilments while another may report the fitness metric). An example of the Conformance Checking view is given in Figure 14.

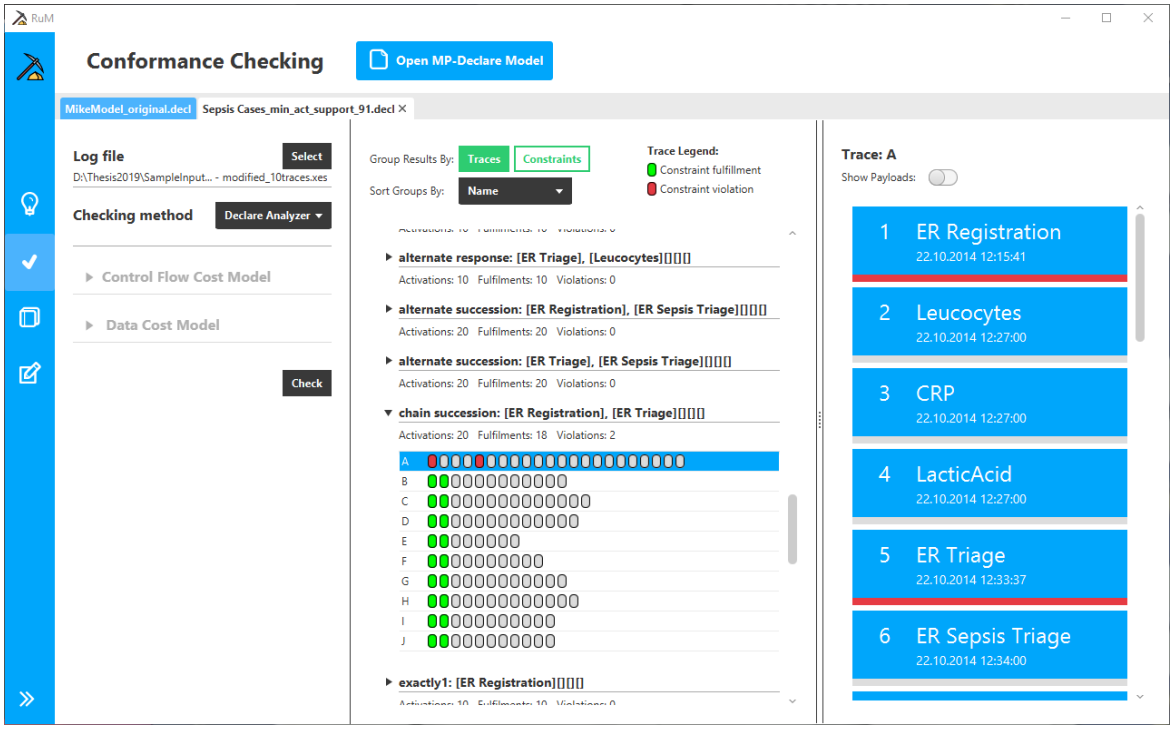


Figure 14. Conformance checking view with results using the default method (sidebar is minimized).

5.2.1 Available methods

There are three methods available for conformance checking: Declare Analyzer [23], Declare Replayer [31] and Data-Aware Declare Replayer [32].

The Declare Analyzer reports constraint activations, violations and fulfillments (which constraint is fulfilled or violated in which trace and by which event) and has no parameters other than the model and the event log.

The Declare Replayer and the Data-Aware Declare Replayer report trace alignments (which event should be inserted into or deleted from the trace to satisfy the model). The Data-Aware Declare Replayer can also account for the data perspective (what data values should activities have in order to conform to the model).

Parameters for both the Declare Replayer and the Data-Aware Declare Replayer affect how the final alignments will look like. It is possible to define the default cost for every insertion and deletion, but also for each activity specifically. In the case of the Data-Aware Declare Replayer, it is additionally possible to define costs for missing data values and faulty data values separately. For example, the user may define high costs for insertion and/or deletion of a specific activity so that the conformance checking algorithm would prefer inserting or deleting other activities instead.

5.2.2 Result views

The layout of the results view is the same for all three conformance checking methods. The results are always presented in groups where each group represents the results for a specific

trace or a specific constraint. The user can freely switch between which grouping is used without rerunning the conformance checker. For each group, the name of the group is displayed along with some statistics about the group. The statistics depend on the conformance checking method that was used.

Each group can be expanded to see the result details of that group. If the results are grouped by trace, then the details of a group will show how the corresponding trace is affected by each constraint in the model. If the results are grouped by constraint, then the details of a group will show how this specific constraint affects each trace in the event log. This allows users to explore the results at a high level of detail while also being relatively compact in terms of user interface.

It is also possible to see the events of a specific trace, by selecting the corresponding result detail row. For each event its sequence number, name and timestamp are displayed by default. The bottom borders of each event are colored based on the feedback given by the specific conformance checker. It is also possible to show the payloads (attributes) of each event as they appear in the event log.

If the conformance checking result is a trace alignment (Declare Replayer and Data-Aware Declare Replayer), then it is also possible to show or hide both the activities that are inserted into the trace or removed from the trace as a result of the alignment. An example of the results view displaying an alignment is shown in Figure 15.

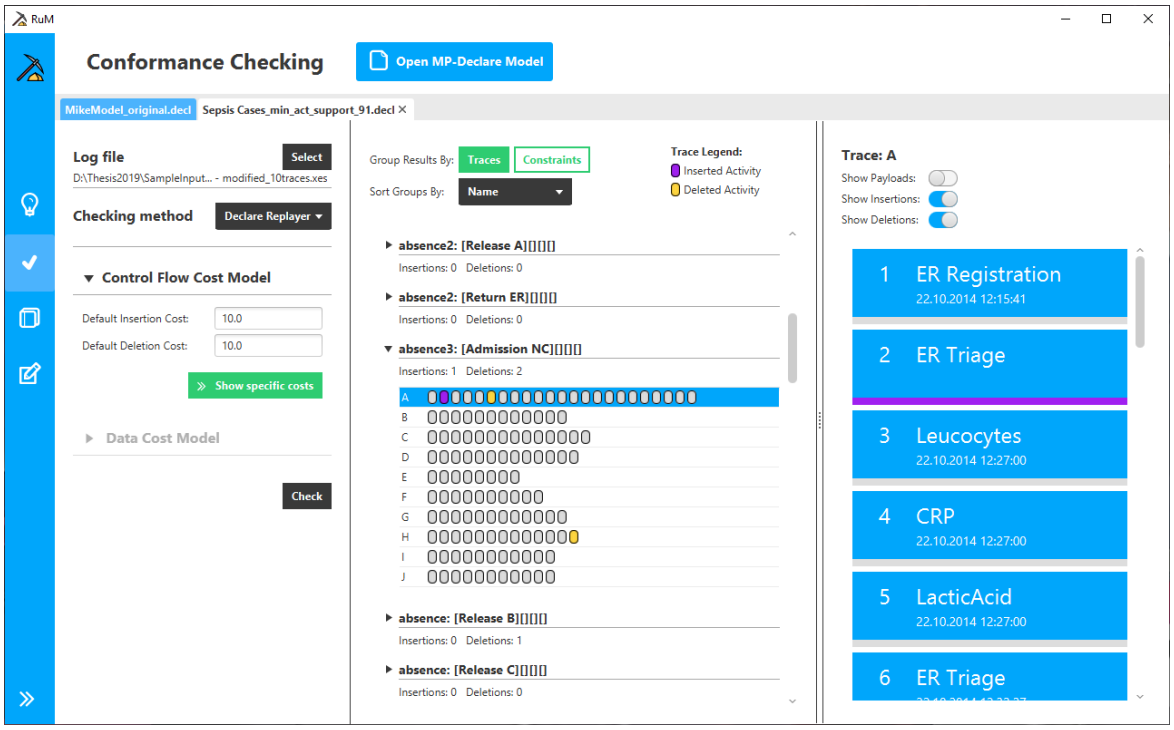


Figure 15. Conformance checking view with results using the Declare Replayer Method (sidebar is minimized).

