2015-11-01

Integrating Process Mining with Discrete-Event Simulation Modeling

Siyao Liu
Brigham Young University - Provo

Integrating Process Mining with Discrete-Event Simulation Modeling

Siyao Tony Liu

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

School of Technology

Brigham Young University

November 2015

ABSTRACT

Integrating Process Mining with Discrete-Event Simulation Modeling

Siyao Tony Liu
School of Technology, BYU
Master of Science

Discrete-event simulation (DES) is an invaluable tool which organizations can use to help better understand, diagnose, and optimize their operational processes. Studies have shown that for the typical DES exercise, the greatest amount of time is spent on developing an accurate model of the process that is to be studied. Process mining, a similar field of study, focuses on using historical data stored in software databases to accurate recreate and analyze business processes. Utilizing process mining techniques to help rapidly develop DES models can drastically reduce the amount of time spent building simulation models, which ultimately will enable organizations to more quickly identify and correct shortcomings in their operations.

Although there have been significant advances in process mining research, there are still several issues with current process mining methods which prevent them from seeing widespread industry adoption. One such issue, which this study examines, is the lack of cross-compatibility between process mining tools and other process analysis tools. Specifically, this study develops and characterizes a method through which mined process models can be converted into discrete-event simulation models. The developed method utilizes a plugin written for the ProM Framework, an existing collection of process mining tools, which takes a mined process model as its input and outputs an Excel workbook which provides the process data in a format more easily read by DES packages.

Two event logs which mimic real-world processes were used in the development and validation of the plugin. The developed plugin successfully extracted the critical process data from the mined process model and converted it into a format more easily utilized by DES packages. There are several limitations which will limit model accuracy, but the plugin developed by this study shows that the conversion of process models to basic simulation models is possible. Future research can focus on addressing the limitations to improve model accuracy.

Keywords: Siyao Tony Liu, process mining, discrete-event simulation, ProM, ProModel

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

1.1 Background

Simulation is defined as the imitation of the operation of a real-world process or system over time. In simple terms, simulation allows its users to construct a virtual model of a real-world process or system for the purpose of observing and analyzing certain phenomena within that process or system. Simulation is used across a wide variety of industries and disciplines, including engineering, healthcare, food service, supply chain management, economics, finance, and many other fields. It is an extremely versatile tool which, when used properly, can assist a great deal in validating, modifying, and debugging an organization's processes.

Two main types of simulation are continuous simulation and discrete-event simulation. Continuous simulation models are useful when a system's state varies continuously with time. Examples of this are weather system models or fluid models used in engineering analysis. By contrast, discrete-event simulation models are useful when a system's state changes as certain events occur over time. An example of this is a warehouse, where events such as inventory coming in or going out change the state of the system and can be measured using discrete moments in time. Between the two methods, discrete-event simulation is better suited to analyze business and operational processes due to their inherently task-based (discrete) nature. As a result, discrete-event simulation has seen more widespread use throughout industry, as its potential applications are much broader.

Modern discrete-event simulation frameworks have granted users the flexibility to build highly complex models, containing many different types of variables which can very precisely mirror the attributes of a real-world system. Given enough data and computational power, a user could conceivably build a model which incorporates every possible element of a system and simulates its behavior.

The question of how complex a model should be is a difficult one to answer. Typically, when building a simulation model, the modeler must spend a significant amount of time gathering the appropriate data to accurately represent the system which he/she is modeling. Such data could include things like cycle time for different operations, arrival intervals, capacity constraints, etc. In many organizations, obtaining this data can be a cumbersome activity. Onggo and Hill identify numerous case studies and issues with data collection in simulation projects.[1] Additionally, Trybula asserts that typical modelers will spend up to 40% of the total project time on data gathering and validation.[2] Modelers collect simulation data by either combing through volumes of historical information and piecing the data together, or by physically standing by each operation and recording the necessary data in real-time as it occurs. In order to more accurately represent the process, the modeler must collect several large, independent samples which can be extremely time consuming using either method. This manner of data collection is also very prone to errors and different types of bias.

This reality creates an obvious challenge for modelers who want to simulate real-world processes as accurately as possible while still doing so in a timely manner. While simulation models with higher complexity can provide more accurate and useful information, the

[1] B.S.S. Onggo and J. Hill, "Data Identification and Data Collection Methods in Simulation: A Case Study at Orh Ltd," *Journal of Simulation* 8, no. 3 (2014).

[2] W. J. Trybula, *Building Simulation Models without Data*, vol. 1, *Systems, Man, and Cybernetics, 1994. Humans, Information and Technology., 1994 IEEE International Conference on* (1994).

additional time and resources it takes to develop these models often can outweigh the value gained from building them. In other words, modelers often reach diminishing returns in their efforts very quickly.

Enterprise Resource Planning (ERP) systems developed by companies such as SAP, Oracle, and IBM have enabled businesses of all sizes to record and manage the performance of any aspect of their operations. With the ubiquity of information systems today, organizations are able to collect more data from their different processes than they can fully interpret. These huge caches of data contain valuable information that, if used properly, can provide great insights into how these organizations' processes perform.

Extensive research has been done in both academia and industry to help organizations better utilize this data. Collectively, this body of research is known as Process Mining. Process mining has been described as "the missing link between model-based process analysis and data-oriented analysis techniques."[3] In other words, process mining allows for the analysis of real-world business processes by utilizing data mining techniques. With process mining, organizations can now more optimally utilize their collected data in a sensible way to better understand the underlying processes which generate that data.

1.2 Problem Statement & Research Objective

Although there have been significant advances in process mining research, there are still several issues with current process mining methods which prevent them from seeing

[3] W.M.P. van der Aalst, "Process Mining - Data Science in Action," accessed March 26, 2015. http://www.tue.nl/en/university/departments/mathematics-and-computer-science/research/research-institutes/data-science-center-eindhoven-dsce/news/14-10-2014-mooc-by-wil-vd-aalst-process-mining-data-science-in-action/.

widespread industry adoption. One such issue, which this study will examine, is the lack of cross-compatibility between process mining tools and other process analysis tools.[4]

Process mining tools have the ability to quickly characterize operational processes using historical data, yet there is not an easy way to broadly export this data in a way that is interpretable by other tools (such as simulation tools). Similarly, simulation tools are excellent for creating predictive and prescriptive models of existing processes, yet there is not an easy way for these tools to quickly replicate an existing process. Thus, by enabling the compatibility of these two types of tools, the advantages of each can be more easily leveraged and applied to real-world processes (Figure 1).

