Generating Synthetic Procedural Multi-Perspective Electronic Healthcare Treatment Cases

Master Thesis

for obtaining the academic degree
Master of Science (M.Sc.)

Trier University
Department IV - Computer Science
Chair of Business Informatics II

UNIVERSITÄT
TRIER

1st Reviewer:	Prof. Dr. Ralph Bergmann
2nd Reviewer:	Joscha Grüger, M.Sc.
Supervisor:	Joscha Grüger, M.Sc.

Submitted on April 22, 2022 by:

David Jilg

Abstract

This thesis aims to develop an approach that allows generating realistic event logs based on *Data Petri nets (DPN)* that can be used as a substitute for missing real data when applying AI methods. Based on a requirement analysis, a literature review of already existing methods for synthetic data generation based on process models is conducted, and the *token-based simulation* method is chosen. Finally, the approach is developed, implemented, and evaluated.

The approach was implemented in the tool called *DALG: The Data Aware Event Log Generator*[1]. During the evaluation, it was found that it can generate event logs that conform to both the control-flow and the data perspective of a DPN. However, achieving the generation of realistic event logs could not be reached. It was found that it is difficult to describe processes accurately enough in DPNs to generate realistic data since they lack the necessary expressiveness. Nonetheless, the approach shows promise for generating realistic event logs. However, further research regarding the problems uncovered in this thesis is necessary to improve the realism of the synthetic data. The main contributions of this thesis are the identification of challenges that occur when trying to generate realistic data based on Data Petri nets, solutions for several of the uncovered problems, and the tool DALG.

Keywords: Process Mining · Data Petri Net · Event Log · Synthetic Data · Event Log Generation · Token-based simulation · Artificial Intelligence

1. https://github.com/DavidJilg/DALG

Table of Contents

1 Introduction **1**
1.1 Research Objective 2
1.2 Methodology 4
1.3 Thesis Structure 4

2 Fundamentals **5**
2.1 Process Modelling 5
2.1.1 Procedural and Declarative Process Modelling 5
2.1.2 Petri Nets 6
2.1.3 Data Petri Nets 9
2.1.4 Petri Net Markup Language 10
2.2 Process Mining 12
2.2.1 Event Logs 12
2.2.2 Process Discovery 14
2.2.3 Conformance Checking 14
2.2.4 Process Enhancement 16
2.3 Related Work 16

3 Preliminary Work for the Approach Development **18**
3.1 Requirement Analysis 18
3.1.1 DPN Modelling Tool Identification 18
3.1.2 Challenge Identification 19
3.1.3 Choosing a Exemplary Data Petri net Model 20
3.1.4 Requirement Specification 21
3.2 Literature Review 26
3.2.1 Database Identification and Selection 26
3.2.2 Search Results 29
3.2.3 Identified Approaches 31
3.3 Selection Process 34
3.3.1 Approach Comparison 34
3.3.2 Approach Selection 38

4 Approach Development **40**
4.1 Methodology 40
4.2 Generating Conforming Traces 41
4.3 Generating Realistic Variable Values 42
4.4 Approach for Generating Event Logs from Data Petri Nets 44

5 Conception and Implementation **46**

5.1 Conception Methodology . . . 46
5.2 Concept Overview . . . 47
5.3 Entry and Exit Point . . . 49
5.4 Model Representation . . . 52
5.4.1 Places and Markings . . . 52
5.4.2 Transitions . . . 54
5.4.3 Variables . . . 54
5.5 Simulation . . . 56
5.5.1 Simulation Approach . . . 57
5.5.2 Simulation Configuration . . . 58
5.6 Algorithmic Approach . . . 60
5.6.1 Evaluating Transition Guards . . . 60
5.6.2 Generating Random Traces . . . 63
5.6.3 Generating Variable Values . . . 65
5.6.4 Generating All Possible Traces . . . 67
5.7 Implementation . . . 69
5.7.1 Implementation Methodology . . . 69
5.7.2 DALG: The Data Aware Event Log Generator . . . 70

6 Evaluation 73
6.1 Overview . . . 73
6.2 Conformance of the Generated Traces . . . 74
6.2.1 Process Models . . . 74
6.2.2 Methodology . . . 74
6.2.3 Results . . . 75
6.3 Generating All Traces Possible in a DPN . . . 76
6.3.1 Methodology . . . 77
6.3.2 Results . . . 77
6.4 Realism of the Generated Event Logs . . . 78
6.4.1 Methodology . . . 78
6.4.2 Comparison to Real Event Logs . . . 79
6.4.3 Meaningfulness from the Perspective of a Domain Expert . . . 85
6.5 Configuration Effort . . . 89
6.6 Discussion . . . 90

7 Conclusion and Future Work 94

A Relevant Literature Works xvi

B Synthetic Traces for the Evaluation xix

List of Figures

2.1 Valid connections in a Petri net 6
2.2 Transition firing in a Petri net 7
2.3 And-Split and AND-Join in a Petri net 8
2.4 XOR-Split and XOR-Join in a Petri net 8
2.5 Exemplary Data Petri net 10
2.6 Exemplary Petri Net Markup Language file 11
2.7 Definition of places, transitions, and arcs in a PNML file 12
2.8 Exemplary XES file 13
2.9 Conformance checking through alignments 15

3.1 Data Petri net depicting the treatment of melanoma. 23

4.1 Approach to generate event logs from DPNs 44

5.1 Class diagram for the concept 48
5.2 Entry and exit point of the concept 50
5.3 Definition of a transition in a PNML file 51
5.4 Internal model representation of the concept 53
5.5 Simulation part of the concept 56
5.6 Algorithmic approach of the MILP-Solver 63
5.7 Graphical user interface of DALG 72

6.1 Illustration of the requirement model 75
6.2 A trace that describes the daily activities of a person 80
6.3 Occurrences of activities in the real and synthetic event log 81
6.4 Lead times in the real and synthetic event log 82
6.5 Exemplary trace for the treatment of melanoma 85

List of Tables

3.1 Challenges present in the requirement model 22
3.2 Results of the database assessment 28
3.3 Suitability of the identified approaches 39

6.1 Results of conformance checking with the MPE 76
6.2 Results of comparing the synthetic and real event log 84

A.1 Relevant literature found during the literature review xvi

List of Algorithms

1 Algorithmic approach of the MILP-Solver 61
2 Algorithmic approach of the random trace generation 64
3 Algorithmic approach for generating realistic variable values 66
4 Algorithmic approach to generate all traces of a DPN 68

List of Definitions

1 Petri net 6
2 Data Petri net 9
3 SCIFF Specification [LCCM20] 32

List of Abbreviations

AI Artificial Intelligence

ALP Abductive Logic Programming

BPMN Business Process Model and Notation

CDSS Clinical Decision Support Systems

CNF Conjunctive Normal Form

CPN Colored Petri Net

CSP Constraint Satisfaction Problem

DALG The Data Aware Event Log Generator

DFKI German Research Center for Artificial Intelligence

DOAJ Directory of Open Access Journals

DPN Data Petri net

FSA Finite State Automata

GNU GLP GNU General Public License

GUI Graphical User Interface

IC Integrity Constraint

ID Identification

IEEE Institute of Electrical and Electronic Engineers

JSON JavaScript Object Notation

MDPI Multidisciplinary Digital Publishing Institute

MILP Mixed Integer Linear Programming

MPE Multi-perspective Process Explorer

PM Process Mining

PNML Petri Net Markup Language

ProM Process Mining Toolkit

RNG Random Number Generator

RQ Research question

SAT Boolean Satisfiability Problem

SESE Single Entry Single Exit

SN Student Number

XES eXtensible Event Stream

1. Introduction

Artificial Intelligence (AI) is a disruptive technology in healthcare, which is of increasing importance in this domain [LLXN19]. It has the potential to improve the efficiency of health care systems and create benefits for physicians and patients [LLXN19]. For example, AI allows for more accurate and rapid diagnosis, decision-making support (for example, through *Clinical Decision Support Systems (CDSS)* [Alug16]), and treatment that is explicitly conformed to the individual patient needs [HRMK19].

One AI method that is of growing importance in the healthcare domain and that is the focus of this thesis is *process mining* [GVLC$^+$18]. Process mining aims to analyse procedural data to discover, monitor and improve real-world processes [vdAa16]. Most AI methods and especially data-driven AI methods, such as process mining, heavily rely on large datasets. Massive data is a fundamental aspect of these methods [PMSP$^+$21] and is needed to provide robust and reliable results.

The reliance of AI methods on large datasets poses a problem in the healthcare domain since there are several socio-economic, technical, and legal restrictions on data collection [PMSP$^+$21], as well as privacy concerns regarding personal patient data.

A possible solution to this problem could be to use *synthetic data* as a replacement for real datasets that are too sensitive for release [RBBW$^+$20]. In a study conducted by Rankin et al., it was shown that training models with synthetic health care data only leads to small and manageable decreases in accuracy compared to models that were trained with real data [RBBW$^+$20]. Synthetic data, therefore, could unlock new potential within the healthcare domain [RBBW$^+$20].

In the case of process mining, the procedural data it relies on are so-called *event logs*. Event logs contain sequences of activity that are part of a process, such as the treatment of patients. Event logs can be used to discover models of the processes contained in event logs. One of the most common process model types used to describe processes are *Petri nets*. There are already several approaches, such as *token-based simulation* (for example, [vBVB14]), and tools that allow the generation of event logs from Petri nets that are needed for process enhancement or improving of process discovery algorithms (see section 1.1 for some examples). However, these approaches focus on process models that only describe the *control-flow perspective* of processes. This means they only consider the sequence of activities in a process and do not consider what information leads to the specific activity sequence. However, there are also process models which consider additional perspectives. For example, *Data Petri nets* consider the control-flow and the data perspective. The data perspective can be very useful, especially in the medical domain, since the decision to perform certain treatment activities is performed based on diagnostic data.

For example, in a process model depicting the treatment of patients, it would be advantageous to know why a specific treatment activity is performed. A process model covering the data perspective could provide this information by containing variables depicting the diagnosis results. To the best of the authors' knowledge, there is no automatic way to generate synthetic multi-perspective event logs based on Data Petri nets. The lack of approaches and especially the lack of tools to create synthetic data from process models which are not limited to the control-flow perspective creates a problem when dealing with Data Petri nets in the medical domain. Due to the previously mentioned constraints that make data collection in this field very difficult, there is a distinct lack of data that, for example, can be used to enhance processes that are modelled with the use of Data Petri net models.

The contribution that this thesis aims to provide is the mitigation of the previously illustrated problem. Therefore, the main research objective of this thesis is to develop an approach to automatically generate synthetic procedural data, i.e. event logs, from Data Petri nets.

1.1 Research Objective

As previously mentioned, to the best of the authors' knowledge, there is no automatic way to generate synthetic event logs based on Data Petri nets. Therefore, the research objective of this thesis consists of developing (theoretically) and implementing a approach to generate synthetic multi-perspective event logs from Data Petri nets. The event logs should conform to the model and be indistinguishable from authentic event logs. In the following, the main challenges associated with the research problem are briefly described.

The approach that is to be developed to generate the event logs needs to be able to deal with the additional challenges stemming from including the data-perspective in the process models. These additional challenges include, among other things, dealing with variables, e.g. writing and reading variables, evaluating transition guards, and being able to generate **realistic data** that conforms to the process model. There are already many algorithms/tools to generate synthetic event logs from Petri nets that are limited to the control-flow perspective. For example, there are several plug-ins for the *Process Mining Toolkit (ProM)*[1] [vBVB14, ShMi14], as well as stand-alone tools like *CPN Tools* [RWLL+03], *SecSy* [StAc13], and Gena 2 [ShMi14] or even whole frameworks like *PM4PY*[2] that allow synthetic event log generation from Petri nets. Several of the previously mentioned tools and frameworks are at least partially open-source or their algorithms are described in scientific literature (ProM Plug-in Loggenerator[3], Gena 2[4], CPN Tools[5], PM4PY[6]). Therefore, the existing algorithms of these tools may be used as a basis for the developed approach. If an existing algorithm is deemed suitable as a basis, the challenge arises to adapt it to work with Data Petri nets. If no suitable algorithms are found, the challenge is to develop one from the ground up.

1. https://promtools.org/ (ProM is introduced in section 2.3)
2. https://pm4py.fit.fraunhofer.de/
3. https://github.com/Macuyiko/processmining-loggenerator
4. https://github.com/pavelperc/newgena
5. https://cpntools.org/2018/01/19/source-code/
6. https://github.com/pm4py

Another challenge arises from evaluating the logical expression present in the transition guards of Data Petri nets. A *Mixed Integer Linear Program (MILP)* solver may be needed to solve these expressions. However, the development of such a solver is not trivial. Therefore, the implementation of the MILP presents a non-trivial challenge.

Research Questions

This thesis will try to answer the following research questions (RQs) that arise from the previously described research problem.

RQ1 Is it possible to automatically generate realistic synthetic data, i.e. event logs, from Data Petri nets?

- **RQ1.1** How can realistic synthetic data that conforms to the modelled process be automatically generated from a Data Petri net?

RQ2 What challenges arise when trying to generate realistic synthetic data from Data Petri nets automatically?

RQ3 How can it be evaluated if synthetic data that was automatically generated from a Data Petri net is semantically correct?

The first research question concerns whether it is possible to automatically generate realistic data from Data Petri nets. Although the research goal of this work is to develop a approach for realistic data generation from Data Petri nets, it is not yet known whether this is possible. In order to answer this research question, it also needs to be determined how the generation of realistic event logs from Data Petri nets can be achieved. In this context *realistic* refers to whether the data generated for the attributes of events and the sequence of events make sense on a semantic level. For example, consider a process model from the medical domain that describes the treatment of patients and has a variable depicting the patients' age. In order to generate realistic event logs from this model, a realistic value has to be generated for this variable. The age variable in this exemplary model would probably be of the type 'integer', and the range of legitimate values is therefore very large. However, the value space that makes sense on the semantic level is much smaller since having a patient over 120 years makes little sense.

The second research question concerns the challenges of generating synthetic data from Data Petri net. To develop an approach, the challenges that arise when trying to generate realistic event logs based on Data Petri nets need to be identified. They are needed to assess the suitability of existing approaches and direct the development.

Finally, the third research question will need to be answered to evaluate whether the approach can generate realistic data. During the evaluation, the data generated by using this approach has to be evaluated in terms of being semantically correct and conforming to the process model from which it was generated. Therefore, it is necessary to determine how the synthetic data's semantic correctness, i.e. realism, can be evaluated.

1.2 Methodology

This section briefly presents the methodological approach used to achieve the previously described research objectives. The first step in the methodology consists of conducting a requirement analysis to identify the requirements that the approach developed to generate synthetic event logs from Data Petri nets has to fulfil to generate realistic data. Subsequently, a comprehensive literature review regarding synthetic procedural data generation is conducted to identify methods that could be used as a basis for the approach developed in this thesis. The identified methods are then compared by using the previously established requirements. If an already existing method is suitable as a basis, it is used to develop the approach. If no method is deemed suitable, the approach is developed from the ground up. The developed approach is then implemented, and the implementation is used to evaluate whether the approach can satisfy the requirements and generate realistic synthetic data.

1.3 Thesis Structure

This section provides a brief overview of the structure of this thesis. Chapter 2 explains the fundamentals of Process Modelling and Process Mining. Additionally, related literature works are introduced. Chapter 3 provides a detailed description of the preliminary work for the development of the approach to generate synthetic data. This includes the requirement analysis, literature review, and the method selection process. During the selection process, a method is selected, which is used as a basis for the developed approach. Chapter 4 covers the development of the approach and the approach itself. Chapter 5 concerns the implementation of the developed approach. Therefore, this chapter presents the concept developed for the implementation and the implementation itself. Chapter 6, concerns the evaluation of the approach. First, the methodology used for the evaluation is described. Subsequently, the evaluation results are presented, and a conclusion is drawn. Lastly, chapter 7 presents a concluding assessment of the thesis and discusses potential future works.

2. Fundamentals

This chapter provides the fundamental knowledge necessary to comprehend this thesis. The chapter is structured as follows. First, section 2.1 provides basic information about process modelling, including the difference between procedural and declarative modelling, Petri nets, and the Petri Net Markup Language. Subsequently, section 2.2 covers the process mining domain and therefore provides basic information about event logs and process mining techniques. Lastly, section 2.3 provides an overview of scientific works related to this thesis.

2.1 Process Modelling

New technical innovations like computers and digital communication infrastructure have influenced business processes since the early 1950s and are still the main driver behind changing business processes today [vdAa16]. The complexity of business processes has, therefore, greatly improved, and these processes are relying more than ever on information systems spanning multiple organisational groups [vdAa16]. *Process modelling* and therefore *process models* can provide the necessary insights to deal with the increasing complexity of these processes.

The first step in process modelling is to identify the purpose of the model and specify the research objective. The purpose can then be used to determine the abstraction granularity of the model and the features of the process that should be included in the model to achieve the research objective. The last step before the implementation of the conceptualised model is to choose a *Process Modelling Language*, e.g., *Business Process Model and Notation (BPMN)* or Petri Nets, that fits the needs of the model has to be chosen.

In the following, section 2.1.1 describes the two types of process modelling techniques used to create process models. Subsequently, section 2.1.2 and section 2.1.3 explain the process model languages, i.e., *Petri nets* and *Data Petri nets*, that are fundamental to this thesis. Additional, section 2.1.4 covers the *Petri Net Markup Language* that is widely used to store and exchange Petri net models[Kind04].

2.1.1 Procedural and Declarative Process Modelling

There are two common approaches to modelling a process, namely *procedural* (also called *imperative*) and *declarative* modelling. When modelling a process with a procedural language, such as Petri nets, the goal is to specify the procedures of processes. Therefore, a procedural model specifies all the valid sequences of activities that are possible. This means that every sequence of activities not specified in the model is not valid. One example of a procedural modelling language is the Petri net notation, which will be introduced in section 2.1.2. In contrast to procedural models, declarative models describe a process by specifying what should be done in order to achieve a goal and thus not specifically describing how the end state should be reached [GoVC13]. To achieve this, these types of models contain a set of constraints that describe properties and dependencies between the activities in the modelled process [GoVC13]. Declarative models, therefore, do not directly specify all sequences of activities possible in a process.

2.1.2 Petri Nets

The concept of Petri nets was first introduced by Carl Adam Petri in 1962 [Mura89] as part of his dissertation [Petr62] . Petri nets are a graphical as well as a mathematical tool that have been proposed for a wide variety of applications, like the modelling of distributed and concurrent systems [Mura89] or, more recently, the modelling of business processes and as a fundamental part of process mining. This section will introduce the fundamental properties of Petri nets in the context of process modelling and process mining.

Petri nets are process models that describe the control-flow perspective of a process while ignoring all other perspectives [Mann18]. The following definition was taken directly from the thesis ‘Multi-perspective Process Mining’ by Felix Mannhardt [Mann18].

Definition 1 (Petri net)**.** *A Petri net is a triple (P, T, F) where*

- *P is a finite set of places,*
- *T is a finite set of transitions, and*
- *$F \subseteq (P \times T) \cup (T \times P)$ is a set of flow relations that describe a bipartite graph between places and transitions.*

As described in definition 1 a Petri net consists of places and transitions that are connected by direct edges, which are usually called *arcs.* A Petri net, therefore, is a bipartite graph between places and transitions [Mann18]. The edges can only connect a place to a transition and transition to a place. Places can therefore not be directly connected to other places, and the same goes for transitions. Figure 2.1 shows a transition connecting two places through valid arcs. In this case the place ‘p1’ is called the *input place* of the transition ‘t1’ and ‘p2’ is the *output place.* As

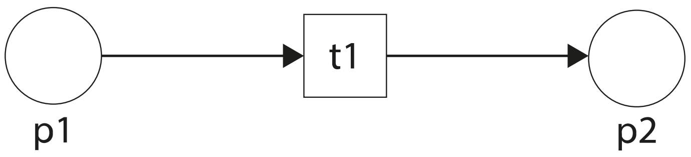


Figure 2.1: Valid connections between the places ‘p1’ and ‘p2’ and the transition ‘t1’.

can be seen in figure 2.1, places are usually portrayed as a circle, and transitions are portrayed as a square or a rectangle. The arcs are usually represented by a line with an arrow at one end that indicates the direction of the arc. In Petri nets, places are passive elements used to indicate the model's state, while transitions are active elements that allow the model to change its states.

In the following, the functionality of Petri net states and state transitions is further described. A place of a Petri net can contain so-called *tokens*. The tokens are graphically represented by a black dot inside the circle that represents the place. These tokens represent the current state of a Petri net. A state is called a *marking* and describes the number of tokens in every place of the net. The placement of tokens determines which transition can be used to change the state of the Petri net. A transition is *enabled* if there is at least one token in each of its input places. If a transition is enabled, it can *fire* to change the state of the Petri net. There is no defined order of firings if multiple transitions are enabled simultaneously. Petri nets that have states with multiple enabled transitions are therefore non-deterministic. When a transition fires, it consumes one token from each of its input places and creates a token in each of its output places. Therefore, the firing of a transition changes the marking and thus the state of a Petri net. Figure 2.2 shows three possible states for an exemplary Petri net to illustrate the previously described concepts. In the state on the left in figure 2.2, the transition 't1' is not enabled

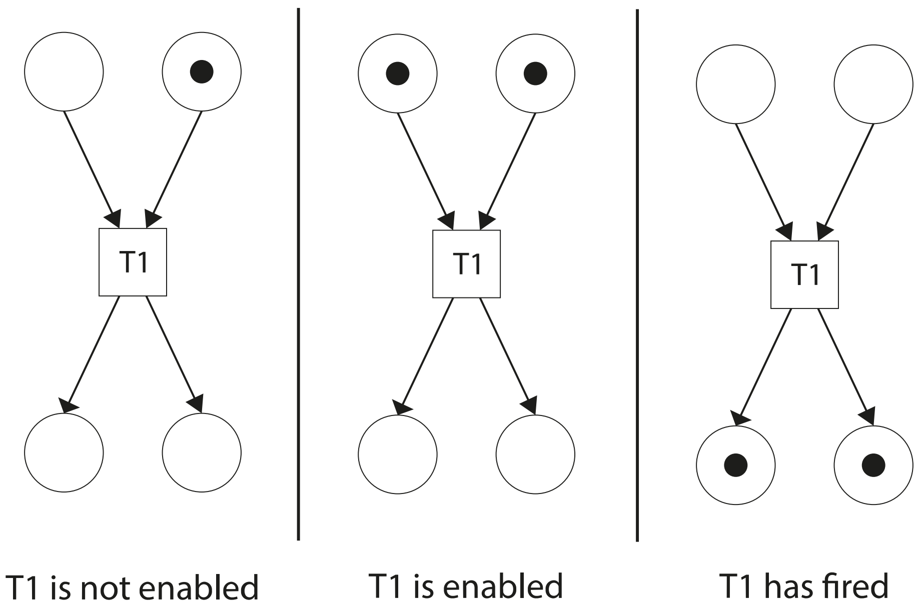


Figure 2.2: Basic example for the firing concept of transitions in a Petri net

because there is no token in its input place 'p1'. In the state, in the middle of the figure, the transition is enabled. The illustration on the right shows the state of the Petri net after the transition 't1' has fired. The state of a Petri net before any transition has fired is called the *initial marking*. A Petri net state where no transition is enabled and therefore no state change can take place is called a *final making*. The set of markings that can be reached from a given marking by firing transitions is called the set of *reachable markings* of the given marking.

Four basic modelling constructs can be used to direct the control-flow of a Petri net. The *AND-Split* and *AND-Join* constructs allow the creation of concurrent paths in a Petri net that can be executed in parallel. Therefore, these constructs are used when multiple activities or activity sequences should be executed simultaneously. Figure 2.3 shows an exemplary usage of an AND-Split to split a single path into two concurrent paths that are later merged again with an AND-Join. In this

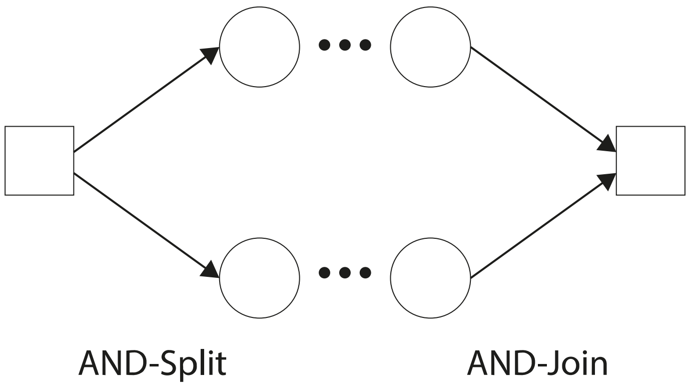


Figure 2.3: Concept of the AND-Split and AND-Join in a Petri net

example, the transition 't1' that is coming from the single path creates a token in each of its output places 'p1' and 'p2' when it is fired. Therefore, the activities that follow these places can now be executed simultaneously and independently. In order to merge the two concurrent paths into a single path again, the transition 't2' has to be enabled. Therefore, the join requires both paths to be completed before they can be merged again.

The *XOR-Split* and *XOR-Join* constructs allow the creation of exclusive paths in a Petri net from which only one path can be executed. Therefore, these constructs are used when there are multiple activities or multiple sequences of activities from which only one should be executed. Figure 2.4 shows an exemplary usage of an XOR-Split to split a single path into two exclusive paths that are later merged again with an XOR-Join. When there is only one token in the place 'p1', only one of the two

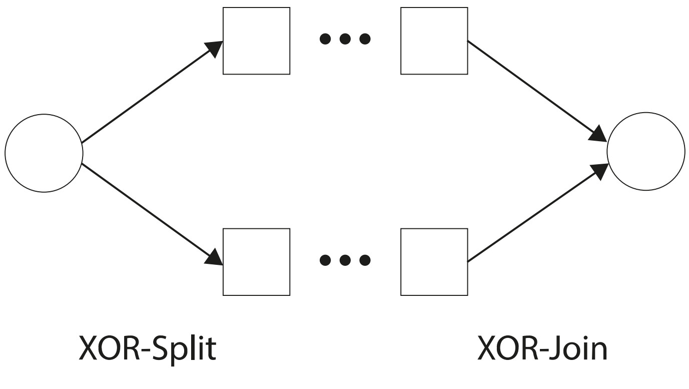


Figure 2.4: Concept of the XOR-Split and XOR-Join in a Petri net

transitions 't1' and 't2' can be fired, thus resulting in an exclusive choice about which of the activities are executed. Unlike the AND-Join, the XOR-Join only needs one of the paths to be completed in order to proceed since both transitions 't3' and 't4' can create a token in the place *p2*.

2.1.3 Data Petri Nets

The previously described Petri net concept uses only one type of tokens and is therefore only able to model the control-flow perspective of a process [Reis13]. These type of Petri nets are called *elementary* Petri nets [Reis13]. Since the introduction of the 'classical' Petri net notation, several expansions have emerged. For example, so-called *coloured Petri nets* allow distinguishing between individual tokens by giving each token a data value of a specific type [Jens97]. These extensions have made it possible to expand Petri nets beyond the single perspective that only concerns the control-flow. In this section, the notion of *Data Petri nets* is presented.

A *Data Petri net (DPN)* is a Petri net that also models the data perspective of a process in addition to the control-flow perspective. There are several slightly different definitions and notations for Data Petri nets in the scientific literature. However, they tend to have in common that a Data Petri net has global variables that can be read and written by transitions and used in so-called *transition guards* to direct the control-flow. The following definition for Data Petri nets was taken from the paper 'Data-Aware Process Mining: Discovering Decisions in Processes Using Alignments' by Massimiliano de Leoni and Wil M.P. van der Aalst [dLvdA13].

Definition 2 (Data Petri net)**.** *A Data Petri net (DPN) N = (P, T, F, V, U, R, W, G) consists of:*

- *a Petri net (P, T, F);*
- *a set V of variables;*
- *a function U that defines the values admissible for each variable $v \in V$, i.e. if $U(v) = D_v, D_v$ is the domain of variable v;*
- *a read function $R \in T \to 2^V$ that labels each transition with the set of variables that it must read;*
- *a write function $W \in T \to 2^V$ that labels each transition with the set of variables that it must write;*
- *a guard function $G \in T \to G_V$ that associates a guard with each transition.*

As described in the definition, a Data Petri net has a set of variables that each has a domain that defines the possible values a variable can have. Additionally, a guard is assigned to every transition (If a transition has no guard, an empty guard is assigned). Lastly, every transition can write values to variables and read the current values of variables. In Data Petri nets, the status of a transition, i.e., enabled or not enabled, is determined by the number of tokens in its input places **and** additionally by the current values of the variables that it reads in regards to its guard. A guard consist of a logical expression that uses the read variables (for example, '*variable* $x \geq 10$'). If there are enough tokens in the transitions input places and the logical expression in the guard can be evaluated as *true* with regards to the current variable values, the transition is enabled. The remaining concepts, such as firing transitions, are the same as those introduced for Petri nets limited to the control-flow perspective.

Unlike Petri nets, Data Petri nets do not have one graphical representation that has been established as a standard. The graphical representation used in this thesis is based on how ProM displays Data Petri nets. In graphical representations of Data Petri nets in ProM, variables are usually displayed as yellow elongated hexagons with the variable name written inside the hexagon. A transition writing to a variable is usually represented by a dotted arrow pointing from the transition to the variable. Transitions that read variables are represented by a rectangle that has two outlines. Some graphical representations also have dotted arrows pointing from the variables to the transition that read them. The transition guards are usually written in the plain text next to the rectangle. Transition guards are sometimes also written in a table next to the model.

Figure 2.5 shows a simple exemplary Data Petri net representing a workflow for the treatment of patients coming into an emergency room. In this example, the

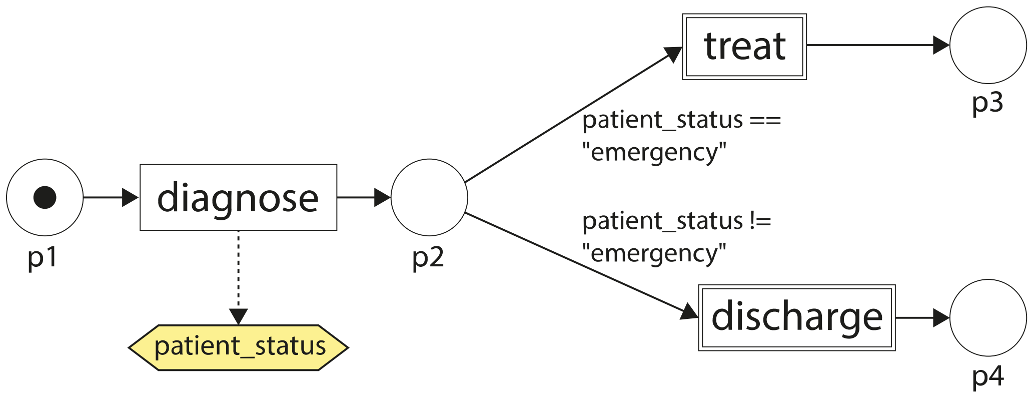


Figure 2.5: Exemplary Data Petri net for patients coming into an emergency room

patient is examined through the 'diagnose' transition, and it is determined whether the patients' condition constitutes an emergency. The transition writes the results of the examination to the global variable 'patient_status'. Subsequently, the next step in the treatment process is determined by the value of this variable. Both transitions that could be enabled after the transition 'diagnose' is fired are equipped with a transition guard. When the condition of the patient constitutes an emergency, only the transition 'treat' is enabled. If the condition does not constitute an emergency, the transition 'discharge' is enabled.

2.1.4 Petri Net Markup Language

The *Petri Net Markup Language (PNML)*[1] is interchange format for Petri nets that is widely accepted [Kind04]. It is an XML-based language that supports any Petri net type since it allows the definition of custom types [WeKi03]. In the following, the structure of an PNML file is described. Figure 2.6 shows the PNML file of the Data Petri net that was previously illustrated in figure 2.5. Due to the size of

the file, the figure is missing the definitions of places, transitions, arcs, and one of the two final markings. An example of how the definition of places, transitions,

1. https://www.pnml.org/

```
 1 <pnml>
 2    <net id="net1" type="http://www.pnml.org/version-2009/grammar/pnmlcoremodel">
 3       <name>
 4          <text>DPN_Name</text>
 5       </name>
 6
 7       <page id="n0">
 8          [...]<!-- Place, Transition and Arc defnitions -->
 9       </page>
10
11       <finalmarkings>
12          <marking>
13             <place idref="p1"> <text>0</text> </place>
14             <place idref="p2"> <text>0</text> </place>
15             <place idref="p3"> <text>0</text> </place>
16             <place idref="p4"> <text>1</text> </place>
17          </marking>
18          [...]<!-- Second final marking -->
19       </finalmarkings>
20
21       <variables>
22          <variable type="java.lang.String">
23             <name>patient_status</name>
24          </variable>
25       </variables>
26    </net>
27 </pnml>
```

Figure 2.6: Petri Net Markup Language file of the Data Petri net shown in figure 2.5

and arcs would look like in a PNML file can be seen in figure 2.7. Additionally, a PNML file usually contains graphical information that defines how the Petri net should be visualized. This information was also removed from the file due to the size constraints in the figure. However, the graphical information does not affect the process described in the model. Therefore, it is irrelevant when generating synthetic event logs from a Petri net and can be disregarded.

As can be seen in figure 2.6 a PNML file consists of one or more Petri nets that are encapsulated in a `<net>` tag. Each Petri net has an *id* as well as a *type* that are defined in the opening `<net>` tag. Additionally, a Petri net can have a name that is defined inside a `<name>` tag. The definition of the actual Data Petri net consists of three main parts. First, the places, transitions, and arcs are defined. Usually, these definitions are organized in pages with optional subpages that are defined by the `<page>` tag that can be seen between lines 7 and 9 in figure 2.6. The second part of a PNML file for a Data Petri net defines the valid final markings. These can be seen between lines 11 and 19 in figure 2.6. The definition of a final marking consists of the number of tokens present in every place of the Petri net. The third part of the Data Petri net description concerns the variables. The variable definitions are encapsulated in the `<variables>` tag that can be seen in line 21 in in figure 2.6. Every variable has a name by which it is referenced and a type. Additionally, a numeric variable can have a defined minimum and maximum value.