5.3 MP-Declare Editor

The MP-Declare Editor section of the application focuses on the process enhancement branch of process mining[31]. This section allows the user to open and modify an existing process model or to create an entirely new model. It is possible to modify the model manually or to use voice recognition for entering constraints and data conditions.

The entire model editor is presented in a single view. Two slide-in panels are used, the first one for editing the activities and the second one for editing the attributes. Editing the constraints is done in a single table where each row corresponds to a single constraint. The entire model is also represented visually in the same view and the visualization is updated on the fly as the user is editing the model. An example of the MP-Declare Editor is given in Figure 16.

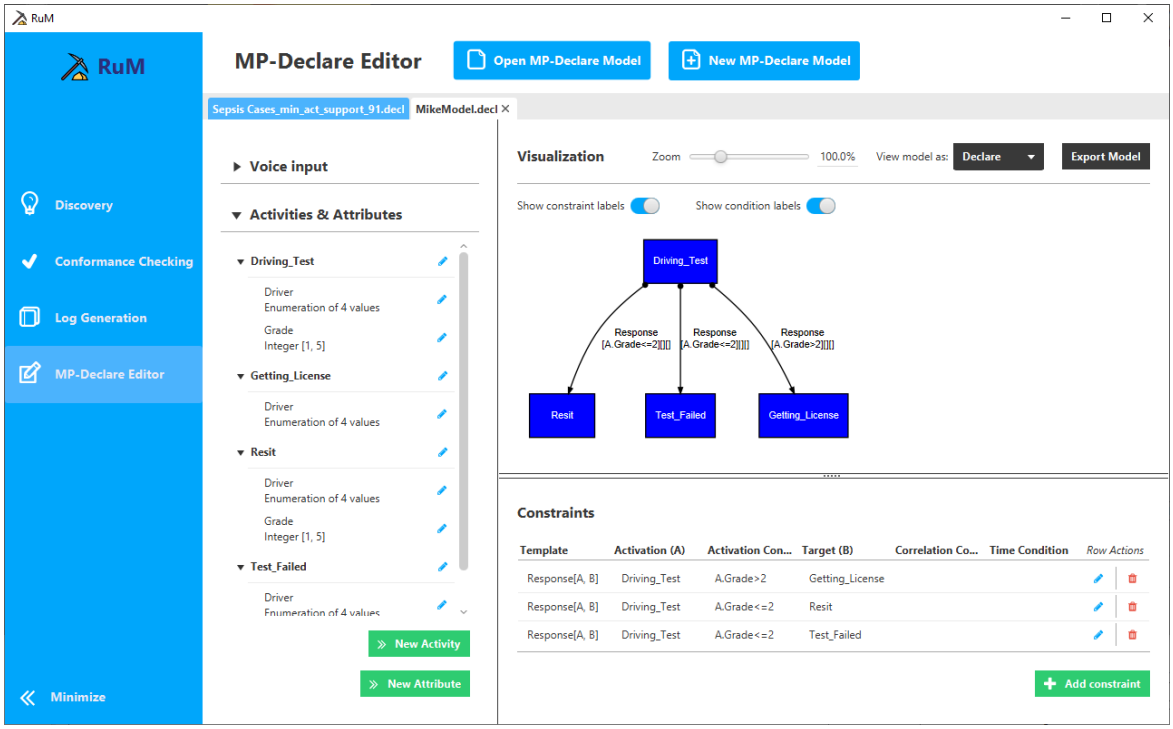


Figure 16. MP-Declare Editor section after opening a small process model.

The MP-Declare Editor uses the decl file format to open and export the models. All the aspects of the format are supported (activity definitions, attribute definitions, activity-attribute bindings and constraints with all of the allowed condition types).

5.3.1 Voice Input

The MP-Declare Editor supports entering declarative constraints by voice recognition. This means that the user can record a sentence which will then be processed by RuM and any constraints found in the sentence will be extracted and added to the model along with the related activities if the activities do not already exist in the model. The voice input

[31] It could be argued that the MP-Declare Editor is not really a process enhancement tool, because it only takes the process model as input and has no capabilities for processing event logs. However, RuM as a whole provides ways to gain information that can then be used to manually enhance the model and, therefore, we consider the editor to be suitable for process enhancement.

functionality is provided by Speech2RuM [33]. An example of the voice input is provided in Figure 17.

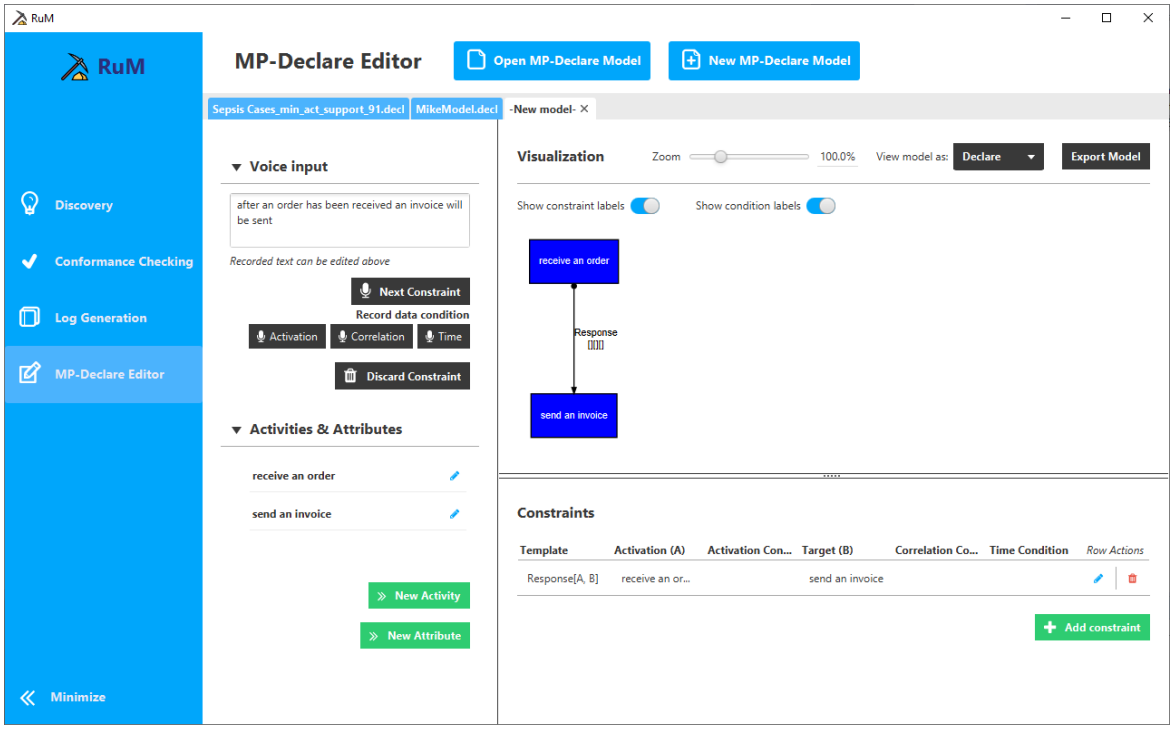


Figure 17. MP-Declare Editor section after opening a new model and entering one constraint by using voice input.

It is important to note that Speech2RuM was developed separately from this thesis and the integration of Speech2RuM into RuM was done in cooperation with the author of Speech2RuM. The main contribution of this thesis was the creation of the user interface and related controllers in RuM. It is also important to note that the author of Speech2RuM carried out a usability test that focused entirely on using voice recognition to create MP-Declare models in RuM and some of the feedback from that test was used for developing the user interface of RuM during this thesis.

Voice input can be used for both modifying existing MP-Declare models and creating new models. Furthermore, it is allowed to use a mix of both voice input and manual editing in the same view. For example, the user can input the first constraint by voice, the second manually, the third again by voice and so on. The main limitation of voice input in the current user interface is that only new constraints can be added by voice, while editing and removing constraints has to be done manually.