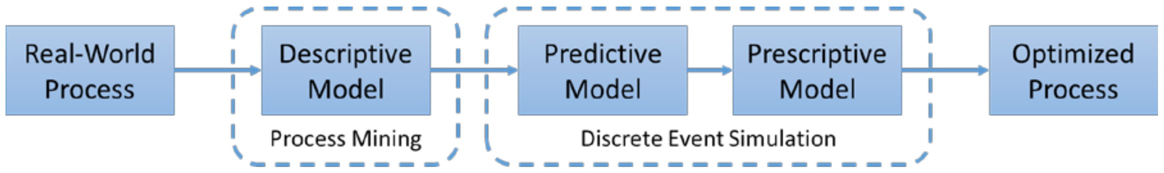


Figure 1: Process Optimization

With these benefits in mind, the primary objective of this research is to develop and characterize one method which would allow for mined process data to be more widely compatible with existing discrete-event simulation tools. Greater external compatibility would allow modelers to more easily develop simulation models from mined processes, which will ultimately help process mining techniques gain greater acceptance in real-world applications. Having this capability enables organizations to use the data that they already collect in a more meaningful way. Process mining coupled with discrete-event simulation is a powerful

[4] W.M.P. van der Aalst, "Process Mining Manifesto" (paper presented at the BPM 2011 Workshops, Clermont-Ferrand, France2012).

combination which would allow organizations to more rapidly and accurately validate and optimize their operational processes.

1.3 Research Scope

The purpose of this research is to establish a basic understanding of the possibilities of cross-compatibility between process mining and discrete-event simulation. As such, this research will primarily focus on discovering and characterizing one specific method which can be used to facilitate the integration of existing process mining tools into discrete-event simulation tools. The method discovered by this research in no way attempts to be a comprehensive tool which covers all usage scenarios. Nor is it intended to be very user friendly. It will, however, help in building a greater understanding of the limitations and challenges of translating a process mining framework into a more generalized discrete-event simulation framework. Specifically, this research project will attempt to disprove the hypothesis that a generalized discrete-event simulation model cannot be automatically created from a mined process. Ultimately, the results of this research will help determine the viability of using process mining to create general simulation models, under what sort of circumstances it can be applied, and how accurate such a model is compared to the real-world process that it mimics.

1.4 Outline

The remainder of this thesis is structured as follows: Chapter 2 will provide a review of literature of relevant research in the areas of discrete-event simulation and process mining. Key concepts and conventions within these two disciplines will be discussed here in order to provide the reader with the appropriate background knowledge. Chapter 3 will cover the methodology. Chapter 4 will reveal the results of the research and provide an analysis of the effectiveness of the developed method. Finally, Chapter 5 will provide a recap of the research objective and what was accomplished, as well as pose any lingering questions which may guide future research efforts.

2 LITERATURE REVIEW

2.1 Process Modeling Notation

2.1.1 The LEAP Framework

On a conceptual level, all simulation models generally work the same way. The most basic simulation model consists of just four elements:

Locations – Areas within the system where work-in-process units may be located (e.g., a waiting room or an examination room at a hospital)

Entities – The actual units being worked on by the system (e.g., the patients in a hospital)

Arrivals – The time intervals at which new entities enter into the system (e.g., a new patient enters the hospital every 30 minutes)

Processes – The time it takes to process entities at each location, also known as *Timing* (e.g., how long it takes to draw blood or perform a physical examination), and the way entities move between locations, also known as *routing* (e.g., a patient goes from the waiting room to an exam room)

These four fundamental building blocks make up the LEAP framework.[5] While different simulation packages may have different names for each of these building blocks, the general concept remains the same. In addition, most commercially available simulation packages build upon the LEAP concept to add more robust functionality in their models, however the most basic of simulation models will simply contain these four elements described by LEAP.

[5] C.R. Harrell, *Simulation Using Promodel* (McGraw-Hill Education, 2011).

2.1.2 Petri Nets

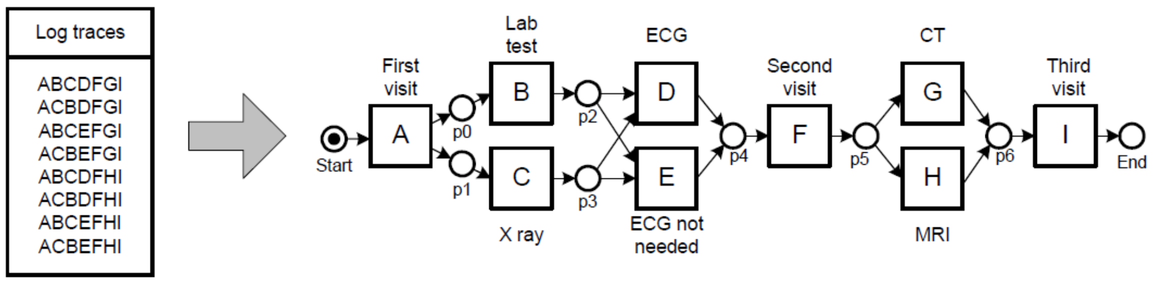


Figure 2: Example Petri Net Model

In the world of process mining research, Petri nets are the prevailing standard by which processes are modelled. Originally described in 1962, Petri nets were introduced as a way to visually depict distributed systems. A Petri net consists of four components: places (circles), transitions (rectangles), arcs (arrows), and tokens (dots). The places represent the possible "states" within a system, the transitions are "events" which cause a change of state, arcs denote the flow between places and transitions, and tokens can be thought of as entities within the system.[6] Rozinat et. al. show an example of a Petri net representing the process studied in their paper (Figure 2).[7]

The generalized nature of Petri nets makes them applicable across a variety of applications. However, the Petri net paradigm of places, transitions, arcs, and tokens can be difficult for casual users to understand intuitively, particularly when applied to physical systems. For example, Figure 2 above is a Petri net depicting the process flow within a hospital system. Thinking of the token as a patient, it can be difficult for users unfamiliar with Petri nets

[6] C.A. Petri and W. Reisig, "Petri Net," *Scholarpedia* 3, no. 4 (2008), http://dx.doi.org/10.4249/scholarpedia.6477.

[7] A. Rozinat et al., "Discovering Simulation Models," *Information Systems* 34, no. 3 (2009).

to understand that after the ‘First visit’, the patient will go to *both* ‘Lab test’ and ‘X ray’. At an intuitive level, it appears that the patient must do *either* ‘Lab test’ or ‘X ray’. Similarly, after the ‘Second visit’ the patient must go to *either* ‘CT’ or ‘MRI’ but not both. The minor subtlety of the place markers after ‘First visit’ and ‘Second visit’ can be lost on many casual users. Figure 3, combined with routing rules, is how most DES software packages would depict the process in Figure 2. This type of model is more intuitive to understand and allows for casual users to more quickly pick up and start modelling real-world processes.

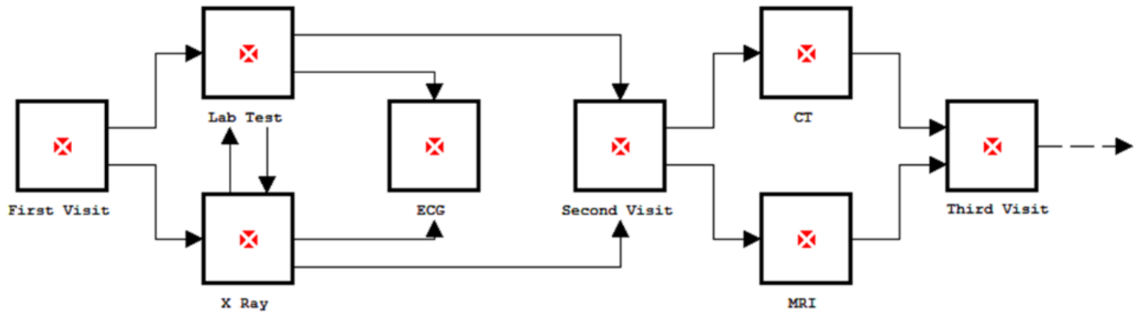


Figure 3: Intuitive Process Model

2.1.3 Colored Petri Nets

Another limitation of Petri nets is their lack of expressiveness to describe additional process information such as timing, resources, and routing rules. Suppose a modeler only had the Petri net in Figure 2 as a reference upon which to build a model of the hospital system. While they could certainly construct the control flow shown in Figure 3, they would lack all the other important information needed to determine routing rules, arrival frequency, processing times, etc.