Figure 2.7 shows a snippet of the place, transition, and arc definitions that were left out in figure 2.6. A place is defined with an id, a name, and the number of tokens

```
<place id="p1">
  <name>
     <text>p1</text>
  </name>
  <initialMarking>
     <text>1</text>
  </initialMarking>
</place>

<transition guard="(patient_status=='emergency')" id="t2">
  <name>
     <text>treat</text>
  </name>
  <readVariable>patient_status</readVariable>
</transition>

<arc id="arc1" source="p1" target="t1">
  <name>
     <text>1</text>
  </name>
</arc>
```

Figure 2.7: Definition of places, transitions, and arcs in a Petri Net Markup Language file

present in this place in the initial marking of the Petri net. If a place contains no tokens in the initial marking, the `<initialMarking>` tags are usually left out of the place definition. A transition is also defined with an id and a name. Additionally, a guard string can be included in the initial `<transition>` tag. If a transition reads or writes variables, the variables are referenced by using a `<readVariable>` or `<writeVariable>` tag that includes the name of the variable.

2.2 Process Mining

Process Mining (PM) is a research field that has emerged in the last decade [vdAa16]. It has the goal to discover, monitor, and enhance processes by gaining knowledge from information systems. PM techniques have become more mature over the last decade as procedural data has become more readily available than ever [vdAa16]. It has the has the potential to optimise processes, detect bottlenecks and reduce costs [BaSo18], by providing valuable insights about processes [GhAm16]. This section is structured as follows. First, section 2.2.1 gives insights into the procedural data used in PM. Subsequently, sections 2.2.2, 2.2.3, and 2.2.4 cover the three types of process mining techniques (*discovery*, *conformance*, *enhancement*) used in PM.

2.2.1 Event Logs

The most common type of procedural data used in process mining are *event logs*. Event logs are records of events related to specific activities in a process. Each event has an attribute that relates to the activity and a timestamp dating the execution. Additional, events can have an arbitrary number of optional attributes, like, for example, information regarding the person executing the activity. Each event is part of a *case*, which represents a chronological *trace* of activities. The different cases of an event log are independent and can occur in parallel.

The standard that is most prominently used to structure event logs is defined in the *IEEE Standard for eXtensible Event Stream (XES) for Achieving Interoperability in Event Logs and Event Streams* [IEEE16]. Figure 2.8 shows an XML file containing an exemplary event log snippet in the XES format. The main element in the XML

```
<log xes.version="1.0" xes.creator="David Jilg">
   <trace>

      <string key="concept:name" value="Case1"/>

      <event>
         <string key="concept:name" value="diagnose patient"/>
         <string key="org:resource" value="Dr. John Doe"/>
         <date key="time:timestamp" value="2021-10-24T14:32:00.000+01:00"/>
         <string key="name" value="Jane Doe"/>
         <int key="age" value="50"/>
      </event>

      <event>
         <string key="concept:name" value="discharge patient"/>
         <string key="org:resource" value="Dr. John Doe"/>
         <date key="time:timestamp" value="2021-10-2415:06:00.000+01:00"/>
         <string key="name" value="Jane Doe"/>
         <int key="age" value="50"/>
         <boolean key="emergency" value="false"/>
      </event>

   </trace>
</log>
```

Figure 2.8: Exemplary event log snippet in the XES format

file is the *log* element. The opening bracket of the log element usually contains information about the XES version, the XES *name-space* and the creator of the log. In this example, the log element contains a single case that is represented by the trace element with the name ‘Case1’. The name of the case is defined by the ‘concept:name’ attribute located inside the `<trace>` element. In most event logs, the `<log>` element would additionally contain general information about the log or the specific structure of the traces and events. For example, the log could contain information about which XES extensions are used or which attributes events and traces are expected to have. In this example the case ‘Case1’ consists of two events, namely ‘diagnose patient’ and ‘discharge patient’. Each event consists of multiple attribute values. Each attribute is defined by an attribute type, a key and a value. The definition always begins with the attribute type after the opening bracket. The types that are allowed for attribute definitions are *string*, *date*, *int*, *float*, *boolean*, *id* and *list*. Subsequently, the key of the attributes is defined. This key represents the attribute's name. Lastly, the attribute's value is defined before a closing bracket ends the definition. In the case of this example, the attribute 'concept:name' refers to the executed activity.

2.2.2 Process Discovery

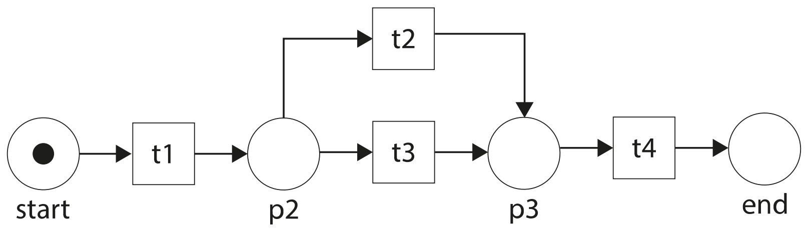
Process Discovery is a process mining technique that aims to generate a process model, e.g. a Petri net, that is representative of the behaviour seen in a given event log [vdAa16]. The process model that is discovered unveils the structure, i.e. the possible activity sequences, behind the seen behaviour and can thereby be used to analyse and optimise processes. Process discovery is thus the first actual step in a process mining workflow, after the collection and eventual preparation of process data, and forms the basis for conformance checking and process enhancement. There are many tools, such as the ProM[2] or *Disco*[3] that allow the discovery of process models from event logs. These tools also allow visualising the process model, analysing its properties, and using it for conformance checking and process enhancement.

2.2.3 Conformance Checking

Conformance checking is used to relate events in event logs to activities in a process model and compare whether there are commonalities or discrepancies between the behaviour described in the model and the behaviour observed through the event logs [vdAa16]. A process model can either be of normative or descriptive character [vdAa16]. If the model is of the latter type, it describes the observed behaviour in an event log. In the case of descriptive models, conformance checking can check whether the models' description of the seen behaviour is accurate. If a process discovery algorithm generated the model, conformance checking could also be used to evaluate the suitability and performance of the process discovery algorithm. Additionally, conformance checking can be used to repair models that do not accurately describe reality [vdAa16]. If the model is of normative character, it presents a guideline of what behaviour is expected to occur. In this case, conformance checking can be used to find undesirable deviations in captured event logs [vdAa16]. For example, conformance checking could be used to find fraud or inefficiencies by replaying an event log on top of the model [vdAa16].

Replaying an event log on top of a model refers to the conformance checking approach *token-based replay*. Token-based replay is one of the most used conformance checking approaches for Petri nets. It is performed by firing transitions in the order that is specified in the given event log [vdAa16] and checking if the activity sequence is valid. A discrepancy is found if, for example, a disabled transition is indicated to fire or if a trace does not reach a valid final marking. Therefore, a token-based replay is an approach that can differentiate between fitting and non-fitting traces [vdAa16]. However, the approach can only be used on Petri nets since it takes advantage of the token mechanism.

Another conformance checking approach that is prominently used and that is not limited to Petri net process models is conformance checking through *alignments*. Figure 2.9 shows a simple example of the optimal alignment of an exemplary event log and process model. In the presentation of the alignments, the top row shows



Optimal alignments for the case t1, t4, t2

t1	≫	t4	t2
t1	t2	t4	≫

t1	≫	t4	t2
t1	t3	t4	≫

Figure 2.9: Example of optimal alignments for a given event log and Petri net.

the given case, and the bottom row shows a valid sequence of transitions. The '≫' symbol represents a misalignment. In this example, both optimal alignments for the case 't1, t4, t2' have two misalignments. This shows how the use of alignments can provide detailed diagnostics for every single case instead of only providing the information that a case is fitting or not [vdAa16]. Additional, these individual assessments can be aggregated into a diagnosis on the process level by, for example, assessing how often particular activities are skipped or performed out of order [vdAa16].

Fitness and Precision Scores

The most common quality criteria that are determined during conformance checking of a model are *fitness* and *precision*. The fitness score shows whether the model allows for the behaviour seen in the given event log [vdAa16]. A model, therefore, has perfect fitness if all traces can be fully replayed on the model [vdAa16]. The precision score shows if the model allows for more activity sequences than there are present in the event log. Therefore, a perfect precision score means that the model does not allow any sequences of activities that are not present in the event log. In this thesis, conformance checking is not used to evaluate the model but to evaluate the generated event logs. Therefore, the fitness score determines if all cases in the generated event logs are valid sequences of activities that conform to the model. The precision score can evaluate whether the developed method can generate all possible traces in the model.

Multi-Perspective Conformance Checking

Multi-perspective conformance checking evaluates the commonalities or discrepancies between the behaviour described in the model and the event log while considering multiple perspectives. For example, in this thesis, conformance checking will consider the control-flow and the data perspective. Therefore, when examining the fitness and the precision of the event logs, the influence of the data perspective on the control-flow must be considered. For example, when examining if a trace in the event log fits the model by using token-based replay, the correctness of the indicated transitions firing has to be evaluated based on the current marking **and** the current

2. https://promtools.org/
3. https://fluxicon.com/disco/

variable status (i.e. evaluating transition guards). There are several tools, such as the *Multi-perspective Process Explorer (MPE)* [MaLR15], that allow to determine the alignment of multi-perspective models and event logs (The MPE is introduced in section 2.3).

2.2.4 Process Enhancement

Process enhancement aims to extend or improve an existing process model by using information about the real process provided in an event log [AAMA$^+$12]. This process mining technique can, for example, be used to repair or modify a model so that the model is a more accurate representation of the real world [vdAa11]. The repair is often done based on conformance checking results. Another example of the use of process enhancement is the extension of a model by, for example, adding a new perspective to it [vdAa11] that has not been included by the used process discovery algorithm. Three perspectives can be considered when extending a model [YaBS18]. The *organisational perspective* includes hidden information about resources in the log (for example, by using the originator field and analysing which subjects are involved and how are they related [DMVW$^+$05]). The *time perspective* concerns the timestamps of events. Lastly, the *case perspective* includes properties of the cases that are part of the log [YaBS18]. The case perspective thereby also includes data properties of the cases [DMVW$^+$05] which are the focus of this thesis.

2.3 Related Work

This section will briefly describe the most relevant works that are related to the research of this thesis.

Process Mining Toolkit

The Process Mining Toolkit (ProM)[4] is a framework that offers the user access to a wide variety of process mining techniques through plug-ins implemented for the framework [pro]. It was developed by the process mining group of the Eindhoven University of Technology. The framework is widely used in the process mining domain by researchers and academics around the world [vBVB14]. The following two paragraphs introduce two ProM plug-ins related to the research of this thesis, and that can also serve as an example of what ProM plug-ins can offer.

Straightforward Petri Net-Based Event Log Generation in ProM

The Paper 'Straightforward Petri Net-Based Event Log Generation in ProM' by vanden Broucke et al. describes a ProM plug-in that allows the generation of event logs from Petri nets by using token-based simulation [vBVB14]. The authors acknowledge that many tools allow for the generation of event logs from Petri nets. However, they argue that there are no tools that allow for a rapid generation of logs from a user-supplied Petri net in a straightforward manner [vBVB14]. The paper also provides an overview of the previously mentioned tools. The implemented plug-in starts from within ProM and is given a Petri net already loaded into ProM. The

4. http://www.promtools.org/

plug-in provides several simulation methods such as random or complete path generation and many configuration options for the token-based simulation. The event logs generated during the simulation are provided in the standardized XES[5] format. The plug-in is also open source, and the source code is provided on github[6].

The difference between the research of this thesis and the described paper is that the paper only focuses on the control-flow perspective of Petri nets. However, this thesis also tries to consider the data perspective of Petri nets and therefore tries to generate synthetic event logs from Data Petri nets.

The Multi-Perspective Process Explorer

The previously mentioned *Multi-Perspective Process Explorer (MPE)* that was developed by Felix Mannhardt[7] is a tool that provides process mining techniques for process discovery and conformance checking of multi-perspective event logs and process models [MaLR15]. It is available as a plug-in for ProM through the *MultiPerspectiveProcessExplorer* package [MaLR15].

The MPE is used as part of the evaluation of this thesis to analyse the event logs generated by the developed approach. It can detect deviations between a model and an event log while considering the control-flow and the data perspective. Therefore, it could be used to determine if synthetic event logs generated by the approach developed in this thesis conform to the model they are generated from. The MPE can detect deviations, such as guard violations and missing or false write operations. Since event logs and process models can have relatively high complexities, manually performing conformance checking is often not feasible. Therefore, the MPE can be used on models and event logs with high complexity to automatically determine any deviation between the behaviour they describe.

5. https://www.xes-standard.org/
6. https://github.com/Macuyiko/processmining-loggenerator
7. https://fmannhardt.de/blog/software/prom/mpe

3. Preliminary Work for the Approach Development

This section covers the preliminary work needed to develop the approach for generating synthetic procedural data from Data Petri nets. First, a requirement analysis is conducted to identify the requirements that the approach has to fulfil to generate realistic data that conforms to a DPN (section 3.1). A comprehensive literature review regarding synthetic data generation with a focus on procedural data is then conducted to identify methods for generating synthetic procedural data from Petri nets (section 3.2). Subsequently, the identified methods are compared, and one method is chosen based on the previously established requirements (3.3). This method is then used for the development of the approach, which is described in chapter 4.

3.1 Requirement Analysis

In this section, the requirement analysis is presented. The requirement analysis provides the basis for the literature review and the process for selecting a method that can be applied or extended for synthetic procedural data generation from Data Petri nets. Since there are differences in notations, definitions and file formats for data Petri nets, an exemplary Data Petri net, is chosen for the requirement analysis. This model should contain as many challenges for data generation from DPNs as possible. Therefore, the first step is to identify potential challenges for generating synthetic procedural data from Data Petri nets. Subsequently, a Data Petri net is chosen based on the identified challenges. Then the model is examined to establish the requirements for the literature review and method selection process.

3.1.1 DPN Modelling Tool Identification

In order to be able to determine possible challenges for generating event logs from DPNs, the possible features of a DPN have to be identified first. As previously explained in section 2.1.3 Data Petri nets consist of Petri nets that have variables that transitions can write and read to influence the control-flow of the model. However, to the best of the authors' knowledge, there is no widely established standard about how exactly the variables and transition guards of a DPN are defined. For example, it needs to be known what kind of logical operators can be present in transition guards or what type of variables can be included in a DPN.

The methodology used for identifying possible features of a DPN consisted of analysing tools that allow the creation of DPN. By analysing tools that enable the creation of DPNs, possible features of DPNs can be identified. Therefore, the first step of identifying possible challenges of generating event logs from DPNs consisted of identifying tools that allow the creation of DPNs.

Multiple resources were used to search for DPN modelling tools. First, the tools in the Petri Nets Tool Database[1] provided by the computer science faculty of the University of Hamburg were examined to determine if they allow the modelling of DPNs. Subsequently, the tools uncovered in a survey of Petri net tools conducted by Weng Jie and Arriff bin Ameedeen were examined [ThAm15] in the same way. Lastly, the plug-ins available for ProM were assessed. In preliminary work regarding the related works of this thesis, ProM was often referenced as a valuable resource for process mining tools implemented as plug-ins for this framework. Therefore, the plug-ins available for ProM were also included in the examined tools.

When examining the tools found during the search, it was determined if they specifically support the creation or editing of Data Petri nets. Many of the tools found during the search offered support for Petri nets that include additional perspectives other than the control-flow perspective (for example, Yasper or CPN Tools [HOPS+06, RWLL+03]). However, most of these tools did not support the creation of DPNs but rather the creation of other high-level Petri net types, such as Coloured Petri nets [Jens97]. Only three tools could be found that allowed the creation or editing of DPNs. These three tools are plug-ins for ProM. The 'Edit DPN (Text language based)' and 'Create DPN (Text language based)' plug-ins developed by Felix Manhardt allow the editing and creation of DPNs. The 'Create/Edit PetriNet With Data' plug-in developed by M. de Leoni also allows to create and edit DPNs.

3.1.2 Challenge Identification

As explained in the previous section, three ProM plug-ins could be found that allow the creation or editing of Data Petri nets. These plug-ins were analysed to determine what challenges can appear in DPNs. The following paragraphs describe the combined challenges for data Generation from DPNs generated by these plug-ins.

Control-Flow Perspective

When trying to generate synthetic procedural data from Data Petri nets, the challenges that have to be considered do not only stem from the data perspective. The challenges that arise when generating synthetic procedural data from Petri nets that are limited to the control-flow perspective must also be considered. Therefore, the first challenge is to process the definition of the control-flow so that the generated traces follow it. However, these challenges are relatively simple since a Petri net limited to the control-flow perspective is only made up of four components. First, the definition of places, transitions, and arcs have to be processed. Additionally, the model's markings, i.e., initial and final marking, have to be processed.

Data Perspective

The first challenge that arises from the extension of a Petri net to include the data perspective is the consideration of variable definitions. The examined tools allow variables of the types *String*, *Boolean*, *Long*, *Double*, and *Date*. Additionally, the

1. https://www2.informatik.uni-hamburg.de/TGI/PetriNets/tools/db.html

definition of minimum and maximum values for Long and Double variables have to be considered for generating values for these variables. The second challenge that arises from the data perspective concerns the influence of the data on the control-flow. Therefore, the writing and reading of variables by transitions have to be considered. Additionally, a non-trivial challenge arises from the guards of transitions that need to be evaluated to determine if a transition is enabled. The examined tools allow guards to consist of variables and values that are connected with operators. The variable types that are allowed in guards are the same as the allowed variable types. The operators that are allowed in the examined tools are **Lesser then** ($<$), **Lesser or equal then** ($\leq$), **Greater then** ($>$), **Greater or equal then** ($\geq$), **Equal** ($==$), **Unequal** ($!=$), **And** ($\&\&$), **Or** ($||$), **Plus** ($+$), **Minus** ($-$), **Multiply** ($*$), and **Divide** ($/$). Data Petri nets modelled with the identified tools can contain invisible transitions. Invisible transitions influence the control-flow just like transitions that are not invisible. However, invisible transitions should not appear in the event logs generated from the model. Therefore, another challenge consists of differentiating between invisible and not invisible transitions.

3.1.3 Choosing a Exemplary Data Petri net Model

Based on the challenges described in the previous section, a search was conducted to find Data Petri nets that could be used as a basis for the requirement analysis. It was thereby preferred to use a DPN modelled in the PNML since this language is widely accepted [Kind04]. The search for DPN model files was conducted by using the Base[2] search engine and the 4TU.ResearchData[3] repository. The advanced search option of Base allows to search for files related to scientific research and can therefore be used to find model files. 4TU.ResearchData is a repository for data and software related to scientific research engineering and design [4TU.]. In addition to Base and 4TU.ResearchData, a DPN acquired from fellow researchers at the faculty of Business Informatics II at the University of Trier, was analysed to determine if it is suitable as a basis for specifying requirements for event log generation from DPNs.

The Base advanced search was first used with the search string 'tit:Data tit:Petri tit:net doctype:7'. This string specifies that the results should be scientific data (document type 7) and contain 'Data', 'Petri', and 'net' in the title of the scientific work that the data is related to. However, this search led to zero results. Therefore, the search query was relaxed by removing the requirement for having 'Data' in the title. This search led to ten results. However, three duplicates in the results brought the total unique search results to seven. Of those seven results, only four contained actual process model files. However, none of the models contained in the discovered datasets were Data Petri nets. The models found were Petri nets limited to the control-flow perspective.

Note: Both previously described searches with the Base search engine were conducted on the 27th of October 2021.

The search of the 4TU.ResearchData repository was conducted using the search string 'Data Petri net'. Due to the number of results (7,004) exceeding the amount that can be feasible examined manually, the filter options of the search function

2. https://www.base-search.net/
3. https://data.4tu.nl/

were used. The results were constrained to data published in the last five years (2017 to 2021). Additionally, the results were filtered to contain only results from the categories 'Information Systems' and 'Business and Management'. With these filters, the number of results could be lowered to 79. These 79 results were examined to find DPN files.

Note: The search of the 4TU.ResearchData repository was conducted on the 28th of October 2021.

From the 79 results, only four datasets contained process model files. Of these four datasets, three contained Petri nets limited to the control-flow perspective[4,5,6]. The only DPN that could be found did only contain invisible transitions and no variables[7]. Therefore, this model is not suitable as a basis for specifying requirements for event log generation from DPNs.

Since no suitable DPNs could be found during the search of the 4TU.ResearchData repository or the search with Base, the previously mentioned model acquired from researchers at the University of Trier was chosen as a basis for the requirements specification. Figure 3.1 depicts a small excerpt of the model. The model describes the computer-interpretable clinical guidelines for treating malignant melanoma. Therefore, the model covers the domain that this thesis focuses on. The model itself is present in the Petri Net Markup Language [WeKi03] file format (.pnml), which is preferred because it is a widely accepted interchange format [Kind04]. Additionally, the model is very complex and therefore contains many of the previously identified challenges. Table 3.1 presents an overview of the challenges present in the model. The only challenges missing are that there are no variables of the type *Date* and no guards containing this value type and that the guard operators **Plus** (+), **Minus** (−), **Multiply** (∗), and **Divide** (/) are missing. Additionally, no variables have defined value spaces, i.e., maximum and minimum values. Furthermore, while the transitions read variables in their guards, it is not defined which transitions read which variables. This is usually done by the following tags in the PMNL file: `<readVariable>variable</readVariable>`. This provides an additional challenge since the read variables will have to be identified by analysing the transition guard.

3.1.4 Requirement Specification

In this section the requirements for the approach developed in this thesis are discussed. The requirements are based on the Data Petri net chosen in section 3.1.3 and the challenges identified in section 3.1.2. Table 3.1 shows how these requirements relate to identified challenges.

The model chosen was modelled in ProM and exported in the Petri Net Markup Language. Additionally, as previously mentioned, a model in the PNML was preferred during the search since the PNML is a widely accepted interchange format

4. https://data.4tu.nl/articles/dataset/Generated_and_industrial_PNML_models_with_generated_log_traces_in_XES_/12696878
5. https://data.4tu.nl/articles/dataset/Recomposing_Conformance_INS_2018_/12680942?file=24014441
6. https://data.4tu.nl/articles/dataset/Validation_of_Precision_Measures_-_Event_Logs_and_Process_Models/12708359
7. https://data.4tu.nl/articles/dataset/Conformance_Checking_Challenge_2019_CCC19_/12714932

Table 3.1: Listing of the challenges present in the chosen model

Challenge	**Present in model**	**Related Requirements**
Control-Flow Perspective		
Place definition	✓	**R1**
Transition definition	✓	**R1**
Arc definition	✓	**R1**
Initial marking	✓	**R1**
Final marking	✓	**R1**
Invisible transitions	✓	**R2**
Variables		
Contains variables	✓	**R3**, **R4**
String (Literal)	✓	**R3**
Boolean	✓	**R3**
Long (Discrete)	✓	**R3**
Double (Continuous)	✓	**R3**
Date	X	**R4**
Minimum/Maximum value	X	–
DPN Transitions		
Transition with guard	✓	**R5**
Transition write variables	✓	–
Transition read variables	✓[*1]	–
Guards		
Contain variables	✓	**R5**
Contain non variable string value	✓	**R6**
Contain non variable boolean value	✓	**R6**
Contain non variable long value	✓	**R5**
Contain non variable double value	✓	**R6**
Contain non variable date value	X	**R5**
Guard Operators		
Lesser then ($<$)	✓	**R6**
Lesser or equal then ($\leq$)	✓	**R6**
Greater then ($>$)	✓	**R6**
Greater or equal then($\geq$)	✓	**R6**
Equal ($==$)	✓	**R6**
Unequal ($!=$)	✓	**R6**
And (&&)	✓	**R6**
Or ($\|\|$)	✓	**R6**
Plus ($+$)	X	**R7**
Minus ($-$)	X	**R7**
Multiply ($*$)	X	**R7**
Divide ($/$)	X	**R7**

[*1] Variables are present in guards of transitions but the `<readVariable>variable</readVariable>` definition is missing.

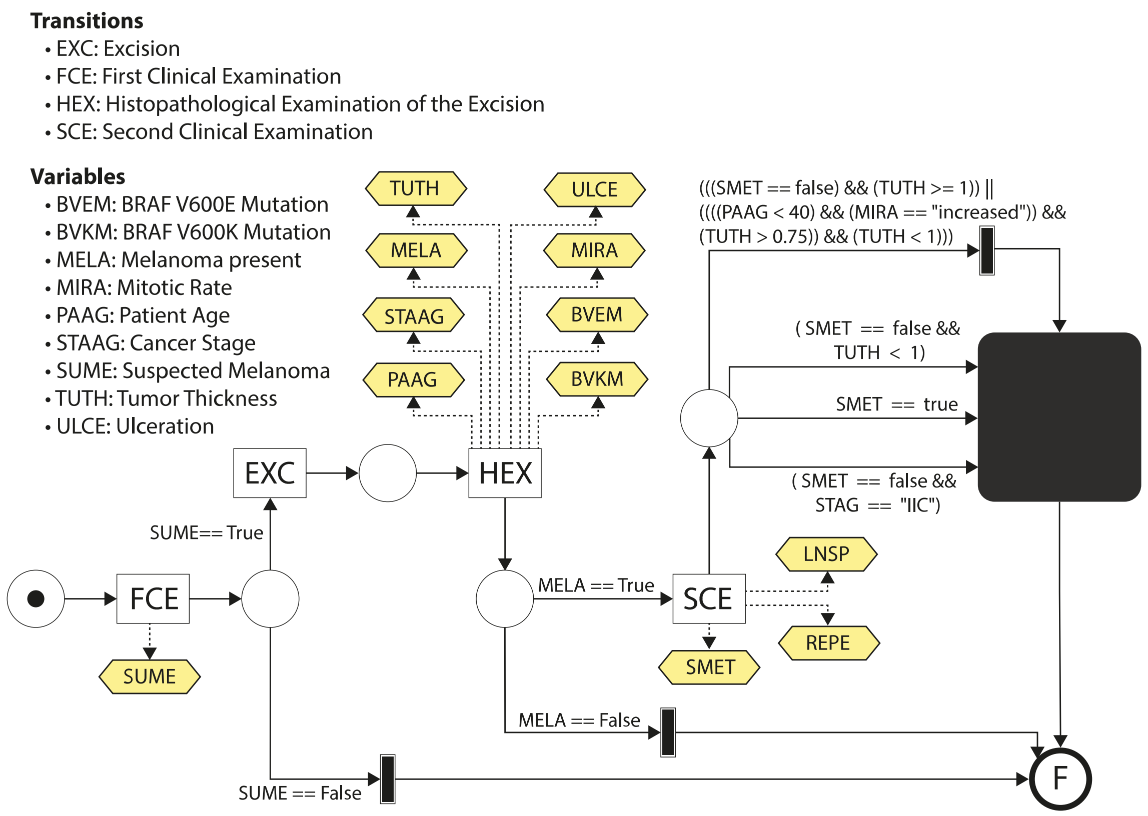


Figure 3.1: Excerpt of the Data Petri net depicting the treatment of melanoma. The black box depicts the missing parts.

for Petri nets [Kind04]. Therefore, the **first requirement** (**R1**) concerns the DPN format that the approach should support. It requires the developed approach to accept Data Petri net model files in the PMNL format (for details on PNML see section 2.1.4). Since this language is a widely accepted standard for storing Petri net models [Kind04], it should allow the developed approach to accept most Petri nets.

The **second requirement** (**R2**) requires the approach to consider whether a transition is invisible. Invisible transitions should not appear in the event log. However, they should still be able to read from and write to variables and change the marking of the model.

The chosen Data Petri net contains variables of the type String, Boolean, Long, and Double. Therefore, the **third requirement** (**R3**) requires the developed approach to be able to deal with variables of these variable types. This included being able to read from and write to the variables. Additionally, the approach needs to be able to generate realistic values for variables of these types. Also, the approach has to be able to evaluate transition guards that contain variables of these types.

Even if there are no variables of the type date in the selected model, it would be advantageous if the developed approach supports this type since all the tools examined for creating Data Petri nets support this type. Therefore, the **fourth requirement** (**R4**) requires the approach to also be able to deal with variables of the type date. However, this requirement is not mandatory as it is not needed to generate event logs from the selected model. However, it would still be advantageous for the approach to support date variables to be able to generate event logs from models containing these types of variables.

In addition to the processing of variables, the approach also has to be able to process transition guards. This is necessary in order for the approach to properly process the influence of the data perspective on the control-flow of the model. Therefore, the **fifth requirement** (**R5**) requires the developed approach to be able to evaluate transition guards and influence the control-flow accordingly.

Transition guards of Data Petri net models can contain various logical operators, such as Lesser then ($<$). The **sixth requirement** (**R6**) specifies the operators that have to be supported by the developed approach in order to be able to process the chosen model. The operators that are present in the model and therefore have to be supported are Lesser than ($<$), Lesser or equal than ($\leq$), Greater than ($>$), Greater or equal than ($\geq$), Equal ($==$), Unequal ($!=$), And ($\&\&$), and Or ($\|$).

During the identification of possible challenges in section 3.1.2 it was discovered that transition guards could also contain arithmetic operators. Even if the selected model does not contain arithmetic operators, it would still be advantageous if the approach supported these operators. Therefore, the **seventh requirement** (**R7**) acts as an addition to the seventh requirement and requires the developed approach to support arithmetic guard operators. The requirement contains the arithmetic operators Plus ($+$), Minus ($-$), Multiply ($*$), and Divide ($/$). However, this requirement is not mandatory as it is not needed for the selected model.

The developed approach has to be able to generate traces that adhere to the supplied process model in order to generate realistic traces. Therefore, the **eighth requirement** (**R8**) requires the approach to generate event logs that only contain valid traces that conform to the model. This means that every trace in the generated event log is a possible firing sequence of the given Petri Net [vdAa16]. There should be no deviations from the allowed activity sequences.

The goal of this thesis is to develop an approach for the generation of event logs from Data Petri nets. As previously described, the motivation behind this goal is that it is hard to obtain synthetic patient data for the development of AI methods. Regarding this purpose, it would be advantageous if the generated event logs contain all the valid behaviour that is described in the model. Therefore, the **ninth requirement** (**R9**) requires the approach to be able to generate event logs that contain all traces that are possible in the given model that has no loops. In this context, all traces refers to all possible combinations of transition sequences that are possible in the model. However, authentic event logs do not necessarily contain all traces possible in a process. Therefore, this requirement is not mandatory since it is not required to generate event logs that are indistinguishable from authentic event logs. Additionally, process models with loops have to be excepted from this requirement since it is not possible to generate all valid traces since there are an infinite amount of different traces if a loop is present. Therefore, the developed approach does not have to be able to generate all possible traces if there are loops present in the model.

If the approach is used with a model that contains loops, it should still allow for the generation of an event log that represents all behaviour described in the model. Therefore, the **tenth requirement** (**R10**) requires the approach to be able to generate all traces possible in a model with a specific amount of loop iterations. When the number of valid loop iterations is specified, the number of maximum possible traces is finite, and it should be possible to generate all valid traces.

Since the goal of this thesis is to develop an approach that can generate event logs that are indistinguishable from authentic event logs the **eleventh requirement** (**R11**) requires the approach to generate realistic data. Therefore, the approach should be able to generate event logs that are meaningful from a perspective of a domain expert and that are indistinguishable from real event logs. This requirement, therefore, is in line with the second part of the first research objective of this thesis (**RQ1.1**).

As described in section 2.2.1 the de facto standard interchange format for event logs is the eXtensible Event Stream format. Most processing mining tools support this format. Therefore, it would be advantageous to generate the event logs in this format. Therefore, the **twelfth requirement** (**R12**) requires the developed approach to generate event logs in the XES format. However, this requirement is not mandatory since it does not have to be fulfilled to generate synthetic event logs from the chosen Data Petri net.

In the following listing, the previously discussed requirements for the approach that is to be developed are formally specified according to ISO 29148 [Inte11]. The term '**shall**' is used for mandatory requirements, and the term '**should**' is used for non-mandatory requirements. The mandatory requirements are necessary to generate realistic synthetic event logs from the chosen model. The non-mandatory requirements are desired requirements that are not necessary for the realistic data generation of the chosen model but serve as a guideline for developing additional features that could be developed to improve the approach.

R1 **PNML Acceptance** The developed approach **shall** be able to accept Data Petri nets in the PNML format.

R2 **Invisible Transitions** The developed approach **shall** consider the influence of invisible transitions on the control-flow of the model. However, the approach **shall** not include invisible transitions in the synthetic data it that it generates.

R3 **Basic Variable Types** The developed approach **shall** be able to deal with models that contain variables of the types String, Boolean, Long, and Double. The approach **shall**, therefore, be able to generate values for these variable types and be able to evaluate transition guards containing variables of these types.

R4 **Date Variables** The developed approach **should** additionally be able to deal with models that contain variables of the type Date. The approach **should**, therefore, be able to generate values for date variables and be able to evaluate transition guards containing date variables.

R5 **Transition Guards** The developed approach **shall** be able to evaluate transition guards and properly consider the influence of these guards on the control-flow of the model.

R6 **Logical Operators** The developed approach **shall** be able to evaluate transition guards that contain logical operators. The logical operators that the approach **shall** support are Lesser then ($<$), Lesser or equal then ($\leq$), Greater then ($>$), Greater or equal then ($\geq$), Equal ($==$), Unequal ($! =$), And ($\&\&$), and Or ($||$)

R7 **Arithmetic Operators** The developed approach **should** be able to evaluate transition guards that contain arithmetic operators. The operators that the approach **should** support are Plus (+), Minus (−), Multiply (∗), and Divide (/)

R8 **Conforming Traces** The developed approach **shall** be able to generate event logs that only contain traces that conform to the supplied model. Therefore, the approach **shall** be able to generate event logs that do not contain any deviations from the model regarding the control-flow as well as the data perspective.

R9 **Full Exploration** The developed approach **should** be able to generate all possible traces from a given model that has no loops. In this context all traces refers to all combinations of transition sequences that are possible in the model.

R10 **Full Exploration with Loops** If a model has a loop the approach **should** allow for the generation of all possible traces with a specified amount of times that each marking is allowed to be present in a trace. In this context all traces refers to all combinations of transition sequences that are possible in the model and do not exceed the maximum number of loop iterations.

R11 **Realism** The developed approach **shall** be able to generate data that is considered meaningful from the perspective of domain experts and that is indistinguishable from real traces.

R12 **Output Format** The developed approach **should** be able to generate event logs in the Extensible Event Stream (XES) format.