Voice input can be started by first expanding the voice input panel and then clicking the button “Record Constraint”[32]. The recording will stop automatically after a sentence has been detected by Speech2RuM and the resulting constraints and activities will be added to the model. After the recording is processed the user can modify the recorded text manually, record the next constraint or record additional data conditions. Modifying the recorded text changes all the constraints and activities related to that specific recording and recording a

[32] The environment variable GOOGLE_APPLICATION_CREDENTIALS must be set beforehand as described in the Speech2RuM readme file.

data condition will add the condition to all the constraints extracted from the current constraint recording.

Voice input also supports limited capabilities for discarding the recording results. It is possible to discard previous recordings up to and including the last recorded constraint. However no overall undo/redo functionality is currently provided.

5.3.2 Entering Activities and Attributes Manually

Activities and attributes can be added manually using the corresponding slide-in panels. It was decided to use the slide-in panels here because attribute definitions require more information than just the name of the attribute and also because the slide-in panels have enough room to conveniently manage activity-attribute bindings. An example of these slide-in panels is shown in Figure 18.

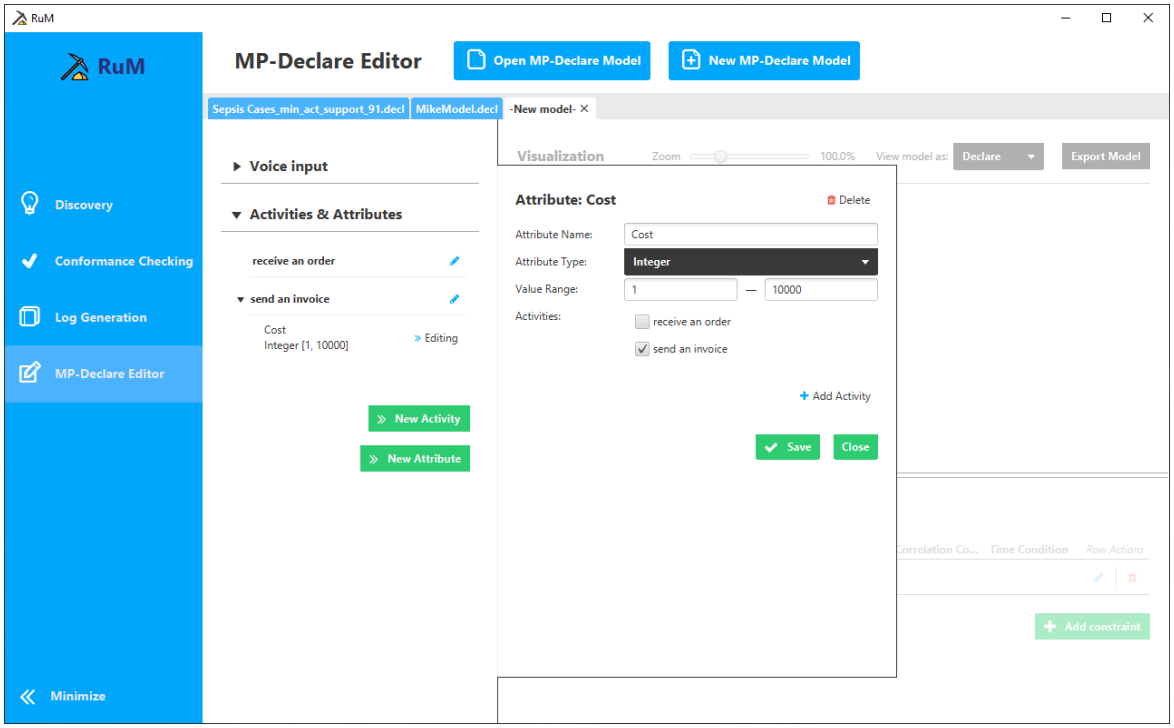


Figure 18. Attribute editing slide-in panel in the MP-Declare Editor.

In order to add a new activity, the user must click the button “New Activity”. This opens the slide-in panel which allows the user to enter the activity name and optionally to define the attributes associated with that activity. The same panel is also used for editing and removing existing activities.

New attributes can be added either through editing an activity or by clicking the button “New Attribute”. The main difference between these two possibilities is that, in the first case, the attribute will be added only to the activity that is being edited, while, in the second case, it is possible to add the attribute to any of the existing activities. In both cases, the information needed for defining an attribute is exactly the same.

The following information is needed in order to define an attribute:

- Attribute name that distinguishes this attribute from all the other attributes.
- Attribute type that can be either Integer, Float or an Enumeration.

- In the case of Integer and Float attributes, it is required to define the value range of the attribute. The bounds of the value range are inclusive.
- In the case of Enumeration attributes, all the possible values of the attribute must be defined.

When working with attributes, it is important to note that no orphaned attributes are allowed by the MP-Declare Editor. Orphaned attributes, in this context, are attributes that are not related to any activity. If an orphaned attribute is created (by for example removing the only activity which the attribute is related to) then that attribute will be removed from the model.

Activity and attribute names in the model must be unique.

5.3.3 Entering Constraints Manually

New constraints can be manually added and existing ones manually modified through the constraints table located at the bottom of the editor. The constraint table is linked to the activities in the model such that only the activities defined in the model can be used for defining a constraint. This has the side effect that in order to successfully add a new constraint at least one activity must be defined in the editor and if an activity is modified then the same modification will also apply to all related constraints. The behavior of the constraint table is otherwise identical to the one of the other tables used throughout the application.

The following information can be entered in order to define a constraint:

- Constraint template is required for every constraint. It can be any of the templates supported by RuM.
- Activation activity is required for every constraint. It can be any of the activities already defined in the same model.
- Activation condition can be defined for every constraint. The condition is validated using the validation capabilities of AlloyLogGenerator.
- Target activity must be defined for every constraint that uses a binary template. In the case of unary templates, the target activity must be empty. The target activity can be any of the activities already defined in the same model.
- Correlation condition can be defined for every constraint that uses a binary template. In the case of unary templates, the correlation condition must be empty. The condition is validated using the validation capabilities of AlloyLogGenerator.
- Time condition can be defined for every constraint. The condition must match the following regular expression “^\\d+,\\d+,[s,m,h,d]$”.

The constraints in the model must be unique.

5.3.4 Model Visualizations

The MP-Declare Editor uses similar visualizations for the models that are used in the Discovery section of the application to view the discovery results. There are only a few minor differences because of the functionality differences between the two sections. For example, the activity support is not shown and all activity rectangles are colored uniformly because the activity support metric is not applicable in the case of the MP-Declare Editor.

The same three visualizations can be used that are also available in the Discovery section: Declare view, Textual view and Automaton view. The main purpose and functionality of all three visualizations is very similar and therefore it would be redundant to describe them again here. This also applies to exporting the model. However, it is important to point out that all visualizations are updated on the fly while the model is being edited so it is possible,

for example, to see how the addition of a new constraint changes the Automaton visualization of the overall model.

5.4 Log generation

In addition to supporting the three main process mining branches, RuM also includes support for generating artificial event logs. This can be useful for testing out new process mining algorithms or gaining a better understanding of the process behaviors allowed by a certain model.

Both the input parameters and the generated log are presented in a single view and no additional slide-in panels are used. An example of the Log Generation view is shown in Figure 19.

