UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Ijlal Hussain

Generating Synthetic Event Logs based on Multi- perspective Business Rules

Master's Thesis (30 ECTS)

Supervisor(s):
Dr. Fabrizio Maria Maggi

Tartu 2016

Generating Synthetic Event Logs based on Multi- perspective Business Rules

Abstract:

Traditional business modelling is imperative in the sense that activities are provided step by step, from start to end, leading towards full business process. It has been proved that the imperative paradigm is most suitable in the context of stable and predictable processes. Declarative models are more suitable for variable processes. A declarative model is made of a set of constrains that cannot be violated during the process execution. In recent years, many techniques have been developed to discover declarative process model from event logs. To test these techniques it is sometime necessary to have tools that generate synthetic logs on which the techniques can be applied. However, majority of the existing tools available in this field use simulation of an imperative process model to generate synthetic event logs. These approaches are not suitable for the evaluation of process discovery techniques using declarative process models. Additionally, there is a need for tools to generate event logs based on the simulation of multi-perspective declarative models. To close this gap, we developed a tool for log generation based on multi- perspective Declare models. This model simulator will base on the translation of Declare constraints into Finite State Automata for the simulation of declarative processes. The tool will allows users to generate logs with predefined characteristics (e.g., number and length of the process instances), which is compliant with a given Declare model.

Keywords: Declare, Declarative Process Models, Process Simulation, Log Generation, Multi-perspective, Integer Linear Programming

CERCS: P170-Computer science, numerical analysis, systems, control

Sünteetiliste sündmuste logide genereerimine baseerudes mitmeperspektiivsetele ärireeglitele

Abstrakt:

Traditsiooniline äriprotsesside modelleerimine kasutab imperatiivset lähenemist, kus äriprotsesse kirjeldatakse üksteise järel sooritatavate tegevuste abil. On näidatud, et imperatiivne lähenemine on sobivam lahendus stabiilsete ja ennustatavate protsesside puhul. Deklaratiivsed mudelid seevastu sobivad muutuvate protsesside kirjeldamiseks. Deklaratiivne mudel sisaldab endas reeglite hulka mida ei tohi eirata protsessi käitamisel. Viimastel aastatel on arendatud mitmeid uusi meetodeid deklaratiivsete protsessimudelite leidmiseks sündmuste logidest. Meetodite testimiseks on vajalik tööriistade olemasolu, mis genereerivad sünteetilisi sündmuste logisid, mille peal neid meetodeid katsetada. Enamus olemasolevaid tööriistu kasutavad imperatiivseid protsessimudelid logide genereerimiseks. Selline lähenemine ei ole sobiv deklaratiivsete protsessimudelite avastamise meetodite testimiseks. Sarnaselt on olemas vajadus tööriistade järgi, mis genereeriks sündmuste logisid kasutades mitmeperspektiivseid Declare mudeleid. Käesolevas töös esitleme tööriista mitmeperspektiivsete Declare mudelite genereerimiseks. See töörist tõlgib Declare piirangud lõpliku olekumasina esitusse,et neid kasutada deklaratiivsete mudelite simuleerimiseks. Tööriist võimaldab kasutajatel genereerida logisid eeldefineeritud omadustega (näiteks protsessi instantside arv ja protsessi pikkus), mis on kooskõlas Declare mudelitega.

Võtmesõnad: Declare, deklaratiivne protsessimudel, protsessi simuleerimine, logide genereerimine, mitmeperspektiive, lineaarne taisarvuline planeerimine

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

Table of Contents

Abbreviations and Acronyms

API	Application Programming Interface
BPM	Business Process Management
BPMN	Business Process Modelling Notation
BPMNs	Business Process Management Systems
CPNs	Colored Petri Nets
FSA	Finite state Automata
LP	Linear Programming
ILP	Integer Linear Programming
ML	Markup Language
MXML	eXtensible Markup Language(MXML)
RE	Regular Expression
XES	eXtensible Event Steam
XML	Extensible Markup Language

1 Introduction

Process mining is a rising process management technique allowing for the analysis of business processes based on event logs. Recently, XES (eXtensible Event Stream) [1] has been introduced as an XML based standard of sorting, exchanging and analysing event logs. According to XML based standard, every event in the log represents as an activity (i.e., an explicit steps in some process)[2] [3] and is linked to a specific case (i.e., an instance of a process). All events related to a case are grouped and can be run in a single execution of the process, it is also known as a trace of events. Event logs may store extra information related to events, for example, the originator or source (i.e., person or device), starting and execution of the activity, duration of the event, or data elements stored with the event.

Automated discovery of process models from event logs is one of the most developed branch of process mining. One of the main purpose of process discovery is to extract useful information from the event logs. Therefore, testing and evaluation of process discovery techniques and tools require the availability of event logs. Unfortunately, the real log files contain noise [4][5] and are not suitable to controlled experiments where logs needs to have some given characteristics. Thus, a typical approach implemented for testing process discovery algorithms is based on synthetic logs generated through simulation. Simulation can create logs with predefined attributes and allow analysts to have more control on the exploratory settings to fine tune the developed algorithm.

In recent years, many techniques have been developed to discover declarative process models from the event logs. In addition, very recent research is focusing on the development of techniques involving multi-perspective declarative models. Such approach have got the attention of the process mining community and is useful to mine processes working in dynamic environments [6][7][8][9] [10][11]. Indeed, differently from procedural process models that work in a closed world assumption and explicitly specify all the allowed behaviours, declarative models are open. Therefore, they enjoy flexibility and are more suitable to describe highly variable behaviours in a compact way. One of the main challenges in the context of testing with declarative models is the capability of supporting multi-perspective specifications.

There exists several model simulators and log generators for process models [12][13][14][15][20][24][32][33]. These available tools simulate process model to generate synthetic logs. The main drawback of these tools are not able to generate log based on multi-perspective constraints. Therefore, tools for the generation of event logs based on the simulation of multi-perspective declarative models are needed. we will develop a tool for log generation based on multi- perspective Declare models [16]. The proposed simulator simulates declarative processes by translating Declare constraints into Finite State Automata. The tool allows user to generate logs with predefined characteristics (e.g., number and length of the process instances), which is compliant with a given Declare model.

2 Background

This section discusses some background elements of proposed research, i.e., the concept of event logs, Declare-based modelling of processes, and Finite state Automata.

2.1 Event logs

Event logs are the starting point of process mining. Event logs represent or contains information of how organizational workflow has been executed in an organization [17]. The information in event log structured in a text file. Such information can be collected from Business Process Management Systems (BPMSs) BPMS which has the ability to store information about the workflow execution [18]. Event log consists of collection of traces each trace is related to a single process instance. If the log consists of 5 traces, then this log contains data about 5 instances of a business process. Traces are single data entries that can be collected from the sequences of events carried out to perform an activity. Events or activities are another important element of event logs. The standard attributes of events are shown in Figure 2.1

Attribute Name	Description
Activity Name	Name of an element. The purpose of this attributes is to make understandable generated log.
Lifecycle transition	It shows the status of event logs for example Start, Finish etc.,
Timestamp	Data and time when an element executed
Originator	Originator and any number of additional data

Figure 2.1: Standard attributes of Events

Example of event logs based on standard attributes are shown in Figure 2.2.

```
<trace>
    <string key="concept:name" value="Synthetic trace no. 0"/>
    <event>
        <string key="Data:A.X" value="28"/>
        <string key="concept:name" value="A0"/>
        <string key="lifecycle:transition" value="complete"/>
        <date key="time:timestamp" value="2015-04-12T02:47:39.367+03:00"/>
    </event>
    <event>
        <string key="Data:A.X" value="5"/>
        <string key="concept:name" value="A0A1"/>
        <string key="lifecycle:transition" value="complete"/>
        <date key="time:timestamp" value="2015-04-12T18:01:00.578+03:00"/>
    </event>
</trace>
```

Figure 2.2: An example of XES containing Trace and events with standard attributes

The IEEE Task Force on Process Mining recognized XES (eXtensible Event Steam) [1]as a standard for event logs. This standard defines how the event log can be stored, exchange and analysed. Since BPMS support execution of a process model therefore, logs correspond to the process model that has been executed by BPMS's execution engine. XES has its open-source reference implementation library known as OpenXES