Collectively named high-level Petri nets, the original Petri net concept has been extended with additional features to address this limitation.[8] These features include timing, hierarchy, and “color”, which allows tokens to contain differentiating data. While technically a misnomer, high-level Petri nets featuring these extensions are generally referred to as Colored Petri nets, due to their use in the CPN Tools[9] software package. Figure 4 below shows the Petri net model from Figure 2 expressed as a Colored Petri net.[10]

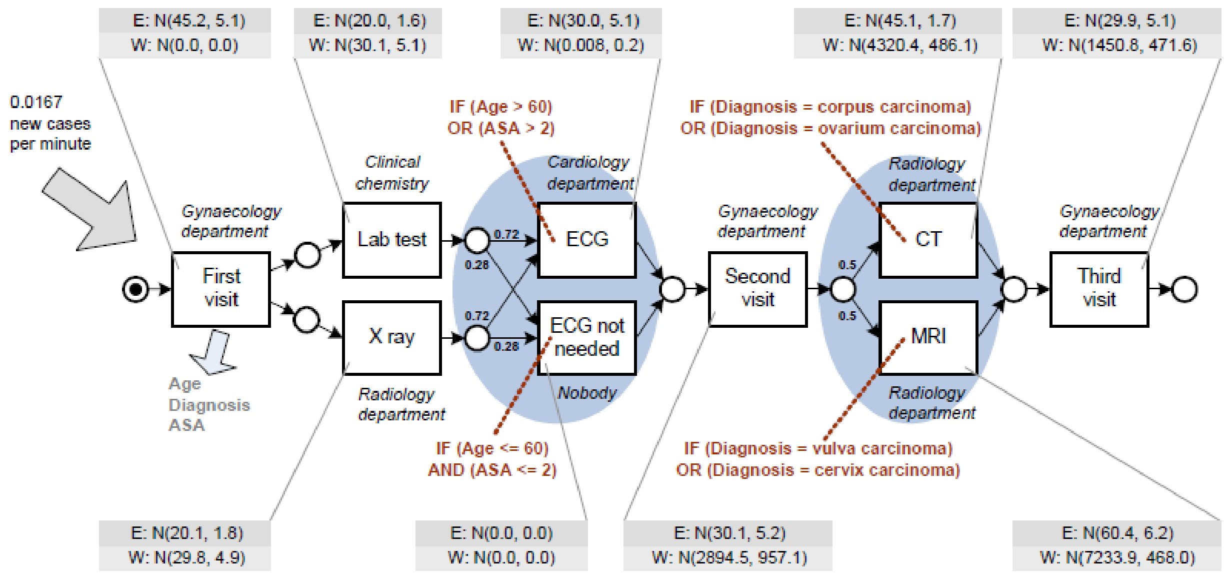


Figure 4: Example Color Petri Net Model

Under the Colored Petri net concept, the original Petri net now begins looking a lot more like a LEAP model, suitable for simulation. The transitions can be thought of as **Locations**, the tokens with their associated data are **Entities**, the **Arrivals** are labeled at the start of the process, and **Processing** times are labelled on each transition. In addition to this

[8] K. Jensen, *High-Level Petri Nets* (Springer, 1983).

[9] M. Westergaard and H.M.W Verbeek, "Cpn Tools Homepage," Eindhoven University of Technology, accessed April 21, 2015. cpntools.org.

[10] Rozinat et al., "Discovering Simulation Models."

basic LEAP information, the Colored Petri net model in Figure 4 also contains routing rules and resource information. This example shows that the Colored Petri net model contains enough data to build a simulation model.

2.2 Discrete-Event Simulation

Discrete-event simulation (DES) is a type of simulation which models a real-world system by updating different state variables which describe that system at discrete moments in time when certain events occur. One possible example of a state variable within a manufacturing system could be the number of work-in-process units that are currently being processed at a particular operation within the system. An event is simply something which triggers the next set of state variable changes, such as the completion of a manufacturing operation or the arrival of a work-in-process unit into the system.

2.2.1 Control Flow

Following the LEAP framework detailed in Section 2.1.1, the first component of developing a DES model is identifying the locations and routing between those locations. This is known as a Control Flow. In Figure 5, six unique locations, A through F, have been identified and are represented by the six boxes in the diagram. The arrows between the boxes dictate where entities may move between the locations. Often, the routings between locations are dictated by rules, where entities will be routed differently based on certain criteria (e.g., painted units are routed to the paint shop while unpainted parts are routed straight to final assembly).

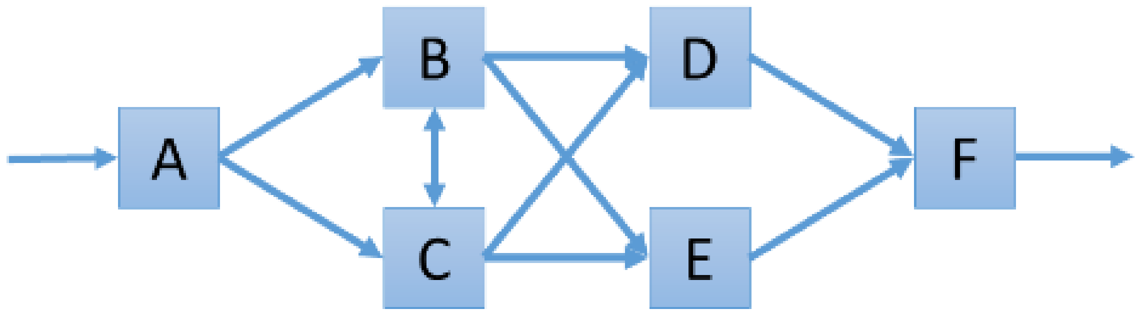


Figure 5: Simple Control Flow

While determining control flow may seem, on its surface, like a relatively straightforward task for a modeler, the reality is that there are often many routing details which are difficult to determine during the data collection process. For example, consider a manufacturing system with several processing steps. At each step, there is a possibility for errors to occur, which results in the entity-in-process to be routed to a 'Rework' location. To determine the probability of entities routed to 'Rework', the modeler will likely need to pore through significant amounts of historical data; a time-consuming and repetitive process.

2.2.2 Event Scheduling and Timing

Using a LEAP model as an input, the simulation software performs the simulation by following the next-event time advance approach, which consists of the following phases[11]:

Step 1: The simulation clock is initialized to zero and the times of occurrence of future events are determined.