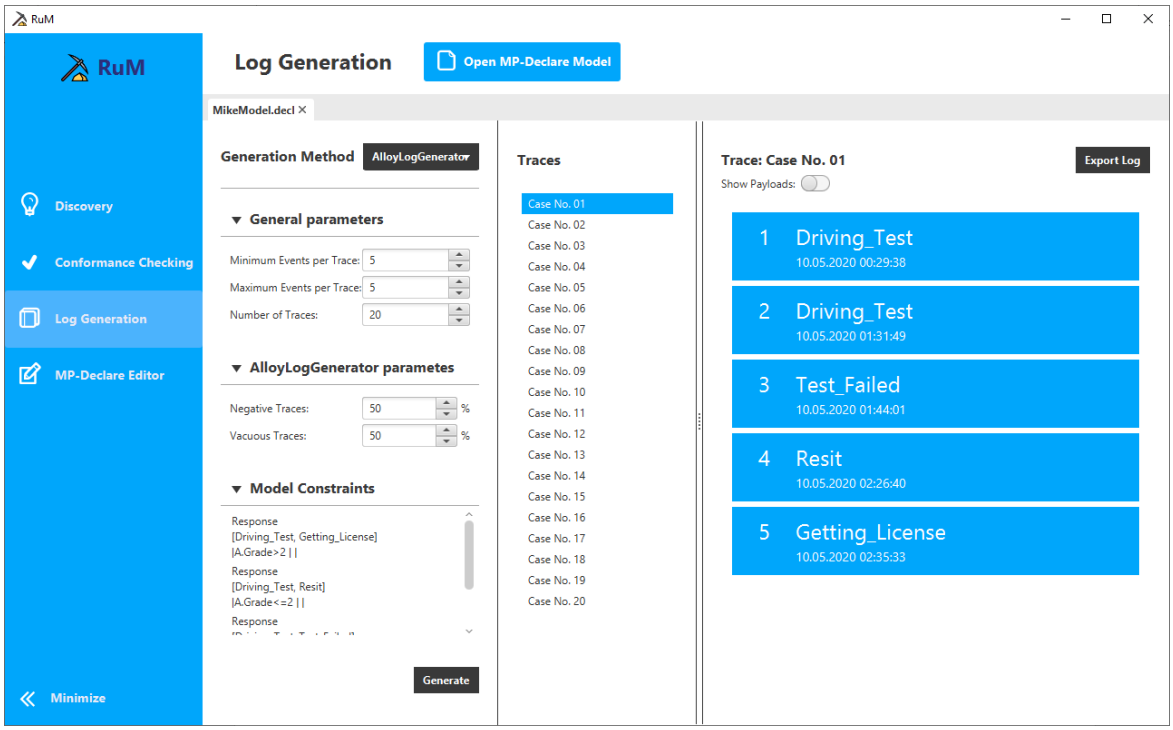


Figure 19. Log Generation section showing a generated event log.

5.4.1 Available methods

There are two log generation methods available: AlloyLogGenerator [24] and MinerFul Log Generator [34]. The main difference between these methods from the user's standpoint is that the AlloyLogGenerator is also able to account for the activation and correlation conditions in the input process model. This is also the reason why AlloyLogGenerator is the only available method if any data conditions are present in the model.

There are also major algorithmic differences between the two methods, but these differences will not be discussed in this thesis.

The following parameters are available for both log generation methods:

- Minimum events per trace defines the minimum number of events that each trace must have.
- Maximum events per trace defines the maximum number of events that each trace can have.

- Number of traces defines the total number of traces in the generated event log.

In case of AlloyLogGenerator there are two additional parameters available:

- The percentage of negative traces defines the proportion of traces in the generated event log that may violate the model in some way.
- The percentage of vacuous traces defines the number of traces in the generated event log that may be vacuously satisfied (traces that do not activate any of the constraints in the model).

5.4.2 Result view

The results of the log generation are displayed in two columns (as already shown in Figure 19). The first column shows the names of the traces in the generated log and allows for selecting a specific trace. The second column shows the events of the selected trace. Trace events are shown using the same user interface elements as in the conformance checking section. It is also possible to view the attributes (payloads) of the generated events.

6 User Evaluation

In order to evaluate the new user interface of RuM, a qualitative user evaluation study was conducted. In this chapter, we give an overview of the study methodology and an overview of the study results. The results of the study will be described in two parts. First, we describe the results of a survey that each study participant filled after using the application. This will give a general overview of how the application was perceived by study participants. After the survey results, we will describe the study findings in more detail.

6.1 Study Methodology

The aim of the study was to (1) gain insights into how the new user interface and also RuM in general is perceived by potential users, (2) determine potential areas of the application that the users may find difficult or overly complicated and (3) find potential ways to improve the user interface. The performance of the participants was not measured in quantitative terms (task-completion times, success rates, number of errors, etc.), instead, the focus was on gathering qualitative data (which aspects of the user interface design are problematic and which work well).

Eight participants were selected for the study all of whom were business process mining experts with academic backgrounds from various universities across Europe. Four of the participants had little to no knowledge of Declare, while the other four can be considered Declare experts. This split based on participants' knowledge of Declare was necessary because RuM is aimed to be both a useful application for experts while also being a starting point for users who are not familiar with Declare.

The user evaluation study consisted of the following main steps for each participant:

1. An introductory email was sent to the participant before the beginning of the test explaining the purpose of the study and providing a link to all the files needed for the study (including the consent form, the application itself, the task list for the study and the input files for the tasks). A redacted example of the introductory email can be found in Appendix I and the consent form can be found in Appendix II.
2. A Skype call was started with the participant[33].
3. The study was introduced to the participant at the start of the test along with the consent form and a verbal consent was asked from the participant.
4. The application was introduced to the participant with a short demonstration. The demonstration was limited in scope. Only the navigation of the main sections and the section headers were introduced, but the work areas of the sections were not demonstrated.
5. The participant was asked to start the application on their computer[34] and to share their screen.
6. The recording of the call was started and the participant was notified of the recording.
7. The participant was asked to carry out the tasks based on the task list we had provided earlier. The task list can be found in Appendix III.

[33] In every call there was at least one observer in addition to the facilitator. The facilitator was responsible for guiding the participant through the application. The observers served a supporting role and intervened in case of more technical questions about the process mining algorithms used in RuM.

[34] In some cases, the participants were either unable to run the application on their computer or did not have time to set up the application for the test. In these cases, we provided the participant access to a separate computer which had RuM installed. The access was provided via the remote desktop software TeamViewer.

8. The participants were encouraged to think out loud, to ask questions and to point out interesting aspects of the application during the test. If the participant had difficulty with some specific task, then first some guiding hints were provided to the participant. If the hints were not sufficient then the solution of the task was explained.
9. After finishing the task list, we continued with a short post-interview. The participant was asked some questions about the application in general and about the tasks where the participant had the most difficulty. The recording of the test was stopped after the post-interview was concluded.
10. The participant was sent a link to a survey which measured the participant perception of and experience with RuM.

The tasks covered all four sections of the application in the following order: Discovery, Conformance Checking, MP-Declare Editor and log generation. The inputs for the tasks were based on the Sepsis Cases [35] event log. However, the event log was modified (some events were changed and the number of traces was significantly reduced). The overarching goal of the tasks was to create an artificial dataset of sepsis cases that is representative of the actual real-life treatment process.

6.2 Survey Overview and Results

A short survey was created and sent to all the participants of the user evaluation. The aim of this survey was to measure the participants perception of and experience with RuM in general terms.

The study was conducted anonymously to allow the participants to answer as truthfully as possible and because we did not plan to connect the answers to specific participants. This, however, means that the survey results cannot be presented according to the two groups of participants originally identified (BPM experts and Declare experts). Instead the participants are divided into two groups based on whether or not they have had any previous experience with RuM[35] as reported by the participants in the same survey. It is important to point out that only some (and not all) of the Declare experts had used the previous version of RuM and the same applies for the BPM experts.

[35] Previous experience in this context means that the participant either had used RuM (2019) before or had participated in the user evaluation of Speech2RuM.

The survey consisted of the following scales: System Usability Scale [36], satisfaction [37], expectation confirmation [37], future use intentions [37] and usefulness [37]. The full survey is added to this thesis as an additional file (Appendix IV file "post-survey.pdf"). The results for each scale are shown in Table 4.

Table 4. Survey results as averages for all included scales overall and grouped based on previous experience using RuM.

	All participants	**Participants with some experience using RuM**	**Participants with no experience using RuM**
System Usability Scale	81.875	76.25	87.5
Satisfaction	4.5	4.25	4.75
Expectation confirmation	4.55(5)	4.5	4.66(6)
Future use intentions	4.166(6)	4.33(3)	4
Usefulness	4.3125	4.5	4.125

The average system usability scale score is estimated to be 69.69. The products that are at least passable should have a score above 70, while everything below 70 is considered to be pointing towards usability issues. Good products are expected to have a score in the high 70s to upper 80s and truly superior products are expected to score better than 90 points [38].