3.2 Literature Review

In this section, the literature review is described. The literature review aims to identify methods for the generation of synthetic procedural data that could be used to develop an approach that allows the generation of synthetic event logs based on Data Petri nets. The first step in the literature review consists of identifying, assessing, and selecting suitable databases that should be used for the literature review. This step is described in section 3.2.1. Subsequently, the suitable databases are searched to find relevant papers by examining the title and abstract of all papers found during the search. The papers identified as relevant during the search will be examined in detail since only their title and abstract have been considered so far. If approaches relevant to the generation of synthetic procedural data from Data Petri nets are found, they are added to a data set for consideration in the selection process described in 3.3. Section 3.2.2 describes the results of searching the suitable databases to find relevant literature and the methodology used to conduct the search. Lastly, the methods identified during the literature review are presented in section 3.2.3.

3.2.1 Database Identification and Selection

The following databases were identified during the search for databases containing scientific literature that could be used for the literature review.

- SpringerLink[8]
- Institute of Electrical and Electronic Engineers (IEEE) Xplore[9]
- Multidisciplinary Digital Publishing Institute (MDPI)[10]
- Web Of Science[11]
- PubMed[12]
- ScienceDirect[13]
- Directory of Open Access Journals (DOAJ)[14]
- Scopus[15]

Except for Scopus and Web Of Science, all the previously mentioned databases can be searched without requiring a subscription or institutional access. The University of Trier, where the author of this thesis studies, offers students access to Web Of Science but not Scopus. Therefore, Scopus has been disregarded for the search since enough accessible databases have been found. A two-step process ensures that the identified databases contain enough literature work relevant to this thesis. First, a simple keyword search with the keywords 'synthetic data generation' is conducted to determine the number of search results generally related to the relevant topic. Since the identified databases offer different search options, a common option is selected. However, the databases have no search option, which is entirely identical for every database. Therefore, the search option that is supported by most of the databases is selected. The search is conducted by searching for literature works with titles that contain the keywords 'synthetic', 'data', and 'generation' in no particular order. All databases but SpringerLink support this search type. SpringerLink only supports the search for titles with keywords that appear in the specified order. Therefore, SpringerLink's search results are smaller than they would be with the search method used for the other databases. In the second step, relevant journals are identified, and it is determined which of the identified databases contain literature from these journals. While the search is not limited to journal articles, the presence of relevant journals in the databases could be a helpful indicator of the amount of relevant literature present in the database. The following journals have been identified as containing scientific work relevant to this thesis and are used for the second step of the database assessment.

8. https://link.springer.com/
9. https://ieeexplore.ieee.org/
10. https://www.mdpi.com/
11. https://www.webofscience.com/
12. https://pubmed.ncbi.nlm.nih.gov/
13. https://www.sciencedirect.com/
14. https://doaj.org/
15. https://www.scopus.com/

1. Journal of Biomedical Informatics[16]
2. International Journal of Medical Informatics[17]
3. Artificial Intelligence in Medicine[18]
4. Journal of Medical Artificial Intelligence[19]
5. npj Digital Medicine[20]
6. JAMIA: Journal of the American Medical Informatics[21]
7. Journal of Biomedical and Health Informatics[22]

The search of the databases for the database assessment was conducted on the 5th of November 2021. The results of the database assessment can be seen in table 3.2. As previously described, the search of the databases to determine the number of relevant literature was conducted using the keywords ‘synthetic’, ‘data’, and ‘generation’. With the exception of SpringerLink, the order in which these keywords had to appear was not specified. No further restrictions, such as restricting the publishing dates, were performed. The databases were queried on the third of November 2021. Web Of Science and IEEE Xplore have significantly more

Table 3.2: Number of hits of the first word search and number of journals available for the assessed databases

Database	**# Search Results**[*1]	**# Journals Present**[*2]
Web Of Science	97	6 (1,2,3,5,6,7)
IEEE Xplore	78	1 (7)
ScienceDirect	37	3 (1,2,3)
DOAJ	24	0
SpringerLink	18[*3]	0
PubMed	8	6 (1,2,3,5,6,7)
MDPI	7	0

[*1] The number search results that contain the keywords ‘synthetic’, ‘data’, and ‘generation’ in their title.
[*2] The number of journals that were previously identified as very relevant and that are indexes by the database. The brackets contain the specific the journals referring to the previous enumeration.
[*3] For SpringerLink the search is restraint to the keywords being in the correct order since the title search option without ordering the keywords is not supported by this database.

16. https://www.journals.elsevier.com/journal-of-biomedical-informatics
17. https://www.journals.elsevier.com/international-journal-of-medical-informatics
18. https://www.sciencedirect.com/journal/artificial-intelligence-in-medicine
19. https://jmai.amegroups.com/
20. https://www.nature.com/npjdigitalmed/
21. https://academic.oup.com/jamia
22. https://www.embs.org/jbhi/

results than the other databases, with 97 and 78 results each. The Multidisciplinary Digital Publishing Institute and PubMed database have very few search results with 8 and 7 results, while ScienceDirect, DOAJ, and SpringerLink are in the middle ground. However, SpringerLink would probably have a significantly higher number of search results if the previously described restrictions did not apply. Web Of Science and PubMed have the highest ranking in the assessment regarding the number of relevant Journals present, with 6 out of 7. The only journal missing is the Journal of Medical Artificial Intelligence, which is not present in any assessed database. Of the remaining databases, only IEEE Xplore and ScienceDirect have relevant Journals present, with 1 in IEEE Xplore and 3 in ScienceDirect.

Based on the results of the database assessment, the Directory of Open Access Journals and the Multidisciplinary Digital Publishing Institute database are disregarded for the literature search for the time being. They only provide a small number of search results, and none of the relevant journals are available. Therefore, the remaining databases used for the search are Web Of Science, IEEE Xplore, SpringerLink, ScienceDirect, and PubMed. However, if the search in the remaining databases does not provide enough relevant papers, the option remains to extend the search to the previously disregarded databases. In addition, the search can also be extended to search engines such as Google Scholar[23] or Base[24] if necessary.

3.2.2 Search Results

This section covers the result of the literature review and provides a brief overview of the methods for synthetic procedural data generation found in scientific literature. As described in section 3.2.1 the search for relevant scientific literature was first conducted in the databases Web Of Science, IEEE Xplore, SpringerLink, ScienceDirect, and PubMed. The search was done using the keywords ‘event log’ and ‘generation’ with different search configurations depending on the available search configurations of the databases. In the following, the search configurations used for the databases are described.

While searching the databases, all search queries were restricted to only containing documents from the last 20 years (2002 to 2021). While searching the Web Of Science database, the search string ‘event log generation’ was used to find documents of the type ‘Articles’. Additionally, the results were filtered to only contain articles from the categories ‘Computer Science Information Systems’, ‘Computer Science Artificial Intelligence’, and ‘Computer Science Interdisciplinary Applications’. Web Of Science was searched on the 4th of November 2021. The IEEE Xplore database was also searched with the string ‘event log generation’. The results were filtered to only contain documents from the publication topics ‘Data Mining’, ‘learning (artificial intelligence)’, and ‘business data processing’. The search of IEEE Explore was conducted on the 5th of November, 2021. The SpringerLink database was searched with the string ‘event AND log AND generation’ with no further restrictions in the search configuration. The search of SpringerLink was conducted on the 6th of November, 2021. The search of ScienceDirect was conducted with the string ‘event log generation’ in the field ‘Title, abstract or author’ and ‘log OR traces OR generation’ in the field ‘Title’. The search was conducted on the 6th of November, 2021.

23. https://scholar.google.com/
24. https://www.base-search.net/

The search of PubMed was conducted with the search string 'event log generation AND log[title]'. The search was conducted on the 8th of November, 2021.

During the previously described search queries, 184 papers could be found, and their title and abstract were examined. Of all the papers whose titles and abstracts were examined, 13 were selected for detailed examination. The papers were selected due to the title and abstract promising the presentation of an approach for synthetic procedural data generation that seems promising for being used as a basis for the approach developed in this thesis. Papers that were not deemed relevant were not further examined. For example, several papers were found that proposed an approach for synthetic procedural data generation by using machine learning on already existing data (For example, [LuKJ02]). These papers were disregarded because it is required that the event log generation is done by using a process model without already existing event logs.

Since the amount of relevant papers found in the first search is relatively low, the search was extended to include the databases of the Directory of Open Access Journals and the Multidisciplinary Digital Publishing Institute, as well as the search engines Google Scholar[25] and Base[26]. The search configurations used to query these databases are described in the following. Just like the first search queries, the extended search queries were also limited to documents published in the last 20 years (2002 to 2021). The MDPI database was queried with the search string 'event log generation'. Additionally, the results were limited to the subject 'Computer Science & Mathematics' to exclude irrelevant results. The search was conducted on the 9th of November, 2021. The DOAJ was also searched on the 9th of November 2021 with the search string 'event log generation'. The BASE search engine was used with the search string 'event log generation subj:event subj:log doctype:(121 13)' on the 10th of November 2021. The document types '121' and '13' refer to conference papers and journal articles. Lastly, Google Scholar was used with the search string 'allintitle: event log generation'. The query was conducted on the 10th of November, 2021.

During the extended search, another 97 documents were found and examined by their title and abstract. Of these papers, 13 were selected because their title and abstract suggested the description of an approach for synthetic event log generation, bringing the total up to 26 papers. The full list of papers that were examined in detail to identify potential methods for synthetic event log generation can be seen in appendix A. Appendix A also contains information about what type of approach each paper proposes. All but two of the examined papers proposed a method or approach for the synthetic event log generation from process models. Out of the remaining 24 papers, 20 propose an simulation based approach to generate event logs. Most of the papers using a simulation-based approach use a token-based simulation approach, which consists of propagating tokens through a model by firing transitions and recording the fired transitions to generate an event log (for example, [KWDS$^+$04, MSKv17, HOPS$^+$06, MeGü04, Bura15, PeMi19, IvAl14, vBVB14, EsKA19, KaKa14]). However, there are also other simulation oppproaches that were found. For example, Ciccio et al. [CBCM15] propose an approach, which consists of transforming a model into a *Finite State Automata (FSA)* before using simulation to generate traces.

25. https://scholar.google.com/
26. https://www.base-search.net/

In addition to the papers that proposed a simulation-based approach, four papers were found that proposed approaches that do not use a simulation. Loreti et al. [LCCM20] and Chesani et al. [CCLM17] propose an approach which uses a form of logical reasoning called *abduction* to generate event logs based on process models. Wiśniewski et al. [WKLS18] propose an approach that consists of converting the search for possible valid traces into a *constraint satisfaction problem*, where every state that satisfies all constraints is a valid sequence of events. Lastly, Skydanienko et al. [SFGM18] propose a similar approach that consists of transforming the search for valid traces into a *boolean satisfiability problem* to extract traces from a process models.

3.2.3 Identified Approaches

This section will cover the approaches identified during the literature review and therefore provide the basis for the selection process covered in section 3.3. Similar approaches have been grouped into one approach. For example, multiple papers proposed an approach that uses some form of token-based simulation. Although there are variations of how this type of approach is used, it will be treated as a single approach in the following paragraphs.

Token-Based Simulation

Multiple papers were found during the literature review that use some form of token base simulation to generate traces from process models (see appendix A for a list of papers). Token-based simulation refers to propagating tokens through a process model and recording the executed transitions to generate event logs. For example, when using Petri nets, the propagation is achieved by firing enabled transitions until all transitions are disabled or a valid final state is reached. Different strategies can be used to determine which transition to fire if multiple transitions are enabled, such as choosing the transitions at random (with or without different weights). The order of the transitions fired is recorded to generate the traces for the event log. The transitions are then referenced with activities to get a valid trace of executed activities that can be added to the event log. The process of propagating tokens and recording transitions firings is repeated until the desired number of traces is reached.

One rather unique use of token-based simulation is proposed by Anna Kalenkova [KaKa14]. The paper proposes an approach to generate graph-based process models by using *graph grammars* and an approach to generate event logs by applying graph grammar *production rules* rules to the generated model. A production rule replaces one part of a graph by another [KaKa14]. The approach uses a simulation that consists of applying production rules that propagate tokens through the graph to generate event logs.

Finite State Automata Simulation

Claudio Di Ciccio et al. propose an approach that uses the simulation of a deterministic *Finite State Automata (FSA)* to generate event logs [CBCM15]. An FSA is a transition system that consists of a set of states and a transition function that acts as a set of transitions that can change the state of the system [CBCM15].

The idea of the approach is to create an FSA with a transition function that only allows the execution of valid traces of the model. The FSA can then be used to generate event logs through the simulation of the FSA. The proposed approach is tailored towards the event log generation of declarative models (see section 2.1.1 for a short explanation of declarative models) and specifically *Declare*[27] models.

The first step in the proposed approach consists of converting the activities of the given process model to a transition function that can be used to traverse the states of the FSA. Subsequently, the constraints by which the Declare model is defined are transformed into *regular expressions*, which are then used to create the FSA. The FSA is then used to generate event logs by traversing randomly through it and recording the executed transitions to generate a trace of activities.

Abduction

Loreti et al. [LCCM20] as well as Ciampolini and Loreti [CCLM17] propose a approach for the generation of synthetic traces with the uses of *abductive logic programming (ALP)* through the *SCIFF Framework*[28]. ALP is a form of programming that uses abductive reasoning. Abductive reasoning is a form of logical reasoning used to find an explanation for a given observation. The explanation should thereby be as simple as possible.

The proposed approach consists of three steps. The first step consists of translating the given process model into a SCIFF specification. Loreti et al. define a SCIFF specification as follows.

Definition 3 (SCIFF Specification [LCCM20])**.** *A SCIFF specification is a triple $\langle KB, A, IC\rangle$ where*

- *KB is a knowledge base (i.e. a Logic Program as for [Lloy84]);*
- *A is a set of abducible predicates with functor* ***ABD****,* ***E****, or* ***EN****;*
- *IC is a set of Integrity Constraints (ICs).*

The knowledge base of a SCIFF specification is used to express declarative knowledge about the specific domain application [ACGL$^+$08]. The abductible predicates are used inside of the ICs to describe behaviour. In the proposed approach, the ICs are used to constrain the valid sequences of events. **E** (positive expectation) and **EN** (negative expectation) are abstractions that used to describe the *expectations*, i.e., the events that are desired to take place [ACGL$^+$08]. The **ABD** predicates correspond to predicates that can be hypothesized [LCCM20]. For example, **ABD** can be used to hypothesize that an activity is executed at a specific time. The following example shows a SCIFF Integrity Constraints that means that activity **a** should be followed by the activity **b** [LCCM20]. T_a and T_b refer to the time that the activities are execute.

$$\mathbf{ABD}(a, T_a) \rightarrow \mathbf{ABD}(b, T_b) \wedge T_b > T_a$$

27. https://www.win.tue.nl/declare/
28. http://lia.deis.unibo.it/sciff/

After the process model has been translated to a SCIFF specification, the SCIFF proof procedure can be used to generate *trace templates* that resemble valid sequences of activities that satisfy all of the specified Integrity Constraints. The SCIFF proof procedure is open source and implemented in the logical programming language Prolog. The third step consists of using these trace templates to generate event logs by grounding the trace templates with variables [LCCM20]. Grounding the trace templates with variables refers to substituting the trace templates' variables (for example, the timestamp variable) with possible valid values.

Constraint Satisfactory Problem

Wiśniewski et al. [WKLS18] propose a method for the generation of synthetic event logs that consists of transforming a process model into a *Constraint Satisfaction Problem (CSP)* where the solutions are valid traces of activities. A CSP is defined over a finite set of variables that are each assigned a domain of possible values and a set of constraints [Dech98]. The aim is to find values for the variables that satisfy all of the defined constraints [BrPS99]. In the context of process mining, the set of variables that need to be assigned could be a sequence of activities. An exemplary constraint could be that a specific activity can only be executed after another activity has been executed. Therefore, a solution to this problem would be a valid sequence of activities.

Since the proposed approach uses a procedural instead of a declarative process model, the transformation of the model into a CSP poses a problem since a procedural model is not already defined by constraints. The authors argue that in order to transform a procedural model into a CSP, all pairs of tasks have to be checked for dependencies to determine the constraints [WKLS18]. This poses a problem on an exponential scale. The proposed solution to that problem consists of splitting the model into multiple sub-graphs to reduce the number of pairs that have to be checked. Each sub-graph has a single path as an entry and a single path as an exit. Therefore, they are called *Single Entry Single Exit (SESE)* blocks. These blocks are supposed to represent the flow of a single token. The transformation of the model to a CSP representation is then done for each SESE, and the blocks are then combined with additional constraints. Lastly, the event log generation is done by solving the CSP and therefore generating valid activities sequences.

Boolean Satisfiability Problem

Skydanienko et al. [SFGM18] as well as Ackermann and Schönig [AcSJ17] propose a approach for the generation of event logs from Declare models that consists of transforming the model into a *conjunctive normal form (CNF)* and and solving it using a *boolean satisfiability problem (SAT) solver* to generate valid traces.

The first step in the approach is to translate the process model into the Alloy specification language [Jack19]. Both papers that propose this approach use a declarative process model. This makes the translation easier since the Alloy language is also a declarative language. However, due to the restrictions of the Alloy language, some preprocessing is required. For example, Alloy does not support numeric variables, so intervals for numeric values have to be generated with constraints [SFGM18]. Subsequently, the *Alloy Analyzer*[29] is used to generate a conjunctive normal form

29. https://alloytools.org/

based on the specified model. The Alloy Analyzer is implemented in Java and is an open source[30] software. The CNF is generated so that every solution is a valid trace of activities in the model. The Alloy Analyzer is used again to generate solutions for the CNF by using an SAT solver to generate valid traces. Lastly, the solutions can be used as a basis for generating an event log.

3.3 Selection Process

The goal of the selection process is to compare the approaches for synthetic event log generation presented in the previous section and select an approach that can be used as a basis for the approach developed in this thesis. The selection is made on the basis of the requirements specified in section 3.1.4. If an approach is used as a basis, it needs to be expanded or modified to allow the generation of event logs from Data Petri nets. Therefore, the selection is made based on how well an approach is suited for the extension or modification. The suitability is assessed by examining if the approaches can be modified or expanded to meet the requirements. The result of the selection process does not have to be that an approach is selected. It can also be discovered that none of the identified approaches is suited as a basis. In that case, no approach is selected, and a new approach has to be developed from the ground up.

3.3.1 Approach Comparison

In the following, the suitability of the approaches regarding each of the 12 requirements is examined. Therefore, a brief description of each of the requirements is presented (see section 3.1.4 for the detailed requirements specification). Subsequently, the suitability of the different approaches is examined. Table 3.3 provides an overview of the results of this examination.

Requirement 1: PNML Acceptance

R1 requires the developed approach to accept Data Petri net model files in the PMNL format. This requirement should not pose a problem for the approach using a **token-based simulation** because this type of process simulation is usually done using a graphical notation for the process model. Therefore, the notation of the given model can be used and no transformation of the model notation has to be performed. The **Finite State Automata** approach should also be suited for the extension to satisfy this requirement since it also uses a graph representation. However, a transformation step has to be added since the FSA notation is different from the Petri net notation. The **abduction**, **CSP**, and **SAT** approaches use a constraint-based representation to describe the modelled process. The abduction and SAT approaches are tailored toward a declarative model. Since Petri nets are procedural models, which, unlike declarative models, do not describe the process with constraints, a complex transformation step has to be added to the approach. This makes the abduction and SAT approach less suited for use as a basis for the method developed in this thesis. The CSP approach is tailored towards procedural models. This approach already incorporates a method for transforming the model into constraints, so this approach is better suited to satisfy the requirement. However, the transformation

30. https://github.com/AlloyTools/

approach is complex and could pose problems when trying to generate all possible traces (**R10**) since the execution order of activities in different sub-graphs can not be controlled. Therefore, the CSP approach is not as well suited as the token-based simulation and FSA approaches.

Requirement 2: Invisible Transitions

R2 requires the method to not include the activities related to invisible transitions in the event log. The **token-based simulation** approach and the **FSA** approach are both well suited to satisfy that requirement when extended or modified. When using these approaches, the trace generation is done in steps since the simulation is done by changing the state of the process model through one activity at a time. Therefore, an activity corresponding to an invisible transition can already be excluded from the event log during the simulation. The **abduction**, **CSP**, and **SAT** approaches are not as well suited to satisfy this requirement as the simulation-based approaches. They can probably still be expanded to meet the requirement, but this extension poses a greater problem. In contrast to the token-based and FSA simulation approaches, the trace generation is not done in steps. Therefore, the activities corresponding to invisible transitions can not be excluded during generation. An additional step has to be added to filter the generated traces and remove these activities.

Requirement 3: Basic Variable Types

R3 requires the developed approach to be able to process variables of the types *String*, *Boolean*, *Long*, and *Double*. This requirement only poses problems for approaches that rely on specific languages or tools. Therefore, the **token-based simulation**, **CSP**, and **FSA** approaches are well suited to be expanded to meet this requirement since they do not rely on any specific languages or tools. The variables supported by these approaches are governed by the extensions or modifications developed for these approaches. The same can not be said for the **abduction** and **SAT** approaches, since they rely on the tools SCIFF[31] and Alloy[32]. The SCIFF proof procedure that is used to generate traces in the abduction approach is open source and implemented in the logical programming language *Prolog*. Therefore, the tool can be extended to support any variable type that Prolog and specifically the used Prolog environment supports. Since Prolog supports the variable types necessary to satisfy the third requirement, the abduction approach is suited to satisfy this requirement. The SAT approach is not well suited to satisfy the fourth requirement since Alloy does not support numeric variables. However, the paper presenting this approach provides a workaround for this problem. Therefore, the approach is still suitable but not as well as the other approaches.

Requirement 4: Date Variables

R4 is a non-mandatory requirement that requires the approach to be able to deal with variables of the type *date*. Since this requirement is very similar to the previous requirement the suitability of the **token-based simulation**, **CSP**, **abduction**, and **FSA** approaches are identical. They are well suited to satisfy this requirement through

31. http://lia.deis.unibo.it/sciff/
32. https://alloytools.org/

extension modification of the approach for the reasons mentioned in the previous paragraph. However, the suitability of the **SAT** approach is worse since Alloy does not have a specific date variable type. Therefore, a workaround, such as transforming date variables to integer representations, has to be added to the approach.

Requirement 5: Transition Guards

R5 requires the developed approach to be able to evaluate transition guards and direct the control-flow accordingly. The **token-based simulation** and **FSA-simulation** approach are well suited for the satisfaction of this requirement. A token-based simulation approach already needs to contain a function that determines if a transition is enabled based on the current token placement. This function could be extended to consider transition guards. A Finite State Automata uses a transition function to change the state of the system. This transition function only returns a state if the input transition can be activated in the current state. This function could also be extended to consider the guards of transitions. The **abduction**, **CSP**, and **SAT** approaches are also well suited for the satisfaction of this requirement since all of them allow to formulate constraints over variables. Therefore, the transition guards can be included directly in the control-flow constraints.

Requirement 6: Logical Operators

R6 concerns the logical operators that can be present in transition guards. The operators that have to be supported are Lesser than ($<$), Lesser or equal than ($\leq$), Greater than ($>$), Greater or equal than ($\geq$), Equal ($==$), Unequal ($!=$), And (&&), and Or ($||$). Just like the suitability of the approaches to satisfy the third requirement (**R3**), the suitability for the sixths requirement is governed by how dependent the approaches are on specific languages or tools. The **token-based simulation**, **CSP**, and **FSA** approaches are therefore well suited to be expanded to meet this requirement since they do not rely on any specific languages or tools. Therefore, the logical operators they support are entirely determined by the implementation of the approaches. The **abduction** and **SAT** approaches rely on SCIFF[33] and Alloy[34]. As previously mentioned, the SCIFF proof procedure that is used to generate traces in the abduction approach is open source and implemented in the logical programming language *Prolog*. Therefore, the approach can be extended to support any logical operator that Prolog and specifically the used Prolog environment supports. Since Prolog supports all operators necessary to satisfy this requirement, the abduction approach is suitable to meet the needs of this requirement. The same is also true for the SAT approach since Alloy and the Alloy Analyzer that the approach relies on supports the logical operators necessary to fulfil the seventh requirement.

Requirement 7: Arithmetic Operators

R7 requires the support of the additional guard operators Plus ($+$), Minus ($-$), Multiply ($*$), and Divide ($/$). Since this requirement is an addition to the seventh requirement, the suitability of the approaches is very similar. Therefore, the **token-based simulation**, **CSP**, and **FSA** approaches are again well suited to be expanded to

33. http://lia.deis.unibo.it/sciff/
34. https://alloytools.org/

meet this requirement since they do not rely on any specific languages or tools. The **abduction** approach is also well suited to be expanded to fulfil this requirement since Prolog supports the necessary operators. The same is true for the **SAT** approach since the Alloy Analyzer also supports these operators.

Requirement 8: Conforming Traces

R8 concerns the generated event logs and requires that they only contain valid traces that conform to the model. The suitability of the **token-based simulation** approach to satisfy this requirement is good. If the evaluation of transition guards can be included in the function that determines if a transition is enabled, only valid traces can be generated since enabled transitions specify valid traces. The same can be said for the **FSA** approach. Only valid traces can be generated if the transition function is modified to only allow transitions to change the state when their guard is satisfied. The **abduction**, **CSP**, and **SAT** approaches are also well suited for the satisfaction of this requirement since they only allow traces that conform to the constraints. Therefore, if the constraints accurately represent the process model, the approaches can only generate valid traces. However, there is one exception regarding invisible transitions. The activities corresponding to invisible transitions have to be filtered out of the generated traces. However, this extension to the approach should pose no serious problem.

Requirement 9: Full Exploration

R9 is a non-mandatory requirement that requires that the developed approach **should** be able to generate event logs that contain all possible traces if the used model has no loop. The **token-based simulation** approach is suitable to satisfy this requirement since some of the papers found using this approach provide a method for generating all traces (for example, [vBVB14]). The **FSA** approach should also be suitable since the graph-based representation can be systematically be explored to find all possible traces. The **abduction** approach is also suitable since the SCIFF proof procedure that is used to generate the traces can already generate all possible traces. The **CSP** approach is deemed not well suited to satisfy this requirement. The approach uses a transformation technique that splits a procedural model into multiple subgraphs, which are then treated separately. This poses a problem when generating all possible sequences of activities since the order of activities in different subgraphs is not controlled in the generated constraints. Lastly, the SAT approach should be suitable for the generation of all valid traces since it can be achieved by finding all solutions to the constraint problems.

Requirement 10: Full Exploration with Loops

R10 is a non-mandatory requirement that requires that the developed method is capable of generating all traces possible with a maximum number of loop iterations from a model that contains loops. The **token-based simulation** approach should be well suited to satisfy this requirement since it should be possible to detect loops and, therefore, limit the number of loop iterations by determining if the current marking has been visited before. The **FSA** approach should also be well suited since the graph-based representation can be used to limit the number of loop iterations by checking if the place in the graph has been visited before. The **abduction** approach is deemed not suitable to satisfy this requirement since the SCIFF proof procedure that is used to generate the traces tries to generate all possible traces, and there does not seem to be a way to limit the number of loop iterations. The **CSP** approach is also deemed not well suited to satisfy this requirement since it already was not deemed suitable to generate all traces from a model without loops. Lastly, the SAT approach should be suitable for the generation of all valid traces since it can be achieved by finding all solutions to the constraint problem with a maximum number of loop iterations included in the solution. However, the adaptations necessary to limit the number of loop iterations will probably be more complex than with the token-based simulation and FSA approach.

Requirement 11: Realism

R11 requires the developed approach to generate data that is indistinguishable from real data. The event logs that are generated, therefore, have to not only adhere to the given process model but also need to be **semantically correct**. The suitability of the approaches to satisfy this requirement can not be properly assessed at this time. Since the first research objective of this thesis is to figure out if the automatic generation of realistic synthetic event logs from Data Petri nets is possible (**RQ1**), it is unknown at this point what capabilities an approach needs in order to satisfy this requirement.

Requirement 12: Output Format

R11 requires the developed method to be able to generate event logs in the XES standard. All the approaches should be well suited to fulfil this requirement since it is not tied to actual the generation of the traces and can be satisfied with a transformation step.

3.3.2 Approach Selection

This section will discuss the approach selection, which is done based on the suitability of the approaches that was described in the last section. In general, all five approaches investigated were found to be suitable for most requirements. However, potential problems were found regarding the acceptance of PNML models (**R1**), invisible transitions (**R2**), variables (**R3**, **R4**), and generating all traces (**R9**, **R10**).

The abduction and SAT approach could have serious problems when using a procedural model since the approaches use a constraint-based model representation. Additionally, these approaches are not well suited for dealing with invisible transitions, and the SAT approach is not well suited for dealing with the required variable types. Additionally, using an abduction approach could be problematic when trying to generate all traces of a model with loops. The CSP approach has similar problems to the abduction and SAT approaches since it also uses a constraint-based model representation. The approach proposes a method for transforming a procedural model into a constraints-based representation to mitigate this problem. However, this method poses severe problems when trying to generate all possible traces, leading to problems regarding the ninth and tenth requirements. Since the token-based simulation and FSA approaches do not have such severe problems, the abduction, CSP, and SAT, approaches are disregarded for the selection.

Table 3.3: Overview of the suitability of the identified approaches to be used as a basis for the method that is developed as part of this thesis

Research Question	**Token-based Simulation**	**FSA Simulation**	**Abduction**	**CSP-Solving**	**SAT-Solving**
R1	✓	O	X	O	X
R2	✓	✓	O	O	O
R3	✓	✓	✓	✓	O
R4	✓	✓	✓	✓	O
R5	✓	✓	✓	✓	✓
R6	✓	✓	✓	✓	✓
R7	✓	✓	✓	✓	✓
R8	✓	✓	✓	✓	✓
R9	✓	✓	✓	X	✓
R10	✓	✓	X	X	O
R11	–	–	–	–	–
R12	✓	✓	✓	✓	✓

- The symbol X is used to denote that the approach is deemed not well suited to satisfy the requirement if it is to to be used as a basis.
- The symbol ✓ is used to denote that the approach is deemed well suited.
- The symbol O is used to denote that the approach is deemed suitable but difficulties are expected during the development.

The token-based simulation and the FSA approach both have been deemed suitable to satisfy all the requirements. However, the FSA approach is deemed to require more extensive extension and modification due to the process representation that differs from a Petri net representation. Since the token-based simulation can use the existing Petri net process model, the approach should be easier to expand to Data Petri nets. Therefore, the token-based simulation approach is used as a basis for the approach developed in this thesis.

4. Approach Development

This chapter describes the development of the approach to generate synthetic procedural data from Data Petri nets and the approach itself. The chapter is structured as follows. First, section 4.1 presents the methodology used for the development of the approach. Subsequently, section 4.2 describes the development process of achieving the generation of traces that conform to the process model they are generated from. Section, 4.3 presents the development of an approach for generating realistic variable values for the write operations of transitions. Lastly, section 4.4 describes the approach developed in this chapter.

4.1 Methodology

This section presents the methodology used to develop the approach to generate realistic event logs from Data Petri nets. Therefore, the significant steps made during the development are presented.

The first step in developing the approach is to identify its purpose. The purpose of the approach is used to set the fundamental direction of the development. In this thesis, the approach is developed to support the research done as part of the thesis. Therefore, it should serve to support the answering of the research questions. The approach can assist with answering the first and the third research question (**RQ1**, **RQ3**). The first research question concerns whether and, if so, how realistic data, i.e. event logs, can be automatically generated from Petri Nets. Therefore, the development of the approach can be used to directly answer the second part of this question, which concerns how realistic data can be generated. Additionally, it can indirectly support answering the first part of this question, which concerns the feasibility of the data generation. In order to access if it is possible to generate realistic data from Data Petri nets, an approach is needed that can be used to generate this data. The generated data can then be evaluated and compared to real data to assess if the approach can generate realistic data. Since the approach can supply synthetic data, it can also assist in answering the third research question, which concerns how the realism of synthetic data can be evaluated.

The second step in developing the approach is to identify what the approach should do to successfully serve its previously identified purpose. In order to serve to support the answering of the research questions, the approach needs to be able to generate process data from Data Petri nets. The data should thereby be as realistic as possible in order to be able to evaluate whether the generation of realistic data from DPNs is possible.

The third step in the development of the approach consists of identifying what exact resources and information the approach is used with and what the desired result of using the approach should be. These are essentially the inputs and outputs of the approach. As previously mentioned, the approach should generate process data from Data Petri nets. The first requirement (**R1**) that was specified in the requirements specification in section 3.1.4 requires the approach to accept process models in the Petri Net Markup Language. Therefore, the approach should start with a model in this format. Additionally, the approach might use additional information provided by a user. The result or the output of the approach should be a synthetic event log in the eXtensible Event Stream format since it is required by the twelfth requirement (**R12**). The resulting event log should be as realistic as possible, meaning that it should be as indistinguishable from a real event log as it is possible.

The final and, at the same time, the most complex step of the development is to figure out how to fill the gap between the present input and the desired result by using the method selected in section 3.3.2. The method selected is the token-based simulation. Due to the complexity of this step, the development is simplified by splitting this step into two sub-steps. First, a way to use a token-based simulation to generate traces that conform to the model has to be developed. The generation of values for the models' variables (including timestamps for events) is ignored during this first step since it poses the main challenge of including the data perspective. Therefore, the second sub-step consists of expanding the approach to generate realistic values for the variables present in the model. The development of an approach for generating conforming traces is described in section 4.2. Section 4.3 presents the development of an approach to generate realistic variable values.

4.2 Generating Conforming Traces

This section will focus on solving the problem of using token-based simulation to generate conformant traces. However, the generation of variable values is ignored for the time being.

In order to perform a token-based simulation, the approach needs a model representation that has the necessary features for such a simulation. In the following, these features are described. First, the representation needs to be able to represent the current state of the model. Therefore, it needs to store the current marking, i.e., the number of tokens in every place and the current values of all the variables in the model. Additionally, the representation needs to allow the model's state to change. Therefore, it has to be able to determine the enabled transitions in regard to the current marking and the variables and be able to fire transitions. In order to determine which transitions are enabled, the representation has to be able to represent places, transitions, and the arcs between them to determine if all input places of a transition contain at least one token. Additionally, the representation has to be able to evaluate transition guards in regard to the current variable values. This is necessary to determine whether a transition is enabled. Additionally, the model representation has to be able to fire transitions correctly. Therefore, it has to be able to update the marking after a transition has fired. The model representation also has to be able to determine if the current marking is a valid final marking to provide a means of properly ending a simulation run. The last thing that the model representation has to be able to do is to reset the model. This is necessary to perform more than just one simulation run. To reset the model, the current marking and the values of the variables have to be reset to their initial configuration. Therefore, the representation also has to be able to store the initial marking and any initial values that exist for the variables in the model.