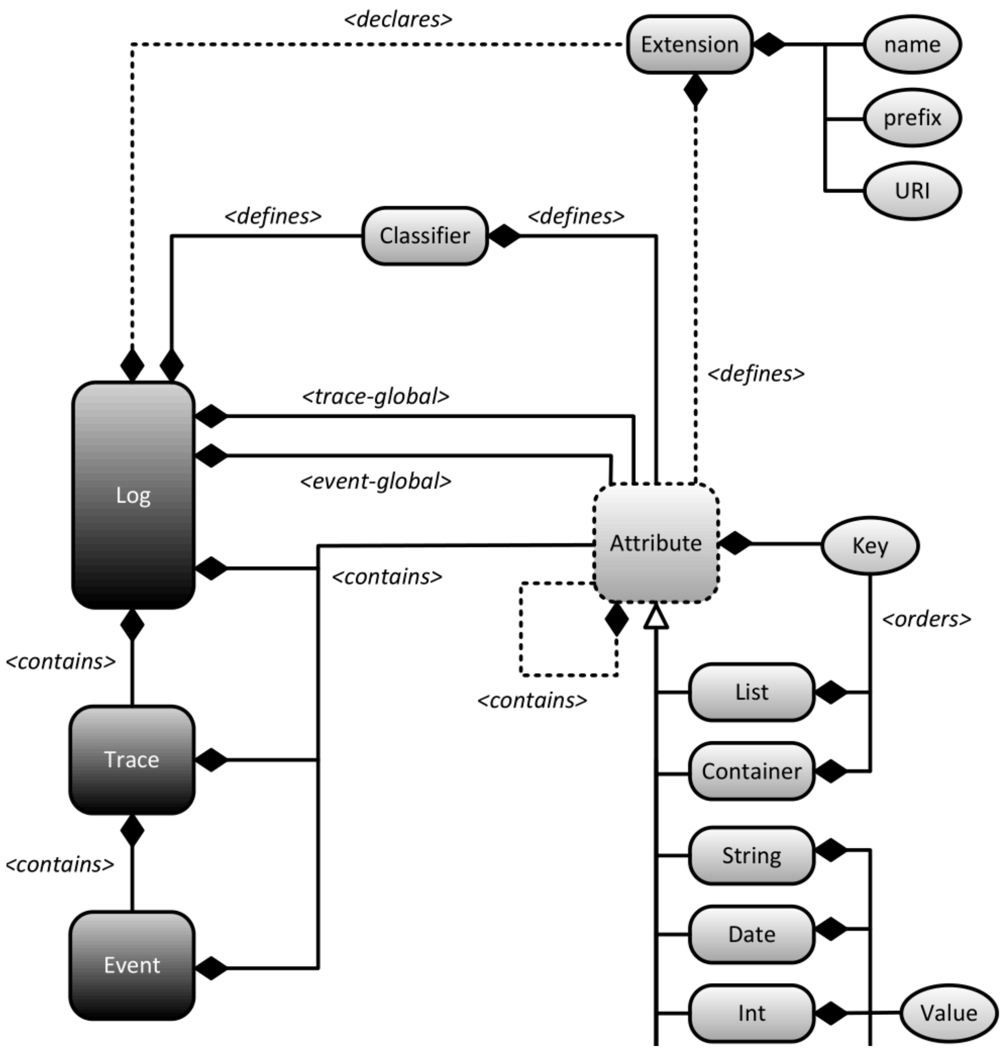
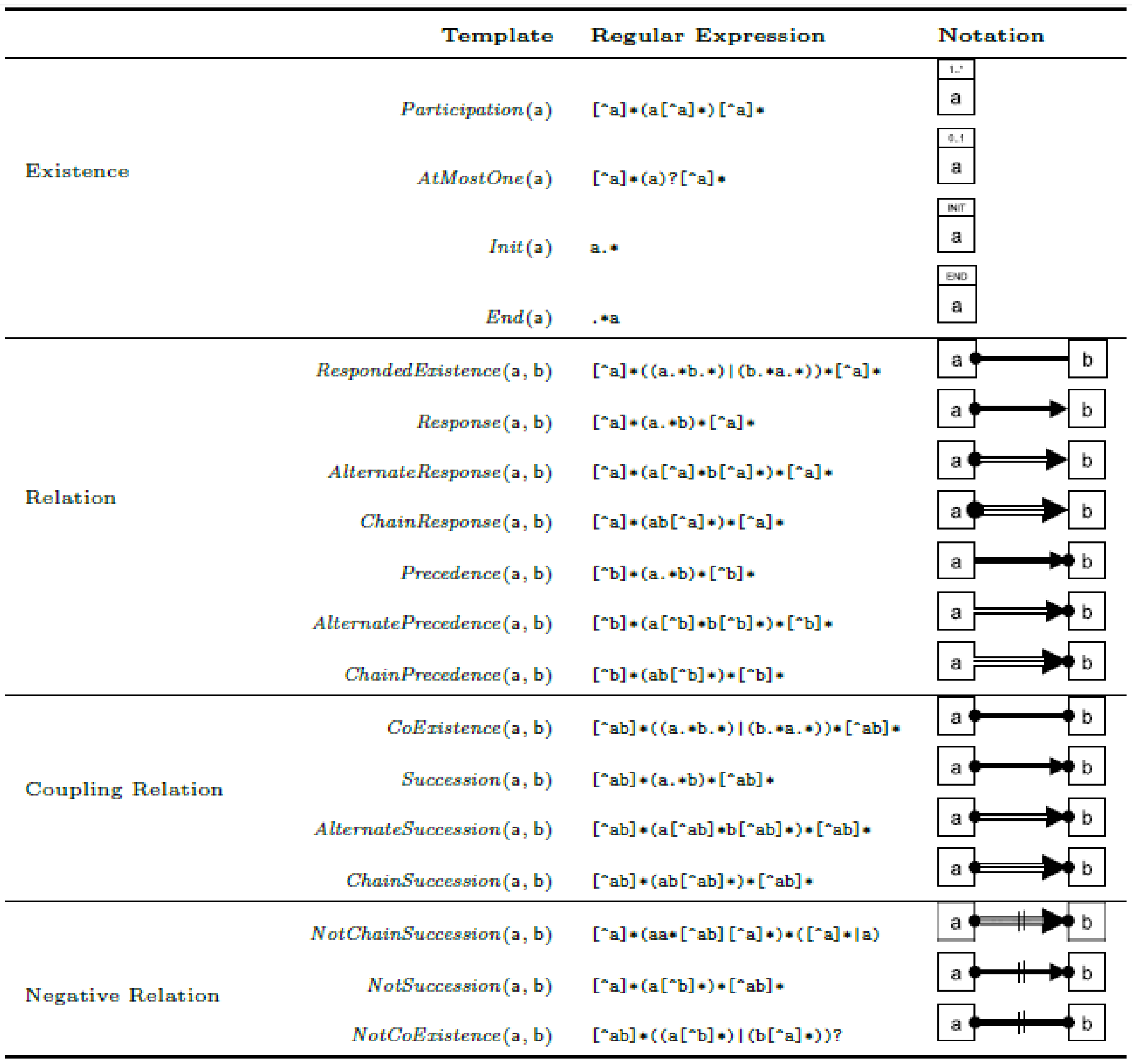


Figure 2.3 The UML 2.0 class diagram for the complete meta-model for the XES standard (Adapted from [23])

2.2 Declare

In this thesis Process models are defined using Declare Process modelling language. This modelling language originally introduced by Pesic and van der Aalst in[19]. In Declare rather than specifying the sequence of activities from the start to end of the process, a set of constraints are defined for the models. The constraints must be true during the process execution. Therefore, only valid activities are allowed that comply with the constraints. Constraints are applied on the set of activities and they are related to temporal ordering. A Declare model consists of at least one constrain and these constrains are based on templates. Templates are very easy to understand for all type of users because of it graphical interface. List of standard templates given in Figure 2.4.



	Template	Regular Expression	Notation
Existence	*Participation*(a)	`[^a]*(a[^a]*)[^a]*`	
	AtMostOne(a)	`[^a]*(a)?[^a]*`	
	Init(a)	`a.*`	
	End(a)	`.*a`	
Relation	*RespondedExistence*(a, b)	`[^a]*((a.*b.*)|(b.*a.*))*[^a]*`	
	Response(a, b)	`[^a]*(a.*b)*[^a]*`	
	AlternateResponse(a, b)	`[^a]*(a[^a]*b[^a]*)*[^a]*`	
	ChainResponse(a, b)	`[^a]*(ab[^a]*)*[^a]*`	
	Precedence(a, b)	`[^b]*(a.*b)*[^b]*`	
	AlternatePrecedence(a, b)	`[^b]*(a[^b]*b[^b]*)*[^b]*`	
	ChainPrecedence(a, b)	`[^b]*(ab[^b]*)*[^b]*`	
Coupling Relation	*CoExistence*(a, b)	`[^ab]*((a.*b.*)|(b.*a.*))*[^ab]*`	
	Succession(a, b)	`[^ab]*(a.*b)*[^ab]*`	
	AlternateSuccession(a, b)	`[^ab]*(a[^ab]*b[^ab]*)*[^ab]*`	
	ChainSuccession(a, b)	`[^ab]*(ab[^ab]*)*[^ab]*`	
Negative Relation	*NotChainSuccession*(a, b)	`[^a]*(aa*[^ab][^a]*)*([^a]*|a)`	
	NotSuccession(a, b)	`[^a]*(a[^b]*)*[^ab]*`	
	NotCoExistence(a, b)	`[^ab]*((a[^b]*)|(b[^a]*))?`	

Figure 2.4: Semantic of Declare Templates [20]

Each constraint in process model corresponds to their respective template. Using these templates makes process model independent of its formal implementation. This approach helps analyst to understand the graphical representation without knowing the hidden formulas. The graphical representation of a Declare process model consists of nodes and arcs and represents activities and constraints respectively.

Contrasted to procedural approaches, Declare models are more applicable to illustrates business processes working in unpredictable environments. Considering all what is not explicitly indicated is permitted, few constraints can determine numerous several available behaviours. Declare template may be divided into three major groups: existence templates, relation templates and choice templates.

2.2.1 Existence Templates

Existence templates is a set of unary templates. These templates can be apply only a single activity. However, some of these template can be branched by replacing a parameter with a disjunction of parameters. Table 2.1 list of Existence Templates

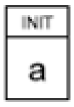
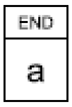
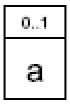
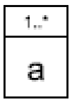
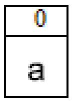
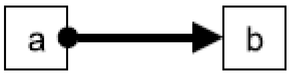
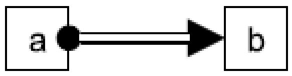
Template Name	Description	Notation
Init(a)	A process instance must start from *a*	
End (a)	*a* will be the last activity of the instance	
$AtMostOne$(a)	*a* should occur only at most once in process	
$Participation$(a)	In each process *a* must occur at least once	
$Absence$(a)	*a* should not occur in a process	

Table 2.1: Existence Templates

2.2.2 Relation templates

The relation templates are used to correlate activities. These templates can be ordered or un-ordered. Ordered means that events should be in a sequence while in un-ordered templates events will occur in any order. The relation templates are divided in two groups: i) positive relation templates and ii) negative relation templates. In negative relation templates the execution of one activity restricts the execution of other activity. Table 2.2 list of relation templates.