Given the overall average system usability score of 81.875, we can say that the new user interface of RuM was received well and the various design choices made during the development were proven to be justified. However, there is a significant difference between the two groups of the survey. The participants who had no previous experience using RuM rated the application significantly higher than the other group. The system usability scale does not give information about what changes should be made to the application or which parts specifically need improvement so we cannot give a definitive explanation to this difference, however, one possible explanation may be differences in expectations.

The other usability scales (satisfaction, expectation confirmation, future use intentions, usefulness) also have relatively high average scores which further confirms the relatively high system usability score. It is interesting to note that participants who had previously used RuM were less satisfied in terms of experience with respect to the other group, while, on the other hand, they regarded the tool to bey more useful and were more likely to continue using it, with respect to people who did not have any experience with the tool. The expectation confirmation score was also slightly lower for participants who had previous experience with RuM, thus indicating that their expectations of RuM were indeed higher.

6.3 Results of Usability Tests

The recordings of the tests were fully reviewed in order to gain as much insight as possible from the user evaluation as a whole. During the review process all the details of the user's interaction with the application that the reviewer deemed relevant were written into notes. Each note consisted of the participants code, name of the application section that the note

was related to and a short description. The notes were created using diagrams.net[36]. Examples of notes are shown in Figure 20.

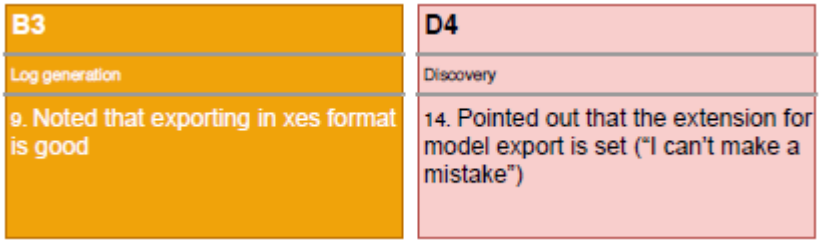


Figure 20. Two test recording notes showing participants feedback while exporting a result file.

Reviewing a single recording took about 1,5 to 2 hours and resulted in approximately 68 notes on average per recording. The total number of notes across all recordings was approximately 540.

Codes for the participants were assigned randomly, but a distinction was made between the participants with little to no knowledge of Declare (codes starting with B) and the participants who can be considered Declare experts (codes starting with D).

All the above mentioned notes were analyzed by creating an affinity diagram [39] in which all the notes were organized into clusters based on common functionality and themes. During the clustering some of the notes were split into two or duplicated if the same note belonged in multiple clusters. This resulted in 10 main clusters based mainly on functionality and a total of 111 subclusters based mostly on common themes. The full affinity diagram is added to this thesis as an additional file (Appendix IV file "affinity-diagram.pdf").

6.3.1 Base User Interface Elements

In this chapter, we describe some of the main findings and feedback related to the base user interface elements of RuM. By base user interface elements, we mean elements or components of the user interface that are specific of RuM, but are not tied to the functionality of any specific section of the application.

Slide-in panels are used in multiple sections across the application in order to allow the user to edit parameters that either do not need to be visible all the time or require too much room otherwise. In general, the slide-in panels were well received (*"the way it's displayed is quite nice"*, B3). Participants had very little issues when opening the panels, but the panels were not closed in multiple cases (B3, B4, D2) and in one case the participant needed help closing the panel (D4). Multiple participants noted that they would expect a specific confirm button for closing the slide-in panel (B1, D3, D4) with B1 stating *"minimize is not the same as save"*.

During the tests, we paid close attention to how participants interacted with tables, because the tables are created using a custom implementation. No bugs related to tables were discovered and participants in general did not have many issues when editing table rows. B3 pointed out that there are *"not many alternative ways how rows could be edited"* and D4 pointed out that the row editing pattern used in RuM is not uncommon in BPM tools. One notable exception is B2 who first entered the row editing mode by double clicking the row *("I want to first see it selected and only then I assume that all the buttons like edit, delete*

[36] https://www.diagrams.net/

work”, B2) and then found it difficult to delete the row, because row deletion is not available while row is in the editing state.

In all sections, except Conformance Checking it is possible to export the results. Exporting is handled in the same way in all cases. While the process of exporting itself was not an issue, the participants had difficulties when the task list did not specifically mention exporting. Not mentioning exporting in some of the tasks was a deliberate choice on our part when designing the task list. For most of the participants, it was unclear that the results can be saved only by exporting (B1, B4, D2, D3, D4). Some participants also expected the application to have quick ways to use the results from one section of the application as an input for another section (B2, D4) with D4 stating *“it’s a bit strange that you have to export the model and then reopen it, the same one, in another tab”.*

Navigation between the different sections of the application did not cause any issues during the study. B2 and D3 even pointed out that they like the ability to quickly switch between the sections (*“So I can switch back and forth between the views, that’s nice”*, D3).

All participants were shown at the start of the test that the sidebar can be minimized, however only D1 used that possibility and most likely only because D1 had limited screen space available during the test.

6.3.2 Model Views

In this chapter, we describe some of the main findings and feedback related to the three model views (Declare, Textual and Automaton) used in RuM. While most of the feedback was received during the discovery task, the same feedback is mostly relevant also for the MP-Declare Editor.

The Declare view was overall well received by the participants, but it is interesting to note that the Declare experts provided significantly more feedback on that view (16 notes from Declare experts compared to 8 notes from the other participants). In relation to the declare view D2 specifically pointed out *“I like the way existence constraint is shown”* and *“it’s a bit more intuitive than in all the papers”*. Some participants (B3, B4, D4) did not initially notice the toggles that can be used to show or hide constraint name and support labels, but in general this functionality was considered useful for example to explain the notation (D3) or to have a more compact visualization of more complicated models (B1).

The main suggestions and issues with the Declare view were related to either navigating or manipulating the view itself. For example, D2 wanted to use left-click to drag the view and also expected to be able to move the elements of the view manually. D1 expected the entire model to be visible given the space available and B2 suggested adding a “fit to window” zoom functionality.

The Textual view was considered by most of the participants as a good addition to RuM. The initial reaction of D3 for example was *“this is nice”* and D4 was especially impressed *“Finally a textual version, I like it”*. D1 noted that the Textual view is *“nice as an explanation of the model”*, but also commented that *“it is not an alternative to the Declare view”*. One major exception here was B2 who’s first reaction after opening the textual view was *“Oofff... [pause] ok”*. B2 later added *“I would never use textual view to get an overview of the process”*.

There were only a few suggestions about the textual view. D1 pointed out that the textual view uses a *“different lexicon”* than the Declare view and D2 pointed out that it is difficult to connect the constraints shown in the textual view to the constraints shown in the Declare view.

The Automaton view received the least feedback out of all three views most likely because it requires a bit more technical knowledge than the other views. For example, D4 assumed that the automaton is *"for theoretical people"* and B3 was unsure about what the automaton actually represented.

The main area of improvement for the Automaton view is the layout of the visualization. D1 and D2 had difficulty following the visualization because the incoming state is not always in the top left part of the visualization (*"visualization should flow from left to right"*, D1). Furthermore, it was pointed out by D1 that the layout sometimes changes when refreshing the view.

6.3.3 Discovery

In this chapter, we describe some of the main findings related specifically to the Discovery section of the application. The chapter mainly focuses on setting the parameters for the discovery. However, a few aspects about the Declare view and export will also be mentioned.

During the development of RuM, we decided that when an event log is opened in the discovery section then the model will be automatically discovered using the default parameters. However, during the tasks, we asked the participants to use a different set of parameters. This caused some confusion for some of the participants (B1, B2, B4), since they assumed that changing the input parameters would automatically change the model. Interestingly, none of Declare experts had the same issue, with only D3 pointing out *"It's interesting it immediately starts to discover something before I could set the parameters"*.

In the Discovery section, it is also possible to filter the activities and constraints shown in the model. Changes in the filters are applied automatically and do not require rediscovering the model. This may be part of the reason why some participants did not rediscover the model after changing the discovery parameters. For example, after being asked to rediscover the model B2 explained *"I did not do that because the map responded to some of the things I changed, namely when I adapted the sliders"*.