Step 2: The simulation clock is advanced to the time of the occurrence of the most imminent (i.e., first) of the future events.

Step 3: The state of the system is updated to account for the fact that an event has occurred.

[11] P. Sloot, "1 Introduction to Simulation and Modeling," accessed July 13, 2014. http://artemis.wszib.edu.pl/~sloot/1_4.html.

Step 4: Knowledge of the times of occurrence of future events is updated and the first step is repeated.

Because of the inherent variability present in real-world systems, the timing of activities is generally defined using probabilistic distributions. In order for these distributions to most accurately characterize the performance of a process, two important conditions must be met: (1) the correct *type* of distribution has been selected, and (2) the selected distribution has been defined correctly.

For example, it is common to see the timing of a manufacturing operation described using a normal distribution with a defined mean and standard deviation. In this case, most manufacturing operations meet the criteria for the central limit theorem, which means modelers can be fairly confident that the first condition is met. The second condition, however, is more difficult to confirm. In order to increase confidence in the mean and standard deviation, numerous samples of the real-world processing time must be taken, which can take a significant amount of time.

2.3 Process Mining

2.3.1 Overview

Process Mining is a specialized branch of data mining which aims to extract information about business processes using data generated from those processes.[12] The output of process mining techniques is known as a process model. Simply put, a process model is "a graphical representation of a business process that describes the dependencies between activities that need to be executed collectively for realizing a specific business objective. It

[12] N. Gehrke and M. Werner, *Process Mining* (Hamburg, Germany: University of Hamburg, 2013).

consists of a set of activity models and constraints between them."[13] Visually, a process model can look very similar to a DES model, but the process model will generally not be interactive.

A simple event log is shown below in Table 1. The ultimate objective of process mining is to use the data contained within an event log such as this and develop a process model as shown in Figure 6.

Table 1: Sample Event Log

Case ID	Event ID	Timestamp	Description
10	2000	8:05:12 AM	Receive Raw Materials
10	2001	8:10:22 AM	Punching
10	2002	8:12:05 AM	Grinding
10	2003	8:20:45 AM	Machining
10	2004	8:45:36 AM	Move to Finished Goods Inventory
20	2005	8:09:55 AM	Receive Raw Materials
20	2006	8:15:34 AM	Punching
20	2007	8:17:01 AM	Grinding
20	2008	8:25:40 AM	Machining
20	2009	8:49:57 AM	Move to Finished Goods Inventory

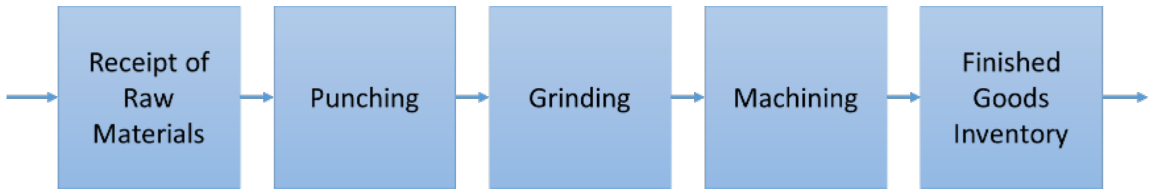


Figure 6: Simple Manufacturing Process

The event log shown in Table 1 features four headings and is representative of the most basic type of data that can be used to mine a process. The Case ID tracks one entity as it goes

[13] M. Weske, *Business Process Management: Concepts, Languages, Architectures* (New York: Springer, 2012).

through the process, while the Event ID is a unique identifier for each instance of a processing step. Additionally, using the Timestamp data and the Description, the processing time between each step of the process can be inferred. Assuming 'Receipt of Raw Materials' is the first step in the process, the frequency of arrival of each new entity can also be determined.

2.3.2 Mining Procedure

The procedure for process mining follows four basic steps:

Step 1: Data Extraction
Step 2: Data Filtering and Loading
Step 3: Data Mining and Reconstruction
Step 4: Analysis

While the first three steps are extremely important in achieving high quality process models, this research primarily focuses on the improving the analytical capability of mined models. Therefore, this research will assume the input data for mining purposes has already been properly extracted, filtered, and can be mined with great accuracy, thus placing the primary focus on Step 4. By enabling the use of discrete-event simulation tools on mined process models, the analytical capability of these models can increase dramatically.

2.3.3 Related Work

Rozinat et al describe a basic case demonstrating a method through which simulation models can be discovered through process mining.[14] In their paper, the authors use a number of existing process mining techniques on a sample dataset to discover three different process models: a timing model, an organizational model, and decision model. These three models

[14] Rozinat et al., "Discovering Simulation Models."

represent three distinct “views” of the process. The three process models are then combined into a comprehensive simulation model.

Following the steps outlined in Section 2.3.2, their research begins by first gathering the event log data which reflects the medical examination process at a major European hospital. This data was extracted in MXML format, which is a widely used standard in process mining research. It was then filtered and standardized to facilitate the mining process.

After extracting and filtering the data, the authors then began mining the data using different algorithms to discover useful information about the process. The first algorithm discovered control-flow, which automatically created a process model detailing the relationships between each of the activities in the event log. Next, a decision point analysis was performed which discovered the routing logic between activities. The third algorithm performed a performance analysis, which calculated the processing times, waiting times, and alternative routing probabilities. Lastly, the role discovery algorithm grouped common resources into specific roles and associated them to particular activities within the process.

Upon completion of the previous steps, the modeler now has enough data to synthesize a basic simulation model. In their research, Rozinat et al represented the simulation model using a Coloured Petri Net (CPN) due to its compatibility with the CPN Tools software. CPN Tools is an open source software designed for “editing, simulating, and analyzing Colored Petri Nets”.[15] While this tools works well for demonstration purposes, commercial DES software such as ProModel provides much greater control and accuracy over open source tools like CPN Tools. The research performed by Rozinat et al keeps all of the mining, simulation, and analysis contained within the ProM Framework, a specialized suite of process mining tools (not

[15] Westergaard and Verbeek, "Cpn Tools Homepage."

to be confused with ProModel). This research differs primarily in its decoupling of the mining process from the simulation and analysis process. This approach should allow for better analysis of models by way of greater flexibility in examining several different “what-if” scenarios.

3 METHODOLOGY

Based on the research objective, the methodology employed in this study can be summarized into two main phases: Development and Conformance Checking. As discussed previously in this paper, a new plugin for the ProM software must be developed (Section 3.1) to export the mined process information in a format readable by the third-party discrete-event simulation software. Then, the output obtained from the plugin will be compared against a manually developed simulation model of the same process in order to characterize the benefits and drawbacks of the developed plugin (Section 3.2).

3.1 Plugin Development

The development of the LEAP export plugin builds upon the method presented by Rozinat et al.[16] As described above in Section 2.3.3, Dr. Rozinat and her co-authors gave an example of one potential method through which a simulation model could be automatically discovered using existing process mining tools built into the ProM 5.2[17] software. In the two last steps of their process, the authors of the paper combined several perspectives mined from the process into a consolidated model using the *Merge Simulation Models* plugin, then exported the consolidated model to a Colored Petri Net (CPN) model using the *CPN Export* plugin.[18]

[16] Rozinat et al., "Discovering Simulation Models."