Template	Description	Notation
Response(a, b)	If activity *a* executed, activity *b* must execute eventually.	
$AlternateResponse$(a, b)	This template is stronger version of response template in which re-	

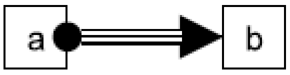
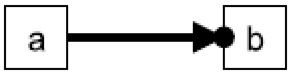
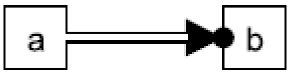
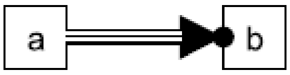
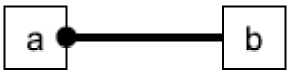
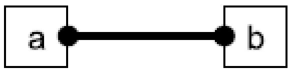
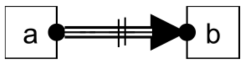
	strict execution of another *a* between an execution of *a* and following *b*.	
$ChainResponse$ (a, b)	Whenever activity *a* executed, activity *b* must be occurs directly after it.	
$Precedence(a, b)$	Before execution of activity *b*, activity *a* must be executed	
$AlternatePrecedence$ (a, b)	every instance of activity B has to be preceded by an instance of activity *a* and the activity *b* cannot be executed again before the activity *a* is also executed	
$ChainPrecedence$ (a, b)	Activity *a* directly precedes each *b*.	
$RespondedExistence$(a, b)	This template specifies that if activity *a* is executed, activity *b* also has to be executed at any time, either in future or past	
$CoExistence$ (a, b)	If one of the activities *a* or *b* is executed, the other one has to be executed as well.	

Table 2.2: Relation Templates

2.2.3 Negative Relation Templates

As the name suggests these templates are negated version of relation templates. For example, while Response(a, b) specifies that If activity *a* executed, activity *b* must execute eventually NotResponse(a, b) is complete opposite it which means event *b* cant execute after execution of event *a*. Table 2.3 shows the symbols, description and graphical representation of all negative relation templates.

Template Name	Description	Notation
$NotChainResponse$ (a, b) $NotChainPrecedence$ (a, b)	*a* and *b* never follow each other directly i.e., $NotChainResponse$ (a, b) If event	

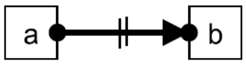
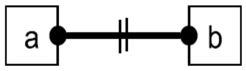
$NotChainSuccession$ (a, b)	a executes, then b should never executed next to a $NotChainPrecedence$ (a, b) a should never precede b directly $NotChainSuccession$ (a, b) is combination of above templates	
$NotResponse$ (a, b) $NotPrecedence$(a, b) $NotSuccession$ (a, b)	After execution of activity *a* activity *b* cannot be executed Before execution of activity *b* there cannot execute activity *a*	
$NotCoExistence$ (a, b) $NotRespondedExistence$ (a, b)	It is a negative relation template that if one of the activities *a* or *b* is executed, the other cannot be executed in the same trace	

Table 2.3 Negative Relation templates

2.2.4 Choice Templates

Choice templates are used to specify that one must choose to execute an activity between the given activities. Choice templates can be specified as 1 of N this means at least one of the activity should be executed from N activities. For example, the 1 of 3 template specifies that no less than one of the three activities A, B, and C must be executed, yet each activities can be executed a variable number of times as long as at least one of these activities occurs at least once.

The exclusive choice templates are stronger than choice templates. The exclusive choice 1 of N means at least one of the activity should be executed one or more than one times from N activities, while other activities (N-1) cannot be executed in any way. For example, 1 of 2 exclusive templates determines one of the two activities A and B must be executed, while the other activity cannot be executed.

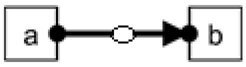
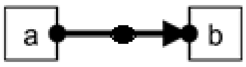
Template Name	Description	Notation
$Choice$ (a, b)	At least *a* or *b* has to executed in a process	
$exclusiveChoice$ (a, b)	At least *a* or *b* has to executed but not both	

Table 2.4: Choice Templates

2.2.5 An Example of a Declare Model

Fracture treatment a case of Declare procedure model is given in Figure 2.5.The process involves activities like checkup patients, check the risks of X-ray, change position, settle a cast, uproot cast, perform operation, and determine recovery its behavior is specified by the following constraints C_1 - C_7:

C_1 *Init* (checkup the patient)

C_2 *AlternatePrecedence* (check X-ray risk; perform X-ray)

C_3 *Precedence* (perform X-ray; change position)

C_4 *Precedence* (perform X-ray; apply cast)

C_5 *Response* (apply cast; remove cast)

C_6 *Precedence* (perform X- ray; perform operation or surgery)

C_7 *Response* (perform operation; determine recovery)

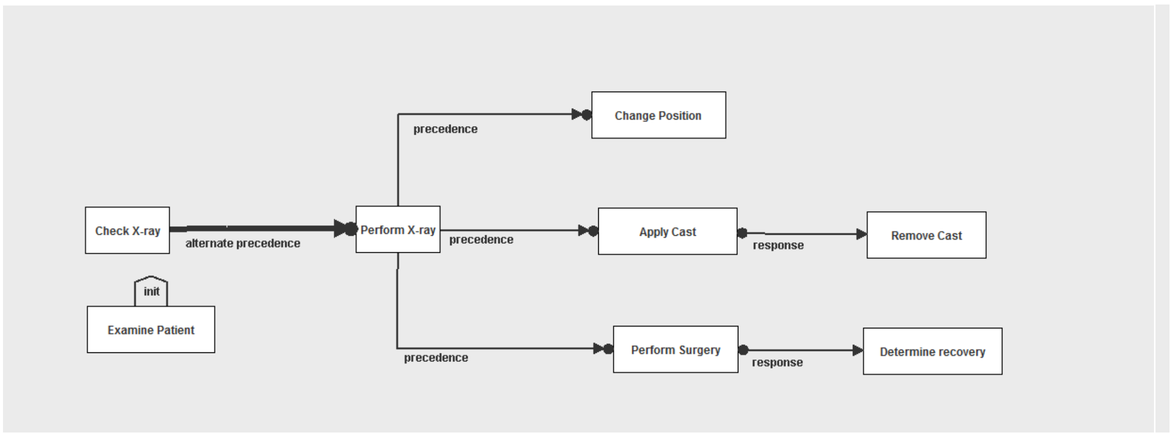


Figure 2.5 The Declare model for treatment process [20]

As indicated by above constraints, each process occurrence begins with an activity checkup patient (C_1) and medical team can be perform many times at any stage of the treatment.

It is necessary to take X-ray of the patient before applying cast, surgery or repositioning. During the treatment X-ray can be taken many times, if required. Due to side effects of rays and health issues it is important to check the risk factors of X-rays i.e., allergies, pregnancy etc. and whenever perform X-ray is required these risks should be check every time. Therefore, activity check X-ray risk must be completed before perform X-ray occurs, without any other execution of perform X-ray in between ($C_{2,}$). When the activity perform X-ray is complete the staff can apply change positions, apply cast or perform surgery ($C_{3,}$ $C_{4,}$ C_6). During the treatment it is possible to take X-ray many times (if required) since the activity check X-ray is completed before performing a new X-ray activity.

The last activity of this process is determine recovery. When activity perform surgery is completed all patients send to determine recovery, it is also possible to perform recovery for those patients who did not undergo a surgery. ($C_{5,}$ C_7)

2.3 Declare with Data

All the Declare constrains we have discussed in the previous sections focuses only the control-flow. However, Declare models also support data-flow or data based on their condition. There is no specific format for these condition but, for the sake of simplicity or differentiate between activity and payload data name we can used special character for example ".". In the proposed thesis we have define like "A.X" where A is an activity name and X is the payload of event A.

Activation Condition is used to activate a constraint event. For example, if activations condition of an event is set as *[A.x > 2]*, then the event A will a constraint only when the value of payload x is greater than 2.

For example, in the fracture treatment example *Response (Applycast; Removecast)* with the data condition *[A:CastStatus == 1]*.This constrains will be activated when *Applycast.CastStatus* equal to 1 (1 for true and 0 for false).If this occurs then we need to execute *Removecast*

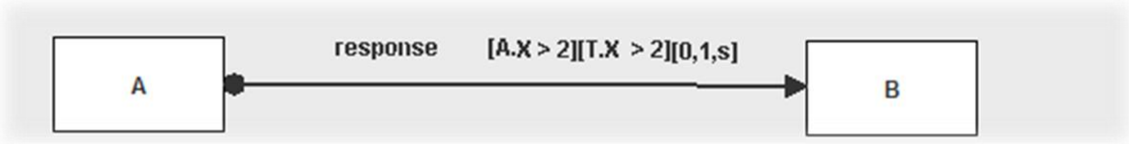


Figure 2.6: Example of Declare Model with data

2.4 Finite State Automata

A deterministic FSA is a labelled transition system A $= (A, S, \partial, s_0, S_f)$ defined over states S and an alphabet A, having $\partial : S \ x \ A \ \rightarrow S$ as transition function, i.e., a function that, given a starting state and ɑ character, returns the target state (if defined). $s_0 \in$ S is the initial state of A, and $S_f \subseteq$ S is the non-empty set of its accepting states ($S_f \neq \phi$). For the sake of simplicity, we will omit the qualification "deterministic" in the remainder of this thesis. A finite path π of length n over A is a sequence $\pi = (\pi^1, \ldots \ldots .., \pi^n)$ of tuples $\pi^1 = (s^{i-1}, \partial^i, s^i)$ $\in$ Ϭ for which the following condition hold true: (i) π^1 , the first tuple, is such that $s^0 = s_0$ (it starts from the initial state of A), and (ii) the starting state of π^i is the largest state of π^{i-1} : π^1= ((s0, ∂^1 , s1)(s1, ∂^2 , s2),……(sn-1, ∂^n , s^n)) [20].