During the discovery task, we asked the participants to select a specific set of templates for the discovery. It took a bit of time for most of the participants (B1, B2, B3, B4, D1) to find the correct templates from the list shown in RuM. While the templates are ordered, the ordering was not apparent to the participants (*"ok, so they are not alphabetically sorted"*, D1). The ordering is based on template categories beginning with unary templates, followed by positive binary templates and then negative binary templates. This order was discussed for example with B4 in the post-interview and B4 suggested adding category headers (*"if there was some kind of brief headings there, then it would have been easier to quickly categorize it"*, B4). It was also pointed out that the high number of templates might be difficult to use (B3) and that the template meanings are not explained in the user interface (B2).

The export of the discovery results takes into account the currently opened model view and also the support filters. This means that if the user wants to export a model in decl format then the Declare view must be selected and only the parts of the model that are not filtered out with the support sliders are exported. This came up only with three participants (B1, B2, B3), since all the other participants were already in the correct view and had set the correct filters when exporting. B1 assumed correctly that filters affect the exporting and set the correct values without help. However, B2 and B3 did not make that assumption and needed help to export the correct model. This can point to a problem with the user interface, but neither B2 nor B3 mentioned it as a problem in the post-interview.

6.3.4 Conformance Checking

In this chapter, we describe some of the main findings related specifically to the Conformance Checking section of the application.

The Conformance Checking section received the most feedback (27 clusters of notes in the Affinity diagram compared to for example only 8 clusters of notes related to Discovery). This was somewhat expected due to the complexity of the result view and the amount of information shown. While the conformance checking section does not need a total rework, multiple areas of improvement were identified during the user evaluation.

One of the main differences between the Conformance Checking section and the other sections of the application is the fact that conformance checking uses two input files (the model and the event log). This was solved in the user interface by treating the model file as the main file of the section and the event log file as a parameter. We were unsure if this would become a problem for the users or not. During the evaluation it turned out not to be a problem. Only two participants (B2 and D2) paused for a moment before finding how to select the event log.

The results of the conformance checking are grouped by traces by default such that one group per trace is displayed and the group name is the trace name as it appears in the event log without any additional indicators. In the event log that we used for the tests the traces were named as A, B, C, etc. This was a major source of confusion in most of the tests (*"I'm not sure what these letters mean"*, B1). When asked to guess then B1 for example assumed that these names refer to activities, while B2 assumed these names represent constraints and D3 assumed they are quality metrics. A common suggestion by the participants was to add a prefix like "Trace:" or "Case ID:" (B1, B4, D3).

The details of each group are displayed in a table where each row shows a single trace in the context of the corresponding trace and constraint. The events of that trace are displayed as small "bubbles" such that each bubble represents a single event in the trace. The meaning of these bubbles was at first confusing to many participants (B3, B4, D4). For example, D4 noted, after exploring the view for a bit, that *"the green are the constraint fulfillment, I don't know what are the grey ones"*.

It is also possible to see the events of each trace while exploring the conformance checking results. This however came as a surprise to some participants (B1, B3, B4). When asked about it then, for example, B1 noted that it is *"maybe not intuitive"* and suggested either adding a "show details" button or selecting the first trace by default when a group is opened.

One of the tasks that the participants had the most difficulty with was showing the original trace and the aligned trace after performing conformance checking with the Declare Replayer. Part of the reason was that the user interface of RuM and the wording of the task did not match, but also some issues with the user interface itself were identified. Half of the participants (B4, D1, D2, D3) needed help in order to figure out that the show insertions and deletions toggles can be used to see the original and the aligned trace. Furthermore, some participants had to pause and think about how the toggles must be set in order to see the original trace (B4, D3) with D3 also pointing out that *"showing the deletion is like a negation of a negation"*. Another source of confusion related to this functionality was that the sequence number of a specific event changes if any of the previous events in the trace are hidden (*"The numbering is changing"*, D1).

Despite the relatively high number of notes gathered about the Conformance Checking section the general perception was still more positive than negative. For example, D2 commented that *"it is presented in a pleasing way"* and commended the level of details in

the result view while D3 summarized the conformance checking task with *"I like this conformance checking, well done!"*.

6.3.5 MP-Declare Editor

In this chapter, we describe some of the main findings related to the MP-Declare Editor section of the application. The focus of this chapter is mainly on editing activities and constraints. However, a few specific aspects of the Declare view will also be mentioned.

During the model editing task a clear difference in the preferred editing styles became apparent. Most of the Declare experts (D2, D3, D4) tried initially to edit the model through the Declare view (*"because it's called an editor you would kinda expect that everything you see, especially the thing you see in the main is the thing you are editing"*, D3). In relation to this, it was also suggested by D2 and D3 that clicking on the visualization should highlight the corresponding row in the constraint table. Two participants pointed out (B1, B3) that placing the constraint table below in the visualization instead than on the left is a bit confusing because this part of the user interface is used only for viewing in the other sections of the application.

Some participants also had issues while saving an attribute. B1 added the attribute correctly but closed the panel without saving and the attribute was lost. B3 almost missed saving the attribute, but did not need help, while B4 saved the attribute only when the facilitator started to point it out. B4 later said that having the button "Done" for the attribute and then the button "Save" for the activity was confusing because both buttons use the same checkmark icon.

In relation to adding attributes, it is interesting to note that an attribute can be added in two ways. Either by editing an activity or by clicking the button "New attribute" below the list of activities. While the latter seems to be more explicit, all the participants added the attribute through editing an activity.

The main source of confusion when editing the constraints was the fact that in the case of precedence templates the activities are presented in "reverse" order compared to other templates (target activity before the activation activity). This created a situation where multiple participants were unsure if they were removing the correct template (B4, D2, D3), because the same order was not used in the conformance checking results. On the one hand, this points to a clear issue in the conformance checking result view. But, on the other hand, it also shows that changing the order of A and B in the parameters of the precedence templates in the editor is possibly not enough to point out that in the case of precedence templates activation and target appears in a different order with respect to the other templates (*"if I am not very attentive to how it's expressed I'm just going to assume it's the same flow"*, B4).

Editing the activation condition also caused some questions related to the syntax of the condition. D1 and D2 wanted to use an equal sign instead of the word "is" while B3 commented while entering the condition *"I hope this is the way to write it … is it so English like?"*. A couple of suggestions were given related to the activation condition. D2 suggested adding a help button that would describe the syntax and D3 suggested that attributes in the model should be recognized by the editor while typing the activation condition and suggested implementing an auto-completion mechanism for that.

6.3.6 Log Generation

In this chapter, we describe some of the main findings related to the Log Generation section of the application.

The Log Generation section received the least feedback (5 clusters of notes in the Affinity diagram compared to for example 8 clusters of notes related to Discovery). This was somewhat expected since this section also has the least amount of functionality.

Three of the participants noted that only one method is available (B2, B3, D3). This is because the model used to generate the log contained data conditions so that only AlloyLogGenerator could be used.

The Log Generation section also contains an overview of the constraints that are in the process model. Multiple participants expected that this overview is somehow functional and can be used either to affect the log generation (B1, B3, D2) or to check the generated log somehow (B1, D4). Part of the reason can be that this area of the user interface is reserved for parameters in other sections of the application.

6.3.7 Terminology issues

In this chapter, we describe some of the terminology issues that we encountered during the tests.

The most prominent issue was related to the term "trace". Some of the participants saw that term as having a different meaning than is used in RuM and would prefer to use the term "case" instead (*"Almost at every conference again I have at least one time the discussion what is for you a trace and what is for me a trace"*, B2).

Multiple participants were unsure what the term "vacuous" refers to (B1, B2, D3). While the word "vacuous" does exist in the English language with basically the same meaning in RuM, it is quite clear that this word is uncommonly used and therefore foreign to many potential users.

One of the participants (D3) was unsure what the percentage of negative traces actually means. After explaining the parameter, it was suggested by the participant that either "non-conforming traces" or "noisy traces" should be used instead.

For some participants, it was also unclear why some of the templates are numbered (B2, B3, B4). An example of this are the templates existence2 and existence3.

When exploring the alignments, it was pointed out by B1 that "move in log" and "move in model" would for some users be more familiar terms than "inserted activity" an "deleted activity".