[17] Process Mining Group, *Prom*, 5.2 ed. (Eindhoven, Netherlands: Eindhoven University of Technology, 2010).

[18] A. Rozinat et al., "Discovering Colored Petri Nets from Event Logs," *International Journal on Software Tools for Technology Transfer* 10, no. 1 (January 2008).

The resultant output from this process is a *.CPN file with data formatted in an XML-type format. This file format is highly specific to the CPN Tools software[19] and is generally unreadable by other DES software packages. The new plugin developed in this study will take the place of the CPN Export plugin to export the consolidated model to a more general format which can then be used more easily by other DES packages. The dashed line in Figure 7 below shows how the method proposed in this study deviates from the original method used by Rozinat et al.[20]

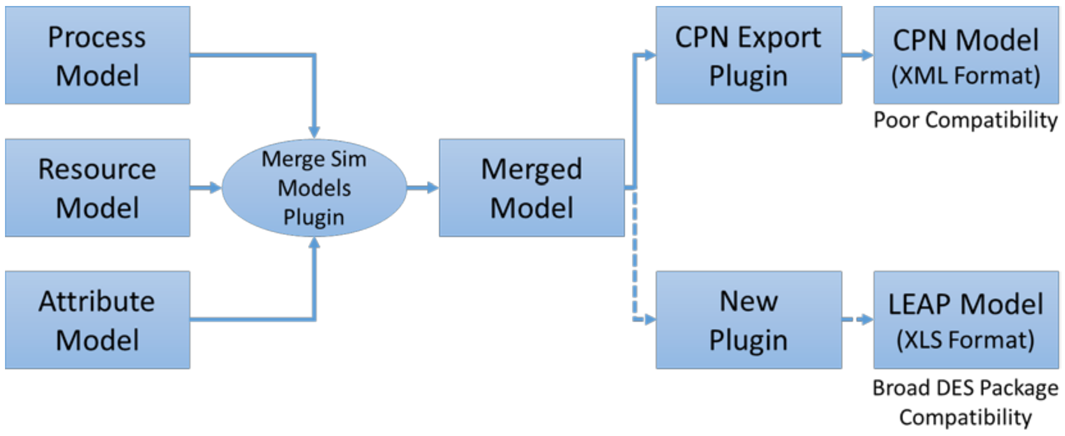


Figure 7: Simulation Model Development Process

For this study, ProModel Corporation's ProModel 2014[21] was selected as a reference DES package upon which all testing would occur. To develop the plugin, a suitable data output format had first to be selected. The selected output displays the process data in an easily readable tabular format and is similar in structure to the format given in ProModel's ProActiveX spreadsheet[22] due to its ease of use and its flexibility. The output file is a Microsoft

[19] Westergaard and Verbeek, "Cpn Tools Homepage."
[20] Rozinat et al., "Discovering Simulation Models."
[21] ProModel Corporation, *Promodel*, 9.1.0.1639 ed., vol. 2014 (Orem, UT: ProModel Corporation, 2014).
[22] ProModel Corporation, "Proactivex," ProModel Corporation, accessed May 13, 2015.

Excel Binary File Format (*.xls).[23] Using Excel allows for the data to be more easily manipulated and integrated with third-party DES software packages such as ProModel through the use of its built in VBA programming language.

With the output format established, the plugin could then be built. Within the ProM 5.2 framework, all plugins are written in the Java language and follow a standard implementation format to ensure compatibility with the ProM 5.2 software. This format is included with the ProM 5.2 documentation.[24] Using this standard implementation format, a new plugin was built and named *ProModel Export*, taking the consolidated model as an input and gives a standardized tabular data format as its output. The resulting plugin and its limitations are discussed further in Section 3.2.1 of this paper.

3.2 Conformance Checking

Once the plugin was developed and tested for basic functionality, it was applied to sample data in order to compare its output against a manually developed simulation model. For this study, two event logs were used: a log from an outpatient clinic[25] and a log from an insurance company.[26] Both of these logs are artificially generated logs based on real-world processes and have been used as examples in other Process Mining studies. Therefore, these logs should provide a reasonable baseline of how the plugin will function under limited real-world circumstances. Many of the limitations and benefits of the plugin will become evident through this process of comparing the output of the plugin against the manually developed

[23] Microsoft Corporation, "Microsoft Office Excel 97-2007 Binary File Format (.Xls) Specification," (2007).

[24] P. van den Brand, "Implementing and Integrating Plugins in the Process Mining Framework," (2004).

[25] A. Rozinat et al., "Outpatient Clinic Example," Eindhoven University of Technology, accessed 2015. http://www.processmining.org/_media/documentation/cpnexport/outpatientclinicexample.mxml.gz.

[26] A. Rozinat et al., "Insurance Company Example," Eindhoven University of Technology, accessed 2015. http://www.processmining.org/_media/documentation/cpnexport/insuranceclaimexample.mxml.gz.

simulation models. By characterizing these limitations and assessing the benefits, the objectives of the study will be met.

3.2.1 Case Studies

The Outpatient Clinic case used in this study is the same case study used by Rozinat et al in their study.[27] Rozinat et al note in their paper that this case has been artificially generated based on a real-life process of the AMC hospital in the Netherlands. Figure 8 below shows the process flow represented as a Petri net. One can see from this figure that this Outpatient Clinic process features several branched routings and represents how a medical clinic could realistically work.

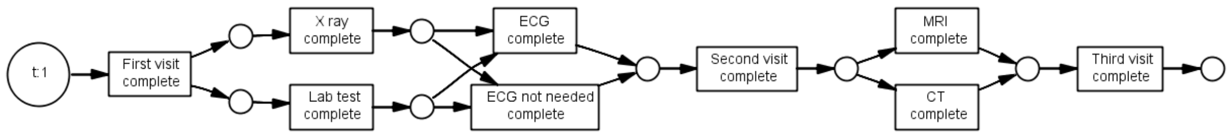
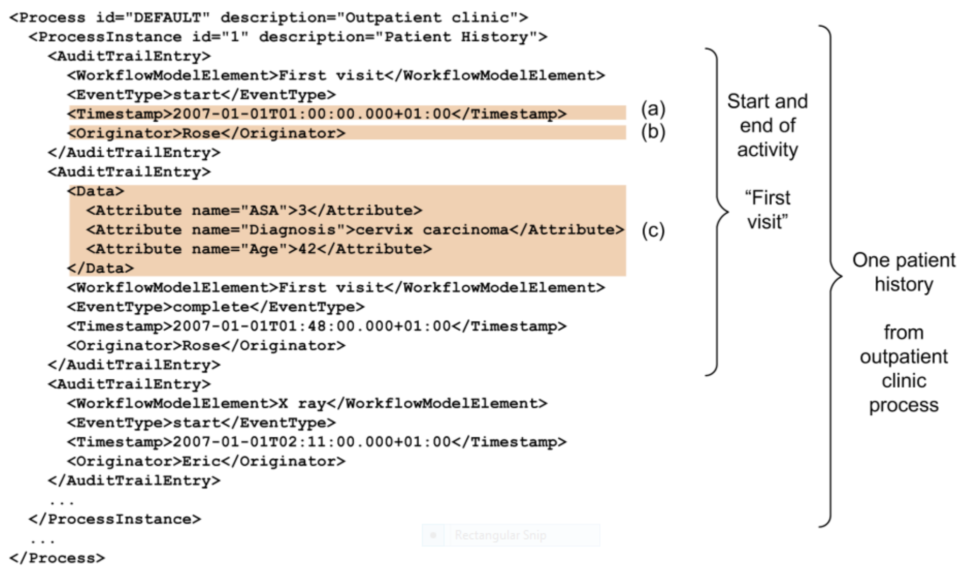