As previously discussed, the input of the approach includes a Data Petri net stored in a Petri Net Markup Language file. Therefore, the approach must contain a step that reads this file and parses it into the previously discussed model representation that allows to perform a token-based simulation.

Due to the nature of token-based simulation, the generation of conforming traces is no longer a challenge after the model representation has been created. Whether a trace conforms to the model is determined by whether only enabled transitions are fired. Additionally, depending on the scope of the 'conforming' definition, it might also be considered whether a trace reaches a valid final marking or ends prematurely. A trace can end without reaching a final marking by ending up in a deadlock in which no transitions are enabled. These restrictions can be easily met by using the previously discussed features of the developed model representation. During a simulation run, the representation can be used to determine which transitions are enabled and therefore, it can be ensured that only enabled transitions can be fired. Additionally, the ability of the representation to determine if a final marking is reached can be used to end the simulation properly or exclude traces that end prematurely.

For the token-based simulation, a strategy is needed to decide which transition to fire if multiple transitions are enabled and how many simulation runs are to be performed. Since there are many different possible strategies, it would be advantageous if the simulation could be configured. As previously discussed, the approach can contain user input as an additional input. Therefore, the configuration for the simulation can be controlled by the user's input.

In order to generate traces from the token-based simulation, the sequence of fired transitions during each simulation run needs to be stored. However, only transitions that are not invisible should be included. Additionally, the current variable values must also be stored at every event (i.e., transition firing). Therefore, the approach needs to contain a log representation similar to the model representation that allows storing the previously described information. Lastly, every event needs to have a timestamp. Therefore, the simulation has to include a simulated clock that advances with every fired transition.

As previously discussed, the desired output of the approach is an event log in the Extensible Event Stream format. Therefore, another step has to be added after the token-based simulation. This step is the last step necessary to reach the desired output of the approach. It synthesises an XES-File from the event log representation that contains the generated traces.

4.3 Generating Realistic Variable Values

During the token-based simulation step of the approach described in the previous section, transitions are fired that write values to variables. Realistic values have to be generated for these write operations in order to achieve the goal of generating synthetic event logs that are indistinguishable from authentic event logs.

Variables in a PNML file are defined with a name and a type (e.g., String). Numerical variables may additionally have a minimum and maximum value. There is no other information about the values of variables directly defined in a standard DPN model described in the PNML. This poses a problem since a DPN does not contain enough semantic information to generate realistic values.

The only other information about possible values for variables in a standard DPN modelled in the PNML is present in transitions guards. Therefore, the approach should include a step in which the transition guards of the model are analysed to infer semantic information about variables. This ensures that all the information contained in the DPN is extracted. For example, if a transition has the guard 'cancer_type == "Leukemia" ' it can be inferred that 'Leukemia' is a realistic value for the variable 'cancer_type'. In addition to variable values, transition guards may also contain information about value ranges for numeric or date variables. For example, if a transition has the guard 'oxygen_saturation < 90' it may make sense to generate values above and below 90 for the variable 'oxygen_saturation' in order to have some traces that satisfy this guard and some traces that do not.

However, it is unlikely that all possible values for the discrete variables are present in transition guards. Additional, the dependencies between variables are not described. Suppose a model describing the process of diagnosing and treating patients in a hospital has a variable 'gender' that is assigned the value 'female'. In this case, the variable 'cancer_type' should not be 'prostate cancer' since a female patient does not have a prostate. These kinds of dependencies have to be considered if the goal is to generate event logs that are indistinguishable from authentic event logs. In addition to the possible values and dependencies of variables, the distribution of numeric and date variables has to be considered. For example, in the same model, as used in the last example, the distribution of the variable depicting the patient's age may not be uniform since the age of patients coming into a hospital is usually older than the population's average. Therefore, the generation of values for this variable should consider using distributions that are not uniform. However, the information that can be inferred about numeric and date variables does not allow to determine accurate distributions that result in realistic values.

The previously illustrated examples show that the problems regarding realistic variable value generation can not be solved by using only information extracted from the provided model. Therefore, another information source for semantic information has to be added to the approach to generate realistic event logs. The obvious choice is to have the additional information be provided by the user who is also providing the model since he should have more knowledge about the variables in the model. The provided information should thereby consist of possible values for variables, dependencies between variables, and distribution of numerical variables. Additionally, information about the timestamp of events has to be provided to generate event logs with realistic timelines. As previously illustrated, some of the necessary information about variables can be inferred from the model itself. Therefore, the information acquired in the previously described analysis step of the approach can be used to provide the user with a basis for the input of additional information. The additional information can then be added to the internal model representation so it can be used in the token-based simulation to generate more realistic values for the variables.

4.4 Approach for Generating Event Logs from Data Petri Nets

This section describes the approach for generating realistic synthetic event logs from Data Petri net, whose development has been described in this chapter. Figure 4.1 shows an illustration of the approach.

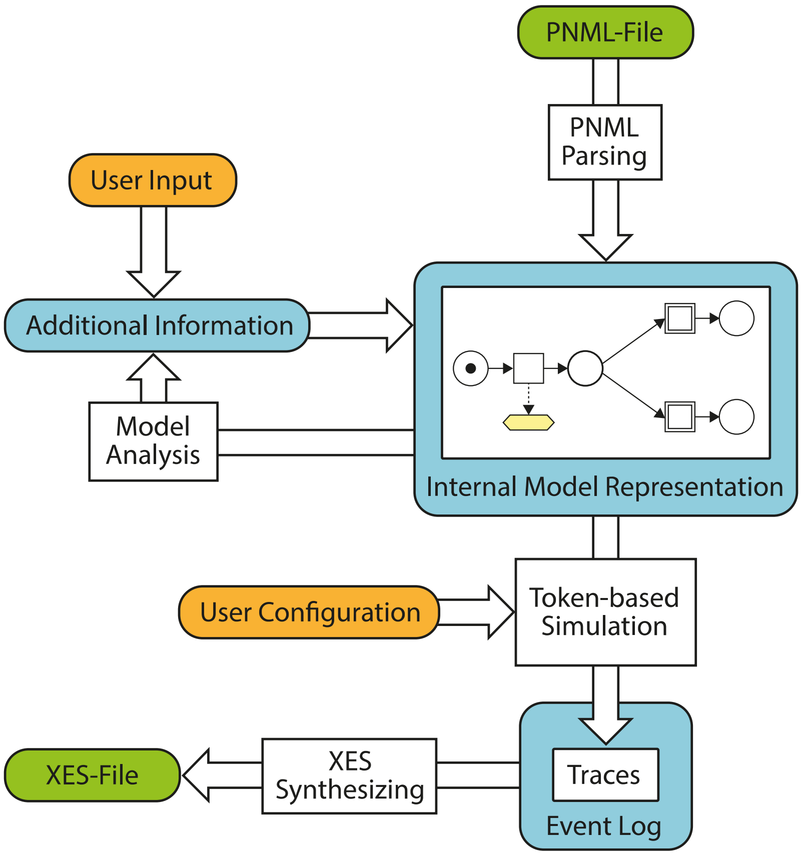


Figure 4.1: Overview of the developed approach to generate realistic synthetic event logs from Data Petri nets

The approach starts with a Data Petri net modelled in the Petri Net Markup Language. It is supplied by the user as a '.pnml' file. This model file is parsed into an internal model representation. The internal model representation provides features that allow the model to be used for a token-based simulation. These features allow controlling the state of the model. This includes functions for determining the currently enabled transitions, firing transitions, and determining if the model is in a final state. Additionally, the model representation allows the model to be reset to its initial state to allow for multiple simulation runs to generate multiple traces.

After the model has been parsed, it is analysed to acquire all the semantic information available in the model. The information that is directly specified in the model is already obtained during the parsing into the internal model representation. However, the model contains information in the transitions guards that can not be read directly and has to be extracted through the analysis of the guards. The information extracted from the model is provided to the user as a basis for the additional semantic information he is expected to provide. The additional input is necessary since the information present in the model is not sufficient to generate realistic event logs.

The user can provide realistic values for the variables of the model and dependencies between the variables. Additionally, he can provide maximum values, minimum values, and information about the distribution of numeric variables. For the transitions in the model, he can provide weights that are used when multiple transitions are enabled, and a random decision has to be made. Additionally, he can provide information about how the firing of each transition should affect the simulation clock. Therefore, this information can be used to control the timestamps of events in the generated log.

The additional information provided by the user is included in the model representation so it can be used during the token-based simulation. In addition to semantic information, the user can also configure the simulation. This includes, for example, the amount of traces that should be generated or the minimum length a trace should have to be included in the event log.

Once the user has configured everything, the token-based can generate the desired event log(s). A single simulation run is performed for each trace that needs to be generated. Each simulation run consists of firing random enabled transitions until a final marking, or a deadlock has been reached. The random decision performed if multiple transitions are enabled is influenced by the weights provided by the user. An event is created in the trace for each transition fired during a simulation run. If the fired transition writes values to any variables, the semantic information provided by the user is used to generate realistic values for these write operations.

After the simulation has finished, only one step remains in the approach. This step consists of synthesising an event log in the Extensible Event Stream format by combining the generated traces. The log is thereby saved in an '.xes' file.

5. Conception and Implementation

This chapter presents the concept developed to implement the approach described in chapter 4. Additionally, the implementation of the concept is described. The chapter is structured as follows. First section 5.1 describes the methodology used to develop the concept. Subsequently, section 5.2 provides an overview of the developed concept. The three parts of the concept, namely the entry and exit point, the model representation, and the simulation part, are then explained in sections 5.3, 5.4, and 5.5. Section 5.6 provides a detailed description of the algorithmic approaches developed for the concept. Lastly, section 5.7 describes the software that was implemented based on the developed concept.

5.1 Conception Methodology

This section will cover the methodology used to develop the concept for implementing the approach. The development of the concept is the first step toward evaluating the approach for generating realistic event logs from DPNs. This step aims to create a basis that can be used to direct and structure the implementation. Additionally, it is used to elaborate on the approach since it is still missing many details necessary to implement it.

During the concept development, the first step consisted of translating the approach to a basic class structure. Each step of the approach was therefore put into a designated class. The more complex steps, such as the token-based simulation, were split into multiple subclasses. For example, the simulation has a subclass responsible for generating variable values since this is a rather complex task. Subsequently, the necessary classes for storing the data, such as the created event logs, were added. Therefore, the result of the first step is a class structure based on the approach.

The second step consisted of developing additional features that are not part of the approach but that are beneficial for implementing software that allows a user to generate realistic event logs. For example, a class that provides a *graphical user interface (GUI)* to the user was added to the previously developed class structure. Therefore, the first two steps result in a complete class structure that shows which classes provide which functions and how the classes interact and work together.

The third, last, and most complex step of the concept development consisted of developing detailed concepts for the classes' functions. As previously mentioned, the developed approach only consists of a sequence of steps without specifying how these steps should operate. Therefore, the function of these steps needs to be elaborated. The development of these elaborations was influenced by the goal of creating realistic event logs as well as the requirements specified in section 3.1.4. The approach was specified to allow these requirements to be met by an implementation of the approach. However, due to how general the specification of the approach is, it is possible to develop an implementation of the approach that does not satisfy the requirements. Therefore, the requirements have to be considered during the development of the concept. In the following, the most complex functions of the approach that were elaborated on during the development of the concept are described.

One of the more complex parts of the concept that had to be elaborated on was the token-based simulation step. Therefore, an approach had to be developed to determine which transitions are fired during the simulation. The ninth and tenth requirements (**R9**, **R10**) require that it should be possible to generate all traces from a model. Therefore, a token-based simulation approach for this purpose had to be developed. However, this type of approach is contractionary to the goal of generating realistic event logs since it is not realistic that an event log contains all possible traces. Therefore, it was decided to develop multiple simulation approaches and give the user a choice to choose an approach. Another complex approach that needed to be developed was the MILP-Solver that is used in the model representation to determine if a transition is enabled. Again the requirements were used as a basis to determine what the algorithm for the evaluation of transition guards has to be able to do (**R3**,**R4**,**R6**, and **R7**).

The last rather complex part of the concept that had to be elaborated on is the semantic information that the user can provide and the configuration options for the simulation. The methodology used for specifying the options and semantic information that the user can configure follows the goal of giving the user as much control as possible. The developed approach builds on the premise that the user knows best about how a realistic event log should look like for the model he provides. Therefore, the more control the user has over the simulation and the generation of variable values, the higher the chance that the generated event logs are as realistic as possible. When developing the concept, care was taken to add as many configuration options for the simulation as possible. However, this could also overwhelm the user. For that reason, it was decided that most options should have standard values that the user does not have to touch if he does not want to change them. The same procedure was followed for the specification of the simulation configuration options.

5.2 Concept Overview

Figure 5.1 shows a simplified class diagram of the concept that illustrates its parts and how they work together. Due to size constraints, the diagram is missing the class attributes and methods. In the figure, the concept is split into three parts highlighted by different colours. These parts will each be covered in detail in individual sections. In the following overview, the three parts will be referenced by their colour in this figure.

The classes highlighted in blue serve as the program's entry and exit point and control its execution. The 'main' class serves as the entry point of the program when the program is controlled through a command-line interface. However, when the program is controlled using the graphical user interface, the 'Gui' class is used to control the 'main' class. The 'main' class is provided with a path to a PNML-file and uses the 'PnmlReader' class to create an internal representation of the model described in the given file. It then uses the 'ModelAnalyser' class to infer additional

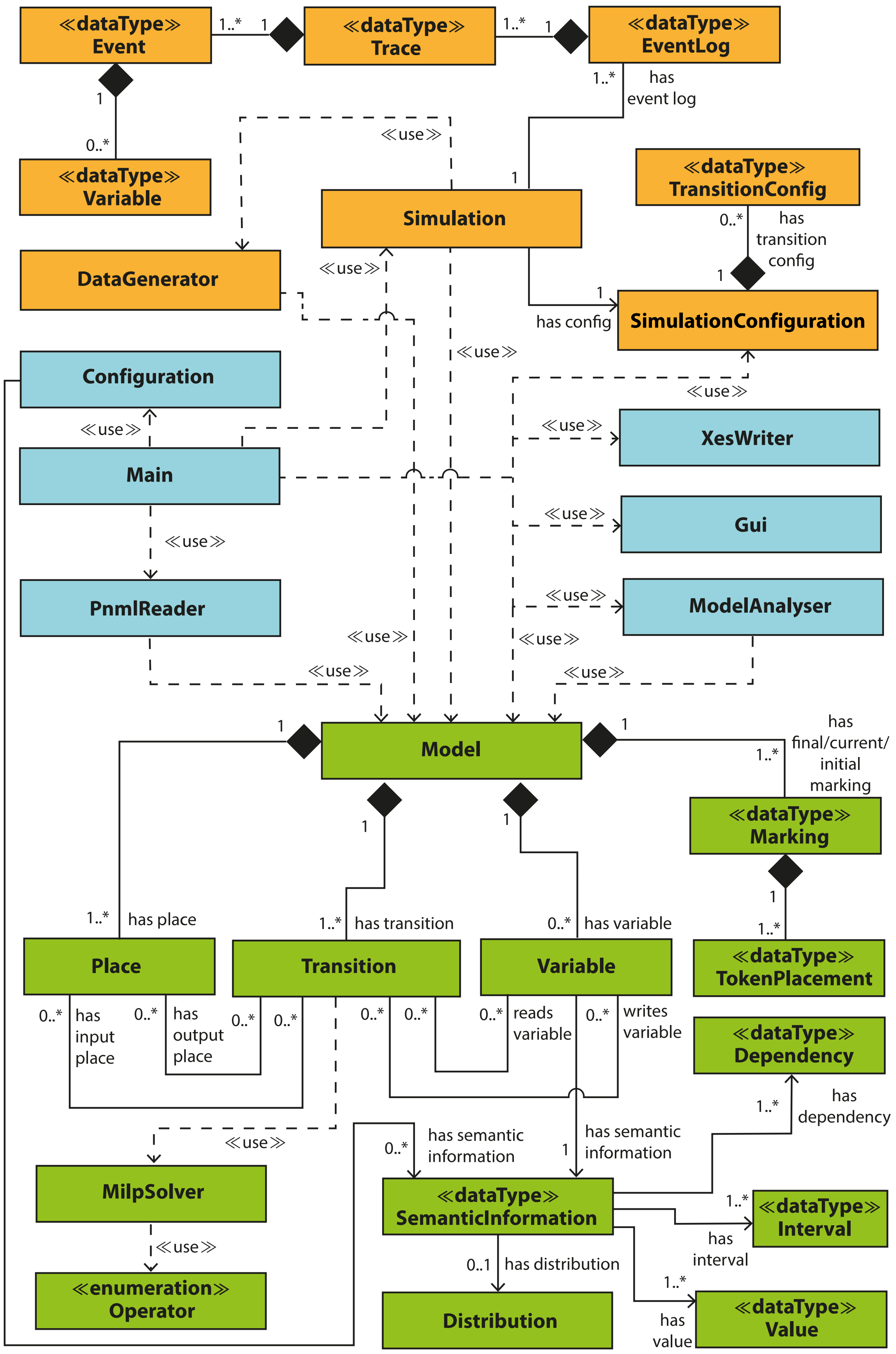


Figure 5.1: Class diagram for the concept

information for the variable generation that is then provided to the user as a basis for his input. It also allows the user to provide additional semantic information and configure the simulation. The 'Configuration' class is used to store the configuration of the program. It also provides the functionality to save the configuration to a file or to read a configuration file. Lastly, the 'main' class runs the simulation and exports the generated event logs using the 'XesWriter' class.

The classes highlighted in green are used for the internal DPN model representation. As described in section 4.2 the model representation is needed to run a token-based simulation. The model representation consists of places, transitions, variables, final markings, and an initial marking. The arcs of the DPN are represented by places and transitions having references to the transitions or places they are connected to. The class that represents variables uses three additional classes for the additional semantic information, such as valid values or dependencies to other variables. The representation also includes a MILP-Solver class used to evaluate guard transitions and determine if a transition is enabled. This is necessary to generate traces that conform to the data perspective of a DPN.

Lastly, the classes highlighted in orange are used for the token-based simulation. The 'Simulation' class is used by the 'main' class to run the simulation. The configuration that the user provides for the simulation is stored in the 'SimulationConfiguration' class. During the simulation, the 'simulation' class makes use of the model representation, the 'DataGenerator' class, and the 'EventLog' class. The 'DataGenerator' class is used to generate values for variables that are written to when executing transitions. The 'EventLog' class is used to store the generated traces until they are written to an event log file.

5.3 Entry and Exit Point

This section will cover the part of the concept highlighted in blue in figure 5.1. Figure 5.2 shows a more detailed class diagram of this part that includes class attributes and methods. As previously mentioned, the 'main' class acts as the entry and exit point of the program and controls its execution. The user utilises this class to control the program either through a GUI provided by the 'Gui' class or through a command-line interface. Therefore, the GUI is separated from the actual program and is not necessary to generate event logs.

The concept follows the approach described in chapter 4. Therefore, the first step in the program that this concept describes is to load a model file. The model file that contains the Data Petri net is provided by the user. The model file needs to be present in the PNML file format ('.pnml'). The program opens the PNML file and parses the model to an internal representation by using the 'PnmlReader' class. The internal model representation will be described in detail in section 5.4. In addition to the methods used for reading a PNML file, the 'PnmlReader' class also provides methods for checking whether a model contains any inconsistencies or problems. If severe problems are found during the parsing, the reading of the model file might be stopped. In any way, the user is informed about any problems that might affect the functionality of the program to perform a token-based simulation.

After the model file has been parsed, the 'main' class uses methods provided by the 'ModelAnalyser' class to analyse the model. This is done in accordance with the developed approach and has the goal of extracting all information from the model that

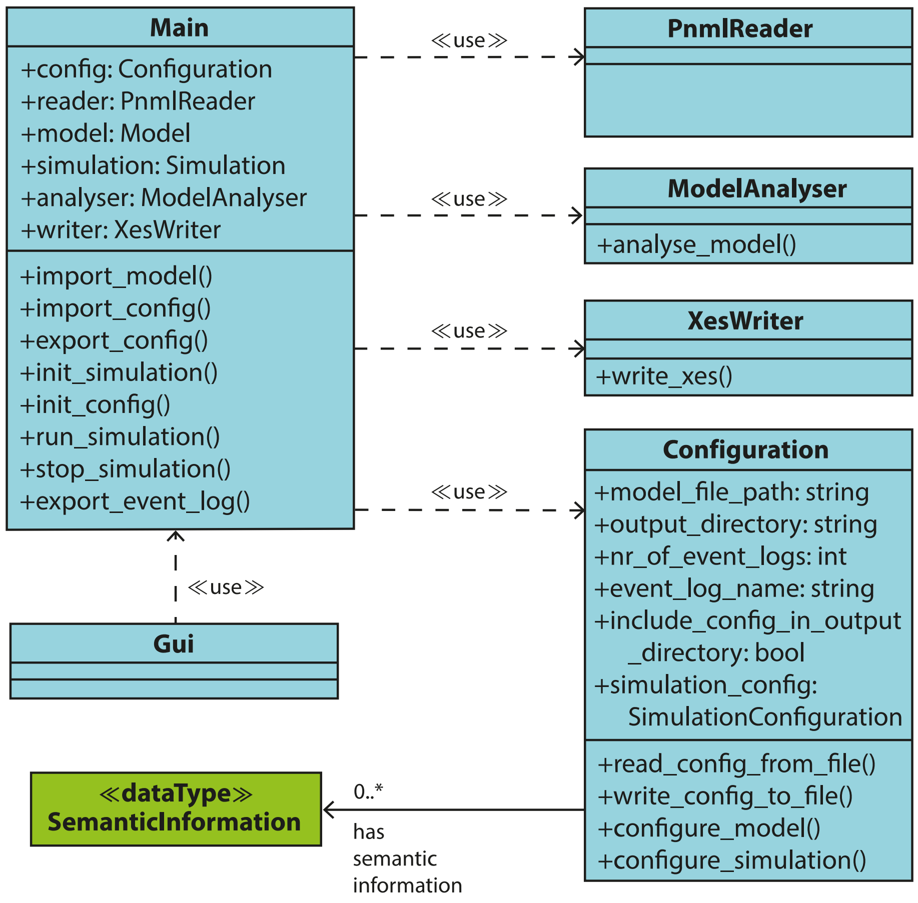


Figure 5.2: Entry and exit point of the concept

can be used to generate more realistic event logs. Therefore, the 'ModelAnalyser' class inspects every transition guard that is present in the model to infer additional information about the variables of the model. This information consists of possible values for variables that make sense on a semantic level and possible value intervals for numeric and date variables. Identifying intervals for numeric variables is advantageous since it makes it easier to generate values that satisfy transition guards. This is helpful when trying to fully explore all possible paths in a DPN. The reason why these intervals are analysed will be explained in more detail in the section 5.5 in which the simulation part of the concept is explained. The following paragraph will give some insights into how the analysis of the transition guards works.

The analysis of transition guards works by searching for variables being compared to values with the equal operator ('==') or the greater or lesser operators ('<', '>', '<=' and '>='). If a transition guard checks whether a variable is equal to a specific value, it can be inferred that this value is a realistic value for that variable. If it is checked if a variable is above or below a specific value, the value is the boundary of an interval that is inferred from the transition guard. The following example illustrates the analysis of transition guards. Figure 5.3 shows an example of how a transition with a transition guard is defined in a PNML file. The analysis of the transition guard in this example would result in the value 'emergency' being considered a realistic value for the variable 'patient_status'. Additionally, the interval '[blood_oxygen_minimum, 90)' would be stored.

```
<transition guard="(patient_status=="emergency") || (blood_o2 < 90)" id="1">
  <name>
     <text>treat</text>
  </name>
</transition>
```

Figure 5.3: Exemplary definition of a transition with a transition guard in a Data Petri net modelled in the PNML

The additional information that could be inferred through the analysis of the transition guards is used for the generation of realistic variable values. However, the user has the option to disregard any of the inferred information since the analysis might result in values or intervals that are not desired to occur in the event log. Information about how the additional information is stored in the model representation will be provided in the section 5.4 which covers the internal model representation. Additionally, section 5.6.3 will provide insights on how this information is used for the generation of variable values.

As discussed in section 4.3 the information that is contained in a model is in most cases not enough to generate realistic event logs. Even with the additional information gained from analysing the transition guards, generating realistic event logs without another information source is not feasible. Therefore, additional information about what realistic event logs should look like for the loaded model has to be provided by the user. The 'main' class, therefore, also allows the user to provide such information. The class accepts the additional semantic information from the user and includes it into the model representation so it can be used in the token-based simulation. The information that can be provided consists of semantic information about the variables of the model and information about the timestamp of events. What kind of information can be provided and in what form it needs to be provided is explained in detail in the section 5.4 which concerns the model representation.

In addition to the configuration of the variable information, the 'main' class also allows the user to configure the simulation as well as some general settings that affect the created event log files (e.g., event log file name). The configuration options for the simulation will be covered in section 5.5. The configuration options for the generated files are displayed as class attributes of the 'Configuration' class in figure 5.2. They consist of the output directory, the name of the generated event log files, the number of generated event logs, and whether to include the configuration in the output directory. All the configuration options set by the user are stored in an 'Configuration' class object. This class also provides methods that allow the user to save the entire configuration to a file. The saved configuration file can then be loaded again at any time to restore the configuration to the saved state. This is advantageous since the configuration can be relatively complex to set up and is otherwise lost when the program is closed. The *JavaScript Object Notation (JSON)*[1] is used to save the configuration file. This format was chosen because it is widely supported and easily readable by humans. Therefore, a configuration file can be edited without using the GUI by simple editing it with a text editor.

1. https://www.json.org/json-en.html

Once everything has been configured, the 'main' class allows the user to run the simulation to generate the desired event logs. The 'main' class, therefore, makes use of the simulation class to run the simulation. After the simulation has ended or the user aborted it, the 'main' uses the 'XesWriter' class to write the generated event logs to XES files in the output directory.

5.4 Model Representation

This section will cover the internal model representation that is highlighted in green in figure 5.1. Figure 5.4 shows a more detailed class diagram of this part that includes class attributes and methods. The internal model representation serves the purpose of allowing the 'Simulation' class to perform the token-based simulation. It, therefore, provides the features necessary to perform such a simulation.

The internal model representation is embodied by the 'Model' class. The class has all the methods necessary to control the model state during a token-based simulation. It has a method to check whether the Petri net is in a final state, a method to get a list of enabled transitions, and a method that allows to fire a transition and, therefore, change the model's state. Additionally, the class has a method that resets the model to the initial state. This is necessary to generate more than one trace during the simulation. The actual model representation consists of class objects of the 'Place', 'Transition', 'Variable', and 'Marking' classes. The 'Marking' class is used to represent the current marking, the initial marking, and the final markings of the model. Each part of the model representation is described in detail in the following sections.

5.4.1 Places and Markings

The places of the DPN are represented by objects of the 'Place' class. Each place is referenced by an ID and has a name. Additionally, every place has references to the transitions that it is connected to. These references represent the arcs in the model. These references can be differentiated by the place being an input or an output place of the transition. The number of tokens that are in each place in a given model state is not stored in the place objects but in the objects of the 'Marking' class.

The marking class represents the initial, current, and final markings. The initial marking needs to be stored in order to be able to reset the model for multiple simulation runs. Each marking object consists of multiple objects of the 'Token-Placement' data type. Each token-placement object consists of a place ID and an integer value representing the number of tokens in that place. Therefore, a marking of a model consists of the number of tokens in every place.

Comparing the Current Marking to the Final Markings

In order to determine if a model is in a final state that is specified in the model file, the current marking of the model is compared to the specified final markings. This comparison is made by comparing the number of tokens in every place in the model. However, some DPNs modelled in the PNML do not include all places in the definition of the final markings. There are two options available to deal with the missing places. First, it can be assumed that all places not present in a defined

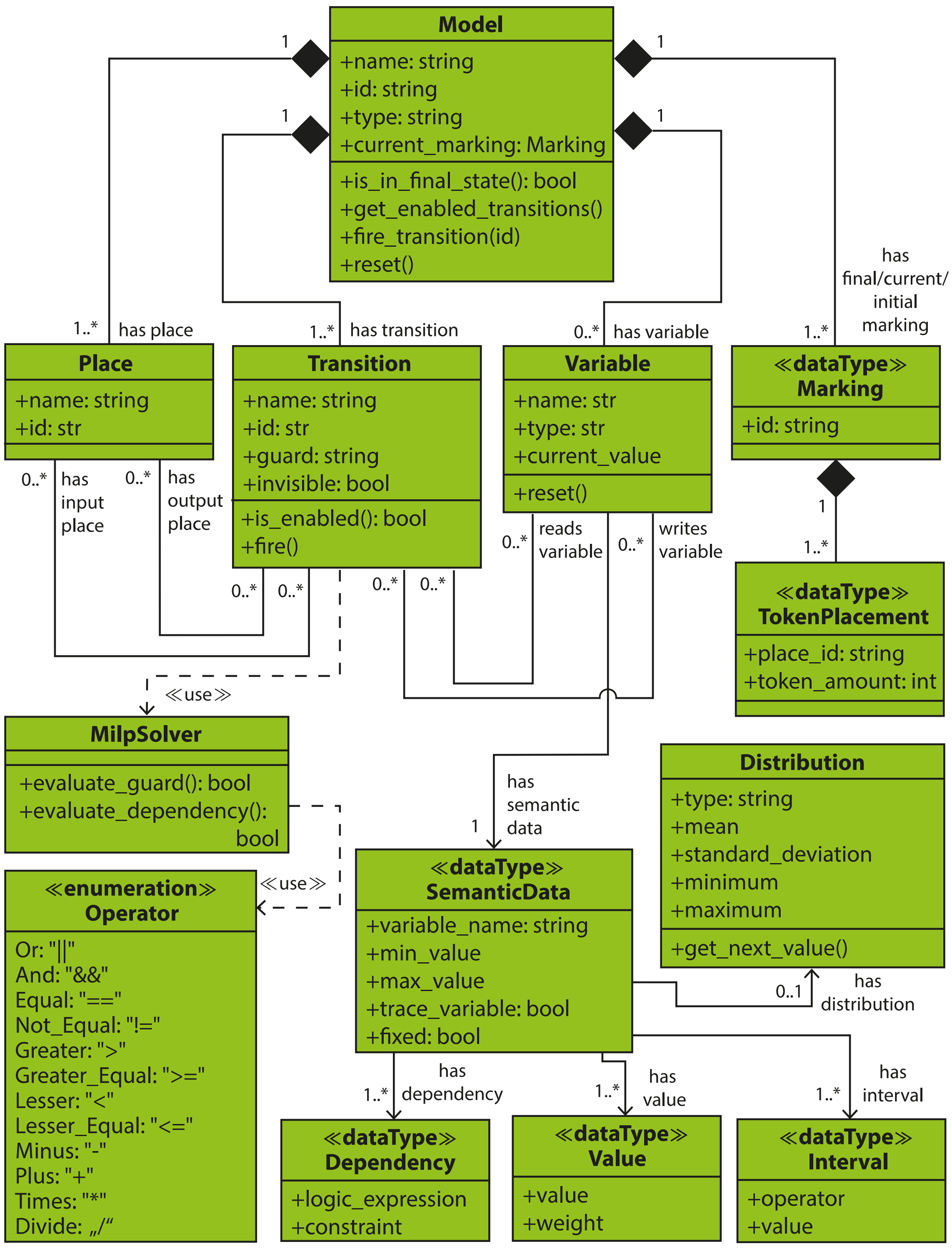


Figure 5.4: Class diagram representing the internal model representation of the developed concept

final marking contain zero tokens. The second option consists of only comparing the number of tokens of the places present in a given final marking. The latter option was chosen while developing the concept described in this chapter. This option was chosen because it provides more options to the person who creates the model. Consider the example following example. The model's creator wants the traces of a model to end if a token is present in a place designated as the final place of the model. There could still be tokens left in the model in different places when a token reaches a final place. If the first method of comparing markings is chosen, the model would need to contain every possible combination of tokens left in different places in the model. However, with the method chosen, the model has to contain only one final marking that contains a token in the final place. Therefore, the effort of modelling is significantly decreased.

5.4.2 Transitions

The transitions of the DPN are represented by objects of the 'Transition' class. Like the places of a model, transitions are also referenced by an ID and have a name. The names usually refer to the activities that the transitions represent. If the transition is fired, the transition's name is also used as a name for the event created in the event log. Additionally, a transition can be flagged as invisible and have a transition guard. The guard of a transition is stored as a string attribute. As previously mentioned, the arcs of a DPN are represented by references between places and transitions. Therefore, every transition has references to the place objects representing its input and output places. In addition to these references, transitions also have references to the variables they read and the variables they write to. The 'Transition' class has two methods that are used by the 'Model' class to change the state of the mode. The first method determines if the transition is enabled. The second method fires the transition and changes the marking of the model. However, the transition does not generate values for the variable it writes to. The generation is handled by the 'Simulation' class. In order to evaluate whether the transition is enabled, it has to be checked if the input places contain enough tokens, and the transition guard has to be evaluated. Since the evaluation of the transition guard is relatively complex, it is outsourced into a separate class. The 'MilpSolver' class is used by the transition when its guard has to be evaluated. It can evaluate the logical expression in transition guards with regard to the current values of variables in the model. The transition guards of a DPN can have nested logical expressions, which need to be evaluated. Therefore, the rough idea for the MILP-Solver consists of recursively evaluating the nested logical expressions one by one. The algorithmic approach for the MILP-Solver will be covered in detail in section 5.6.1.

5.4.3 Variables

The variables of the DPN are represented by objects of the 'Variable' class. The variables are referenced by a name since variables in the Petri Net Markup Language do not have Ids. Each variable also has a type, such as Integer and a current value. Additionally, it can be specified that the variable is a trace variable. In this case, the variable will not be part of any events but rather the trace itself. In addition to these settings, variables can have a data object of the class 'SemanticData'. This class contains the additional information that the user provides and that is used to generate realistic values for this variable. In the following, an overview of the semantic information that can be contained in the data object is presented.

There are four types of semantic information that can be provided for the variables of the DPN. Additionally, semantic information objects for numeric and date variables can contain a minimum and maximum value. In the following, the four types of semantic information are described.

Values The first type of semantic information that can be provided for the variables consists of realistic values. When a value needs to be generated for a variable, the 'Simulation' class can choose one of the provided values. This ensures that the generated values are realistic. This type of information is especially useful for variables of the type String since, otherwise, random strings would have to be generated. In addition to the list of values, a list of weights can be provided. The weight of a value determines how likely a value is chosen.