A finite string of length $n \geq 1$, i.e., a concatenation t = $t_1 \ldots . t_n$ of characters $t_i \in$ A is accepted by A if a path π of length n is defined over A and is such that (i) for every I $\in$ [1,n], $\pi^i = (s^{i-1}, t_i, s^i)$ and (ii)], $\pi^n = (s^{i-1}, t_n, s^n)$ is s.t. $s^n \in s_f$

FSAs are closed under the product operation x. A product of two FSAs takes the connection of languages (sets of accepted strings) recognized by every operand. The product of FSAs is an isomorphism for the combination of RE, i.e., the product of FSAs respectively corresponding to two REs is equivalent to the FSA that derives from the conjunction of the REs. [20]

2.5 Integer Linear Programming

Linear programming (LP) is an approach for optimization of a linear objective function of variables x1, x2……, xn, with respect to linear equality or linear inequality constraints, A LP can be defined as the problem of either maximizing or minimizing a linear function subject to its constrains[34]. In LP all fractional solutions are not accurate, and we must consider the optimization problem

$$\text{Maximize:} \sum_{j=1}^{n} c_j x_j$$

$$\text{Subject to:} \sum_{j=1}^{n} a_{ij} x_j = b_i \qquad (i = 1,2, \dots \dots \dots., m\,),$$

$$x_j \geq 0 \quad (j = 1,2, \dots \dots . \dots . n)$$

$$x_j \text{ integer for some or all} \quad (j = 1,2, \dots \dots . \dots . n)$$

The goal is to maximize variable values (i.e., profits in case business) by utilizing available resources. This problem is known as Linear Integer programming problem. When all decision variables are in integers called pure integer program, and if some, but not all, variables are not limited to be an integer is known as mixed integer program.

In the proposed thesis, for solving ILP problems, an open-source ILP solver *lp_solve* [25] is used. *lp_solve* is an LP and ILP based solver and it is freely available under the GNU Lesser General Public License. The lp_solve jar file is used to solve ILP problem and this jar file can be download from here [26].

3 Related Work

The main goal of automated generation of event log is to test process mining algorithms and business rules in the process models. Recently, a lot of new techniques and methods are proposed by different researcher. The first log generation tool was proposed by Hee and Liu and they presented a framework to generate Petri nets representing processes based on different set of user topological rules [14].

Colored Petri Nets (CPNs) Tools [13] [30] is a widely utilized framework to simulate CPNs. It provides graphical interface to support the modification of CPNs. Moreover, this tool can simulate and allow users to generate traces. CPN tools generates random events log from CPN and the log results are produced in (MXML). The approach [30] extended CP-nets to generate XML event logs that can be mined by process mining tools supporting XML format.

Burattin and Sperduti [12] proposed an approach for logs generation. The tool allows users to generate logs from a BPMN model.

SecSy tool [22] has been developed as a standalone application to generate logs. It allows useful settings for process models and their executions. It creates sets of logs on each run and includes few deviations from original model. The result can be produced in both MXML and XES standard. The main goal of this tool is to execute models with security suited frameworks. It allows to produce special event logs with specific constraints useful for security analysis of processes.

A newly proposed easy to use tool [32], provides different simulation strategies and configuration options covering most standard use cases. The generated event log from this tool can be immediately utilized in subsequent steps within a process mining analysis workflow. The approach presented in [32] is developed as a ProM plugin that allows direct generation of event logs. The plugin uses token-based simulation, which is driven by Petri net models.

In [33] an approach is presented for generation of sets of event logs. This approach is implemented as a ProM plug-in which can be easily used by process miners, researchers, and developers. It allows not only to generate the simple event logs, but also to generate a set of event logs, or event logs with noise. All these functions allow to run experiments in the relatively easy way with different algorithms implemented as a ProM plug-ins. Generated logs can be exported using standard ProM plug-ins to use them in other applications. Noise generation is also quite useful during plug-in testing process. All the above methods described are suitable for procedural methods

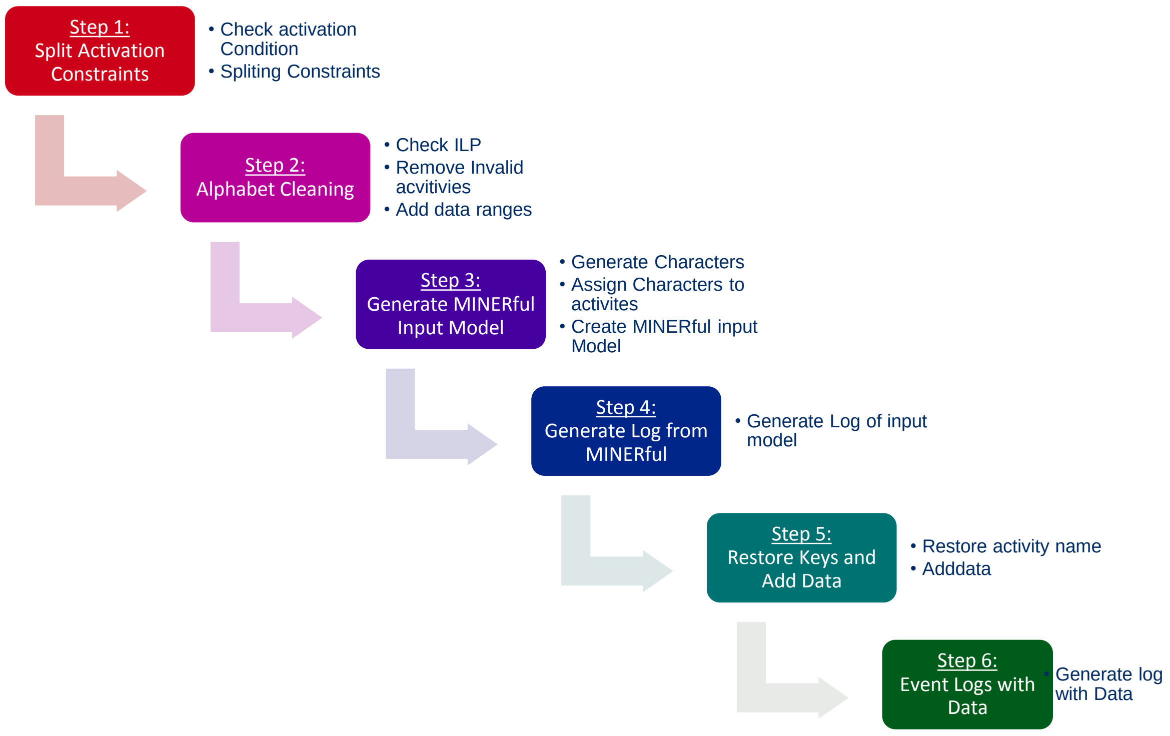
The latest version of CPN Tools [11][30] has graphical support to add Declare constraints to the transitions of a CPN, to generate hybrid models. These tools allow both the user-driven and random executions of such models. CPN tools is an extension of a procedural-based model and approach proposed in this thesis is an inherent tool to manage Declare models, particularly in generating event logs. For example, in the proposed tool, users can specify the number of traces that need to be generated as an input parameter.

An approach presented [20] for the automated generation of event logs, starting from Declare process models. An evaluation of the implemented tool is presented, showing its effectiveness in both the generation of new logs and the replication of the behavior of existing ones. The presented evaluation also shows the capability of the tool of generating very large logs in a reasonably small amount of time, and its integration with state-of-the-art Declare modeling and discovery tools. In this approach [20], they have proposed a graph-based structure as a source to create benchmarking data (event logs). These tools are based on standard Declare and do not allow users to generate logs from multi-perspective declarative specification.

Laurent Y [24] presented an approach to generate automated event logs of declarative process models using Alloy model-finding method. This approach provides valuable help to a process modeler both in the design and execution-time phases. During design time it provides an early understanding while being modeled and the execution-phase is primordial to confirm the execution of traces. This tool is based on Alloy language. User needs to learn an Alloy command to write these scripts and Alloy to understand the generated log

4 Approach

This section describes the proposed approach used for log generation. The approach is depicted in Figure 4.1. To illustrate it we use the model shown in Figure 4.2.This model is used to generate simple event logs. In this figure a Declare model is consist of only four activities, namely A, B, C and D, therefore, the model has three constraints *response(A, B), response(A, C)* and *response (A, D).*



Step 1:
Split Activation Constraints
• Check activation Condition
• Spliting Constraints
Step 2:
Alphabet Cleaning
• Check ILP
• Remove Invalid acvitivies
• Add data ranges
Step 3:
Generate MINERful Input Model
• Generate Characters
• Assign Characters to activites
• Create MINERful input Model
Step 4:
Generate Log from MINERful
• Generate Log of input model
Step 5:
Restore Keys and Add Data
• Restore activity name
• Adddata
Step 6:
Event Logs with Data
Generate log with Data

Figure 4.1: Flow chart of proposed approach

The proposed approach generate all the possible combination of given activation activities. For example in Table 4, *response (A, B)* is represent as *response* (A_0, B), *response (A, C)* is split as *response* (A_1, *C)* and *response (A, D)* as *response* (A_2, *D).* All generated activation are verified using ILP with their data condition. Each constraint gets their data on the basis of result obtained from ILP. Since log generation uses MINERFul [27], therefore, models are created that complies with Minerful Input model. In the resulting model, the values received from ILP is added and finally events logs with data are generated. Next section discusses the above step in details.