Figure 8: Outpatient Clinic Process

Figure 9 below shows an excerpt of the event log representing one patient's processing history though this clinic. Note that several pieces of information critical to defining the process can be found within this log. A *timestamp* (a) allows for the miner to determine the sequence of activities as well as the duration of each activity. The *originator* (b) tag tells the miner which resource(s) acted during each activity. The *attributes* (c) allow the miner to differentiate between different types of entities within the process, enabling alternative routing.

[27] Rozinat et al., "Outpatient Clinic Example."



```
<Process id="DEFAULT" description="Outpatient clinic">
  <ProcessInstance id="1" description="Patient History">
    <AuditTrailEntry>
      <WorkflowModelElement>First visit</WorkflowModelElement>
      <EventType>start</EventType>
      <Timestamp>2007-01-01T01:00:00.000+01:00</Timestamp>
      <Originator>Rose</Originator>
    </AuditTrailEntry>
    <AuditTrailEntry>
      <Data>
        <Attribute name="ASA">3</Attribute>
        <Attribute name="Diagnosis">cervix carcinoma</Attribute>
        <Attribute name="Age">42</Attribute>
      </Data>
      <WorkflowModelElement>First visit</WorkflowModelElement>
      <EventType>complete</EventType>
      <Timestamp>2007-01-01T01:48:00.000+01:00</Timestamp>
      <Originator>Rose</Originator>
    </AuditTrailEntry>
    <AuditTrailEntry>
      <WorkflowModelElement>X ray</WorkflowModelElement>
      <EventType>start</EventType>
      <Timestamp>2007-01-01T02:11:00.000+01:00</Timestamp>
      <Originator>Eric</Originator>
    </AuditTrailEntry>
    ...
  </ProcessInstance>
  ...
</Process>
```

(a)
(b)
(c)
Start and end of activity
"First visit"
One patient history
from outpatient clinic process

Figure 9: Outpatient Clinic Log Excerpt

The Insurance Company case used in this study is a case which was also used by Rozinat et al in earlier publications.[28] This case is an artificially generated case which shows how a claim would be processed at a hypothetical insurance company. This process is shown below in Figure 10. Similar to the Outpatient Clinic case above, this case also features branched routings and decision points which closely mimic real-world processes.

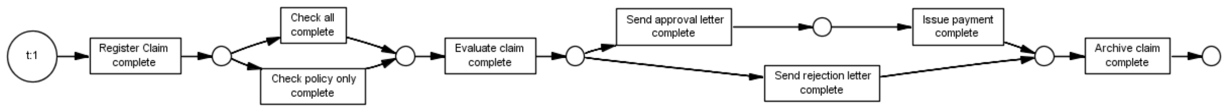


Figure 10: Insurance Company Claim Process

[28] A. Rozinat and W.M.P. van der Aalst, *Decision Mining in Prom* (Springer, 2006).

4 RESULTS AND ANALYSIS

As discussed above in Section 3.1, a new export plugin for ProM 5.2 was developed for this study. This section will review the key functionality of the plugin, discuss the major issues and obstacles encountered when developing the plugin, and describe the potential use cases of the method developed herein.

4.1 Plugin Functionality and Results

The primary assumption behind this plugin is that the user has already created a consolidated process model using the steps outlined by Rozinat et al.[29] The input model must be of type 'HLPetriNet'. The plugin takes this model as an input and provides an output in the Microsoft Excel (.xls) format. To perform this conversion, the developed plugin (named *ProModel Expor*) uses existing objects from within the ProM framework to read and manipulate the data needed to produce the correct output. The following sections will review specifically how the conversion is performed for each element of LEAP, as well as discuss any major issues or obstacles surrounding this conversion.

4.1.1 Arrivals Module

To configure a DES model's arrival scheme, two basic pieces of data are needed: one or more arrival locations and the frequency of arrival at those locations. To obtain this data, the

[29] Rozinat et al., "Discovering Simulation Models."

starting node(s) of the model are first extracted from the HLPetriNet input model using the 'getStartNodes()' function, which is an existing function built into the HLPetriNet object. This function returns an array containing all of the starting nodes in a given HLPetriNet. The developed plugin simply iterates through this array to obtain all of the arrival location names.

Next, the rate of arrival must be determined. The HLPetriNet model contains a distribution function which can be extracted using the 'getCaseGenerationScheme()' function. Once extracted, the distribution function object is then fed into another function within the developed plugin which identifies the type of distribution and its parameters, then outputs the distribution formula in a way which can be read by ProModel. Table 2 and Table 3 below show the arrivals output of the two test cases.

Table 2: Arrivals Output (Clinic Case)

Location	Frequency
First visit complete	N(3583.0631,3506.2535)

Table 3: Arrivals Output (Insurance Case)

Location	Frequency
Register Claim complete	N(2304.0000,3451.5909)

While testing the arrivals functionality, it was discovered that the two test cases were insufficient to test the plugin's ability to handle arrivals at multiple locations. Therefore, a simple test case was constructed to test this functionality. Figure 11 below shows this simple process, which contains only location and routing data. Running this test case through the *ProModel Export* plugin yielded the result shown in Table 4 and Table 5.

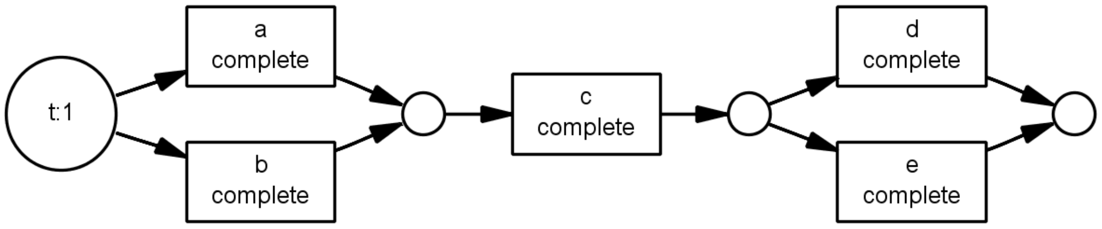


Figure 11: Multiple Arrival Test Case

Table 4: Multiple Arrival Test – Arrival Table Result

Location	Frequency
entryDummy	N(0.0000,0.0000)

Table 5: Multiple Arrival Test - Processing Table Result

Activities	Processing	Resource	Routing	Routing Rules
entryDummy			b complete	
			a complete	
b complete	0.0000	minedGroup0	c complete	
c complete	0.0000	minedGroup0	e complete	
			d complete	
e complete	0.0000	minedGroup0	EXIT	
d complete	0.0000	minedGroup0	EXIT	
a complete	0.0000	minedGroup0	c complete	

In order to address multiple arrival locations, a workaround was developed which entered an ‘entryDummy’ location as the first arrival, then places the actual arrival location into the routing table as processing locations. This allows for the built-in case generation scheme to remain intact, using routing rules to correctly reflect the actual arrival intervals at each location. While this result is cosmetically different than a LEAP model, the resulting performance is the same.