Dependencies The second type of semantic information that can be provided for variables consists of dependencies between variables. As previously discussed in section 4.3, the dependencies between variables need to be considered in order to generate realistic event logs. Therefore, the user can provide information about these dependencies. Each dependency consists of a logical expression and a constraint. The logical expression is a string with the same syntax as the transition guards of a DPN. Therefore, it needs to be possible to evaluate it to *true* or *false* in regard to the current values of the model's variables. If the logical expression is evaluated to be true, then the dependency constraint restricts the variable's possible values. The constraint consists of an operator and a value. The operators that are allowed are '==', '!=', '>', '>=', '<', and '<='. To better illustrate the concept of dependencies, consider the following example for a variable representing the type of cancer a patient has been diagnosed with.

```
'gender == female' => (!=,'prostate cancer')
```

In this example, the dependency describes that it would make no sense that a female patient has prostate cancer. The constraint uses the '!=' operator to represent this fact.

Distributions The third type of semantic information that can be provided for variables are distributions. This type of information can be only provided for numeric and date variables. An object of the 'Distribution' class acts as a *(pseudo) Random Number Generator (RNG)* that generates values following a specific distribution, such as the normal distribution. The user can thereby specify the minimum value, maximum value, mean value, and standard deviation of the distribution.

Intervals The final type of semantic information that can be provided for the variables consists of numeric intervals. This type of information can be only provided for numeric and date variables. Each interval consists of an operator, such as '<', and a boundary value. The operators that are allowed for intervals are '<', '>', '<=', and '>='. For example, the interval `(<, 5)` reaches from the minimum of the variable to the excluded value 5. This type of information is useful to achieve the generation of values that satisfy specific transition guards. The following section provides more details on why this type of information is useful.

5.5 Simulation

This section will cover the simulation part of the concept that is highlighted in orange in figure 5.1. Figure 5.5 shows a more detailed class diagram of this part that includes class attributes and methods. This section is structured as follows. First, the components, i.e., the classes, of the simulation part and their functions are described briefly. Subsequently, the basic approach used to perform the token-based simulation is explained in section 5.5.1. Lastly, the configuration options for the simulation are presented in section 5.5.2.

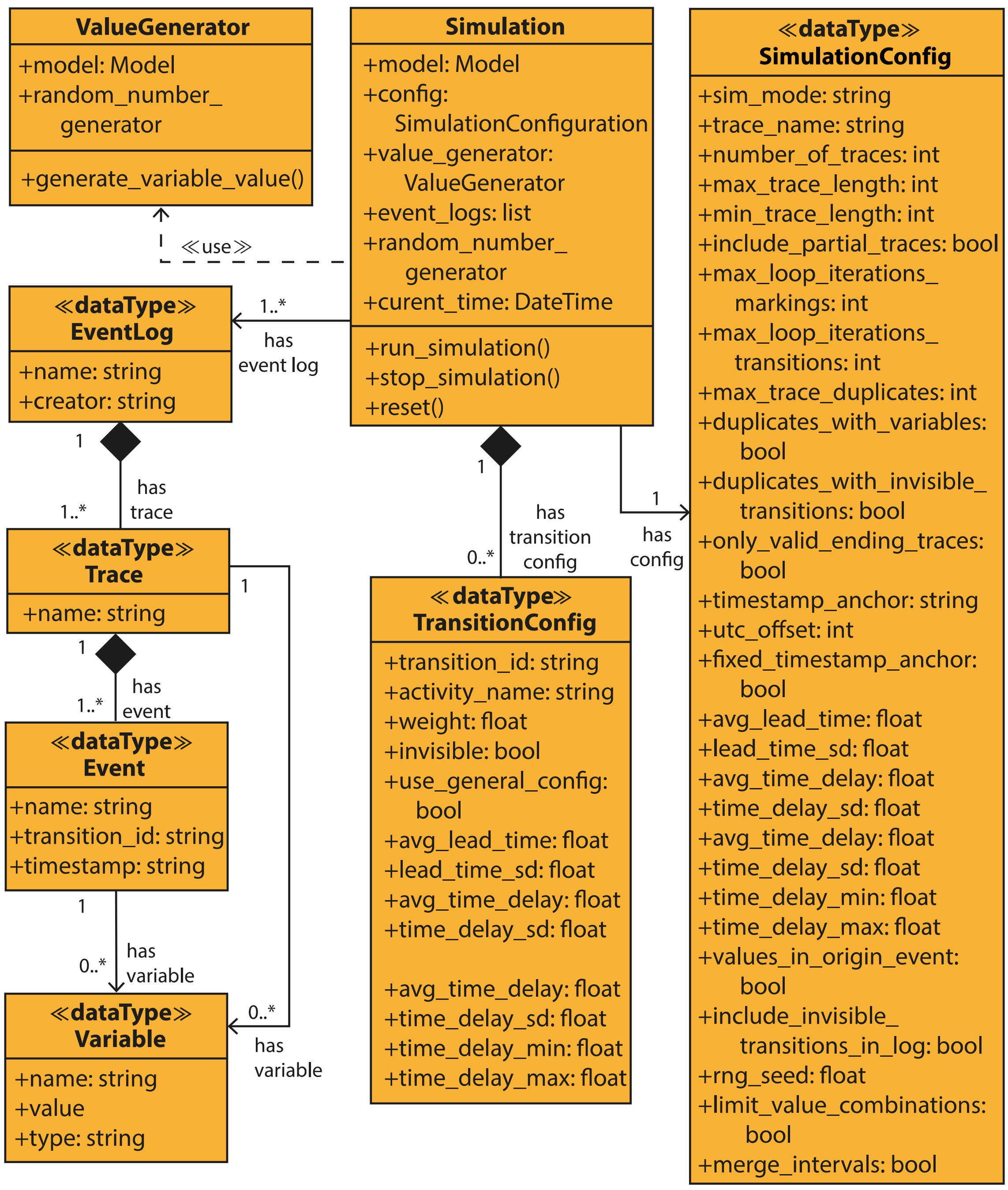


Figure 5.5: Class diagram representing the simulation part of the concept

The 'Simulation' class is used by the 'Main' class to generate event logs through a token-based simulation. Therefore, it provides methods for starting, stopping, and resetting the simulation. During the simulation, the 'DataGenerator' class is used to generate values for all write operations of transitions. The generation considers the semantic information provided for all the variables and tries to generate variable values that are as realistic as possible.

The event logs generated during the simulation are stored in data objects of the 'EventLog' class. Each event log object consists of multiple 'Trace' data objects. Each 'Trace' consists of a sequence of events which are represented by the 'Event' data type. Additionally, each trace can have a list of trace variables. Each event has a name, a timestamp, and a list of variables representing the state of the global variables when the event occurred. The name of the event represents the transition whose firing resulted in this event.

5.5.1 Simulation Approach

This section will cover the basic approaches used to perform the token-based simulation to generate traces from the DPN. The detailed algorithmic approaches developed for the token-based simulation will be covered in section 5.6. During the development of this concept, two approaches for the simulation were developed. The user can configure which of the approaches should be used. The first approach consists of generating traces by firing random transitions until a deadlock or final making is reached. The second approach tries to generate all possible traces of a model by systematically exploring the model's paths. The following two paragraphs will explain the basic idea of these approaches and the simulation modes that are provided to the user based on these approaches.

Random Trace Generation

As previously mentioned, the random trace generation approach randomly traverses through the Petri net to generate sequences of executed activities. The approach consists of a series of simulation runs that each generate a single trace.

A simulation run starts with the initial marking of the model. The approach then consists of firing random transitions until a deadlock or a final marking has been reached. Therefore, in each step of the simulation, the transitions that are enabled with regard to the current marking and the current values of the variables are determined. Subsequently, a weighted random decision is made to determine which enabled transition is fired. The chosen transition is then fired to change the state of the model. Additionally, the 'DataGenerator' class is used to generate values for each variable that the transition writes to. In order to be able to generate an event log from the simulation, the sequences of fired transitions are recorded. Additionally, the state of the model's variables when a transition is fired is also recorded.

The random trace generation approach can also be used to explore a DPN. Therefore, the user can choose between two simulation modes based on this approach. The first mode generates the desired amount of random traces with the specified configuration. The second approach continuously explores the DPN until a maximum number of traces is reached or the user stops the simulation. The exploration of the DPN using the previously described approach can be done by continuously generating random traces and only keeping the traces that have not been generated before. If the simulation is run for a long enough time, this simulation mode can generate all possible traces. However, this type of exploration could be very slow if the DPN has traces that require very specific variable value combinations since the values of the variables are randomly generated. Therefore, as previously mentioned, the concept contains a second approach targeted at systematically generating all traces possible in a Data Petri net. This approach will be explained in the following paragraph.

Generating all Traces

The approach developed to fully explore a DPN is radically different from the previously described random trace generation approach. The approach systematically explores the DPN to make sure that all possible traces are found.

The idea of the approach consists of determining all traces possible while only considering the control-flow perspective (and invisible transitions) and then trying to extend these traces so they also fit the data perspective. Except for invisible transitions, the data perspective can not add extra traces to the model. Extending the model with this perspective can only restrict the set of possible traces through transition guards. Therefore, the set of possible traces in a DPN is a subset of the set of traces possible when ignoring the data perspective except for invisible transitions. The approach, therefore, calculates this subset by determining which of the traces possible in the control-flow perspective can be extended with the data perspective. The extension of a trace consists of finding value combinations for all write operations that satisfy all the transition guards necessary to make the trace valid.

5.5.2 Simulation Configuration

This section will briefly explain all the settings that the user can configure for the simulation. Additionally, it is presented how these settings affect the previously described algorithms. The configuration of the simulation is stored in the 'SimulationConfig' and 'TransitionConfig' classes. A 'TransitionConfig' object is used for each transition in the model to store the individual configuration options. A 'SimulationConfig' object is used to store all the configurations options not specifically related to an individual transition or variable. In the following, the configuration options for individual transitions and the general configuration options are explained.

Transition Configuration

The first configuration option available for individual transitions is the activity name. This option is used to name the events generated when the transition is fired. Additionally, a weight can be specified for each transition. If the random trace generation approach is used, the weights of the enabled transitions are used to perform a weighted random choice that determines which transition is fired. The user can also specify whether a transition should be invisible. An invisible transition will create no event in the event log when it is fired. However, it can be specified that invisible transitions should be included in the event log. The remaining configuration options for individual transitions allow the user to control how the firing of transitions influences the simulation clock. However, these settings do not have to be entered for every individual transition. The user has the option to provide a general configuration. The general configuration is used for every transition that has no individual values. There are two groups of settings for this configuration. First, the lead time of a transition can be specified through the minimum value, maximum value, mean value, and standard deviation. This is the amount of time that the execution of the activity that is represented by the transition should take. Additionally, a time delay can be defined for each transition. The time delay influences how much time passes before a transition fires after the last transition has been fired.

General Configuration

In this section, the general simulation configuration options are explained. A full list of the available settings can be seen in the 'SimulationConfig' class in figure 5.5. The most influential option of the general configuration options is the simulation mode. The simulation mode determines which of the approaches presented in section 5.5.1 is used for the trace generation. There are three modes to choose from. The 'Random Trace Generation' mode consists of randomly firing enabled transitions until a final marking or a deadlock is reached. The 'Random Exploration' mode tries to find as many unique traces as possible while using the same approach as the random trace generation. The third simulation mode uses the second approach described in section 5.5.1, which consists of performing a full exploration of the DPN.

When using the first simulation mode, the number of traces that are generated can be set to a fixed value. If one of the two exploration modes is used, the maximum number of traces can be set. Additionally, when using one of the exploration modes, it can be specified that partial traces should also be included. For example, if the trace 'A, B, C' is generated, the partial traces would be 'A, B' and 'A'. There are two additional settings when using the simulation mode that tries to generate all traces. Both of the settings aim to reduce the number of possible value combinations to increase the efficiency of the trace generation. The first setting limits the value of variables to only include values necessary to satisfy transition guards present in the model. The second setting allows the simulation to merge different intervals for numeric variables that a single interval can represent. The minimum and maximum trace lengths can be specified in all three modes. Additionally, the number of trace duplicates can be configured. It can be defined if the values of variables should be included while determining if two traces are duplicates or if only the sequence of transitions should be considered. Additionally, it can be specified if only transitions that are not invisible should be considered. Two settings allow the user to control the number of loop iterations that are allowed in a trace. The first setting controls how often the same marking is allowed to appear in a trace. The second option controls how often the same transition is allowed to be fired in a trace. The user can also specify that only traces that reach a valid final marking are added to the generated event log.

Another aspect of the generated event logs that can be configured is the timestamp of events. The user can provide a timestamp anchor that is used as a starting value for the simulation clock. The user can specify if the starting time should be the same for all traces or if the timestamp should continuously increase with every trace. This setting, therefore, controls whether the traces run in parallel or whether they follow each other. Additionally, the time between the transition can be specified. As mentioned in the previous section, a general configuration for the time aspect of the transitions can be provided.

Since many decisions are made during the simulation based on pseudo-random number generation, the last setting allows the user to provide a seed for the used RNG.

This allows the creation of reproducible results, which is beneficial for scientific research.

5.6 Algorithmic Approach

This section covers the algorithmic approaches developed for the concept. First, the algorithmic approach used in the model representation to evaluate transition guards is explained in section 5.6.1. Subsequently, the simulation approach used to generate random traces is described in section 5.6.2. Additionally, section 5.6.3 presents the approach used to generate realistic values during the simulation. Lastly, section 5.6.4 explains the algorithmic approach developed to generate all traces possible in a DPN.

5.6.1 Evaluating Transition Guards

In order to evaluate whether a transition is enabled an approach had to be developed to evaluate the guards of transitions. This section describes the algorithmic approach developed to perform this evaluation. Algorithm 1 shows a pseudocode implementation of this approach.

The first step of the approach consists of preparing the guard string for evaluation. This preparation is used to avoid problems during the evaluation and consists of three operations. First, all spaces in the string are removed. The spaces are only included in the string to make it easier to read for a human and serve no purpose when a program processes the string. Subsequently, all negative values, such as '-1', present in the guard are replaced by temporary variables. This is done to prevent confusion between '-' characters that denote that a number is negative and '-' characters that are used as an arithmetic operator. The problem originates from the circumstance that transitions guards are stored as strings. In the developed algorithmic approach, there is the need to check whether there are specific operators in a part of a guard string. This is done by simply checking if the characters used to represent these operators are present in the string. For example, the line '`if "-" in guard_string then do`' is used to check if the operator '-' is present in the guard string. If the negative numbers are present in the guard string, this condition can return undesired results. Therefore, all negative numbers in the guard string are replaced by temporary variables.

The last operation that is done during the preparation of the guard string concerns variables with missing values. Depending on the model used for generating event logs, it is possible that transitions read from variables that do not have been written to yet and therefore have no value. This problem had to be addressed when developing the algorithmic approach for MILP-Solver. The developed algorithmic approach deals with this problem by replacing parts of the guard string affected by missing values with boolean values. Consider the following exemplary guard string where *vararariable1* has no current value.

```
'(variable1 == 1) || (variable2 > 5)'
```

In this example, the guard string can be satisfied by *variable2* having a value greater than five. Therefore, the missing value should not stop the transition from being

Algorithm 1: Algorithmic approach of the MILP-Solver

```
1: function prepare_and_evaluate_guard(guard_string, variables)
2:     guard_string ← guard_string.replace(‘ ’, ‘’)                ▷ Remove spaces

3:     guard_string ← replace_negative_values(guard_string, variables)
             ▷ replacing values such as ‘-1’ with temporary variables to avoid
               confusion with the operator ‘-’

4:     guard_string ← replace_missing_values(guard_string, variables)
             ▷ Replacing parts of the guard string that contain variables that do
               not have a current value. Example where var1 has no current value:
               ‘(var1 == 1) || (var2 == 2)’ => ‘False || (var2 == 2)’

5:     return evaluate_guard(guard_string, variables)

6: function evaluate_guard(guard_string, variables)
7:     if ‘(’ ∈ guard_string or ‘)’ ∈ guard_string then
8:         indices ← get_outer_brackets_indices(guard_string)
9:         parts, operator ← split_outer_brackets(guard_string, indices)
                 ▷ Example for line 8/9:
                       guard_string = ‘(var1 == 1) && (var2 == (var3 + 2))’
                       parts = [‘var1 == 1’, ‘var2 == (var3 + 2)’], operator = ‘&&’

10:        if operator == ‘||’ then return evaluate_guard(parts[0], variables) or
                                            evaluate_guard(parts[1], variables)
11:        else if operator == ‘&&’ then
12:            ...                                  ▷ Analogous for all other operators

13:    else
14:        for each operator ∈ operators do              ▷ Check all allowed operators
15:            if operator in guard_string then
16:                parts ← guard_string.split(operator)
17:                if operator == ‘||’ then
18:                    return evaluate_guard(parts[0], variables) or
                              evaluate_guard(parts[1], variables)
19:                else if operator == ‘&&’ then
20:                    ...                          ▷ Analogous for all other operators

21:        for each variable ∈ variables do
22:            if guard_string == variable.name then
23:                return variable.value
24:        return get_non_variable_value(guard_string)
```

enabled. If *variable1* would have no current value, the preprocessing of the guard that is done in the developed algorithmic approach would return the following guard string.

```
'False || (variable2 > 5)'
```

For all operators except the '!=' operator, the boolean value 'False' is used to replace the affected substring. For the '!=' operator, the affected substring is replaced by the boolean value 'True' since a variable having no value can be interpreted as being not equal to any specific value. If the missing value affects arithmetic operators, the affected part of the guard can be larger than just a single comparison seen in the previous example. Consider the following example where *variable2* is missing a value.

```
'(variable1 == (variable2 + 5)) || (variable2 > 5)'
```

In this example, the entire substring to the left of the '||' operator is affected by the missing value. Therefore, the preprocessing would return the following guard string if *variable2* has no value:

```
'False || (variable2 > 5)'
```

Once the preparation of the guard string has been completed, the actual evaluation of the guard can be performed. As shown in the examples presented in the previous paragraphs, transition guards can contain nested logical expressions. The algorithmic approach developed for the MILP-Solver consists of recursively evaluating the nested comparisons. Figure 5.6 illustrated the principle of this approach with an exemplary execution of the algorithm.

The recursive part of the algorithmic approach starts by determining if there are still nested expressions present in the guard string that must be evaluated independently. This is done by checking if there are still bracket characters (i.e., '(' or ')') present in the string. If this is the case, the outermost brackets and the operator connecting them are determined. The string then gets split by the identified brackets, and the algorithm is recursively called for each part. To better understand this concept, consider the following exemplary guard string.

```
'(variable1 == 1) && (variable2 > (variable3 + 5))'
```

In this example, the outermost brackets are coloured blue. The developed approach would split the string into the following two substrings.

```
'variable1 == 1'
'variable2 > (variable3 + 5)'
```

Additionally, the '&&' operator would be identified. Subsequently, the algorithm would be called recursively for each part, and the results would be connected with the identified operator.

If there are no nested logical expressions left in the guard string, the recursive part of the developed algorithmic approach first determines if there is still an operator left in the string. If this is the case, the string gets split by that operator, and the

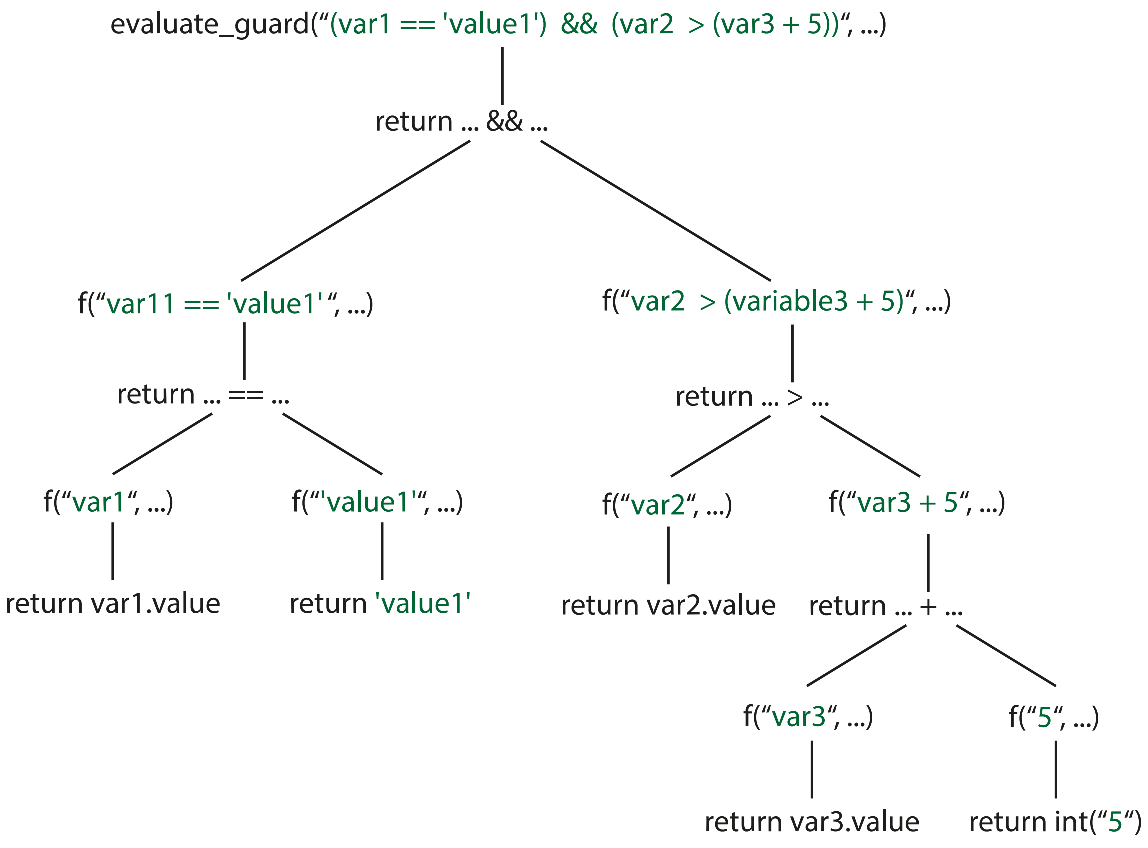


Figure 5.6: Example that illustrates how the algorithmic approach for the MILP-Solver works

Due to size constraints the function 'f' resembles the function 'evaluate_guard' and the function is missing the variables argument.

algorithm is again recursively called for the two parts. If there is no operator left at this, the recursion anchor has been reached. In this case, the algorithm checks if the string is the name of a variable. If this is the case, the current value of the variable is returned. If this is not the case, it means that there is only a value left that is compared to a variable. Therefore, the value is parsed and returned. As soon as all recursion calls have been collected, a final boolean value is available that shows if the transition guard is satisfied.

5.6.2 Generating Random Traces

This section describes the algorithmic approach developed to generate random traces through token-based simulations. Algorithm 2 shows a pseudocode implementation of this approach. The basic idea for this approach consists of performing simulations runs in which random enabled transitions are fired until a deadlock or final marking is reached.

The main part of this algorithmic approach consists of a loop that runs until the desired number of traces have been generated. A token-based simulation run is conducted in each loop iteration to generate a trace. Each simulation run consists of a loop that runs until a final marking is reached. In each loop iteration, the transitions enabled in regard to the current state of the model are determined. If no transitions

Algorithm 2: Concept for the algorithmic approach of the random trace generation through token-based simulation

```
1:  function random_trace_generation(model, config)
2:      $event\_log \leftarrow EventLog()$
3:      $current\_time \leftarrow config.timestamp\_anchor$
4:      while traces.length() < config.nr_of_traces do
5:          $initial\_time \leftarrow current\_time$
6:          $trace, reached\_final\_marking, current\_time \leftarrow generate\_single\_trace($
                                                   $model, config, current\_time)$
7:          if check_if_trace_conforms_to_config(trace, config, traces) then
8:              $event\_log.traces.append(trace)$
9:          else
10:             $current\_time \leftarrow initial\_time$       ▷ Resetting time progress since the
                                                              trace is disregarded
11:         $model.reset(config)$
12:         if config.fixed_timestamp_anchor then
13:             $current\_time \leftarrow config.timestamp\_anchor$
14:     return $event\_log$

15: function generate_single_trace(model, config, time)
    ▷ Returns a trace, a boolean value denoting if a final marking has been reached,
      and the simulation time
16:     $trace \leftarrow Trace()$
17:     while not model.final_marking_reached() do
18:         $enabled\_transitions \leftarrow model.get\_enabled\_transitions(config)$
19:         if enabled_transitions.empty() then
20:             $trace.add\_trace\_variables(model)$      ▷ Adding current values of trace
                                                           variables to the trace object
21:             return $trace, False, current\_time$     ▷ No transitions enabled or final
                                                           marking reached => Deadlock
22:         else
23:             $transition \leftarrow choose\_random\_transition(enabled\_transitions, config)$
24:             $model.fire\_transition(transition, config)$
25:             $current\_time \leftarrow foward\_time(transition, config, current\_time)$
26:             if not transition.write_variables.empty() then
27:                 $generate\_variable\_values(transition, model)$
28:             $trace.add\_event(transition, model)$
29:     $trace.add\_trace\_variables(model)$
30:     return $trace, True, current\_time$    ▷ True => Reached valid Final Marking

31: function check_if_trace_conforms_to_config(trace, config, traces)
32:     if trace.length() > config.max_trace_length then
33:         return $False$
34:     if trace.length() < config.min_trace_length then
35:         ...                                           ▷ Analogous for all other settings
36:     return $True$
```

are enabled, it means a deadlock has been reached. In this case, the loop stops, and the trace returns. If there are enabled transitions, a transition gets chosen through a random decision influenced by the weights of the transitions in question. The chosen transition then gets fired to change the state of the model. Additionally, the simulation clock is forwarded in accordance with the information the user supplied for that transition. If the fired transition writes to any variables, the approach generates values for these write operations using the semantic information supplied by the user. The algorithmic approach for generating realistic variable values is covered in section 5.6.3. After the transition has been fired and the variable values have been generated, an event is added to the trace. This event represents the transition that has been fired and the state of the model's variables when the transition was fired. The previously described behaviour of firing transitions is repeated until the firing of a transition results in a final marking or a deadlock.

When a simulation run ends, the algorithm determines whether the generated trace conforms to the restrictions that have been set by the user. For example, it gets disregarded if the trace exceeds the maximum allowed trace length. If the trace conforms to the restrictions, it gets saved in an event log object that gets written to an XES file once the desired number of traces have been generated or the simulation is aborted. If adding this trace to the event log results in the desired number of traces being reached, the outer loop of the algorithm gets stopped. In this case, the generated traces get written to an XES file in the specified output directory. If the desired amount of traces has not been reached yet, a new simulation run is started. However, before a new simulation run can be started, the model needs to be reset to its initial state. Resetting the model includes changing the current marking of the model to the initial marking and resetting each variable to remove the last value written to it in the previous simulation run. Additionally, the simulation clock is reset when the fixed timestamp anchor option is enabled.

5.6.3 Generating Variable Values

This section describes the algorithmic approach to generate realistic values during the token-based simulation algorithm described in section 5.6.2. Algorithm 3 shows a pseudocode implementation of this approach.

The algorithmic approach relies on the semantic information provided by the user. It starts by determining if any of the variable's dependencies can be applied since the logical expression can be satisfied. Since the syntax of the logical expression used to define a dependency is identical to the syntax of transition guards, the MILP-Solver can evaluate the expression. Suppose one or more *equal dependencies* (dependencies that force a variable to be equal to a specific value) can be applied. In that case, a random applicable equal dependency is picked, and the value that this dependency specifies is assigned to the variable. If no equal dependency can be applied, the *not equal dependencies* (dependencies that restrict the valid variables of a variable) that can be applied are considered to determine what values are invalid values for the variable. Subsequently, it is checked whether any *interval dependencies* (dependencies that restrict numeric or date variables to a specific interval) can be applied. If this is the case, a random applicable interval dependency is chosen, and a value that lies in that interval is generated. However, it is assured that the generated value is not in the list of invalid variables.

Algorithm 3: Concept for the algorithmic approach of generating realistic variable values needed during the token-based simulation

```
 1: function generate_variable_values(transition, model)
 2:     for each variable ∈ transition.write_variables do
 3:         $variable.value \leftarrow generate\_variable\_value(variable, model)$

 4: function generate_variable_value(variable, model)
 5:     if variable.has_dependencies then
 6:         $valid\_dependencies \leftarrow empty\_list$
 7:         for each dependeny ∈ variable.semantic_information.dependencies do
 8:             if MilpSolver.prepare_and_evaluate_guard(dependency.guard) then
 9:                 $valid\_dependencies.append(dependency)$
                                 ▷ Evaluating the logical expressions (guards) of the
                                   dependencies to get the applicable ones. Since the logical
                                   expression use the same syntax as the guards of
                                   transitions the MILP-Solver can be used.

10:         if not valid_dependencies.emty() then
11:             $equal\_constraints, interval\_constraints, not\_equal\_constraints \leftarrow$
                                        $sort\_constraints(valid\_dependencies)$
12:             if not equal_constraints.empty() then
13:                 $constraint \leftarrow choose\_random\_constraint(interval\_constraints)$
14:                 return $constraint.value$
15:             $invalid\_values \leftarrow empty\_list$
16:             for each constraint ∈ not_equal_constraints do
17:                 $invalid\_values.append(constraint.value)$
18:             if not interval_constraints.empty() then
19:                 $constraint \leftarrow choose\_random\_constraint(interval\_constraints)$
20:                 return $generate\_interval\_value(constraint, invalid\_values)$

21:     if variable.semantic_information.use_values then
22:         return $choose\_random\_value(variable.semantic\_information.values,$
                                    $invalid\_values)$
23:     else if variable.semantic_information.use_intervals then
24:         $interval \leftarrow choose\_random\_interval($
                                    $variable.semantic\_information.intervals)$
25:         return $generate\_interval\_value(interval, invalid\_values)$
26:     else
27:         return $variable.semantic\_information.distribution.next\_value($
                                                        $invalid\_values)$
```

If there are no equal or interval dependencies that can be applied, the value for the variable is generated from the other semantic information provided by the user. This can either be the provided values, the provided intervals, or the provided distribution. Which information is used in this case can be individually specified by the user for each variable. If it is specified to use the provided values, a random value from the provided values will be picked. However, it is assured that the invalid values stemming from not equal dependencies are not considered. If the user has specified the use of intervals for the value generation, a random interval will be picked from the set of provided intervals. Subsequently, a random value from inside this interval will be generated. However, it is again assured that the generated value is not invalid. Lastly, the user can specify the parameters of a distribution that are used to generate values for the variable. A random value will be drawn from that distribution if this is the case. However, it is again assured that the generated value is not invalid.

5.6.4 Generating All Possible Traces

This section describes the algorithmic approach developed to generate all traces possible in a DPN and, therefore, fully explore the DPN. Algorithm 4 shows a pseudocode implementation of this approach.

As previously described in section 5.5.1 the approach for generating all traces consists of first exploring the DPN without considering the data perspective and then extending the discovered traces with data. Therefore, the first step of the algorithmic approach consists of calculating all traces that are possible when ignoring the guards of transitions. In the pseudocode implementation shown in algorithm 4 the 'recursive_trace_generation' function is used for this step. The step is done by performing a recursive depth-first search through the graph of the model. The search starts by calling the recursive function with the model in its initial state and the configuration supplied by the user.

In each iteration of the recursive function, it is first determined if the provided model is in a final marking. If this is the case, one of the recursion anchors is reached, and the trace gets added to a list of traces that reached a final marking. However, if the trace does not conform to the user's restrictions, it gets disregarded (For example, if the trace does not reach the minimum trace length required). If the model is not in a final state, it is first checked whether the trace length is equal to the allowed maximum trace length or if the maximum number of loop iterations has been exceeded. If this is the case, the second recursion anchor is reached, and the trace gets added to a list of traces that did not reach a final marking. If the trace has not exceeded these restrictions, the transitions enabled with the current marking of the model are determined. If no transitions are enabled, the third and final recursion anchor is reached, and the trace is added to the list of traces that did not reach a final marking. If transitions are enabled, a copy of the model and the current trace is created for each enabled transition. Subsequently, each transition is fired in the corresponding copy of the model. The algorithm then calls itself recursively for every copy of the model that was created. This behaviour ensures that all sequences of activities possible in the model are found since every possible path is searched.

As soon as all recursive function calls have been collected, the algorithm has calculated all traces possible in the DPN when ignoring transition guards. The algorithm then iterates over all traces found and attempts to extend them with variable values so that they conform to both perspectives of the DPN. However, if the user has configured the simulation to only consider traces that reach a final marking, only traces conforming to this setting are considered.

Algorithm 4: Concept for the algorithmic approach of the simulation mode that generates all traces possible in a DPN

```
1:  function generate_all_traces(model, config)
2:      valid_traces ← empty_list
3:      deadlock_traces ← empty_list
4:      current_trace ← Trace()
5:      recursive_trace_generation(model, valid_traces, deadlock_traces,
                                  trace_copy, current_trace, config)
6:      event_log ← EventLog()
7:      for each trace ∈ valid_traces do
8:          var_cominations ← calculate_var_combinations(trace)
9:          for each combination ∈ var_combinations do
10:             if combination_fits(combination, trace) then
11:                 event_log.traces.append(generate_finished_trace(trace,
                                                             combination)
12:     if not config.only_valid_ending_trace then
13:         for each trace ∈ deadlock_traces do
14:             ...                                  ▷ Identical to lines 8-11
15:     return event_log

16: function recursive_trace_generation(model, valid_traces, deadlock_traces,
                                        current_trace, config)
17:     if model.is_in_final_state() then
18:         if current_trace.length() >= config.min_trace_length then
19:             valid_traces.append(current_trace)
20:     else
21:         if current_trace.length() >= config.max_trace_length then
22:             deadlock_traces.append(current_trace)
23:         else if max_loop_iterations_exceeded(trace) then
24:             trace.pop()                          ▷ Remove last event of trace
25:             deadlock_traces.append(current_trace)
26:         else
27:             enabled_transitions ← model.get_enabled_transitions(
                                                 consider_guards = False)
28:             if enabled_transitions.empty() then
29:                 deadlock_traces.append(current_trace)
30:             else
31:                 for each transition ∈ enabled_transitions do
32:                     model_copy ← copy(model)
33:                     trace_copy ← copy(current_trace)
34:                     model_copy.fire_transition(transition)
35:                     trace_copy.append(transition)
36:                     recursive_trace_generation(model_copy, valid_traces,
                                    deadlock_traces, trace_copy, config)
```

When trying to extend a trace with variable values, the trace is first analysed to determine all the write operations that occur in the trace. Subsequently, all possible combinations of variable values possible for this set of write operations are generated. This is the step of the approach where the previously mentioned intervals for numeric variables are imported. The numeric variable types, such as integer variables, have a practically infinite number of possible values. Therefore, generating all possible value combinations with these types of variables is not feasible. Therefore, all continuous variables are transformed into discrete variables by grouping the possible values into intervals. However, it is ensured that all intervals necessary to satisfy transition guards in the model are included in the created intervals. After the possible combinations have been generated, the algorithm iterated over these combinations. It is checked whether any combination can satisfy all transition guards present in the trace. If a working combination is found, the trace gets expanded with the working variable values, and the trace is added to the event log object so it can be written to an XES file later. If none of the calculated variable combinations can satisfy all transition guards, the trace is only possible when ignoring the data perspective, and it can be disregarded.