6.3.8 General impressions

In general, the feedback in relation to the new user interface of RuM was positive. None of the participants seemed to be frustrated after completing the task list and in general tended to highlight the positive aspects of the new user interface.

The simple start page of the application was praised by B1 *"nice interface"* and D4 *"I was impressed for example by the starting dashboard"*, while D1 and D2 noted that they like how the application is set up in general. B3 also pointed out that the that the user interface *"is way better than expected"* and that everything in the user interface was understandable *"after clicking around for a few minutes"*.

Additionally, B2 and D1 pointed out that there is a need for such a tool in the process mining community, while D2 expressed an interest in using RuM for future student projects.

7 Conclusion and Future Work

In this thesis, we have presented the new interface of RuM that was fully redesigned and reimplemented. Along with the new user interface, some architectural changes were made to RuM that should ensure that RuM can be continuously developed in the future, and much of the application logic was reimplemented.

We have also presented a functional overview of RuM including all the different sections of the user interface (Discovery, Conformance Checking, MP-Declare Editor and Log Generation). The new user interface allows the users to quickly navigate between the different sections of the application and to work with multiple files at the same time.

The new user interface of RuM was thoroughly evaluated with 4 experts of Declare and 4 experts of business process mining in general. The results of the evaluation were analyzed and multiple areas of improvement were identified. In general, the new user interface was well received by the participants of the user evaluation and received a system usability score of 81.875, which is well above the estimated average of 69.69 and qualifies RuM as a well usable application overall.

One of the next steps in developing RuM will be to add monitoring capabilities. Some preliminary work on adding monitoring has already been done, but there are still major hurdles to overcome, such as allowing multiple monitoring tasks to run at the same time.

We also plan to further improve the user interface of RuM based on the results of the user evaluation and to submit a demo of the application to the BPM 2020 conference demo track.

References

[1] International Organization for Standardization, “ISO quality management principles,” 2015. [Online]. Available: https://www.iso.org/files/live/sites/isoorg/files/store/en/PUB100080.pdf.

[2] W. van der Aalst, A. Adriansyah and et. al., “Process Mining Manifesto,” *Lecture Notes in Business Information Processing,* vol. 99, pp. 169-194, 2011.

[3] W. van der Aalst, “Process Mining: Overview and Opportunities,” *ACM Trans. Manage. Inf. Syst.*, vol. 3, no. 2, p. Article 7, 2012.

[4] W. van der Aalst, Process mining: discovery, conformance and enhancement of business processes, Berlin Heidelberg: Springer, 2011.

[5] M. Pesic, *Constraint-based workflow management systems : shifting control to users,* Phd Thesis: Technische Universiteit Eindhoven, 2008.

[6] W. van der Aalst, M. Pesic and H. Schonenberg, “Declarative workflows: Balancing between flexibility and support,” *Computer Science - Research and Development,* no. 23, p. 99–113, 2009.

[7] D. Kapisiz, *Rule Mining with RuM,* Masters Thesis: University of Tartu, 2019.

[8] “RuM (2019) repository,” [Online]. Available: https://github.com/alpdenizz/RuleMiningTool.

[9] “MINERfulGUI repository,” [Online]. Available: https://github.com/haasd/MINERfulGUI.

[10] M. Dumas, M. La Rosa, J. Mendling and H. Reijers, “Process Intelligence,” in *Fundamentals of Business Process Management,* Springer, 2013, pp. 353-383.

[11] M. Weske, Business Process Management: Concepts, Languages, Architectures, Berlin Heidelberg: Springer, 2007.

[12] C. Günther and E. Verbeek, “XES Standard definition,” March 2014. [Online]. Available: http://www.xes-standard.org/_media/xes/xesstandarddefinition-2.0.pdf.

[13] A. Burattin, F. M. Maggi, W. van der Aalst and A. Sperduti, “Techniques for a Posteriori Analysis of Declarative Processes,” in *IEEE 16th International Enterprise Distributed Object Computing Conference*, 2012.

[14] C. Di Ciccio, J. Mendling and F. M. Maggi, “Discovery of Multi-perspective Declarative Process Models,” in *Service-Oriented Computing*, Springer International Publishing, 2016, pp. 87-103.

[15] “Disco website,” [Online]. Available: https://fluxicon.com/disco/.

[16] C. W. Günther and A. Rozinat, “Disco: Discover Your Processes,” in *BPM (Demos)*, 2012.

[17] “ProM website,” [Online]. Available: http://www.promtools.org.

[18] B. van Dongen, A. de Medeiros, V. H.M.W., W. A.J.M.M. and W. van der Aalst, “The ProM Framework: A New Era in Process Mining Tool Support,” *Lecture Notes in Computer Science*, no. 3536, pp. 444-454, 2005.

[19] “Apromore website,” [Online]. Available: https://apromore.org.

[20] C. Di Ciccio and M. Massimo, *MINERful, a Mining Algorithm for Declarative Process Constraints in MailOfMine,* 2012.

[21] M. Pesic, H. Schonenberg and W. van der Aalst, “DECLARE: Full Support for Loosely-Structured Processes,” in *11th IEEE International Enterprise Distributed Object Computing Conference,* 2007, pp. 287-287.

[22] O. Kupferman and M. Vardi, “Vacuity detection in temporal model checking,” *International Journal on Software Tools for Technology Transfer,* no. 4, p. 224–233, 2003.

[23] A. Burattin, F. M.Maggi and A. Sperdutic, “Conformance checking based on multi-perspective declarative process models,” *Expert Systems with Applications,* no. 65, pp. 194-211, 2016.

[24] V. Skydanienko, *Data-aware Synthetic Log Generation for Declarative Process Models,* Masters Thesis: University of Tartu, 2018.

[25] “Viz.js A hack to put Graphviz on the web,” [Online]. Available: https://github.com/mdaines/viz.js.

[26] G. Krasner and S. Pope, “A description of the model-view-controller user interfa-ce paradigm in the smalltalk-80 system,” *Journal of object oriented programming,* no. 3, pp. 26-49, 1988.

[27] “CSS Guidelines webpage,” [Online]. Available: https://cssguidelin.es/.

[28] F. M. Maggi, C. Di Ciccio, C. Di Francescomarino and T. Kala, “Parallel algorithms for the automated discovery of declarative process models,” *Information Systems,* pp. 136-152, 2018.

[29] C. Di Ciccio and M. Mecella, “On the discovery of declarative control flows for artful processes,” *ACM Transactions on Management Information Systems,* no. 5, p. 24, 2015.

[30] S. Christian, S. Schönig and C. Di Ciccio, “Distributed Multi-Perspective Declare Discovery,” in *Proceedings of the BPM Demo Track and BPM Dissertation Award co-located with 15th International Conference on Business Process Modeling,* 2017.

[31] M. de Leoni, F. M. Maggi and W. 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.

[32] C. Mawoko, *Aligning Data-Aware Declarative Process Models and Event Logs,* Masters Thesis: University of Tartu, 2019.

[33] “Speech2RuM repository,” [Online]. Available: https://github.com/Baldre/Speech2RuM.

[34] 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.

[35] F. Mannhardt, “Sepsis Cases - Event Log en,” 2016. [Online]. Available: https://data.4tu.nl/repository/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460.

[36] J. Brooke, SUS-A quick and dirty usability scale. Usability evaluation in industry, 1996.

[37] A. Bhattacherjee, “Understanding information systems continuance: an expectation-confirmation model,” *MIS quarterly,* pp. 351-370, 2001.

[38] P. T. Kortum and A. Bangor, “Usability Ratings for Everyday Products Measured With the System Usability Scale,” *International Journal of Human–Computer Interaction,* vol. 29, pp. 67-76, 2013.

[39] K. Holtzblatt, J. B. Wendell and S. Wood, “Rapid contextual design: a how-to guide to key techniques for user-centered design,” *Ubiquity*, 2005.

Appendix

I. User Evaluation Introductory Email

Dear … ,

Thank you for finding the time to participate in the RuM user evaluation study. The purpose of the study is to evaluate the usability of a new declarative process mining application RuM. This is a joint study conducted by researchers from the University of Tartu (Estonia), the Free University of Bozen-Bolzano (Italy) and La Sapienza University of Rome (Italy).