4.1 Split Activation Constraints

The main goal of this step is to split of activations based on validity of the condition and generate all possible combinations without any duplications. This step takes Declare model as an input and splits the activation activities without any duplications. In our example (see Figure 4.2), all possible combinations of proposed model are given below:

alphabet = {A, A_0, A_1, A_2, A_0A_1, *A0A2, A1A2*, $A_0A_1A_2$, *B, C, D*}

Where

A_0	Occurrence of *A* with C_0 true
A_1	Occurrence of *A* with C_1 true
A_2	Occurrence of *A* with C_2 true
A_0A_1	Occurrence of *A* with C_0 and C_1 true
A_0A_2	Occurrence of *A* with C_0 and C_2 true
A_1A_2	Occurrence of *A* with C_1 and C_2 true
$A_0A_1A_2$	Occurrence of *A* with C_0, C_1 and C_2 true

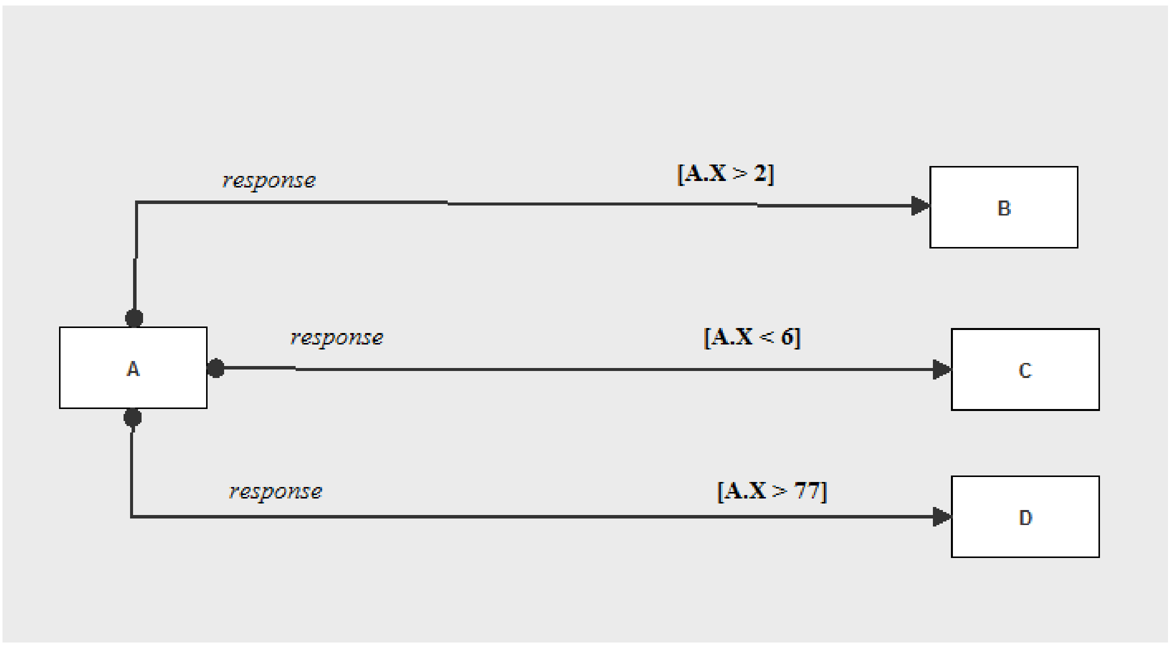


Figure 4.2: Example of Log genretaion appraoch model

The activation of *response (A, B)* is denoted by A_0, for *response (A, C)* is denoted by A_1 and so on. Similarly, activation A_0A_1 is activation for both constraints *response (A, B)* and *response (A, C)*

Thus, we check if the condition can be true together. For example, for activation constraint A_0A_1 we check for activation condition for *response (A, B)* and *response (A, C)* will be true together. According to the activation condition of *A* the payload value must be less than two and greater than six which is not possible.

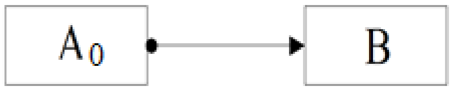
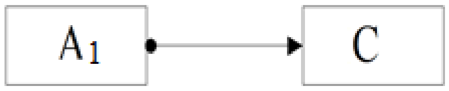
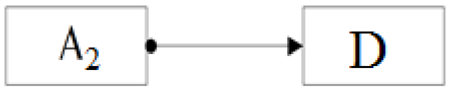
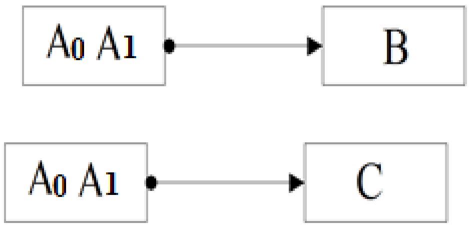
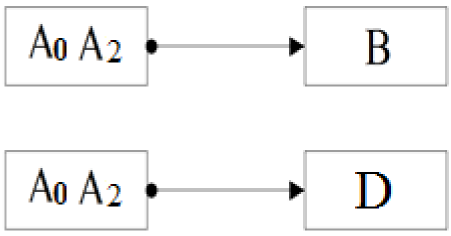
	Notation	Description
Template *(A_0)*		*A_0* followed by *B*
Template *(A_1)*		*A_1* followed by *C*
Template *(A_2)*		*A_2* followed by *D*
Template *(A_0A_1)*		*A_0A_1* is followed by both *B* and *C*
Template *(A_0A_2)*		*A_0A_2* is followed by both *B* and *D*

Figure 4.3: Spliting concept of templates

4.1.1 Alphabet Cleaning

One of the main contribution of this thesis is the use of ILP. Every activity that contains activation condition is checked from ILP. ILP is used to remove elements of the alphabet that correspond to conditions that cannot be true together. The tool developed in this thesis uses *lp_solve* [33] to check ILP. Each activation condition translated to ILP equation and these equations are solved. We get the ILP results for all given activities.

The detailed ILP result from example (in Figure 4.2) is listed below (Table 4.1).

Activity Name	Activation Condition	ILP Status	Data Range
A	(A.X < 2)&&(A.X > 6)&&(A.X < 77)	Invalid	Null
A_0	(A.X < 2)&&(A.X < 6)&&(A.X < 77)	Valid	7-66
A_1	(A.X < 2)&&(A.X < 6)&&(A.X < 77)	Valid	2 – Max

A_2	(A.X < 2)&&(A.X > 6)&&(A.X > 77)	Invalid	Null
A_0A_1	(A.X > 2)&&(A.X < 6)&&(A.X < 77)	Valid	2-6
A_0A_2	(A.X > 2)&&(A.X > 6)&&(A.X > 77)	Valid	Min – 77
A_1A_2	(A.X < 2)&&(A.X < 6)&&(A.X > 77)	Invalid	Null
$A_0A_1A_2$	(A.X > 2)&&(A.X < 6)&&(A.X > 77	Invalid	Null

Table 4.1: ILP Status and Data Ranges of activities

All activities with ILP status Invalid will be remove from the model i.e. *A*, $A_{2,}$ and $A_0A_1A_2$ etc. Invalid ILP Status means that the given condition will never become true for example according to the activation condition of *A* the payload or data value must be less than two and greater than six which is not possible. If the condition is true then data range is assign based on ILP result. For example, activity A_1 can occur when data value is greater than two.

4.2 Generate MINERFul Input Model

We use Minerful [29] to create logs. Minerful only allow single character as an activity name in the process model. For this purpose, we map process activity name with Minerful equivalent character (*TaskChar*). For example, in Figure. 4.1, process model consists of seven activities and these activities are mapped by *a, b, c, d, e, f,* and *g* respectively. Table 4.2 lists all activities of input model with their corresponding TaskChar.

Activity Name	MINERFul TaskChar
A_0	A
A_0A_1	B
A_0A_2	C
A_1	D
B	E
C	F
D	G

Table 4.2: Activity name with MINERFul equivalent TaskChar

4.3 Generate Log via MINERFul

To generate log from MINERFul three inputs are required, *i*) Input Model *ii*) Minimum and Maximum size of events per trace iii) total number of traces that the generated log must contain. The Maximum size per trace and total number of traces per events must be greater or equal to one. MINERFul will produce event logs based on predefined parameters. The resulted output from MINERFul is core structure of proposed log generation format.

4.4 Restore Keys and add Data

The log generated from MINERFul is consists of character that differs from the actual activity name of process model. In this step, we restore original names of these activities without altering their traces and events order.

All these activities available in the generated logs are valid and are verified from ILP. ILP assigns maximum and minimum ranges for each activity. On the basis of ILP result, tool generates random number selected between given ranges.

4.5 Events Log with data

This is the final step of log generation. The generated log with data is stored at a location specified by user and in a user selected format i.e., XES format. To complete this task we have used log storing services provided by MINERFul

5 Evaluation

In this section, we use our developed tool for generating event logs.