Once the algorithm has iterated over all traces found during the first step, the event log object contains all traces possible in the DPN. The only step remaining is to write the generated event log to an XES file.

5.7 Implementation

This section describes the implementation of the previously described concept. Section 5.7.1 describes the methodology used to implement the developed concept and section 5.7.2 describes the implemented software.

5.7.1 Implementation Methodology

The first step of implementing the concept consisted of considering whether to implement a stand-alone software or to extend pre-existing software, such as the ProM framework[2]. It was considered to implement the approach as a plug-in for the ProM framework or as an extension to the PM4Py framework[3]. The decision was made to implement independent software. This decision was primarily influenced by the limited time available for the implementation. Since the implementation was done as part of a master thesis, there was a strict deadline by which the entire research had to be concluded. Therefore, the implementation had to be finished within a limited time frame. Since it would take a considerable amount of time to get to know the frameworks, the consideration for the two frameworks was dropped.

The second step consisted of choosing a programming language to implement the concept. This decision was again influenced by the limited time available for the implementation. Therefore, the decision was made to only consider programming languages that the author was already familiar with. The considered languages were *Java*, *Python*, and *C#*. The language that was finally chosen for the implementation was Python. Python is the most known for its implementation speed of these programming languages. However, there is almost no research comparing the amount of development time needed when using different programming languages. The only scientific work that was found regarding this topic was a study conducted by Lutz

2. http://www.promtools.org/doku.php
3. https://pm4py.fit.fraunhofer.de/

Prechelt showed that a program written in Python needs significantly less development time than a program written in Java [Prec00]. Therefore, this is an opinion based on the author's personal experiences. Additionally, Python offers a broad selection of GUI frameworks, of which many offer cross-platform compatibility. This is advantageous since a goal for implementing the software was to develop a Linux and Windows version.

The third step of the implementation consisted of choosing libraries and frameworks that were used for the implementation. Care was taken to ensure that the libraries work on Linux and Windows. This ensured that a single version of the software could be developed that works on both operating systems, which ensured that the implementation could be done in the available time. Additionally, it was ensured that the licences of the libraries allow the source code to be published. For example, the QT6 framework[4] was chosen for the GUI since it supports both Linux and Windows and is available under the *GNU General Public License[5] (GNU GLP).*

The fourth and final step of the implementation consisted of programming the software. The implementation was carried out very close to the concept. There were only some small deviations from the developed algorithmic approaches of the concept. However, these deviations did not change what these algorithms do. The only significant deviations made from the concept were in the form of expansions to the concept. These expansions were primarily based on feedback from researchers at the Chair Of Business Information Systems II at the University of Trier and the *German Research Center for Artificial Intelligence (DFKI)*[6] that might use the developed software for their research. Additionally, some expansions were based on the further analysis of real-life event logs. These expansions were made to allow a better imitation of real event logs. For example, a setting was added that allows adding metadata to the created event log.

5.7.2 DALG: The Data Aware Event Log Generator

The result of the implementation is a software called *The Data Aware Event Log Generator (DALG).* DALG is available under the GNU General Public Licence version 3.0[7] and is, therefore, open-source. The software and the source code are available through a GitHub repository[8]. DALG is written in Python and is available for Windows and Linux. However, since the QT6 framework[9] is used for the GUI, the only operating systems currently officially supported by DALG are Windows 10, Windows 11 and Ubuntu 20.04.

Concept deviations

The implementation of the software was carried out very close to the concept. Therefore, this chapter will only describe the deviations made from the concept since the concept was already described in detail. There were only small deviations from the algorithmic approaches presented in the previous chapter. However, these deviations

4. https://www.qt.io/product/qt6
5. https://www.gnu.org/licenses/gpl-3.0.en.html
6. https://www.dfki.de/en/web
7. https://www.gnu.org/licenses/gpl-3.0.en.html
8. https://github.com/DavidJilg/DALG
9. https://www.qt.io/product/qt6

did not change what these algorithms do. The only significant deviations from the concept were made in the form of expansions. In the following, these expansions of the concept are described.

The biggest deviation of the concept consists of an expansion to the 'Random Exploration' simulation mode that was previously presented. When using this mode, the user has the option to have a trace estimation performed before the actual simulation is conducted. The trace estimation has the goal of giving the user information about the maximum possible number of traces. This can be used to determine the progress of the simulation and therefore makes the software easier to use. The trace estimation determines all traces of a model that are possible when only considering the control-flow perspective. This is done by repurposing the algorithm developed for the 'All Traces' simulation mode, which was described in the previous chapter.

Another deviation that was made from the concept concerns the writing operations of transitions. In the implementation, it is possible to have variables change in regards to themself if they are written to. For example, there is an option to have a variable increase by a value in a specified value range every time the variable is written to.

Another eviation from the concept allows variables to have initial values. That means that variables can have values before they have been written to. There are two options provided to the user. The user can either provide an initial value for the variable or generate it with the provided semantic information.

There was also a deviation made regarding the configuration options for the intervals of numeric variables. In order to make it easier for the user to provide intervals, there is the option to automatically include inverse intervals of the provided intervals.

Another deviation was made in regard to the transition configuration. Some event logs have the characteristic that not all events contain all variables. Therefore, a setting was added to the transition configuration that allows the user to control which variables are present in the corresponding event. Additionally, most event logs contain some metadata. Therefore, a setting was added that allows the user to add metadata to the generated event log.

There were two additional setting options added for controlling the simulation clock that were not originally part of the concept. The options allow to control the precision of the timestamps (second or microsecond precision) and whether a transition forwards the simulation clock.

The last deviation made from the concept consists of the option to have variables with fixed values. The user has the option to set a variable as a fixed variable. In this case, the value of the variable is fixed once the variable has been written to once.

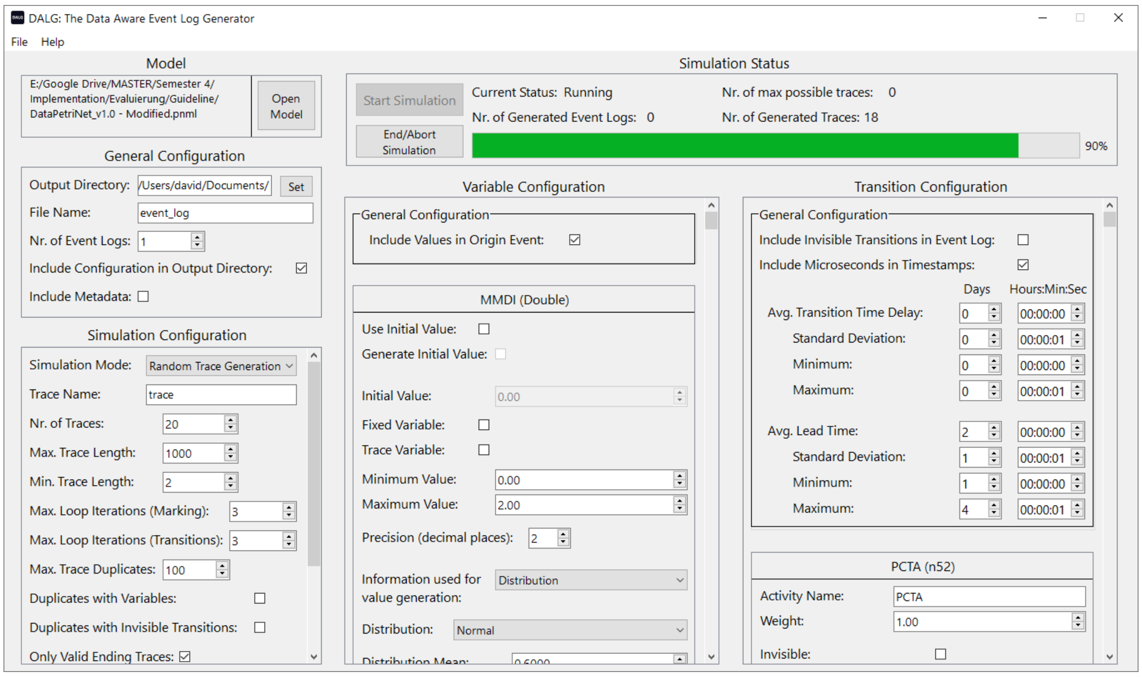
DALG Overview

As previously mentioned, DALG is available for both Windows and Linux-based operating systems. However, the QT6 framework used for the GUI of the software officially only supports a limited number of Linux distributions and only the tenth and eleventh iteration of Windows[10]. Therefore, the decision was made to only

10. https://doc.qt.io/qt-6/supported-platforms.html

officially support and test DALG on Windows 10, Windows 11, and Ubuntu 20.04. However, DALG can also be controlled through a command-line interface. Therefore, it should be possible to use this software on most Windows and Linux machines.

Even though DALG can be controlled through the command line, the preferred way to operate it is through the provided GUI. Only then can the user access all of the 'comfort' features that make it easier to use the software. The main window of the windows version of DALG can be seen in figure 5.7.



Model

E:/Google Drive/MASTER/Semester 4/Implementation/Evaluierung/Guideline/DataPetriNet_v1.0 - Modified.pnml

Simulation Status

Current Status: Running
Nr. of max possible traces: 0
Nr. of Generated Event Logs: 0
Nr. of Generated Traces: 18
90%

General Configuration

Output Directory: /Users/david/Documents/
File Name: event_log
Nr. of Event Logs: 1
Include Configuration in Output Directory: ☑
Include Metadata: ☐

Simulation Configuration

Simulation Mode: Random Trace Generation
Trace Name: trace
Nr. of Traces: 20
Max. Trace Length: 1000
Min. Trace Length: 2
Max. Loop Iterations (Marking): 3
Max. Loop Iterations (Transitions): 3
Max. Trace Duplicates: 100
Duplicates with Variables: ☐
Duplicates with Invisible Transitions: ☐
Only Valid Ending Traces: ☑

Variable Configuration

General Configuration
Include Values in Origin Event: ☑

MMDI (Double)

Use Initial Value: ☐
Generate Initial Value: ☐
Initial Value: 0.00
Fixed Variable: ☐
Trace Variable: ☐
Minimum Value: 0.00
Maximum Value: 2.00
Precision (decimal places): 2
Information used for value generation: Distribution
Distribution: Normal
Distribution Mean: 0.6000

Transition Configuration

General Configuration
Include Invisible Transitions in Event Log: ☐
Include Microseconds in Timestamps: ☑

	Days	Hours:Min:Sec
Avg. Transition Time Delay:	0	00:00:00
Standard Deviation:	0	00:00:01
Minimum:	0	00:00:00
Maximum:	0	00:00:01
Avg. Lead Time:	2	00:00:00
Standard Deviation:	1	00:00:01
Minimum:	1	00:00:00
Maximum:	4	00:00:01

PCTA (n52)

Activity Name: PCTA
Weight: 1.00
Invisible: ☐

Figure 5.7: Main window of the graphical user interface provided by DALG (Windows 10 Version)

DALG allows the user to load any Petri Net modelled in the Petri Net Markup Language. The GUI allows the user to easily configure all the settings for the simulation and provide all the necessary semantic information about the variables. As explained in the concept description, the program allows the user to save the configuration to a JSON file. The user can also load a previously saved configuration file. Lastly, the GUI allows the user to start, monitor, and stop the simulation. More information about the features and operation of the software can be read in DALGs user manual. The user manual is included in DALGs GitHub repository[11].

11. https://github.com/DavidJilg/DALG/blob/main/src/documentation/manual.pdf

6. Evaluation

This chapter describes the evaluation of the approach developed in this thesis. The chapter is structured as follows. First, section 6.1 gives an overview of the methodology used to evaluate the approach. Subsequently, the different evaluation steps described in the methodology section are presented in sections 6.2 through 6.4. Additionally, section 6.5 discusses the manual effort required to generate synthetic data with DALG. Lastly, the results of the evaluation are discussed in section 6.6.

6.1 Overview

To evaluate the developed approach, the implementation of the approach described in chapter 5 was used. It was evaluated whether the implementation can satisfy the requirements specified for the development of the approach in section 3.1.4.

It was possible to evaluate most of the requirements in a single evaluation step that consisted of evaluating if the implementation could satisfy the eighth requirement (**R8**). The eighth requirement requires the approach to generate traces that conform to the model they are generated from. The implementation had to be used to generate traces from a DPN, to evaluate whether it could satisfy this requirement. These traces could then be compared to the model to determine if there are deviations from the behaviour described in it. During the generation of traces, the first eight requirements (**R1** - **R8**) and the twelfth requirement (**R12**) could be evaluated. For example, the sixth requirement(**R6**) requires the approach to be able to deal with specific operators in transition guards. Therefore, this requirement can be evaluated using a model containing these operators during the generation of the traces for evaluating the eighth requirement. If the generated traces conform to the model, it means that the approach can properly evaluate transition guards with the required operators. The same principle was applied for all the requirements except the ninth, tenth, and eleventh requirement (**R9**, **R10**, **R11**). It was more feasible to evaluate these requirements separately since their evaluation is significantly more complex and can not be inferred from the fact that the approach can generate traces that conform to a model. Therefore, the evaluation was done in three steps.

First, all but the ninth, tenth, and eleventh requirements were evaluated while evaluating the conformance of the generated traces. This evaluation step is described in section 6.2. Subsequently, it was evaluated whether the implementation can generate all traces possible in a DPN. Section, 6.3 describes this step. Lastly, the realism of the generated event logs was evaluated to determine if the approach can satisfy the eleventh requirement. This step is covered in section 6.4.

6.2 Conformance of the Generated Traces

To check the conformance of traces generated with the developed approach, DALG was used to generate traces from specific process models. Subsequently, the traces were compared to the model they were generated to check if there were any deviations from the models. The process models used for this evaluation step are described in section 6.2.1. Subsequently, the methodology used to perform the conformance checking is described in section 6.2.2. Lastly, the results of this evaluation step are presented in section 6.2.3.

6.2.1 Process Models

The first model used for this step is the same model that was already used as a part of the requirement analysis (see section 3.1.3). Since multiple process models were used for the evaluation, each model is given a unique name to avoid confusion. The model used for the requirements analysis and this step of the evaluation is referred to as the *requirement model* throughout the remaining thesis.

As previously explained in section 3.1.3 the requirement model does not include all challenges identified during the requirement analysis. Therefore, it does not contain all challenges necessary to evaluate all of the requirements. An additional model with the missing challenges was created to check whether the developed approach can also generate conforming traces when these additional challenges are present. This model contains transitions guards that contain all arithmetic operators required by requirement **R7** ('+', '-', '/', and '*'). Additionally, it contains a variable of the type date, which is written to and used in a transition guard. The date variable is needed to evaluate the fourth requirement (**R4**). The model created to evaluate whether the approach can satisfy the requirements concerning the additional challenges will be referenced to as *additional challenges model* in the remaining thesis.

In addition to using the additional challenges model for the previously described purpose, it was also used for checking whether the approach can generate all traces possible in DPN. Therefore, the model was extended with a loop. Additionally, a path was added that is possible when only considering the control-flow perspective but not when considering the data perspective. Figure 6.1 shows an illustration of the final additional challenges model. As can be seen in the figure the transitions 't1', 't2', 't3', and 't4' contain the arithmetic operators required by the seventh requirement (**R7**) in the their transitions guards. Additionally, the transition 't5' contains a guard that can not be satisfied in any way, making the path 't0, t5' not possible when considering the data perspective. The figure also contains information about all possible paths in the model. This information will be relevant in section 6.3 where the evaluation concerning the generation of all paths will be described.

6.2.2 Methodology

DALG (version 1.0) was first used to generate 1000 random traces from the requirement model to determine if the approach can generate conforming traces. This model is very complex, and the number of possible unique traces is exponentially high. Therefore, the generated traces do not cover all possible traces. However, it was assured that all transitions appear at least once in the generated logs to ensure that the requirements regarding specific operators only present in some transitions

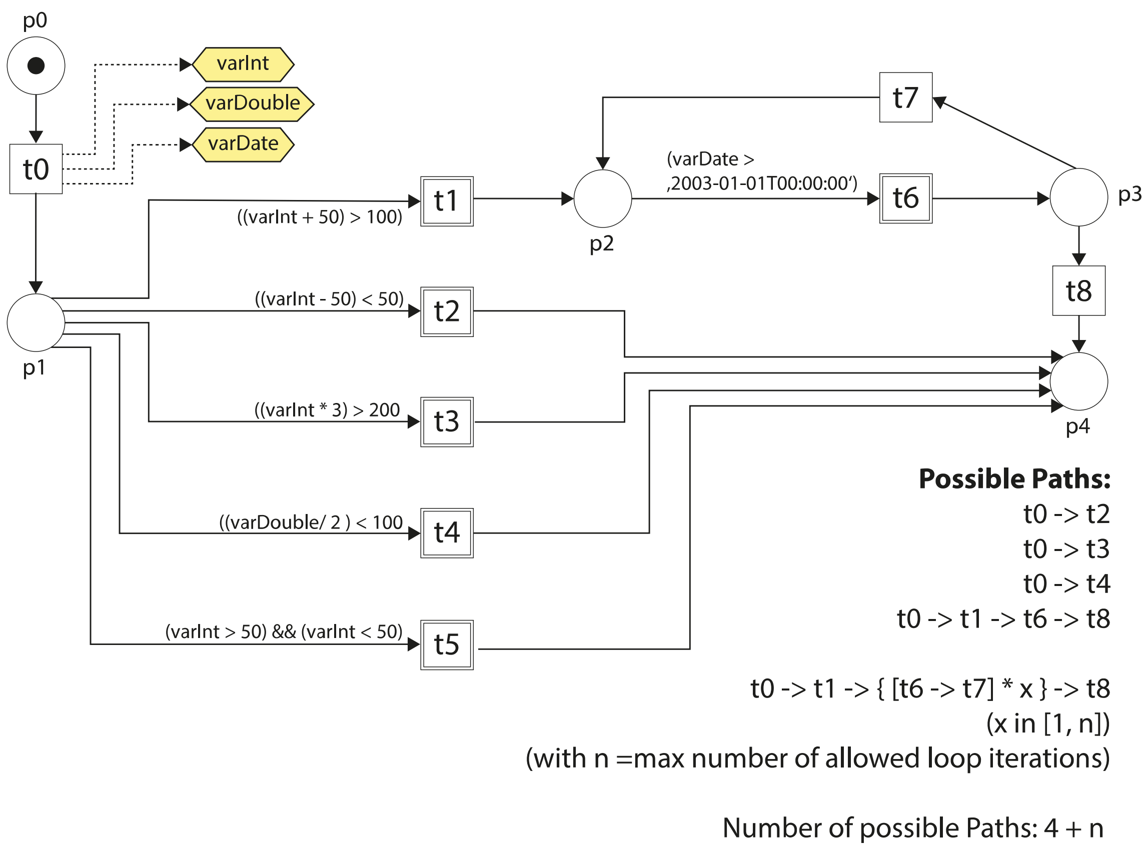


Figure 6.1: DPN created to evaluate whether the approach can deal with the challenges missing in the requirement model and generate all traces

could be evaluated. Due to the high number of traces necessary to cover all relevant parts of the model, the conformance checking of these traces could not be done manually. Therefore, the conformance checking of these traces was done with the previously introduced Multi-perspective Process Explorer (see section 2.3). The MPE is a plug-in for ProM that allows performing conformance checking on multi-perspective process models and event logs. The MPE can detect deviations between a model and an event log while considering the control-flow and the data perspective. Therefore, it could be used to determine if the generated event logs conform to the model they are generated from.

However, it could not be used to check the conformance of the traces generated from the additional challenges model. The MPE does not support the use of date variables in transition guards and also struggles with arithmetic operators. The additional challenges model was created to check the support of date variables and arithmetic operators. Therefore, the MPE could not check the conformance of traces generated from this model. However, the second model is relatively simple, with only a small number of possible paths. Therefore, the conformance of the traces could be checked by only generating nine unique traces, which fully covered all possible paths in the model that do not exceed five loop iterations. The nine traces were then examined manually to determine if they deviated from the model.

6.2.3 Results

No deviations were found during the examination of the traces generated from the requirement model using the MPE. Therefore, every transition fired in the traces was enabled regarding the current token placement of the model and the transition guard. Additionally, no write deviation occurred, meaning that all transitions wrote values to the correct variables, and no write operation was missing. Table 6.1 shows the results of running a conformance check on the traces using the MPE. The

Table 6.1: Results of checking the conformance of traces generated with DALG through the use of the MPE

Indicator	**Value**
Avg fitness	100%
# Correct writes	27.667
# Wrong Writes	0
# Missing Writes	0
# Correct Events	21.015
# Wrong Events	0
# Missing Events	0

evaluation results with the additional challenges model were identical to the results of the evaluation with the requirement model. There were no deviations from the model in the generated event log.

The evaluation of the traces generated from both models shows that DALG can generate traces that conform to the control-flow and data perspective of a DPN. This means that the developed approach can satisfy the **eighths requirement** (**R8**).

Other Requirements Necessary to Generate Conforming Traces

In order to generate conforming traces from both the requirement and the additional challenges model, DALG had to satisfy several other requirements. Therefore, it can be inferred that DALG can also satisfy the other requirements necessary for generating conforming traces from the chosen models. Both models used were modelled in the Petri Net Markup Language. Therefore, it can be concluded that DALG can satisfy the **first requirement** (**R1**). The requirement model contains invisible transitions. Therefore, DALG can satisfy the **second requirement** (**R2**). It can also be concluded that DALG can satisfy the **third** (**R3**) and **fourth requirement** (**R4**) since the additional challenges model contains variables of all required types. In order to generate conforming traces, DALG also had to be able to properly evaluate the transition guards of the models used in this step. Therefore, DALG can also satisfy the **fifth requirement** (**R5**). Additionally, this means that DALG can also satisfy the **sixth** (**R6**) and **seventh requirement** (**R7**) since the required operators are present in the transition guards of the requirement and additional challenge model. Lastly, the generation of conforming traces also confirms that DALG can satisfy the **twelth requirement** (**R12**) since the event logs were generated in the XES format and could successfully be imported into ProM.

6.3 Generating All Traces Possible in a DPN

This section describes the evaluation step in which it was examined if DALG can generate all traces possible in a Data Petri net. This section is structured as follows. First section 6.3.1 describes the methodology used for this evaluation step. Subsequently, 6.3.2 presents the results of this step.

6.3.1 Methodology

The **ninth requirement** (**R9**) requires the approach to be able to generate all traces possible in a model with no loop. The **tenth requirement** (**R10**) specifies that if the model has a loop, it should be possible to generate all traces with a maximum number of loop iterations. In this context, 'all traces' refers to all sequences of transitions possible in a DPN. It was first tried to generate all possible traces from the requirement model to evaluate the approach regarding these requirements. However, this model contains a loop, and the MPE can not calculate if all sequences of transitions possible with a maximum number of loop iterations are contained in the log. Therefore, a less complicated model was needed since the requirement model is too complex to manually evaluate whether an event log contains all possible traces. As described in the previous section, the additional challenges model was created in a way that also allows it to be used for evaluating whether the approach can generate all traces.

Therefore, DALG (version 1.0) was used with the additional challenge model to generate all traces. Multiple simulation runs with zero to five maximum loop iterations were performed. The traces generated during these simulation runs were examined manually to determine if all traces possible with the maximum number of loop iterations are present in the generated log.

6.3.2 Results

During the evaluation of the generated traces, it was confirmed that all traces possible in the model were present in the generated event logs. This also includes all traces possible with the different maximum loop iterations that were used. Additionally, the path in the model not possible when considering the data perspective was not present in any event log. This shows that DALG can satisfy the **ninth** (**R9**) and **tenth requirement** (**R10**) for models that have a relatively low complexity. However, DALG might not be able to generate all possible traces of a model with high complexity in a feasible time frame. As previously explained in chapter 5 DALG has two simulation modes targeted at exploring a DPN. The first mode consists of randomly generating traces and only keeping those that have not been found before. In a high-complexity model, this mode could take a very long time to find all traces. This is especially true if the model contains transition guards requiring specific variable values to be satisfied. Since the variable values in this mode are generated randomly, finding the right values could not be feasible in a reasonable time frame. The second simulation mode that allows exploring a DPN does not support all DPNs. It cannot properly deal with DPNs that have variables that get written to more than once in a single trace. The implementation of the all traces simulation mode is only able to generate combinations of variable values where each variable only appears once. Therefore, a variable that gets written to more than once is not supported. However, a more complex implementation of this approach may be able to solve this problem. Additionally, the all traces mode tries to find variable combinations that satisfy all transitions in a trace by brute-forcing valid combinations. Therefore, this mode might also not be able to generate all traces of a highly complex DPN in a reasonable time frame.

In summary, it can be said that the developed approach can find all possible paths of a DPN without a loop. Additionally, it was found that the approach can find all traces possible in a model with a loop that do not exceed a certain maximum number of loop iterations. Therefore, DALG can satisfy the **ninth** (**R9**) and **tenth** (**R10**) requirement. However, the approach may not be suited to find all traces of a DPN with a high complexity since the generation might exceed a feasible time frame.

6.4 Realism of the Generated Event Logs

This section describes the evaluation step in which it was determined if DALG can generate realistic synthetic data. This section is structured as follows. First, the methodology used for this step is described in section 6.4.1. Subsequently, the results of this evaluation step are described in sections 6.4.2 and 6.4.3.

6.4.1 Methodology

The eleventh requirement requires the generated event logs to be realistic. In this context, realistic is defined as event logs that are indistinguishable from real logs and considered meaningful from the perspective of domain experts. In order to evaluate whether event logs generated with the approach are indistinguishable from real event logs, they have to be compared to real-world event logs. Therefore, the first step of evaluating the realism consisted of acquiring a real event log and evaluating whether it is possible to use the approach to generate an event log that is indistinguishable from it. However, trying to imitate already existing event logs is not the primary use case of the developed approach. The approach is targeted at situations where no real event logs exist. Therefore, the realism of the generated event logs was also evaluated by generating event logs from a DPN and having a domain expert for that model judge whether the generated logs are meaningful on a semantic level. The following two paragraphs describe the methodology for these two steps.

Comparison to Real-world Event Logs

The first step to compare generated event logs to real event logs was to acquire a suitable real-world event log. The IEEE Task Force on Process Mining provides real-world event logs in the XES format[1]. The provided event logs were assessed to determine how suited they are for the evaluation. The event log that was chosen for the evaluation describes the daily living activities of several individuals[2]. This log was chosen since it does not require specific domain knowledge to understand it since it deals with everyday activities that everyone is familiar with. Additionally, it describes relatively simple behaviour, which allows for the manual comparison to generated event logs, and easier modelling of the behaviour.

After acquiring the real event log, a heuristic miner was used to discover the basic structure of the behaviour in the log. Subsequently, a Data Petri net was manually modelled to represent the behaviour in the event log as accurately as possible. This model will be referenced by the name *activities model* to avoid confusion with the other models used during the evaluation. After the activities model was created, the

1. https://www.tf-pm.org/resources/xes-standard/about-xes/event-logs
2. https://data.4tu.nl/articles/dataset/Activities_of_daily_living_of_several_individuals/12674873/1

event log was further analysed through a Python script to obtain the information necessary for configuring the simulation and the semantic information needed to generate event logs with the developed approach. For example, the timestamps of the events were analysed to determine the distribution of the transitions lead times.

The obtained information was used together with the activities model to generate a synthetic event log using DALG (version 1.0). Subsequently, the generated event log was compared to the real log to determine if differences could be detected. The comparison was made both manually and automatically. The sequences of activities and the overall structure of the logs were compared manually. Additionally, a Python script was used to compare the statistical properties of the event logs. For example, the average number of events per trace and the lead times of transitions were compared.

Evaluating Semantic Meaningfulness

The semantic meaningfulness and realism of the event logs that the approach can generate were evaluated with the help of a domain expert of the domain the used model stems from. The requirement model was used for this step. This model describes the diagnosis, treatment and aftercare of melanoma. Since this work focuses on the generation of synthetic procedural data in the medical field and specifically the generation of treatment cases, the model is well suited for the evaluation. The semantic information for configuring the implementation was acquired from the S3-Guideline for Diagnostics, Treatment and Aftercare of melanoma (original title: 'S3-Leitlinie Diagnostik, Therapie und Nachsorge des Melanoms')[3] published by the German Cancer Society[4] and the German Dermatological Society[5].

The configuration was used together with DALG version 1.1 to generate five traces from the requirement model. The generated traces are contained in appendix B. These traces were then presented to Dr med. Stephan Alexander Braun, who is an attending physician and the deputy head of the skin cancer center[6] at the Münster University Hospital[7]. Dr Braun was asked to identify aspects of the traces that make no sense from a semantic perspective. He was tasked with checking whether a real doctor would perform the same treatments with the information present in the treatment traces or if a real doctor would at least find the performed steps reasonable. Additionally, Dr Braun was asked to check whether the values generated for the variables of the model made sense.

6.4.2 Comparison to Real Event Logs

This section will present the results of trying to imitate a real-world event log by using DALG on a DPN representing the behaviour described in the log. This section is structured as follows. First, some basic information about the used real-world event log is presented. Subsequently, commonalities and differences between the real and the synthetic event log are presented. Lastly, a summary of comparing the real and the synthetic log is given.

3. https://www.leitlinienprogramm-onkologie.de/leitlinien/melanom/ (Website and guideline in german)
4. https://www.krebsgesellschaft.de/german-cancer-society.html
5. https://derma.de/ (Website in german)
6. https://web.ukm.de/index.php?id=hauttumorzentrum (Website in german)
7. https://www.ukm.de/university-hospital-muenster

Event Log Basics

The event log used is provided by the IEEE Task Force on Process Mining[8] and describes the daily living activities of several individuals. The dataset that contains the used event log can be accessed in the 4TU.ResearchData repository[9]. The dataset contains multiple event logs. For the evaluation, the event log with the filename 'activitylog_uci_detailed_labour.xes' that is contained in the Gnu Zipped Archive with the same name was used. The dataset was downloaded on the 1st of February 2022.

Each trace in the event log represents one individual's activities in one day. The

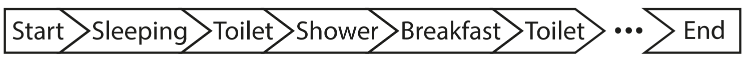


Figure 6.2: Simplified exemplary trace of the event log describing daily activities

traces consist of events that represent either the start or the end of an activity. A variable in the event defines if the event corresponds to the start or end of the activity. In addition to the events representing activities, each trace starts with an 'Start' activity and ends with an 'End' activity. Figure 6.2 shows a simplified exemplary trace that represents the basic structure of the traces present in the log.

Metadata

In addition to the traces, real event logs usually contain metadata. This is also the case for the real event log used for this evaluation step. There are significant differences between the metadata of the real event log and the synthetic log. It was, therefore, not possible to accurately recreate the metadata of the real event log in the synthetic even log. The real log contains specifications that define what variables and attributes each trace and each event in the log has. The trace specification was successfully recreated in the synthetic event log since the real event log uses the most common trace definition in which traces only have a name. However, it was impossible to recreate the event specification in the synthetic event log since DALG does not allow the user to customize this metadata. The metadata in the event log also specifies that a variable in the events is a classifier for the executed activity. It was not possible to recreate this metadata in the synthetic log. The real event log also defines which XES extensions are used. This metadata was mostly recreated since the DALG includes the extensions for the timestamps and names of events. However, the real event log also uses the 'Lifecycle' XES extension to define if an event is a start or the end of an activity. This extension definition is missing in the synthetic log. The last metadata included in the real log provides information about the program and libraries used to process the event log data. For example, the real event log specifies that it was processed with Fluxicon Disco[10]. It was not possible to recreate this metadata in the synthetic event log.

In summary, it was not possible to accurately recreate the metadata of the real log in the synthetic log. However, this was to be expected. A large variety of metadata can

8. https://www.tf-pm.org/resources/xes-standard/about-xes/event-logs
9. https://data.4tu.nl/articles/dataset/Activities_of_daily_living_of_several_individuals/12674873/1
10. https://fluxicon.com/disco/

be included in XES event logs. It is simply not feasible to offer configuration options for all possible metadata combinations in a program such as DALG. However, the metadata is only a very small part of an event log since event logs mainly consist of a large number of traces. Therefore, it is feasible to manually add the metadata to a synthetic event log.

Statistical Comparison

This paragraph will present the results of comparing the statistical analysis of the real and the synthetic event log. First, the occurrences of events will be compared. Subsequently, the lead times of the activities in the logs are examined.

The traces of the real event log have an average of 56 events. This means that there are, on average, 26 activities executed in a single day since each activity is represented by two events (activity start and activity end) and there are starting and ending events. The synthetic event log has an almost identical average trace length with 55 events or 25.5 activities. The analysis of the occurrences of the individual events leads to similar results. There is no significant difference between the average occurrences of individual activities in the real and the synthetic event log. Figure 6.3 shows the differences for each individual activity present in the event log. Additionally, there is no significant deviation between the event logs regarding

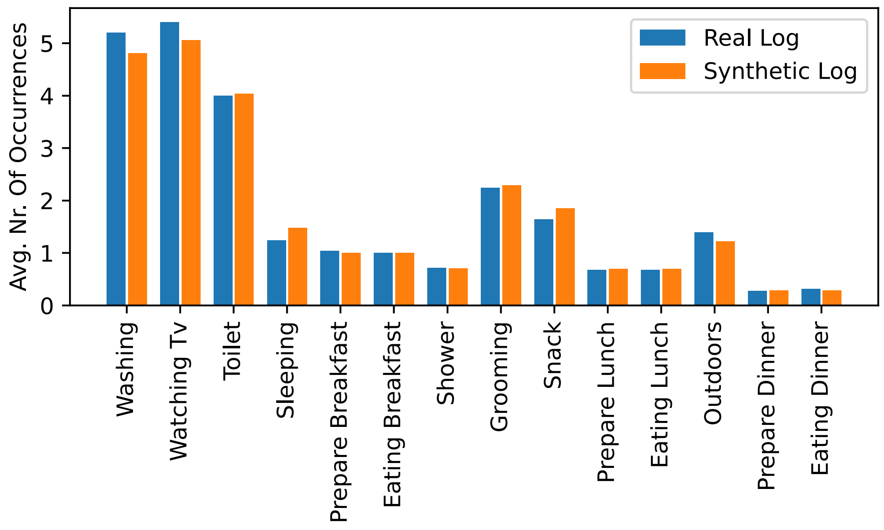


Figure 6.3: Comparison of the average number of occurrences of activities in the real and synthetic event log

the minimum and the maximum number of activity occurrences. Both logs' average trace length and activity occurrences are similar because DALG allows the user to provide weights for the transitions. These weights could be calibrated so that each activity's average number of occurrences matches the real event log. Additionally, the transitions' weights leading to the model's final place could be calibrated so the average trace length matches the real log.