Your test will be carried out over Skype on … . The test will take approximately 45 to 60 minutes and the test will be recorded.

I am sending you the link to all of the files needed for the upcoming test. The files can be found in a Google Drive folder here:
https://drive.google.com/open?id=11UdA0wJncKan2pK3EmZa_pOxAWJxNXKY

In this folder you will find a consent form, RuM application, list of tasks that will be performed during the test and the necessary input files for these tasks.

To expedite the test procedure, we kindly ask you to prepare for the test by performing the following tasks (10-15 minutes):

1. Download rum-0.4.5.jar
2. Install Java JDK 11 if not already installed on your machine
3. Try to run the application using the following command: java -jar .\rum-0.4.5.jar
4. Download the input files 1_Training_Set_Sepsis.xes and 2_Test_Set_Sepsis.xes and have these available for the test
5. Print out the task list and have it available for the test
6. Skim over the task list to get a general idea of the test process, however I would ask you to not do the tasks before the test
7. Read, sign, and scan the consent form

After the test we will also send you a link to a short survey about your general perception of RuM. The survey will take about 5-10 minutes.

On behalf of the research team,
Anti Alman

II. User Evaluation Consent Form

Consent to Act as a Participant in a Research Study

Study title: RuM user evaluation

Principal Investigator: Anti Alman

Co-Investigators: Claudio Di Ciccio, Fabrizio Maria Maggi, Alexander Nolte

Introduction: As an experienced process analyst you were selected to participate in this study. This is a joint study conducted by researchers from the University of Tartu (Estonia), the Free University of Bozen-Bolzano (Italy) and La Sapienza University of Rome (Italy).

The aim of this study is to evaluate the usability of RuM, a new declarative process mining application. We will particularly focus on identifying problematic aspects of the application's user interface and the general workflow of the application.

The study focuses on the four main functionalities of the application which are process model discovery, conformance checking, MP-Declare editor and log generation.

During the study we will ask you to use RuM to perform process mining related tasks. The description of the tasks and the application have been made available to you before the start of your test.

We will record your interaction with RuM and our conversation during the study. After the study we will ask you a few follow-up questions in a short interview, and we will ask you to fill out a short survey. The results of the test will be used to further improve the user interface of RuM.

Participation requirements: Any person 18 or older who has experience with process mining.

The expected duration of the study: The study will take about 45-60 minutes of your time.

Risks and Benefits: The risks that are associated with this research are no greater than those ordinarily encountered in daily life. There are no direct benefits to participants, but the researchers anticipate future improvements to RuM to potentially benefit process mining researchers and practitioners.

Privacy and Confidentiality: In order to protect the participants' identities during this study the research team will follow the following procedure: The original recordings will only be accessible to the Principal and Co-Investigators. The video files will be used to analyze the interaction of the participant with the RuM application. Audio contained in the recordings will be transcribed, potential identifiers will be removed or aggregated and the original recordings will be deleted afterwards.

Your data and consent form will be kept separate. Your consent form will be stored securely and will not be disclosed to third parties.

By participating, you understand and agree that the data and information gathered during this study may be used by the participating universities for publication purposes. However, any identifiable information will not be mentioned in any such publication or dissemination of the research data and/or results. The University of Tartu requires all research records to be maintained for at least 5 years following final reporting or publication of a project. Aggregated data will thus be archived by the Principal Investigator for that timespan.

Questions about the Study: If you have any questions, comments, or concerns about the study either before, during, or after participation, please contact the Principal Investigator (anti.alman@ut.ee) or the Co-Investigators.

Voluntary Participation: Your participation in this research is voluntary. You may discontinue participation at any time during the research activity. Your decision regarding whether or not to participate in this study will not result in any loss of benefits to which you are otherwise entitled.

I am of age 18 or older. I have read and understood the information above and I want to participate in this study:

☐ Yes ☐ No

Participant: The above information has been explained to me and all of my current questions have been answered. I understand that I am encouraged to ask questions, voice concerns or complaints about any aspect of this study during its course, and that such future questions, concerns or complaints will be answered by a qualified individual or by the investigator(s) listed on the first page of this consent document.

Co-investigator: I certify that I have explained the nature and purpose of this research study to the participant, and I have discussed the potential benefits and possible risks of study participation. Any questions the participant had about this study have been answered, and we will always be available to address future questions, concerns or complaints as they arise. I further certify that no research component of this study has started before this consent form was signed.

Participant_____Co-investigator_____

III. User Evaluation Task List

Introduction

You work in a hospital that has agreed to provide sepsis cases data to a private company for analysis. The hospital has recorded sepsis cases as a set of activities that were performed for a specific patient. Personal information of the patients has already been removed from the dataset. However, the management board of the hospital has decided that this alone is not enough to protect the privacy of their patients.

It was agreed that the hospital will create an artificial dataset of sepsis cases that is representative of the actual real-life treatment process. You have been given the task of creating that dataset.

To achieve this you have been given access to the anonymized dataset of sepsis cases (separated into training and test set) and a process mining tool called RuM.

Task I

We will begin by discovering a model of the sepsis cases treatment process. We will use the training set as input for discovery.

1. Discover an initial model from “1_Training_Set_Sepsis.xes” using the following parameters
    a. Method: “Declare Miner”
    b. Templates: Existence, Precedence; Response; Chain Precedence; Chain Response
    c. Minimum constraint support: 80%
    d. Pruning type: All Reductions
    e. Vacuity detection: on
2. Filter out all activities with support less than 90%
3. Navigate the different views available to inspect the model
4. Play around with the different labels that can be attached to constraints in the “Declare” view
5. Export the model as a Declare (*.decl) file

Task II

Before using the model to generate the artificial dataset we want to make sure that the model was discovered correctly. We can do this by checking if the model is conformant with the data contained in the test set.

1. Check if the model from the previous task conforms to the “2_Test_Set_Sepsis.xes” using the Declare Analyzer.
2. Navigate the output of the Analyzer. What information can you get from it?
3. Check if the model from the previous task conforms to the “2_Test_Set_Sepsis.xes” using the Declare Replayer using the following parameters:
    a. Checking method: Declare replayer
    b. Default costs
        i. Insertion cost: 5.0
        ii. Deletion cost: 10.0
    c. Specific costs for activity ER Registration

        . Insertion cost 2.0
        i. Deletion cost 15.0
4. Try to play a bit by switching between original trace views and alignment views
5. Try to find some activities that are have been inserted in some of the traces to align the model with the test set
6. Try to find the constraint that causes the highest number of deletions

Task III

It seems that the discovered model needs some modifications before it can be used to generate the artificial dataset. We will do these modifications manually using the MP-Declare editor.

1. Edit the model from Task I
    a. Remove the constraint that caused the most deletions
    b. Add the following attribute to activity ER Triage
        i. Name: Treatment
        ii. Type: Enumeration
        iii. Possible values: T1, T2
    c. Change constraint Chain Response [ER Triage, ER Sepsis Triage]
        . Template to: Response
        i. Activation condition: A.Treatment is T1
2. Use this model for the log generation task

Task IV

Now that the model of the process is completed we can use it for generating the artificial dataset.

1. Generate a log based on the model from Task III with the following parameters
    a. Generation Method: AlloyLogGenerator
    b. Minimum events per trace: 5
    c. Maximum events per trace: 10
    d. Number of traces: 20
    e. Negative traces: 20%
    f. Non-Vacuous traces: 50%
2. Check if activity ER_Triage was generated with the attribute Treatment and that some of the constraints in the model are actually satisfied in the generated traces
3. Export the generated log

This concludes the tasks for this evaluation. Next we will continue with a small interview discussing your impression of RuM - what has been done well and what could be improved?

IV. Additional Files

- Affinity diagram.pdf
- Post-survey.pdf

V. License

Non-exclusive licence to reproduce thesis and make thesis public

I, Anti Alman,

(*author's name*)

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,

A Desktop Application for Advanced Business Rule Mining,

(*title of thesis*)

supervised by Fabrizio Maria Maggi.

(*supervisor's name*)

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.

Anti Alman

15/05/2020