5.1 Multi-Perspective Process Model

In section 2 we have discussed different templates of Declare and it is not possible to include all these templates for evaluation of our implementation. For sake of simplicity we create a sample model containing several templates (see Figure 5.1)

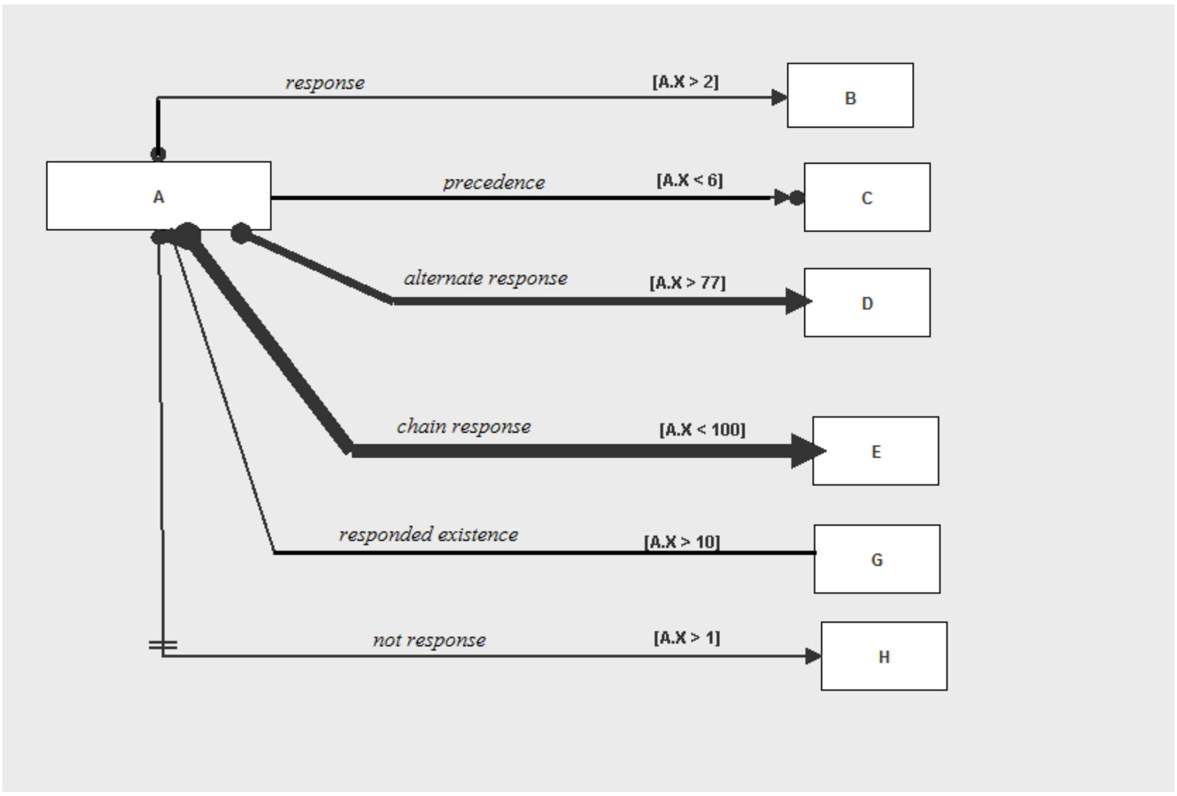
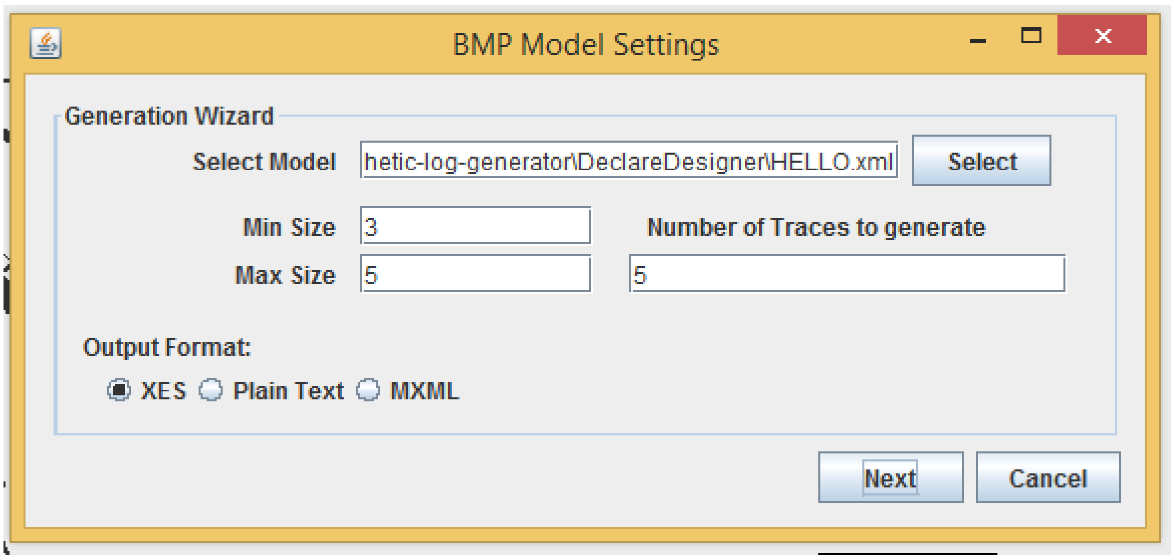


Figure 5.1 Sample Declarative process model for evaluation

The sample model contains six templates *response*, *precedence*, *alternate response*, *chain response*, *responded existence* and *not response.*

5.2 Parameter Settings

To evaluate the event log we have set event size between 3 to five per trace. Number of traces also fixed to 5 thesis (see Figure 5.2).


BMP Model Settings

Generation Wizard

Select Model: hetic-log-generator\DeclareDesigner\HELLO.xml

Min Size: 3

Max Size: 5

Number of Traces to generate: 5

Output Format: [✓ XES] Plain Text MXML

Figure 5.2: Parameter setting for Evaluation

5.3 Declare Input Model

As we discussed in the previous section about MINERFul input model and based on ILP results the generated MINERFul equivalent mapped TaskChars for the valid activation activities are shown in the Table 5.1. All activities with null ILP status is removed from the input model.

Activity Name	MINERFul Mapped TaskChar
A	A
A0A1A2A3A4	B
A0A1A3A4	C
A0A2A3A4	D
A0A2A4	E
A2	F
A2A4	G
B	H
C	I
D	J
E	K
G	L

H	M

Table 5.1 Activities with mapped TaskChars

In the input model, we are including all split activation activities based on activation conditions. For the given evaluation model activation activities with correlated constrains are shown in the Table 5.2.

Activities	Description

Response(B, H)
AlternateResponse(B, J) :
ChainResponse(B, K)
RespondedExistence(B, L)
NotSuccession(B, M)
	The TaskChar B is representing activation activity *A0A1A2A3A4* and this activation activity must be compliant with all given constrains. The input model generated all constrains that is required for activation condition *A0A1A2A3A4*

Response(C, H)
AlternateResponse(C, J)
RespondedExistence(C, L)
NotSuccession(C, M)
	The TaskChar C is representing activation activity *A0A1A3A4* and in input model all required constraint related to this activation activity is generated.

Response(D, H)
ChainResponse(D, K)
RespondedExistence(D, L)
NotSuccession(D, M)
	TaskChar D is denoted by *A0A2A3A4*

Response(E, H)
ChainResponse(E, K)
NotSuccession(E, M)
	TaskChar E is denoted by *A0A2A4*

ChainResponse(F, K)	TaskChar F is denoted by *A2*

ChainResponse(G, K)
NotSuccession(G, M)
	TaskChar D is denoted by *A2A4*

Precedence(A, I)	The only activation activity is the given model *precedence (A,C)* generated in the input model

Table 5.2 MINERFul Input Model of Evaluation Model

5.4 Generated Event logs

The generated XML file format converted into a tabular format. In this section we introduce each trace of generated log.

Attribute	Value
Event No	1
Activity Name	A0A2A4
Payload Value	7
lifecycle: transition	Complete
time: timestamp	2014-09-25T06:00:50.825+03:00

Attribute	Value
Event No	2
Activity Name	E
Payload Value	81
lifecycle: transition	Complete
time: timestamp	2014-09-26T03:39:03.700+03:00

Attribute	Value
Event No	3
Activity Name	A
Payload Value	706019319
lifecycle: transition	Complete
time: timestamp	2014-09-26T05:45:13.481+03:00

Attribute	Value
Event No	4

Activity Name	D
Payload Value	93
lifecycle: transition	Complete
time: timestamp	2014-09-27T04:02:20.403+03:00

Attribute	Value
Event No	5
Activity Name	B
Payload Value	86
lifecycle: transition	Complete
time: timestamp	2014-09-28T00:21:38.168+03:00

This trace contains five events with only one activation activity highlighted in green colour. The activity *A0A2A4* means this activity must be executed by all given correlated constrains. According to the template A0 *response (A, B)* which means *A* must be followed *B, A2, chain response (A, E)* , A must be immediately followed by *E* and *not response (A, H),* A never followed by H. In this trace you can see that in the event number two E is immediate following A, B is also following A at event number 5. H is never exists in this trace. Hence, we can say that this trace is compliant with the given model.