4.1.2 Resources Module

Resource information is extracted from the HLPetriNet model using the built in 'getGroups()' function. 'getGroups()' returns an array of workgroups discovered by the organizational miner plugin. Each of these workgroups represent a set of resources which perform similar tasks. By finding these workgroups, the component resources can be extracted as well. Once discovered, the developed plugin writes the workgroups and their component resources to a new sheet within the Excel workbook. ProModel can then input the list of resources and assign them to workgroups via its Macros function. Resource information retrieved from both cases are shown in Table 6 and Table 7.

Table 6: Resources Output (Clinic Case)

Group	Resource
minedGroup3	Jan
	Martin
	Rose
	Vanessa
minedGroup2	Fred
	Vic
	Wilma
minedGroup1	Claire
	Jo
	Valentine
minedGroup0	Alex
	Eric
	Jane
	Maria
	Nigel
	Ralph

Table 7: Resources Output (Insurance Case)

Group	Resource
minedGroup2	John
	Mona
	Robert
minedGroup1	Howard
	Vincent
minedGroup0	Fred
	Linda

To achieve the desired conversion output, two minor differences had to be addressed. The first difference was with null resource groups. In cases where there are processes which require no resource interaction, a workgroup with a member named ‘nobody’ will be created and added to that process. For example, in the Outpatient Clinic case, the ‘ECG not needed’ activity has no processing time and exists simply as a placeholder for routing purposes. Despite this, the organizational miner still assigned a workgroup (minedGroup4) to that process, with the only member of that workgroup being ‘nobody’. This behavior can create problems with DES software as the ‘nobody’ resource will be viewed as a valid resource and used as such. To correct this, the export plugin simply ignores all resources named ‘nobody’.

The second difference was with combined resource groups. In the Insurance Company case, there existed several activities which resources from multiple workgroups could work on. To show this relationship, the high level Petri net model records this information as a concatenation of multiple workgroups in plain text, delimited by colons (‘:’). While there is nothing logically incorrect about this, it creates a syntactic problem with ProModel as well as clutters the model with unnecessary elements. To correct this, all such groups are omitted from the final output of the export plugin and all references to combined resource groups in the ‘Processing’ module are converted to a ‘minedGroup0 OR minedGroup2’ format which is functionally equivalent for modelling purposes.

4.1.3 Attributes Module

Within the ProM process mining tools, all of the cases in an event log are thought of as a single entity type. To differentiate one from another, each entity is assigned a number of attributes. For example, in the Outpatient Clinic case, all of the entities can be thought of as ‘Patient’ entities, with each one being assigned a ‘Diagnosis’ attribute which helps to differentiate how each is routed through the system.

To extract the attribute information from the HLPetriNet model, the 'getAttributes()' function is run, which returns an array of HLAttribute objects. Using an iterative process, each attribute is extracted and written to a new spreadsheet within the Excel workbook. Table 8 and Table 9 show the completed attribute output of the two test cases.

Table 8: Attributes Output (Clinic Case)

Attributes	Frequency
Age	U(55.0000,35.0000)
ASA	U(2.5000,1.5000)
Diagnosis	DiagnosisDist()

Table 9: Attributes Output (Insurance Case)

Attributes	Frequency
PolicyType	PolicyTypeDist()
Status	StatusDist()
CustomerID	CustomerIDDist()
Amount	U(525.0000,475.0000)

In addition to the attributes themselves, each attribute's frequency of occurrence must also be extracted from the process model. Within the high level Petri net model, these frequency distributions can either be numeric or nominal. The extraction of numeric distributions is relatively straightforward – the export plugin programmatically determines the type of distribution and its parameters, then outputs them in a format that is readable by ProModel. Such is the case in Table 8 with the 'Age' and 'ASA' frequencies, which both lie on a Uniform distribution.

Nominal distributions are slightly more complex and require more intermediate steps to successfully convert to a ProModel-friendly format. These distributions are expressed as a table of possible values with their frequencies of occurrence. Table 10 shows a nominal distribution for the 'Diagnosis' attribute in the Outpatient Clinic case. Because ProModel does not support qualitative attributes, exporting these nominal distributions requires a slight modification to the way the data is formatted. Specifically, each of the possible values from the nominal distribution table must first be assigned to a unique integer index number. Then, the frequency of occurrence for each value is calculated as a percent of total. The resultant converted output is shown in Table 11.

Table 10: Nominal Distribution Table (Clinic Case)

Value	Frequency
corpus_carcinoma	236
cervix_carcinoma	295
vulva_carcinoma	205
ovarium_carcinoma	264

Table 11: User Distribution Table (Clinic Case)

ID	Percentage	Index	Note
DiagnosisDist	23.60%	1	corpus_carcinoma
	29.50%	2	cervix_carcinoma
	20.50%	3	vulva_carcinoma
	26.40%	4	ovarium_carcinoma

4.1.4 Locations and Processing Module

Location and processing information is extracted similarly to how arrivals are extracted. The plugin first takes the extracted starting nodes and writes each subsequent node to the spreadsheet until no more nodes remain, then it moves back up the chain to determine if any alternative routings exist and iteratively writes each branch of the process until all locations are found. While writing each location, the plugin also writes the associated processing time distribution for each location, any resources used by the activity at that location, and the routing locations and rules. All of this information is extracted directly from the HLPetriNet model using existing functions. The resulting output of the two test cases is shown in Table 12 and Table 13.

Table 12: Processing Output (Clinic Case)

Activities	Processing	Resource	Routing	Routing Rules
First visit complete	N(2710.7400,303.2494)	minedGroup3	X ray complete	
			Lab test complete	
X ray complete	N(1206.3000,104.9062)	minedGroup0	ECG complete	((((ASA <= 2) AND (Age > 60))) OR (ASA > 2))
			ECG not needed complete	((ASA <= 2) AND (Age <= 60))
ECG complete	N(1798.1564,303.8217)	minedGroup2	Second visit complete	
Second visit complete	N(1805.9400,310.5195)	minedGroup3	CT complete	((Diagnosis == 1) OR (Diagnosis == 4))
			MRI complete	((Diagnosis == 3) OR (Diagnosis == 2))
CT complete	N(2705.1600,102.5359)	minedGroup0	Third visit complete	
Third visit complete	N(1790.9400,307.3877)	minedGroup3	EXIT	
MRI complete	N(3622.2000,371.1359)	minedGroup0	Third visit complete	
ECG not needed complete	N(0.0000,0.0000)		Second visit complete	
Lab test complete	N(1198.7400,97.5106)	minedGroup1	ECG complete	((((ASA <= 2) AND (Age > 60))) OR (ASA > 2))
			ECG not needed complete	((ASA <= 2) AND (Age <= 60))