Just as in the analysis of the activity occurrences, no significant differences could be found while analysing the lead times of activities in the real and synthetic even log. Figure 6.4 shows the differences between the average lead times of activities in the real and synthetic logs. Additionally, the activities' standard deviation, minimum,

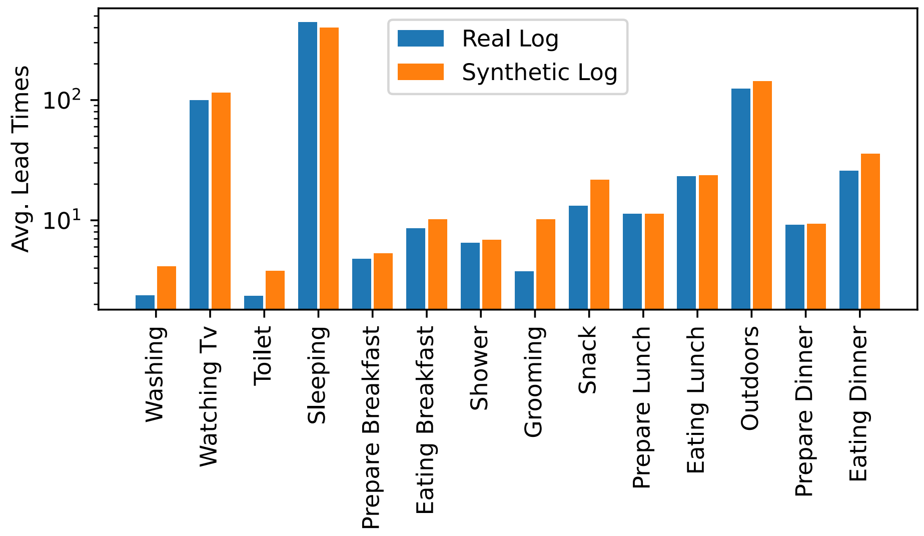


Figure 6.4: Comparison of the average lead times (in minutes) of activities in the real and synthetic event log

and maximum lead times are also similar between the real and the synthetic event log. The lead times of events are similar because DALG allows the user to provide an average lead time for each transition. Additionally, the standard deviation, the minimum, and the maximum can be defined for the lead time. Therefore, the lead times of the activities in the real event log were accurately recreated in the synthetic log.

In summary, no significant differences were discovered during the statistical analysis of the real and the synthetic event log. The occurrence of activities and the lead times of these activities were accurately recreated in the synthetic log.

Event Log Errors

Real-world event logs can contain errors. For example, event logs are often recorded from sensors that can deliver false readings. The chosen event log also contains what seems to be errors. For example, the log contains a trace where there is an event after the 'End' event. As it was explained before, each trace has a 'Start' event at the beginning and an 'End' event at the end of a trace. However, a single trace in the real log does not conform to that rule. This trace has a single event after the 'End' event. This is most likely an error. The synthetic event log generated by DALG does not contain such errors since DALG currently does not support any deviation from the model. However, it is not uncommon for event logs to contain such errors. Therefore, DALG is not able to recreate this aspect of real-world event logs.

Semantic Analysis

This paragraph will present the results of comparing the real and the synthetic event log on a semantic level. First, the order of activities in the traces is compared. Subsequently, the time frame of the traces is analysed.

As previously mentioned in the real event log, each activity executed by an individual corresponds to two events in the log. These events represent the start and the end of the activity. The events have the same name but differ in the ‘lifecycle’ variable, which specifies if the event is the start or the end of the activity. It was possible to recreate this aspect of the real event log accurately in the synthetic event log. The activities model contains invisible transitions that loop around each non-invisible transition so that each transition always gets fired twice. Transition guards were used to control that each transition fires exactly two times before the model moves towards the next transition. Additionally, the variable configuration of DALG could be used to ensure that the values generated for the ‘lifecycle’ variable always show the correct state of the activity (start or end).

As previously explained, each trace in the real event log starts with an ‘Start’ activity and ends with an ‘End’ activity. There is only one exception to this rule which seems to be an error. The traces in the synthetic event log follow this rule and, therefore, accurately recreate this aspect of the real log. When it comes to the order of activities in the real event log, only a few activities follow an exact order. These activities relate to the three main meals an individual can consume in a day (breakfast, lunch, and dinner). Each of these three meals has a preparation and a consumption activity. In the real log, the preparation activity always precedes the consumption activity. This is also the case in the synthetic event log. Additionally, the three meals always appear in the same order in the real event log (first breakfast, then lunch, and finally dinner). This is also the case in the synthetic event log. The correct order of these activities was achieved through transition guards. For example, there is a transition guard that only allows the ‘prepare lunch’ transition to be fired if the ‘eating breakfast’ transition has already been fired before.

The remaining activities do not follow any strict order in the real event log, with the only exception being that the same activity is not executed twice in a row. It was possible to recreate this restriction in the synthetic event log through careful modelling of the DPN. However, a real individual executes the activities, so there are some hidden rules by which humans act. Since these rules can not easily be formally specified, it was not possible to model a DPN that accurately conforms to them. Therefore, some traces in the synthetic event log contain activity sequences that seem odd to a ‘normal’ human. For example, there is a trace in the synthetic event log with 22 activities, of which 7 are ‘Watching Tv’. This kind of behaviour is not seen in the real event log. However, this behaviour could still be realistic since there are individuals who spend most of their day watching television.

As previously mentioned, each trace in the real event log represents a day in the life of one individual. Therefore, each trace in the real log roughly starts and ends at midnight. It was not possible to recreate this aspect of the real event log in the synthetic log. DALG only allows the user to determine how far apart events in the log should be and what the starting time of each trace should be. There is no option to have, for example, transition guards referring to the current simulation time to end the trace when a specific time is reached. Therefore, the sequence of activities in the synthetic traces does not always end at midnight. Additionally, the synthetic event log traces do not always cover a single day (24 hours).

To summarise, the real and the synthetic event log are similar on a semantic level. The sequences of activities in the synthetic log make sense on a semantic level. The only major difference between the real and synthetic log is the time frame of the traces, which do not represent a single day in the synthetic logs.

Summary

Table 6.2 shows an overview of the results of comparing the synthetic to the real event log. During the comparison of the real and synthetic event logs, significant dif-

Table 6.2: Results of comparing the synthetic and real event log regarding different aspects

Aspect	**Comparison Result**
Metadata	Significant deviations between the real and synthetic log. The real log contains mostly different metadata then the synthetic log.
Statistics	
Activity occurrences	Very similar average number of occurrences per trace for every activity.
Lead Times	Very similar average lead times for every activity.
Errors	Significant deviations between the real and synthetic log. The synthetic log does not contain any errors but the real log does.
Semantic Analysis	
General	Only relatively small deviations between the real and synthetic log. Each trace starts and ends with a 'Start' and ends with a 'End' activity in both logs. Each activity has a start and end event in both logs. Some traces in the synthetic log contain 'odd' behaviour not seen in the log (For example, a individual watching tv for most of the day).
Activity order	No significant deviations between the real and synthetic log. Preparation and consumption activities of meals and the different meal types share the same orders in both logs. Additionally, no activity got executed twice in a row in both logs.
Time frame	Significant deviations between the real and synthetic log. Each trace in the real log covers one day (24 hours) and starts and ends at midnight. The traces in the real event log do not cover 24 hours and do not start and end at midnight.

ferences were found between the metadata in both logs. The real event log contained significantly more metadata than the synthetic log. During the statistical comparison, it was found that both logs are very similar from a statistical perspective. Both the occurrences of activities and the lead times of activities are very similar in both logs. When examining the real event log, it was found that it contained errors. The synthetic log does not contain these errors. During the semantic comparison of the real and synthetic log, it was found that there are significant differences between the time frames of the traces in the real and synthetic event logs. The real traces each cover a single day and start/end at midnight. This is not the case in the synthetic event log. However, the other aspects of the semantic analysis, such as the order of activities, were found to be very similar in the real and the synthetic event log.

To summarize, it was found that the real and synthetic log are similar. However, there are significant deviations regarding the metadata, errors and the time frame of traces.

6.4.3 Meaningfulness from the Perspective of a Domain Expert

This section will present the results of evaluating the traces generated by DALG with the help of a domain expert. The requirement model was used for this step, and Dr med. Stephan Alexander Braun evaluated the traces generated from this model by DALG. The generated traces are contained in appendix B. This section is structured as follows. First, some basic information about the used model is presented. Subsequently, the aspects of the event log that were deemed not realistic by Dr Braun are described. Lastly, a summary of the results of these evaluation steps is given.

The aspects that were deemed not realistic by Dr Braun are sorted into three distinct groups regarding the reason that caused them to be not realistic. During the evaluation, it was found that most unrealistic aspects of the traces were caused by inaccuracies in the model or the configuration and not a problem of DALG itself. The first group of unrealistic aspects are caused by the process model not accurately depicting realistic treatment processes. The second group consists of unrealistic aspects that resulted from a misconfiguration of the semantic information provided to DALG. The third and last group of aspects are caused by DALG not being suitable to generate realistic event logs concerning these aspects.

Process Model Basics

The requirement model is used for the evaluation step described in this section. This paragraph provides a basic overview of the behaviour described in the model and the generated logs so that the unrealistic aspects described in the following paragraphs can be better understood. The requirement model describes the diagnosis, treatment, and aftercare of people suspected of having melanoma (skin cancer). Therefore, the transitions in the model represent diagnosis and treatment steps, such as the first clinical examination.

Figure 6.5 illustrates an exemplary treatment process of a patient with melanoma that shows the basic structure of the behaviour described in the model. A more detailed illustration of this treatment case is contained on the first page of appendix B. In the case illustrated in figure 6.5, the first activity consists of a clinical examination

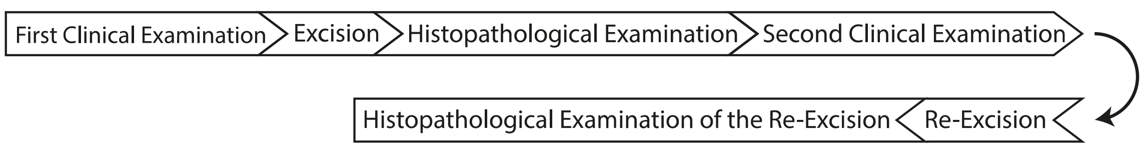

First Clinical Examination
Excision
Histopathological Examination
Second Clinical Examination
Re-Excision
Histopathological Examination of the Re-Excision

Figure 6.5: Simplified exemplary trace of the process model describing the treatment of melanoma

of the patient. Subsequently, an excision of the lesion suspected to be a melanoma is performed. The removed tissue is then analysed to, for example, determine if the lesion is a melanoma. In this exemplary case, the patient has melanoma. Therefore, a second clinical examination is performed to determine if metastasis is suspected. In this exemplary case, no metastasis is suspected. However, it was not possible to remove the entire melanoma during the first excision. Therefore, a re-excision is performed to remove the melanoma. The treatment ended after a histopathological examination was performed, during which it was determined that the entire melanoma was successfully removed after the re-excision.

The previously described case is a relatively simple example of the behaviour possible in the model. For example, if a metastasis had been suspected, a much longer and more complex treatment process would have been conducted.

Unrealistic Aspects Caused by inaccuracies in the Model

This paragraph explains aspects of the event log that were deemed unrealistic by Dr Braun and that are caused by inaccuracies in the requirement model used to generate the log. As previously mentioned, five traces were generated for the evaluation by Dr Braun. During the description of the unrealistic aspects, these traces are referenced by their trace number and can be viewed in appendix B.

Dr Braun noted that the examination for metastases is already carried out at the first clinical examination if a melanoma is suspected. In the model, it is only possible to perform this examination in the second clinical examination. Therefore, it was impossible to generate realistic traces regarding this aspect.

Another aspect noted by Dr Braun regards the excision of lesions. There are two possible outcomes for the excision. It was either possible to remove the entire melanoma or there are still parts of it left. Dr Braun noted that a re-excision is performed to remove the remaining parts in the latter case. However, in the model, it is possible not to have a re-excision performed even if there are still parts of the melanoma left.

The third trace generated for this evaluation step ends without finishing the patient's treatment. In this trace, it is confirmed that the patient has melanoma, and two excisions are performed. After the re-excision, it was determined that there are still parts of the melanoma left. However, only examination activities are conducted in the rest of the trace, and no further treatments, such as another excision, are performed. Dr Braun noted that a real treatment in such a case would have contained further treatment activities. However, the trace is a valid trace according to the model. Therefore, it was not possible to generate realistic traces regarding this aspect.

Another aspect noted by Dr Braun to not be realistic concerns the examination activity that examines the tumour marker 'lactate dehydrogenase'. One of the transitions representing this activity in the model does not write to any variables. However, Dr Braun noted that this examination would result in new patient information regarding an increased or normal value for this marker.

In the requirement model, it is possible to have two PET or CT scans performed with only a few examinations in between. This happened in the fifth trace generated for this evaluation step. However, Dr Braun noted that it is not realistic to have two PET scans performed in such close proximity to each other.

Another aspect noted regarding the PET, or CT scan activities concerns the information generated by these examinations. In the model, some of the transitions representing these activities only write to the variable depicting the patient's cancer stage. Dr Braun noted that these examinations would result in additional information about the existence of metastases. However, in the generated traces, no additional information is added by these activities since the model does not allow it.

In the fifth trace generated for this evaluation step, the patient is diagnosed with metastases. The patient is then treated with the 'BRAF' and 'MEK' inhibitor drugs. Dr Braun noted that surgery would have been performed in a real case instead of or in addition to the drug treatment. However, according to the requirement model, the generated trace is valid. Therefore, it was not possible to generate realistic traces regarding this aspect.

In the fourth trace generated for this evaluation step, the patient is diagnosed with having a metastasis with the 'BRAF V600E' mutation. The patient is treated with 'Anti-Pd1' antibodies. Dr Braun noted that in a real case with these characteristics, there would probably be additional treatments performed concerning the 'BRAF V600E' mutation. However, the model does not specify any additional treatment in such a case. Therefore, it was not possible to generate realistic traces regarding this aspect.

In the model, the transition representing a sentinel lymph node biopsy can be fired before the transition representing the lymph node sonography. However, Dr Braun noted that the sonography would be performed before the biopsy and that the outcome of the sonography would determine if the biopsy is even performed. Therefore, it was not possible to generate realistic traces regarding this aspect.

Another aspect noted by Dr Braun concerns the examination that determines if the patient has a 'BRAF V600E' or a 'BRAF V600K' mutation. In the model, this examination is performed with tissue removed from an excision of the melanoma. However, Dr Braun noted that the examination of mutations is usually not done based on the excision of the melanoma but based on samples taken from the metastasis. Therefore, it was not possible to generate realistic traces regarding this aspect.

Dr Braun noted another unrealistic aspect regarding the analysis of mutations. In the generated traces, the mutations are analysed even when it has been determined that there are only single cancer cells present in the lymph node metastases. According to Dr Braun, this is not possible. However, in the model, the variables that a transition writes to are fixed and can not change regarding other variable values. Another factor influencing this unrealistic aspect of the traces is that the variables depicting the presence of mutations are boolean variables. For example, if these variables would be strings, DALG could be configured to generate a value, such as 'can not be determined' for the variable and, therefore, make this aspect realistic. However, since these variables are boolean variables, this is not possible since both 'True' and 'False' values would mean that the analysis has been performed.

Unrealistic Aspects Caused by inaccuracies in the Configuration

This paragraph explains aspects of the event log that were deemed unrealistic by Dr Braun and that are caused by inaccuracies in the configuration of DALG used to generate the log. During the evaluation, it was determined that a better configuration of DALG could result in realistic event logs concerning these aspects.

One aspect that a better configuration could improve on is analogous to the last aspect described in the previous paragraph. During the examination of excised tissue, the parameter 'mitotic rate' is determined. However, Dr Braun noted that this parameter is only determined if the tumour thickness exceeds 1 millimetre. This parameter is always determined in the generated traces, even if the tumour thickness is below this threshold. This is again caused by the previously described problem that consists of transitions having a fixed set of variables that they write to. However, in this case, the variable for this parameter is a string variable. Therefore, DALG can be configured to write a value, such as 'can not be determined', to the variable if the tumour thickness is below 1 millimetre. Therefore, an improved configuration of DALG could result in synthetic event logs that are realistic concerning this aspect.

During the examination that determines if the patient has any mutations, the 'BRAF V600E' and the 'BRAF V600K' mutation can be detected. In the second case presented to Dr Braun, the patient has been diagnosed with both mutations. However, according to Dr Braun, this is most likely not possible. This aspect could be improved by specifying a mutual exclusivity between the two variables representing these mutations by using the dependency feature of DALG.

The unrealistic aspect noted most by Dr Braun during the evaluation concerns the cancer stage. In all but the first trace presented to Dr Braun, he noted that values generated for the stage did not change correctly in regard to the other values, such as the presence of metastases or the tumour thickness. A better configuration could improve on this aspect.

Another aspect that a better configuration could improve is the lead times of activities in the traces. For example, Dr Braun noted that results of a histopathological examination of tissue usually take five working days till they are available. This is not the case in synthetic traces. However, as shown by the statistical evaluation described in section 6.4.2 DALG offers all the features necessary to generate traces with realistic lead times for activities. Therefore, realistic traces in this aspect could be generated with an improved configuration.

The reason for the inaccuracies in the configuration that caused the unrealistic aspects described in this paragraph is that the configuration itself was done without a domain expert. The domain expert was only available for the evaluation of the traces. Therefore, it was not possible to create a highly accurately configuration. In a real scenario, the configuration could be done with the help of a domain expert that ensures that the configuration is as accurate as possible.

Unrealistic Aspects Caused by DALG

Out of the set of unrealistic aspects identified by Dr Braun, only one aspect is caused by DALG not being suited to generate event logs that are realistic concerning this aspect. This aspect concerns the timestamps of events. This problem was already discovered during the first evaluation step described in section 6.4.2. The timestamps of events in the generated traces were deemed unrealistic by Dr Braun in all five traces. The timestamps of activities in event logs represent when the activity was performed. Therefore, the difference between timestamps represents the lead time of events. Dr Braun noted that both the lead and execution times were unrealistic.

As described in the previous paragraph, a better configuration could improve the lead times of activities. However, this is not the case for the times at which activities are executed. DALG does not offer enough configuration options for timestamps to achieve realistic execution times. Consider the following example. In the first trace presented to Dr Braun, an excision of a lesion is performed at 4:35 am. Since an excision is not an emergency procedure, this timestamp is unrealistic.

The reason for the unrealistic timestamps of activities is that the time at which an activity is executed can only be configured in regard to the previous activity. There are two options for influencing these timestamps. On the one hand, the lead time of activities can be configured through an average time, standard deviation, minimum time, and maximum time. On the other hand, a delay can be defined through the same four parameters. Therefore, the execution time of an activity is determined by its specified delay and the lead time of the previous activity. These settings do not allow for enough control over the execution times of activities to generate realistic time-frames for traces. Additional settings are necessary to achieve realistic traces.

For example, it could be possible to implement a setting that allows the user to specify the time frames in which an activity can be performed. This setting could then be used to specify that non-emergency diagnosis and treatment options, like an excision, can only be performed between 7:00 am and 6:00 pm. This would result in traces that no longer have non-emergency treatments performed in the middle of the night.

Summary

During the evaluation of traces generated from the requirement model, several aspects were deemed unrealistic by Dr Braun. However, most of these aspects were caused by inaccuracies in the requirement model. These aspects could be improved by refining the used process model. However, a DPN might not allow for enough features to model a process accurately enough to be able to generate realistic traces.

Some unrealistic aspects were caused by inaccuracies in the configuration that stem from the fact that no domain expert took part in the configuration of DALG. It was found that a refined configuration could improve these aspects.

From all the aspects noted by Dr Braun, only the timestamps of activities can be traced back to DALG not being able to generate traces that are realistic concerning this aspect. The reason found for this unsuitably is the limited configuration options available for the time aspect of event logs.

6.5 Configuration Effort

During the generation of the event logs for the different evaluation steps described in the previous sections, it was found that there is a significant configuration effort needed when generating event logs with DALG. Especially when trying to generate realistic traces, the additional domain knowledge required can need a relatively large effort to specify accurately. Additionally, the configuration has to be manually performed by the user. Since this thesis aims to develop an approach that can automatically generate realistic event logs from a DPN, the manual effort needed to configure DALG has to be considered when evaluating the approach.

Consider the example of the requirement model used to generate the traces that Dr Braun evaluated. This model has 26 variables and 76 transitions. Additional information has to be provided for each variable if the goal is to generate as realistic event logs as possible with DALG. The amount of information and the amount of effort necessary varies with the variable type. For boolean and string variables, realistic values and dependencies must be provided. For long, double, and date variables, the minimum, maximum, distribution, and dependencies must be provided. Additionally, initial values and information about the variable being a trace variable or part of events must be provided for variables of all types. For each of the 76 transitions, information about the realistic time frame and weight must be provided.

During the configuration of the requirement model, it was found that the specification of dependencies for the variables required the most effort. The requirement model describes treatment processes, and the variables mostly contain information about the patient's status. Therefore, many variables in this model are interconnected. For example, the variable representing the cancer stage is dependent on several other variables, such as the tumour thickness and the variables representing the presence of mutations. Therefore, many dependencies have to be specified to generate realistic traces in this case. This results in a large manual configuration effort.

To summarize, depending on the number of variables and transitions in the process model, there is a significant manual effort necessary to perform the configuration of DALG. Additionally, the effort depends on how interconnected the variables in the model are since the dependencies between the variables have to be specified.

6.6 Discussion

In this section, the previously described results of the evaluation are discussed. First, a summary of the results is presented, and subsequently, the problems uncovered during the evaluation are discussed.

Summary

The evaluation showed that DALG can generate event logs that conform to the control-flow and data perspective of a DPN. Additionally, it was found that DALG can generate all behaviour possible in a given model. However, for models with high complexity and, therefore, many possible traces, DALG can not achieve the generation of all behaviour in a reasonable time.

During the examination of the realism of the event logs, it was found that DALG cannot generate event logs that are realistic in every aspect considered during the evaluation. It was found that DALG cannot generate realistic metadata for the event logs it can generate. Additionally, errors sometimes contained in real event logs were not possible to be generated with DALG. When it comes to the semantic perspective, it was found that DALG could not generate traces that were completely realistic. However, the main reason for the unrealistic traces were inaccuracies in the used process models and configurations.

Metadata

The fact that DALG cannot generate traces that contain realistic metadata is due to the large variety of metadata that can be included in event logs. DALG was developed to support as many DPNs as possible and be a general way of generating event logs from DPNs. Therefore, DALG includes options with which the most common metadata can be included in the event logs. However, due to the large amount of different metadata that can be present in event logs, there will always be event logs that can not be recreated with DALG. It is simply not feasible to implement configuration options for all metadata possible in event logs. This problem is most significant when trying to generate event logs that are indistinguishable from real logs. However, this use case is not the main use case for DALG. The motivation behind the approach's development is situations where there are no event logs of a specific behaviour present that can be used to apply AI methods. In these situations, there is no metadata that has to be accurately recreated. Additionally, AI methods, such as process discovery algorithms used on event logs usually focus on the traces of event logs and pay little to no attention to the metadata.

Errors

It was shown during the evaluation that real event logs can contain errors, such as false sensor readings. The developed approach does not consider this aspect of realistic event logs. This aspect should be considered in future work. However, when applying AI methods on real data, the datasets are usually subjected to preprocessing in which, for example, errors can be corrected or filtered. Therefore, the approach could still be useful to replace missing real data for using AI methods. However, errors in the event log can be useful for some use cases, such as developing conformance checking and process enhancement algorithms. Therefore, additional research should be conducted regarding the generation of event logs with intentional errors.

Generating All Possible Traces

As previously described, DALG can generate event logs containing all possible behaviour in a model. This characteristic is advantageous for AI methods. However, it was found that it is not possible to generate all possible traces in a reasonable time for models with high complexity. There are two reasons for this. First, the number of possible traces scales exponentially with the number of markings in the model where more than one transition is enabled. Large DPNs, such as the requirement model, have many such markings and, therefore, many possible traces that can take a lot of time to explore. The second reason for the large amount of time necessary to generate all traces of a complex DPN is the algorithmic approach used for the simulation mode developed for this purpose (see section 5.6.4). It is not a straightforward task to generate all traces from a DPN since the valid traces depend on the values generated for variables. The approach that DALG used for this task consists of brute-forcing combinations of values for write operations that fit the transition guards of traces. Therefore, this approach can take a long time to generate all traces, even for DPNs that are not that complex. Further research regarding improving the algorithmic approach for this simulation model could improve this characteristic of the approach.

Semantic Perspective

During the evaluation step, which consisted of imitating a real event log and the evaluation by Dr Braun, it was found that the event logs were not realistic from a semantic perspective. However, it was found that these unrealistic aspects of the generated logs were, for the most part, caused by inaccurate process models and DALG configurations.

While trying to imitate the daily living activities of real individuals, some synthetic traces contained behaviour that was not completely realistic (see section 6.4.2). These traces were the result of the process model not accurately describing real human behaviour. A similar problem caused the traces evaluated by Dr Braun to not be realistic from a semantic perspective. An inaccurate process model lead to unrealistic traces.

These examples shows show the most significant problem with trying to generate completely realistic event logs based on process models. It is difficult, if not impossible, to capture processes in a Data Petri net accurately enough to generate completely realistic event logs. The features that DPNs can have do not allow for precise enough modelling of processes.

For example, the variables that a transition writes to are fixed in a DPN. Therefore, the transitions always generate values for the same variables regardless of the current state of the model. It would be advantageous if a DPN would allow a transition to dynamically change what variables it writes to. This feature could, for example, be used to make the requirement model more precise. The transitions in this model that represent the diagnostic activities in the cancer treatment always write to the same variables. However, Dr Braun noted that in the real-world, the information that can be gathered from specific diagnostic activities changes depending on the status of the patient. However, as previously described, it is difficult to model such behaviour in a DPN since the set of variables that a transition writes to is fixed.

Closing Thoughts

The results of this evaluation show that the developed approach is not capable of generating event logs of which every aspect is realistic. However, it was found that the main reason for this result is that it is difficult to model a process precise enough in a DPN to generate realistic data. It was also found that additional domain knowledge is necessary since DPNs are not only missing the necessary precision but also lack the necessary domain knowledge to generate realistic data. However, it was found that even if the information and process model provided to the approach are of sufficient quality, the approach could still struggle to generate realistic event logs in some cases. It is simply not feasible for a general approach that supports most DPNs to possess the features necessary to generate traces with very specific characteristics. There are always gonna be event logs with uncommon characteristics which can not be recreated with this approach. An implementation like DALG can only offer so many configurations options. However, these configuration options are only necessary in the first place because Data Petri nets do not allow a process to be modelled precisely enough. If DPNs allowed for a more precise description of a process, an implementation such as DALG would need far fewer configuration options since the information could be directly acquired from the model. Therefore, more precise process models could also solve the problem of DALG lacking the configuration options necessary to generate very specific behaviour.

To summarize, the approach shows promising potential to generate event logs that are realistic enough to be useful for replacing missing data needed for applying AI methods. However, the limitations that stem from modelling processes in DPNs prevent the generation of synthetic data that is realistic in regard to every aspect. Therefore, future work regarding the use of different process models that allow for greater precision when modelling a process or the expansion of the features possible in a DPN should be conducted.

7. Conclusion and Future Work

This thesis aims to develop an approach for generating realistic synthetic event logs based on Data Petri nets. The approach is developed to allow the substitution of missing real-world datasets for applying AI methods through synthetic data. Additionally, throughout the development, it is examined what difficulties and challenges need to be overcome to achieve realistic data generation and if it is even possible to generate realistic data based on DPNs.

To achieve these goals, a requirement analysis (see section 3.1) was first carried out to identify what requirements the approach has to satisfy to achieve realistic data generation that can be used as a substitute for real data. During the requirement analysis, tools that allow the creation of DPNs and an exemplary DPN were analysed to determine what features DPNs can have. The results of this step were used to determine potential challenges when generating synthetic data from DPNs. Finally, based on the findings, a set of requirements was specified to direct the further steps in the development of the approach.

Following the requirements analysis, a literature review (see section 3.2) was conducted to find already existing approaches or methods for generating synthetic procedural data based on process models. This step aimed to determine if any pre-existing approach or method could be used as a basis for the approach developed in this thesis. During this step, multiple approaches that were potentially suited to be used as a basis were identified. The approaches identified during the literature review were subsequently evaluated and compared (see section 3.3) based on the requirements specified in the requirements analysis. It was examined how well the approaches are suited to satisfy the requirements if they would be used as a basis. Based on the findings of the evaluation of the approaches, it was decided to use a *token-based simulation* approach as a basis for the approach developed in this thesis.

Following the preliminary work, a way to use a token-based simulation to generate realistic synthetic event logs based on Data Petri nets was developed (see chapter 4). The development focused on having the token-based simulation generate conforming traces and generating realistic values for the variables of a DPN. In order to evaluate the developed approach, it had to be implemented so that the data the approach can generate could be analysed. Therefore, a concept for a software that uses the approach to allow for the generation of synthetic event logs from Data Petri nets was developed (see sections 5.1 through 5.6). During the development of the concept, the rough algorithmic approaches of the developed approach were evolved into full algorithms, and it was determined how these algorithms could be implemented in software. The developed concept was then implemented (see section 5.7) resulting in a software called *DALG: The Data Aware Event Log Generator*[1]. DALG uses the

1. https://github.com/DavidJilg/DALG

developed approach to generate synthetic event logs that conform to the control-flow and the data perspective of Data Petri nets.

In the final step of this thesis (see chapter 6), DALG was used to evaluate if the approach developed in this thesis can satisfy the requirements specified during the requirements analysis and if the approach can generate realistic event logs that are useful for substituting real data. The realism of the synthetic logs that DALG can generate was examined by comparing them to real event logs and having a domain expert determine if all aspects of the logs are realistic. The evaluation found that while some aspects of the synthetic event logs are realistic, several other aspects had unrealistic properties that made it possible to differentiate the synthetic event logs from real logs. It was found that the process models used to generate the event logs for the evaluation were not precise enough no generate realistic event logs. The lack of precision stems from the limitations present when modelling processes in DPNs. It was found that there are not enough features possible in a DPN to model a process accurately enough to generate realistic data based on this description of the process.

The main contributions made in this thesis are identifying challenges associated with trying to generate realistic event logs from Data Petri nets and the tool DALG that allows generating event logs that conform to the control-flow and data perspective of a DPN. Additionally, solutions for several of the identified challenges were conceived while developing the approach.

Regarding the challenges, it was found that it is difficult, if not impossible, to create DPNs that are precise enough to allow for the generation of realistic data since DPNs lack the necessary expressiveness. Since the models used during the generation of synthetic event logs for the evaluation were not precise enough, it was impossible to generate realistic data in every aspect. However, it was found that domain knowledge provided by the user in addition to the process model can improve this problem and lead to more realistic event logs. For this purpose, the information missing in DPNs was identified during this thesis. Additionally, it was researched how this information could be specified by the user. However, it was found that the specification of this additional domain knowledge can be difficult and require a significant manual effort. Additionally, it was found that it is not possible to entirely substitute the information missing in DPNs with information provided by the user since software such as DALG can only offer a finite number of possible configuration options for the additional domain knowledge. There will always be real event logs with very specific features that can not be recreated with DALG since a general implementation lacks the necessary configuration options. Therefore, it was found that more precise process models are necessary in order to achieve the generation of realistic data.

Despite the identified problems, the Data Aware Event Log Generator is still a useful contribution of this thesis. It solves the problem of generating synthetic event logs that conform to both the control-flow and the data perspective of a DPN. Previously developed approaches had only achieved the generation of event logs that conform to the control-flow perspective. Additionally, the generated event logs are realistic concerning several aspects despite the lacking expressiveness of DPNs. This was achieved by developing a method that uses domain knowledge provided by the user in addition to the process model he provides to generate realistic event logs concerning several aspects. This method can be used to substitute missing information in DPNs to a certain degree. However, the developed method for substituting missing information in DPNs can not entirely negate the lacking expressiveness of DPNs. Nonetheless, the generated data can still be useful for scientific research in the process mining domain, even if not all aspects of the synthetic event logs are realistic.

Several aspects of the research done in this thesis require future work. During the evaluation of the approach, it was found that one of the main factors resulting in the approach not being able to generate realistic data is the inaccuracy of the process models used during the evaluation, which stems from a lack of expressiveness of Data Petri nets. The limited features allowed in a DPN make it difficult, if not impossible, to model a process accurately enough so that the approach developed in this thesis can generate realistic data. Therefore, future research regarding the extension of DPNs should be conducted. The extensions should focus on allowing a more precise description of a process.

For example, it should be possible to have transitions that dynamically change the variables they write instead of always writing values to a fixed set of variables. This example would be especially helpful for process models depicting the treatment of patients since the information gathered from diagnosis activities changes depending on the patient's status. For example, consider the requirement model used during the requirement analysis and evaluation. In this model, the transition that represents a histopathological examination of tissue removed during an excision writes to variables that depict the presence of specific cancer mutations. However, during the evaluation of the approach, Dr Braun noted that the identification of mutations can not be achieved when there are only single cancer cells in the lymph nodes. Therefore, it would be advantageous if the transition did not write to the mutation variables if only single cancer cells were found in the lymph nodes. Alternatively, it could also be researched if other process model types, such as BPMNs, could provide the precision necessary to generate realistic data. However, the developed approach uses a token-based simulation specifically targeted at Petri nets. Therefore, future research should focus on first trying to extend DPNs.