Attribute	Value
Event No	6
Activity Name	A0A2A4
Payload Value	6
lifecycle: transition	Complete
time: timestamp	2014-09-28T03:31:51.171+03:00

Attribute	Value
Event No	7

Activity Name	E
Payload Value	93
lifecycle: transition	Complete
time: timestamp	2014-09-28T04:03:59.537+03:00

Attribute	Value
Event No	8
Activity Name	A0A2A4
Payload Value	6
lifecycle: transition	Complete
time: timestamp	2014-09-29T01:23:43.980+03:00

Attribute	Value
Event No	9
Activity Name	E
Payload Value	78
lifecycle: transition	Complete
time: timestamp	2014-09-29T20:45:40.302+03:00

Attribute	Value
Event No	10
Activity Name	B
Payload Value	87
lifecycle: transition	Complete
time: timestamp	2014-09-29T21:01:38.225+03:00

The trace number 1 is also contains five events with two but same activation activity highlighted in green colour. In this trace the activity *E* repeating on event number seven and nine and both events are immediately following *A2*. Activity *B* is available in the last of the trace and it is following by both activating activities i.e., A0A2A4. In the log we can see that number of constraints are following according to activating activity which is exactly requirement of the given process model

Attribute	Value
Event No	11
Activity Name	G
Payload Value	93
lifecycle: transition	Complete
time: timestamp	2014-09-30T17:11:15.854+03:00

Attribute	Value
Event No	12
Activity Name	B
Payload Value	83
lifecycle: transition	Complete
time: timestamp	2014-10-01T05:54:54.613+03:00

Attribute	Value
Event No	13
Activity Name	A0A1A3A4
Payload Value	1811582665
lifecycle: transition	Complete
time: timestamp	2014-10-01T17:08:44.135+03:00

Attribute	Value
Event No	14
Activity Name	D
Payload Value	80
lifecycle: transition	Complete
time: timestamp	014-10-02T12:21:50.018+03:00

Attribute	Value
Event No	15
Activity Name	B
Payload Value	93
lifecycle: transition	Complete
time: timestamp	2014-10-03T11:16:50.199+03:00

The trace number 2 is only one activation activity highlighted in green colour. This trace is different from the previous two traces because in this trace template *responded existence (A, G)* has executed. Responded existence specifies that if event A is executed in the trace, then also event G has to be executed either after or before event A, so the *G* is present before *A* in the trace while *D* and *B* is following proper order. Hence, we can say that this trace is also compliant with the actual process model.

Attribute	Value
Event No	16
Activity Name	B
Payload Value	88
lifecycle: transition	Complete
time: timestamp	2014-10-03T20:55:04.472+03:00

Attribute	Value
Event No	17
Activity Name	E
Payload Value	77
lifecycle: transition	Complete
time: timestamp	2014-10-04T06:37:22.905+03:00

Attribute	Value
Event No	18
Activity Name	A2A4
Payload Value	1
lifecycle: transition	Complete
time: timestamp	2014-10-04T23:49:39.500+03:00

Attribute	Value
Event No	19
Activity Name	E
Payload Value	92
lifecycle: transition	Complete
time: timestamp	2014-10-05T13:51:13.217+03:00

As we have set the size of events per trace between three and five. This trace contains four events and the only activation activity is at the second last of the trace. According to this activation activity, activity *E* must be followed after *A2* while *A4* never followed by any activity *H*. The trace is looking good according to the given model.

Attribute	Value
Event No	20
Activity Name	D
Payload Value	98
lifecycle: transition	Complete
time: timestamp	2014-10-06T10:48:22.064+03:00

Attribute	Value
Event No	21
Activity Name	A0A2A4
Payload Value	5
lifecycle: transition	Complete
time: timestamp	2014-10-06T20:40:44.217+03:00

Attribute	Value
Event No	22
Activity Name	E
Payload Value	88
lifecycle: transition	Complete
time: timestamp	2014-10-07T07:45:40.083+03:00

Attribute	Value
Event No	23
Activity Name	A
Payload Value	448988707

lifecycle: transition	Complete
time: timestamp	2014-10-07T13:50:42.888+03:00

Attribute	Value
Event No	24
Activity Name	B
Payload Value	78
lifecycle: transition	Complete
time: timestamp	2014-10-08T01:17:24.274+03:00

Trace number five is the last trace of this evaluation and it is containing the same activation activity as trace number one and two. The result is very clear, activity E is directly following by A2 and A4 is not following any activity H. Hence, we can see that all generated logs are compliant with the given model.

5.5 Chain Constrains Process Model

In the Figure 5.3 the reader can see that the *response (A, C)* which means that the activity C must be followed by activity A. However, C is also an activation activity of other constraints. The given model in split as *A0, A0A1, A1, B, C, C0, C0C1, C1, D, and E.*

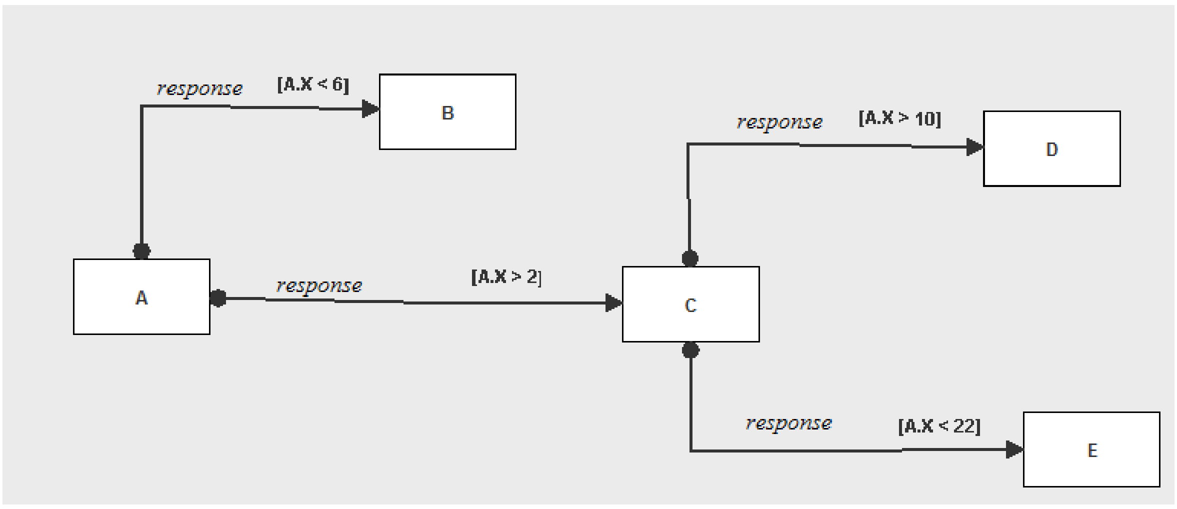


Figure 5.3: Chain of Constraints in a Model

To evaluate the result of chain constraint we have generated a very simple event logs. The maxim and minimum event per trace is three and five respectively and the size of trace is set to only two. The generated event logs based on the set parameters are given below:-

Attribute	Value
Event No	1
Activity Name	A1
Payload Value	1981591828
lifecycle: transition	Complete
time: timestamp	2015-06-05T12:50:20.484+03:00

Attribute	Value
Event No	2
Activity Name	C0
Payload Value	1252252207
lifecycle: transition	Complete
time: timestamp	2015-06-05T18:35:26.668+03:00

Attribute	Value
Event No	3
Activity Name	E
Payload Value	13
lifecycle: transition	Complete
time: timestamp	2015-06-06T13:06:41.238+03:00

Attribute	Value
Event No	4

Activity Name	B
Payload Value	2
lifecycle: transition	Complete
time: timestamp	2015-06-06T20:13:41.174+03:00

Attribute	Value
Event No	5
Activity Name	D
Payload Value	10
lifecycle: transition	Complete
time: timestamp	2015-06-07T18:42:22.365+03:00

Attribute	Value
Event No	6
Activity Name	A1
Payload Value	1398284071
lifecycle: transition	Complete
time: timestamp	2015-06-07T22:04:48.203+03:00

Attribute	Value
Event No	7
Activity Name	E
Payload Value	10
lifecycle: transition	Complete
time: timestamp	2015-06-08T20:17:35.166+03:00

Attribute	Value
Event No	8
Activity Name	A0
Payload Value	2
lifecycle: transition	Complete
time: timestamp	2015-06-09T11:57:56.920+03:00

Attribute	Value
Event No	9
Activity Name	C
Payload Value	3
lifecycle: transition	Complete
time: timestamp	2015-06-10T09:57:41.716+03:00

Attribute	Value
Event No	10
Activity Name	B
Payload Value	5
lifecycle: transition	Complete
time: timestamp	2015-06-10T21:41:50.338+03:00

In this specific case we have that A1 should be followed by C, C0, C1 or C0c1. For this purpose we use a branched response constraint shown in the Figure 5.4.

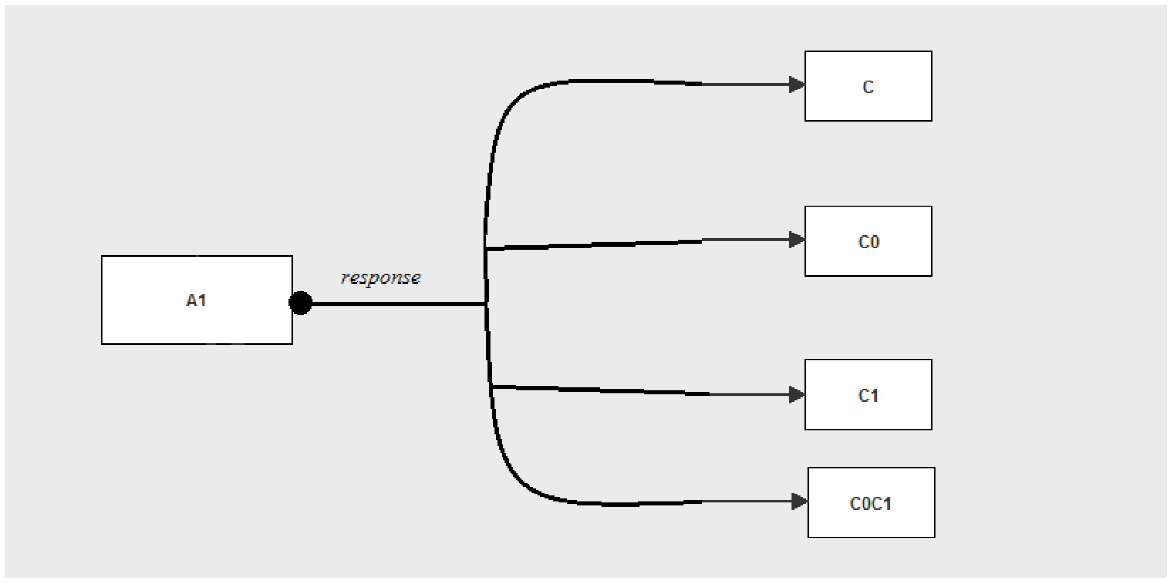


Figure 5.4 Branched Chain Constraints model

5.6 Performance

In order to evaluate the performance of the proposed approach, we have executed different process models containing different constraints and trace sizes to evaluate the duration of log generation. The maximum size of events per trace is ten while all data models have same constraints.

This assessment has been conducted on a machine equipped with Intel (R) Core (TM) i5-3437 CPU with 3.86 usable memory (RAM).We have used Eclipse and Java as the coding language for the implementation of the tool. The log generation time is slightly different for each run so the duration of log generation presented in the Table 5.3 is average of five executions.

Number of Constraints	Number of Traces			
	10	100	1000	2000
3	0.054 seconds	0.137 seconds	0.755 seconds	1.73 seconds
5	0.109 seconds	0.1756 seconds	0.8874 seconds	1.782 seconds
10	3.6144 seconds	3.686 seconds	4.4368seconds	5.3028 seconds

Table 5.3: Generation times with respect to number of constrains and trace size.

The first column of the performance table is representing the number of constrains in the models while second column is divided into sub columns based on trace sizes. The reader can see that the duration of log generation with small set of traces is much faster as compared to higher trace length. We can thus conclude that the performance effects based on the number of constrains and size of traces in the log.

6 Conclusion and Future Work

In this thesis we wanted to cover the following research questions:

- *How can we generate event logs from multi-perspective Declarative process models?*
- *What are the performances of the proposed approach when using Declarative model containing different numbers of constrains to generate event logs of different sizes?*

In response of our first research question, we tried to address the question by developing a tool that generates multi-perspective event logs of declarative process models. In addition, in this thesis, we have developed a method that translate data condition to linear equations and to solve these ILP equation we used *lp_solve*. The usage of ILP in this thesis *i)* detecting violating activities before generating MINERFul input model and *ii)* to set a data range for valid activities. This tools is very simple to use, users can generate event logs easily without any additional knowledge about the templates automaton and theorem.

Our second research question is about the performance of developed tools. Of course, real life process data may be containing a lot of constrains and to generate event logs of those models would be a challenging task. In the evaluation sections we have experimented with different number of constraints and trace size and the performance result showed that this tool is capable to generate large logs in a reasonable amount of time

Future Work

This tool can be improve in future by modification in the current implementation or adding new feature in the proposed application.

- **Data Condition:** Declare models mainly constituent of three data conditions. The approach proposed in this thesis focuses only on activation condition. However, this implementation can be easily extended to implement other data conditions i.e., correlation condition and time condition.
- **Logical Operators.** Currently, we are focusing only simple condition without any Logical operator. There is room to implement such logical operators in the data conditions.
- **Activity Naming**: In current implementation, during splitting the activities spaces or numeric numbers are not allowed in the activity names. The work can also be improved by allowing these characters.
- **Integer Linear programming:** For Integer Linear Programming we have used *lp_solve* and it does not support all the problems of linear equations. Furthermore, large equations or large size processes will affect the performance of this tool.
- **Interface:** The user interface of Declare Designer is not very intuitive and it can be improved in terms of usage. Sometime it is very difficult to differentiate two data conditions because of interface issue.

Acknowledgements

All the praises and thanks to Allah Almighty and all the best blessings be upon Holy Prophet (PBUH) who directed humanity to pursue knowledge without any discrimination.

I would also like to express my gratitude to University of Tartu, for providing excellent research environment at the institute. I would like to thank Supervisor Dr. Fabrizio Maria Maggi and Dr. Claudio (University of Vienna) who ignited critical abilities in me. I am indebted to them for their guidance, and significant assistance

I am very grateful to my colleagues at Betssongroup for their support and flexibility, without their flexibility this task was not possible.

Finally, I would like to thank my parents, family members, teachers and friends for their love, prayers, patience and endless support.

7 References

[1] H. Verbeek, J. Buijs, B. Van Dongen, and V. Der Aalst, “Xes, XESame, and Prom 6,” in *Information Systems Evolution*, vol. 72, Springer, 2011, pp. 60–75.

[2] A. W. Scheer, “Nüttgens. M.: ARIS Architecture and Reference Models for Business Process Management,” *van der Aalst, WMP; Desel, J.; Oberweis, A. Bus. Process Manag. Tech. Empir. Stud.*, vol. 1806, pp. 376–389, 2000.

[3] A. Scheer, “ARIS toolset: a software product is born,” *Inf. Syst.*, vol. 19, no. 8, pp. 607–624, 1994.

[4] van Dongen, “BPI challenge 2011,” 2011.

[5] van Dongen, “BPI challenge 2012,” 2012.

[6] C. Di Ciccio and M. Mecella, “On the discovery of declarative control flows for artful processes,” *ACM Trans. Manag. Inf. Syst.*, vol. 5, no. 4, p. 24, 2015.

[7] F. M. Maggi, “Declarative Process Mining with the Declare Component of ProM.,” in *BPM (Demos)*, 2013.

[8] F. M. Maggi, R. P. J. C. Bose, and W. M. P. van der Aalst, “Efficient discovery of understandable declarative process models from event logs,” in *Advanced Information Systems Engineering*, 2012, pp. 270–285.

[9] M. L. Bernardi, M. Cimitile, C. Di Francescomarino, and F. M. Maggi, “Using Discriminative Rule Mining to Discover Declarative Process Models with Non-atomic Activities,” in *Rules on the Web. From Theory to Applications*, Springer, 2014, pp. 281–295.

[10] M. Pesic and W. M. P. der Aalst, “A declarative approach for flexible business processes management,” in *Business Process Management Workshops*, 2006, pp. 169–180.

[11] P. Pichler, B. Weber, S. Zugal, J. Pinggera, J. Mendling, and H. A. Reijers, “Imperative versus declarative process modeling languages: An empirical

investigation," in *Business Process Management Workshops*, 2012, vol. 99, pp. 383–394.

[12] A. Burattin and A. Sperduti, "PLG: A framework for the generation of business process models and their execution logs," in *Business Process Management Workshops*, 2011, pp. 214–219.

[13] K. Jensen, L. M. Kristensen, and L. Wells, "Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems," *Int. J. Softw. Tools Technol. Transf.*, vol. 9, no. 3–4, pp. 213–254, 2007.

[14] K. M. van Hee and Z. Liu, "Generating Benchmarks by Random Stepwise Refinement of Petri Nets.," in *ACSD/Petri Nets Workshops*, 2010, pp. 403–417.

[15] G. Bergmann, Á. Horváth, I. Ráth, and D. Varró, "A benchmark evaluation of incremental pattern matching in graph transformation," in *Graph Transformations*, Springer, 2008, pp. 396–410.

[16] M. Pesic, H. Schonenberg, and W. M. P. der Aalst, "Declare: Full support for loosely-structured processes," in *Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International*, 2007, pp. 287–300.

[17] W. M. P. Van Der Aalst, *Process mining: discovery, conformance and enhancement of business processes*. Springer, 2011.

f

[18] M. Dumas, M. La Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of business process management*. Springer, 2013.

[19] C. Di Ciccio, M. Mecella, M. Scannapieco, D. Zardetto, and T. Catarci, "MailOfMine--analyzing mail messages for mining artful collaborative processes," in *Data-Driven Process Discovery and Analysis*, vol. 116, Springer, 2012, pp. 55–81.

[20] C. Di Ciccio, M. L. Bernardi, M. Cimitile, and F. M. Maggi, "Generating Event Logs Through the Simulation of Declare Models," in *Enterprise and Organizational Modeling and Simulation*, Springer, 2015, pp. 2–10.

[21] M. Pesic, "Constraint-based workflow management systems: shifting control to users," Technische Universiteit Eindhoven, 2008.

[22] T. Stocker and R. Accorsi, “Secsy: Security-aware synthesis of process event logs,” in *Workshop on Enterprise Modelling and Information Systems Architectures*, 2013, pp. 71–84.

[23] XES-standard. http://www.xes-standard.org. *"vii, 8, 9"*

[24] Laurent Y, Bendraou R, Baarir S, Gervais M-P. Planning for declarative processes. In: *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. ; 2014:1126-1133.

[25] http://javailp.sourceforge.net/

[26] http://lpsolve.sourceforge.net/5.5/

[27] https://github.com/cdc08x/MINERful

[28] C. Di Ciccio, M. H. M. Schouten, M. de Leoni, and J. Mendling, “Declarative Process Discovery with MINERful in ProM,” *BPM Demos*, pp. 60–64, 2015.

[29] https://github.com/cdc08x/MINERful

[30] M. Westergaard and T. Slaats, “Cpn tools 4: A process modeling tool combining declarative and imperative paradigms,” *Autom. Control Comput. Sci.*, vol. 47, no. 7, pp. 393–402, 2013.

[31] van Dongen, “BPI challenge 2014,” 2014.

[32] S. K. L. M. Vanden Broucke, J. Vanthienen, and B. Baesens, “Straightforward Petri Net-based Event Log Generation in ProM,” *Available SSRN 2489051*, 2014.

[33] Shugurov and A. A. Mitsyuk, “Generation of a set of event logs with noise,” in *Proceedings of the 8th Spring/Summer Young Researchers’ Colloquium on Software Engineering (SYRCoSE 2014)*, 2014, pp. 88–95.]

[34] http://web.mit.edu/15.053/www/AMP-Chapter-09.pdf

Appendix

I. Source Code

Find source code, Example models and generated log file at
https://github.com/ijlalhussain/LogGenerator

II. License

Non-exclusive licence to reproduce thesis and make thesis public