Only a small sample of traces was presented to Dr Braun when evaluating the realism of the synthetic event logs. Since the process model used in this step is highly complex, the five traces presented to Dr Braun are not representative of the entire behaviour possible in the model. Additionally, only a single process model was used during this step. In total, only three process models were used during the entire evaluation. Therefore, further work should be done regarding the further evaluation of the developed approach. This should include a broader selection of process models and samples of traces that are more representative of the behaviour described in the models.

Several aspects of the approach and the tool developed in this thesis would benefit from further research. During the evaluation, it was found that the aspect of the synthetic logs that is the most unrealistic is the time aspect. The timestamps of events in the event logs generated by DALG are very unrealistic. This is due to DALG not offering the user enough control over the timestamps. Further research regarding ways to give the user more control over the specification of domain knowledge for this aspect could significantly improve the realism of the synthetic logs.

The specification of domain knowledge, in general, is an aspect of DALG that would benefit from further research. During the development of the approach, it was found that extensive domain knowledge is required in addition to the model the user provides. The user must manually specify all the additional domain knowledge required in the developed approach. The evaluation of the approach showed that this requires a significant manual effort and can also be difficult. Further research regarding the improvement of this issue could significantly improve the usability of the developed approach and make it easier to generate more realistic event logs. For example, it could be researched if the manual specification could at least in part be automated through the use of domain knowledge that already exists in structures, such as ontologies, that computers can automatically process. Additionally, it could be researched how already existing event logs can be automatically analysed to acquire domain knowledge that does not have to be specified manually. During the evaluation step in which it was tried to imitate a real-world event log (see section 6.4.2), a Python script was used to analyse the real event log and acquire domain knowledge. For example, the mean lead times of the activities were calculated so that the data could be used to configure DALG. However, the input of the data acquired through the script had to be further processed and input manually. Nonetheless, this methodology for acquiring domain knowledge shows potential for automating the manual specification necessary in the current state of the approach.

Another aspect of the approach that would benefit from further research is the generation of all traces possible in a DPN. When the requirements were specified that required the approach to be able to generate all traces in a DPN, the expectation was that this would not pose a serious problem for the approach development. This was due to the fact that generating all traces from a Petri net limited to the control-flow perspective can be done with a fairly simple algorithm, and it had already been done in papers such as the Straightforward Petri Net-based Event Log Generation in ProM [vBVB14]. However, adding the data perspective to a Petri net makes this problem much harder since the possible traces are affected by a combination of variable values. Therefore, calculating all possible traces in a DPN is a highly complex constraint satisfactory problem. As explained in section 5.6.4 the algorithmic approach used by DALG to generate all traces is using a brute force approach to generate variable values for the write operations of transitions that fit the transition guards present in a trace. This very slow approach makes it impossible to generate all traces possible in a complex DPN in a reasonable time. Additionally, the approach has trouble dealing with loops in which the same variable gets written to multiple times. Therefore, additional research towards a more efficient approach to this problem could improve the developed approach.

During the comparison of synthetic and real event logs during the evaluation of the approach, it was found that real event logs can contain errors. The developed approach is not capable of generating event logs with errors. However, it could improve the realism of the synthetic event logs if the approach can generate event logs with intentional deviations from the process model. There are already approaches to specifically generate synthetic traces that do not conform to the model (for example, [LCCM20]). However, these approaches are limited to the control-flow perspective. Therefore, further research regarding the generation of synthetic event logs that contain errors could be beneficial for generating more realistic data.

In this thesis, an approach was developed that allows for the generation of synthetic event logs from Data Petri nets. However, the goal of generating realistic event logs could not be reached since the synthetic event logs contain several unrealistic aspects. Nonetheless, the approach looks promising for generating synthetic data that can be used as a substitute for real data. Additionally, several of the uncovered problems could be improved by further research, which could further improve the realism of the event logs generated by the developed approach.

Bibliography

[4TU.] 4TU.Federation. 4TU.ResearchData: Frequently Asked Questions. https://data.4tu.nl/info/en/about-4turesearchdata/frequently-asked-questions. Last Accessed: 14th of February 2022.

[AAMA^{+}12] W. van der Aalst, A. Adriansyah, A. K. A. de Medeiros, F. Arcieri, T. Baier, T. Blickle, J. C. Bose, P. van den Brand, R. Brandtjen, J. Buijs, A. Burattin, J. Carmona, M. Castellanos, J. Claes, J. Cook, N. Costantini, F. Curbera, E. Damiani, M. de Leoni, P. Delias, B. F. van Dongen, M. Dumas, S. Dustdar, D. Fahland, D. R. Ferreira, W. Gaaloul, F. van Geffen, S. Goel, C. Günther, A. Guzzo, P. Harmon, A. ter Hofstede, J. Hoogland, J. E. Ingvaldsen, K. Kato, R. Kuhn, A. Kumar, M. La Rosa, F. Maggi, D. Malerba, R. S. Mans, A. Manuel, M. McCreesh, P. Mello, J. Mendling, M. Montali, H. R. Motahari-Nezhad, M. zur Muehlen, J. Munoz-Gama, L. Pontieri, J. Ribeiro, A. Rozinat, H. Seguel Pérez, R. Seguel Pérez, M. Sepúlveda, J. Sinur, P. Soffer, M. Song, A. Sperduti, G. Stilo, C. Stoel, K. Swenson, M. Talamo, W. Tan, C. Turner, J. Vanthienen, G. Varvaressos, E. Verbeek, M. Verdonk, R. Vigo, J. Wang, B. Weber, M. Weidlich, T. Weijters, L. Wen, M. Westergaard and M. Wynn. Process Mining Manifesto. In F. Daniel, K. Barkaoui and S. Dustdar (eds.), *Business Process Management Workshops*, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg, p. 169–194.

[ACGL^{+}08] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, M. Montali and P. Torroni. Expressing and Verifying Business Contracts with Abductive Logic Programming. *Int. J. Electron. Commer.* 12(4), 2008, p. 9–38.

[AcSJ17] L. Ackermann, S. Schönig and S. Jablonski. Simulation of Multi-perspective Declarative Process Models. In M. Dumas and M. Fantinato (eds.), *Business Process Management Workshops*, Cham, 2017. Springer International Publishing, p. 61–73.

[Alug16] R. Alugubelli. Exploratory Study of Artificial Intelligence in Healthcare. *International Journal of Innovations in Engineering Research and Technology* volume 3, 01 2016, p. 1–10.

[BaSo18] E. Batista and A. Solanas. Process Mining in Healthcare: A Systematic Review. In *2018 9th International Conference on Information, Intelligence, Systems and Applications (IISA)*, 2018, p. 1–6.

[BrPS99] S. C. Brailsford, C. N. Potts and B. M. Smith. Constraint satisfaction problems: Algorithms and applications. *European Journal of Operational Research* 119(3), 1999, p. 557–581.

[Bura15] A. Burattin. PLG2: Multiperspective Processes Randomization and Simulation for Online and Offline Settings. *ArXiv* volume abs/1506.08415, 2015.

[CBCM15] C. D. Ciccio, M. L. Bernardi, M. Cimitile and F. M. Maggi. Generating Event Logs Through the Simulation of Declare Models. In J. Barjis, R. Pergl and E. Babkin (eds.), *Enterprise and Organizational Modeling and Simulation - 11th International Workshop, EOMAS 2015, Held at CAiSE 2015, Stockholm, Sweden, June 8-9, 2015, Selected Papers*, volume 231 the *Lecture Notes in Business Information Processing*. Springer, 2015, p. 20–36.

[CCLM17] F. Chesani, A. Ciampolini, D. Loreti and P. Mello. Abduction for Generating Synthetic Traces. In E. Teniente and M. Weidlich (eds.), *Business Process Management Workshops - BPM 2017 International Workshops, Barcelona, Spain, September 10-11, 2017, Revised Papers*, volume 308 the *Lecture Notes in Business Information Processing*. Springer, 2017, p. 151–159.

[Dech98] R. Dechter. Constraint satisfaction. In *In In the MIT Encyclopedia of the Cognitive Sciences (MITECS*. Citeseer, 1998.

[dLvdA13] M. de Leoni and W. M. P. van der Aalst. Data-Aware Process Mining: Discovering Decisions in Processes Using Alignments. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, New York, NY, USA, 2013. Association for Computing Machinery, p. 1454–1461.

[DMVW+05] B. F. van Dongen, A. K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters and W. M. P. van der Aalst. The ProM Framework: A New Era in Process Mining Tool Support. In G. Ciardo and P. Darondeau (eds.), *Applications and Theory of Petri Nets 2005*, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg, p. 444–454.

[EsKA19] E. Esgin and P. KARAGOZ. Process Profiling based Synthetic Event Log Generation. In *International Conference on Knowledge Discovery and Information Retrieval*, 01 2019, p. 516–524.

[GhAm16] M. Ghasemi and D. Amyot. Process mining in healthcare: A systematised literature review. *International Journal of Electronic Healthcare* volume 9, 01 2016, p. 60.

[GoVC13] S. Goedertier, J. Vanthienen and F. Caron. Declarative business process model-ling: Principles and modelling languages. *Enterprise Information Systems* volume 9, 06 2013, p. 161–185.

[GVLC+18] R. Gatta, M. Vallati, J. Lenkowicz, C. Casà, F. Cellini, A. Damiani and V. Valentini. A Framework for Event Log Generation and

Knowledge Representation for Process Mining in Healthcare. In *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*, 11 2018, p. 647–654.

[HOPS+06] K. Hee, O. Oanea, R. Post, L. Somers and J. M. Van der Werf. Yasper: a tool for workflow modeling and analysis. *2010 10th International Conference on Application of Concurrency to System Design* volume 0, 01 2006, p. 279–282.

[HRMK19] D. Horgan, M. Romao, S. A. Morré and D. Kalra. Artificial Intelligence: Power for Civilisation – and for Better Healthcare. *Public Health Genomics* 22(5-6), 2019, p. 145–161.

[IEEE16] IEEE. IEEE Standard for eXtensible Event Stream (XES) for Achieving Interoperability in Event Logs and Event Streams. *IEEE Std 1849-2016*, 2016, p. 1–50.

[Inte11] International Organization for Standardization. ISO/IEC/IEEE Draft Standard for software and systems engineering – Life cycle processes – Requirements engineering. *ISO/IEC/IEEE P29148 First Edition*, 2011.

[IvAl14] S. Ivan and A. Alexey. Generation of a Set of Event Logs with Noise, 2014.

[Jack19] D. Jackson. Alloy: A Language and Tool for Exploring Software Designs. *Commun. ACM* 62(9), August 2019, p. 66–76.

[Jens97] K. Jensen. A brief introduction to coloured Petri Nets. In E. Brinksma (eds.), *Tools and Algorithms for the Construction and Analysis of Systems*, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg, p. 203–208.

[KaKa14] V. Kataeva and A. Kalenkova. Applying Graph Grammars for the Generation of Process Models and Their Logs, 01 2014.

[Kind04] E. Kindler. Using the Petri Net Markup Language for Exchanging Business Processes – Potential and Limitations –, 2004.

[KWDS+04] O. Kummer, F. Wienberg, M. Duvigneau, J. Schumacher, M. Köhler, D. Moldt, H. Rölke and R. Valk. An Extensible Editor and Simulation Engine for Petri Nets: Renew. In J. Cortadella and W. Reisig (eds.), *Applications and Theory of Petri Nets 2004*, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg, p. 484–493.

[LCCM20] D. Loreti, F. Chesani, A. Ciampolini and P. Mello. Generating synthetic positive and negative business process traces through abduction. *Knowledge and Information Systems* 62(2), Feb 2020, p. 813–839.

[Lloy84] J. Lloyd. *Foundations of Logic Programming.* Symbolic computation. Springer-Verlag. 1984.

[LLXN19] T. Lysaght, H. Lim, V. Xafis and K. Ngiam. AI-Assisted Decision-making in Healthcare: The Application of an Ethics Framework for Big Data in Health and Research. *Asian Bioethics Review* volume 11, 09 2019.

[LuKJ02] E. Lundin, H. Kvarnström and E. Jonsson. A Synthetic Fraud Data Generation Methodology. In R. H. Deng, S. Qing, F. Bao and J. Zhou (eds.), *Information and Communications Security, 4th International Conference, ICICS 2002, Singapore, December 9-12, 2002, Proceedings*, volume 2513 the *Lecture Notes in Computer Science*. Springer, 2002, p. 265–277.

[MaLR15] F. Mannhardt, M. de Leoni and H. A. Reijers. The Multi-perspective Process Explorer. In F. Daniel and S. Zugal (eds.), *Proceedings of the BPM Demo Session 2015 Co-located with the 13th International Conference on Business Process Management (BPM 2015), Innsbruck, Austria, September 2, 2015*, volume 1418 the *CEUR Workshop Proceedings*. CEUR-WS.org, 2015, p. 130–134.

[Mann18] F. Mannhardt. *Multi-perspective process mining*. Dissertation, Mathematics and Computer Science, February 2018.

[MeGü04] A. Medeiros and C. Günther. Process mining: Using CPN tools to create test logs for mining algorithms, 01 2004.

[MSKv17] A. A. Mitsyuk, I. S. Shugurov, A. A. Kalenkova and W. M. van der Aalst. Generating event logs for high-level process models. *Simulation Modelling Practice and Theory* volume 74, 2017, p. 1–16.

[Mura89] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE* 77(4), 1989, p. 541–580.

[PeMi19] P. Pertsukhov and A. Mitsyuk. Simulating Petri Nets with Inhibitor and Reset Arcs. *Proceedings of ISP RAS* volume 31, 10 2019, p. 151–162.

[Petr62] C. A. Petri. *Kommunikation mit Automaten*. Dissertation, Universität Hamburg, 1962.

[PMSP$^+$21] T. Pereira, J. Morgado, F. Silva, M. M. Pelter, V. R. Dias, R. Barros, C. Freitas, E. Negrão, B. Flor de Lima, M. Correia da Silva, A. J. Madureira, I. Ramos, V. Hespanhol, J. L. Costa, A. Cunha and H. P. Oliveira. Sharing Biomedical Data: Strengthening AI Development in Healthcare. *Healthcare* 9(7), 2021.

[Prec00] L. Prechelt. An empirical comparison of seven programming languages. *Computer* 33(10), 2000, p. 23–29.

[pro] ProM Tools Documentation. http://www.promtools.org/doku.php. Last Accessed: 27th of February 2022.

[RBBW+20] D. Rankin, M. Black, R. Bond, J. Wallace, M. Mulvenna and G. Epelde. Reliability of Supervised Machine Learning Using Synthetic Data in Health Care: Model to Preserve Privacy for Data Sharing. *JMIR Med Inform* 8(7), Jul 2020, p. e18910.

[Reis13] W. Reisig. *Understanding Petri nets. Modeling techniques, analysis methods, case studies.* Springer, Berlin, Heidelberg. 07 2013.

[RWLL+03] A. V. Ratzer, L. Wells, H. M. Lassen, M. Laursen, J. F. Qvortrup, M. S. Stissing, M. Westergaard, S. Christensen and K. Jensen. CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets. In *Proceedings of the 24th International Conference on Applications and Theory of Petri Nets*, ICATPN'03, Berlin, Heidelberg, 2003. Springer-Verlag, p. 450–462.

[SFGM18] V. Skydanienko, C. D. Francescomarino, C. Ghidini and F. M. Maggi. A Tool for Generating Event Logs from Multi-Perspective Declare Models. In *BPM*, 2018.

[ShMi14] I. Shugurov and A. Mitsyuk. Generation of a Set of Event Logs with Noise, 01 2014.

[StAc13] T. Stocker and R. Accorsi. SecSy: Synthesizing process event logs. *Lecture Notes in Informatics (LNI), Proceedings - Series of the Gesellschaft fur Informatik (GI)*, 01 2013, p. 71–84.

[ThAm15] W. J. Thong and M. Ameedeen. A Survey of Petri Net Tools. *Lecture Notes in Electrical Engineering* volume 315, 01 2015, p. 537–551.

[vBVB14] S. vanden Broucke, J. Vanthienen and B. Baesens. Straightforward Petri Net-based Event Log Generation in ProM. *SSRN Electronic Journal*, 01 2014.

[vdAa11] W. van der Aalst. *Process mining : discovery, conformance and enhancement of business processes.* Springer, Germany. 2011.

[vdAa16] W. van der Aalst. *Process Mining.* Springer-Verlag Berlin Heidelberg. 2016.

[WeKi03] M. Weber and E. Kindler. The Petri Net Markup Language. In H. Ehrig, W. Reisig, G. Rozenberg and H. Weber (eds.), *Petri Net Technology for Communication-Based Systems: Advances in Petri Nets*, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg, p. 124–144.

[WKLS18] P. Wiśniewski, K. Kluza, A. Ligęza and A. Suchenia. Generation of Synthetic Business Process Traces using Constraint Programming. In *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*, 09 2018, p. 445–453.

[YaBS18] F. Yasmin, F. Bukhsh and P. Silva. Process enhancement in process mining: A literature review. *CEUR workshop proceedings* volume 2270, December 2018, p. 65–72. 8th International Symposium on

Data-driven Process Discovery and Analysis 2018, SIMPDA 2018 ; Conference date: 13-12-2018 Through 14-12-2018.

A. Relevant Literature Works

This appendix contains information about the papers that were examined in detail during the literature review described in section 3.2. Table A.1 list these papers and provides information about the type of approach they propose. Additionally, URLs by which the papers can be accessed are provided through footnotes. All URLs were last accessed on the 15th of April 2022.

Table A.1: Relevant literature that was found and examined in detail beyond just their title and abstract during the literature review

Author, Title, Year	**Approach**
Li et al., *A novel completeness definition of event logs and corresponding generation algorithm*, 2020[1]	Coverability tree simulation
Chesani et al., *Abduction for Generating Synthetic Traces*, 2018[2]	Abduction
Kummer et al., *An Extensible Editor and Simulation Engine for Petri Nets: Renew*, 2004[3]	Token-based simulation
Jensen et al., *Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems*, 2007[4]	Token-based simulation with CPNs
Mitsyuk et al., *Generating event logs for high-level process models*, 2017[5]	Token-based simulation
Ciccio et al., *Generating Event Logs Through the Simulation of Declare Models*, 2015[6]	Finite State Automata simulation
Nakatumba et al., *Generating Event Logs with Workload-Dependent Speeds from Simulation Models*, 2012[7]	Token-based simulation with CPNs

1. https://doi.org/10.1111/exsy.12529
2. http://dx.doi.org/10.1007/978-3-319-74030-0_11
3. https://doi.org/10.1007/978-3-540-27793-4_29
4. https://doi.org/10.1007/s10009-007-0038-x
5. https://doi.org/10.1016/j.simpat.2017.01.003
6. https://doi.org/10.1007/978-3-319-24626-0_2
7. http://dx.doi.org/10.1007/978-3-642-31069-0_31

Loreti et al., *Generating synthetic positive and negative business process traces through abduction*, 2020[8]	Abduction
Wiśniewski et al., *Generation of Synthetic Business Process Traces Using Constraint Programming*, 2018[9]	Constraint satisfactory problem
Alves De Medeiros an Günther, *Process Mining: Using CPN Tools to Create Test Logs for Mining Algorithms*, 2005[10]	Token-based simulation
Nugroho Yahya et al., *RT-PLG: Real Time Process Log Generator*, 2016[11]	Simulation through the use of a complex event processing engine
Ackermann et al., *Simulation of Multi-perspective Declarative Process Models*, 2016[12]	Simulation of declarative models
Van Hee et al., *Yasper: a tool for workflow modeling and analysis*, 2006[13]	Token-based simulation
Extended Search	
Skydanienko et al., *A Tool for Generating Event Logs from Multi-Perspective Declare Models*, 2018[14]	Boolean satisfiability problem
Kataeva and Kalenkova, *Applying Graph Grammars for the Generation of Process Models and Their Logs*, 2014[15]	Token-based simulation based on graph grammars
Shugurov and Mitsyuk, *Generation of a Set of Event Logs with Noise*, 2014[16]	Token-based simulation
Ackermann and Schönig, *MuDePS: Multi-perspective Declarative Process Simulation*, 2016[17]	Simulation of declarative models
Dingle et al., *PIPE2: A Tool for the Performance Evaluation of Generalised Stochastic Petri Nets*, 2009[18]	No approach for generating event logs based on process models

8. https://link.springer.com/article/10.1007/s10115-019-01372-z
9. https://www.researchgate.net/publication/327893296_Generation_of_Synthetic_Business_Process_Traces_using_Constraint_Programming
10. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.60.460
11. https://doi.org/10.1007/978-3-319-28564-1_11
12. https://doi.org/10.1007/978-3-319-58457-7_5
13. http://dx.doi.org/10.1109/ACSD.2006.37
14. https://www.semanticscholar.org/paper/A-Tool-for-Generating-Event-Logs-from-Declare-Skydanienko-Francescomarino/c8ef759bf2817770411ab780267f94d2333307e6
15. https://www.semanticscholar.org/paper/Applying-Graph-Grammars-for-the-Generation-of-and-Kataeva-Kalenkova/7bb0d820a6072a8b54ebe16a2f0ca14e265c0d0b
16. http://dx.doi.org/10.15514/SYRCOSE-2014-8-13
17. https://www.semanticscholar.org/paper/MuDePS%3A-Multi-perspective-Declarative-Process-Ackermann-Sch%C3%B6nig/4a7ed32981c9f7f7adf8ebf47faaa7f9b2c99e14
18. https://doi.org/10.1145/1530873.1530881

Burattin, *PLG2: Multiperspective Processes Randomization and Simulation for Online and Offline Settings*, 2015[19]	Token-based simulation
Van der Aalst, *Process Mining and Simulation: A Match Made in Heaven*, 2018[20]	Simulation through model play-out
Esgin and Karagoz, *Process Profiling based Synthetic Event Log Generation*, 2019[21]	Token-based simulation
Jouck and Depaire, *PTandLogGenerator: a Generator for Artificial Event Data*, 2016[22]	Simulation of artificial process trees
Stocker and Accorsi, *SecSy: Synthesizing Process Event Logs*, 2013[23]	Simulation
Pertsukhov and Mitsyuk, *Simulating Petri Nets with Inhibitor and Reset Arcs*, 2019[24]	Token-based simulation
Vanden Broucke et al., *Straightforward Petri Net-based Event Log Generation in ProM*, 2014[25]	Token-based simulation
Dash et al., *Synthetic Event Time Series Health Data Generation*, 2019[26]	No approach for generating event logs based on process models

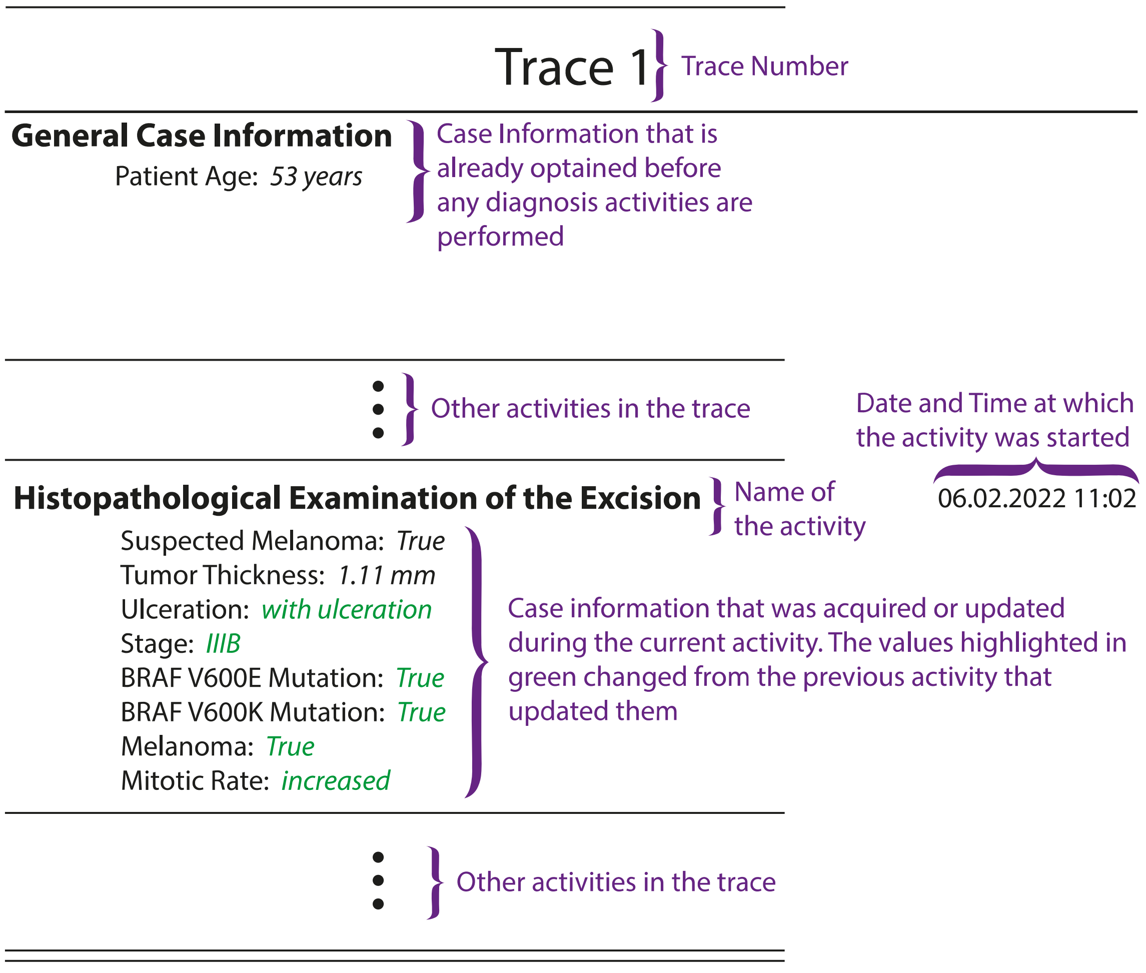
19. https://www.semanticscholar.org/paper/PLG2%3A-Multiperspective-Processes-Randomization-and-Burattin/c1b185ed69d03c1c398aa96c126118f9e5ac3f6b
20. http://dx.doi.org/10.22360/summersim.2018.scsc.005
21. http://dx.doi.org/10.5220/0008363805160524
22. https://www.researchgate.net/publication/313443120_PTandLogGenerator_a_Generator_for_Artificial_Event_Data
23. https://subs.emis.de/LNI/Proceedings/Proceedings222/71.pdf
24. http://dx.doi.org/10.15514/ISPRAS-2019-31(4)-10
25. http://dx.doi.org/10.2139/ssrn.2489051
26. https://www.researchgate.net/publication/337322739_Synthetic_Event_Time_Series_Health_Data_Generation

B. Synthetic Traces for the Evaluation

This appendix contains the traces used to evaluate the realism of the synthetic data that DALG can generate. These traces were presented to Dr Braun to evaluate if DALG can generate realistic event logs. The evaluation step in which these traces were used is described in section 6.4.3. First, a legend explains the structure of the visualisation used to present the traces to Dr Braun. Subsequently, the five traces are presented.

Note: The Timestamp of events in the traces are in the following format:
`day.month.year hour:minute (24 hour time format)`

Legend



Trace 1

General Case Information
Patient Age: *53 years*

Histopathological Examination of the Excision 06.02.2022 11:02
Suspected Melanoma: *True*
Tumor Thickness: *1.11 mm*
Ulceration: *with ulceration*
Stage: *IIIB*
BRAF V600E Mutation: *True*
BRAF V600K Mutation: *True*
Melanoma: *True*
Mitotic Rate: *increased*

Trace Number
Case Information that is already optained before any diagnosis activities are performed
Other activities in the trace
Date and Time at which the activity was started
Name of the activity
Case information that was acquired or updated during the current activity. The values highlighted in green changed from the previous activity that updated them
Other activities in the trace

Trace 1

General Case Information

Patient Age: *61 years*

First Clinical Examination	05.03.2022 06:48
Suspected Melanoma: *True*	
Excision (of the entire lesion)	07.03.2022 04:35
Histopathological Examination of the Excision	07.03.2022 06:22
Tumor Thickness: *0.83 mm*	
Ulceration: *without ulceration*	
Stage: *IA*	
BRAF V600E Mutation: *False*	
BRAF V600K Mutation: *True*	
Melanoma: *True*	
Mitotic Rate: *normal*	
Second Clinical Examination	10.03.2022 04:51
Suspected Metastases: *False*	
Re-Excision (additional tissue was removed)	17.03.2022 01:55
Histopathological Examination of the Re-Excision	17.03.2022 03:29
Residual Tumor Thickness: *0.0 mm*	
Resection Status: *R0*	

Trace 2

General Case Information

Patient Age: *71 years*

First Clinical Examination Suspected Melanoma: *True*	19.03.2022 16:11
Excision (of the entire lesion)	22.03.2022 19:19
Histopathological Examination of the Excision Tumor Thickness: *2.34 mm* Ulceration: *with ulceration* Stage: *IIB* BRAF V600E Mutation: *True* BRAF V600K Mutation: *True* Melanoma: *True* Mitotic Rate: *normal*	22.03.2022 21:08
Second Clinical Examination Suspected Metastases: *False*	24.03.2022 07:18
Sentinel Lymph Node Biopsy	31.03.2022 02:26
Re-Excision (additional tissue was removed)	31.03.2022 04:26
Histopathological Examination of the SLNB Stage: *IIB* BRAF V600E Mutation: *True* BRAF V600K Mutation: *False* Maximum Metastasis Diameter: *0.0 mm* Single Cells in the Sentinel Lymph Node: *True* Resection Status: *R0*	03.04.2022 07:19
Lymph Node Sonography Stage: *IIB* Suspected Metastases: *False*	08.04.2022 04:06
Adjuvant Interferon Therapy / Drug Study	17.04.2022 07:55

Trace 3

General Case Information

Patient Age: *65 years*

First Clinical Examination 17.04.2022 07:55

Suspected Melanoma: *True*

Excision (of the entire lesion) 19.04.2022 23:06

Histopathological Examination of the Excision 20.04.2022 00:49

Tumor Thickness: *1.09 mm*
Ulceration: *without ulceration*
Stage: *IB*
BRAF V600E Mutation: *True*
BRAF V600K Mutation: *True*
Melanoma: *True*
Mitotic Rate: *increased*

Second Clinical Examination 23.04.2022 04:39

Suspected Metastases: *False*

Sentinel Lymph Node Biopsy 01.05.2022 01:58

Re-Excision (additional tissue was removed) 01.05.2022 03:19

Histopathological Examination of the SLNB 02.05.2022 21:16

Stage: *IB*
BRAF V600E Mutation: *False*
BRAF V600K Mutation: *False*
Maximum Metastasis Diameter: *0.0 mm*
Single Cells in the Sentinel Lymph Node: *True*
Resection Status: *R1*

Lymph Node Sonography 05.05.2022 23:01

Stage: *IB*
Suspected Metastases: *False*

Trace 4

General Case Information

Patient Age: *68 years*

First Clinical Examination 13.05.2022 03:26

Suspected Melanoma: *True*

Excision (of the entire lesion) 15.05.2022 10:47

Histopathological Examination of the Excision 15.05.2022 12:04

Tumor Thickness: *4.74 mm*
Ulceration: *without ulceration*
Stage: *IIB*
BRAF V600E Mutation: *False*
BRAF V600K Mutation: *True*
Melanoma: *True*
Mitotic Rate: *increased*

Second Clinical Examination 16.05.2022 12:36

Suspected Metastases: *False*

Re-Excision (additional tissue was removed) 26.05.2022 08:34

Sentinel Lymph Node Biopsy 26.05.2022 10:26

Histopathological Examination of the SLNB 27.05.2022 14:18

Stage: *IIB*
BRAF V600E Mutation: *True*
BRAF V600K Mutation: *False*
Maximum Metastasis Diameter: *0.0 mm*
Single Cells in the Sentinel Lymph Node: *True*
Resection Status: *R0*

Lymph Node Sonography 31.05.2022 01:29

Stage: *IIB*
Suspected Metastases: *True*

Diagnostics to Confirm Metastases 04.06.2022 00:02

Stage: *IIIB*
Metastasis Confirmed: *True*

Tumor Marker Lactate Dehydrogenase Examination 11.06.2022 03:11

Tumor Marker S100B examination 12.06.2022 23:28

Nuclear Magnetic Resonance Imaging Head 13.06.2022 00:44

Stage: *IIIA*

PET/CT or CT Thorax/Abdomen 13.06.2022 03:22

Stage: *IIIB*

Anti-PD1 Antibodies Treatment 19.06.2022 20:50

Trace 5

General Case Information

Patient Age: *49 years*

First Clinical Examination 02.03.2022 08:00

Suspected Melanoma: *True*

Excision (of the entire lesion) 03.03.2022 19:20

Histopathological Examination of the Excision 03.03.2022 22:42

Tumor Thickness: *1.9 mm*
Ulceration: *without ulceration*
Stage: *IB*
BRAF V600E Mutation: *True*
BRAF V600K Mutation: *True*
Melanoma: *True*
Mitotic Rate: *normal*

Second Clinical Examination 07.03.2022 12:03

Suspected Metastases: *False*

Sentinel Lymph Node Biopsy 16.03.2022 01:28

Re-Excision (additional tissue was removed) 16.03.2022 03:42

Histopathological Examination of the SLNB 18.03.2022 08:19

Stage: *IB*
BRAF V600E Mutation: *True*
BRAF V600K Mutation: *False*
Maximum Metastasis Diameter: *0.0 mm*
Single Cells in the Sentinel Lymph Node: *True*
Resection Status: *R0*

Lymph Node Sonography 22.03.2022 20:43

Stage: *IB*
Suspected Metastases: *False*

Tumor Marker S100B examination 24.03.2022 18:19

Tumor Marker S100B Value: *increased*

PET/CT/NMRI 24.03.2022 20:02

Stage: *IB*
Only Suspected Peripheral Lymph Node Metastases: *True*

Diagnostics to Confirm Metastases 26.03.2022 06:21

Stage: *IIIC*
Metastasis Confirmed: *True*

Tumor Marker S100B examination 04.04.2022 10:49

Tumor Marker Lactate Dehydrogenase Examination 04.04.2022 13:18

PET/CT or CT Thorax/Abdomen 04.04.2022 15:59

Stage: *IIIB*

Nuclear Magnetic Resonance Imaging Head 06.04.2022 06:35

Stage: *IIIC*

BRAF and MEK Inhibitor (Drug Treatment) 12.04.2022 10:27