Table 13: Processing Output (Insurance Case)

Activities	Processing	Resource	Routing	Routing Rules
Register Claim complete	N(950.0000,722.4126)	minedGroup2	Check all complete	((PolicyType == 2) OR (((PolicyType == 1) AND (Amount > 500))))
			Check policy only complete	((PolicyType == 1) AND (Amount <= 500))
Check all complete	N(1650.0000,1121.9626)	minedGroup0 OR minedGroup2	Evaluate claim complete	
Evaluate claim complete	N(1270.0000,638.8427)	minedGroup0	Send approval letter complete	Status == 2
			Send rejection letter complete	Status == 1
Send approval letter complete	N(1140.0000,103.9230)	minedGroup0 OR minedGroup2	Issue payment complete	
Issue payment complete	N(1120.0000,517.3007)	minedGroup1	Archive claim complete	
Archive claim complete	N(2460.0000,1076.6615)	minedGroup0 OR minedGroup2	EXIT	
Send rejection letter complete	N(540.0000,374.6999)	minedGroup0 OR minedGroup1 OR minedGroup2	Archive claim complete	
Check policy only complete	N(1200.0000,593.9697)	minedGroup1 OR minedGroup2	Evaluate claim complete	

The Locations and Processing module is the most complex of all the modules and reveals the greatest number of issues with converting a high level Petri net model to a ProModel model. The first issue is with distinguishing between parallel vs non-sequential routing. The Outpatient Clinic process contains an example of this issue. In this process (Figure 8), the ‘First visit’ activity is succeeded by ‘X ray’ and ‘Lab test’. These activities must both occur after ‘First visit’ before the process can proceed. However, it is unclear whether these processes should occur simultaneously or if they simply occur in an unordered fashion. Currently, there is no provision within the HLPetrinet object to distinguish between these two scenarios, which means that it is ultimately up to the modeler to understand the process and make this distinction on their own.

The next issue is with complex processing rules. Within high level Petri net models, entity attributes are only used to determine routing rules. However, in real-world processes, different entity types may be subject to different processing times as well. For example, it is certainly possible that an insurance claim for a greater amount would take longer for an insurance adjuster to evaluate than one for a lesser amount. Because there is no provision in the HLPetrinet object for processing time differentiation, all of the potential processing differences between entity types are simply lumped together into one general distribution which covers all entities. This issue can be somewhat addressed by segregating event logs by entity type and mining the process separately for each in order to receive differentiated timings. Unfortunately, this is a data collection issue and cannot be examined within the scope of this study.

The final issues are regarding a number of features specific to location and processing which cannot be included in the output model because the necessary data simply does not exist within the mined model. This includes features such as complex processing logic, detailed resource usage, and capacity planning.

Petri nets only provide simple process flow information and lack a mechanism to specify more complex processing logic. Using the Outpatient Clinic case as an example, this means that although the Petri net dictates that the ‘Lab test’ and ‘X ray’ activities occur at the same time, the actual real-world entity (the patient) can only physically be in one of the two locations at once and may move on to the next activity regardless of if ‘Lab test’ is complete or not. The fact that high level Petri nets lack this sort of logic means that simulation models derived from mined process models will lack this degree of model accuracy. Because this is an inherent limitation with current process mining tools, the only solution at the moment is for the modeler to be familiar with the process and be able to implement this logic on their own.

With regard to resource usage, high level Petri nets only specify which resource groups are used by which activities and lack a way to describe the logic behind how these resources are used. For example, in the Outpatient Clinic case, the mined process model shows that 'minedGroup0' is in charge of handling the 'X ray', 'ECG', and 'MRI' activities. However, there is no information regarding *how* each resource handles these activities. In other words, although the 'MRI' activity takes about 20 minutes to complete, it is possible that the resource from 'minedGroup0' is only present for the first few minutes to get the machine going and is then free to move on to another activity. For lack of better information, the plugin assumes that resources are bound to their activities for the entire processing duration.

Finally, because process mining deals only with analyzing historical data, design intent is not revealed through this process. This means that the theoretical capacity of each location cannot be included in a high level Petri net. For example, in the Insurance case, it is impossible to determine by simply looking at the event logs just how many claims can be processed in the 'Issue payment' activity. A small insurance company with only a few employees may only be able to issue one or two payments at once while a large insurance company may be able to issue hundreds. Therefore, due to this inherent lack of information, the plugin assumes infinite capacity at each location, which will allow for the greatest amount of analysis in the face of limited data.

4.2 Model Completeness

Given the results outlined in Section 4.1 above, a preliminary assessment of the resulting simulation models' completeness can be performed.

4.2.1 Outpatient Clinic Model

For the Outpatient Clinic case, the process model contained the necessary data to build the majority of the model automatically. The only area which required additional manual input was the sequencing of the 'X Ray Complete' and 'Lab Test Complete' operations (see Figure 8). Due to the inherent limitation of the HLPetriNet process model mentioned above in Section 4.1.4, these two operations are assumed to be non-ordered and sequential. The human modeler must be aware of this assumption and verify its accuracy against reality. In addition, to ensure proper functionality of the non-ordered and sequential operations, the modeler must manually input routing logic into the model which will ensure the entity passes through each operation only once.

In spite of this manual adjustment, the rest of the automatically generated model is functionally correct and accurately reflects the process as it is represented in the event log data.

4.2.2 Insurance Claim Model

In the Insurance Claim case, the results of the automatically generated simulation model are even more promising. Upon evaluation, it appears that this model can be used as-is from the plugin output.

In this particular case, the relative simplicity of the process allowed for the plugin to easily extract a fully robust model. This result is expected as this case is a more simple process which does not contain any complex routing or other logic.

4.3 General Limitations

Given the fact that the developed method relies heavily on process mining, it is naturally bound by the same limitations as other process mining methods. Such limitations include data limitations as well as the limitations associated with process mining tools.

4.3.1 Data

As data collection becomes more ubiquitous, the volume of collected data will naturally increase as well. With this increase in data volume, it becomes increasingly difficult to verify the accuracy and integrity of the collected data. The use of inaccurate data can create serious problems for modelers trying to get the most accurate picture of their processes.

Furthermore, there is the problem of data sufficiency. Despite the large amounts of data being collected, there is still the risk of missing key data points important to the DES process. For example, if a key attribute such as a patient's age was not recorded in the event logs, any process model built using that data would lose a significant degree of accuracy. Similarly, with greater granularity of data, more accurate models can be developed.

In both cases, whether the supplied data is inaccurate or insufficient, the plugin currently has no provision to alert the modeler to any inconsistencies. Therefore, it is up to the modeler to understand the process which they are modeling and learn to recognize potential errors in the automatically generated model. One potential method which can be used to diagnose data problems is to run the simulation model and check for the reasonableness of the output with respect to the real-world process's performance. Through this method, major data errors can be identified.

While these data limitations can severely hamper any automated DES modelling efforts, solving those issues go beyond the scope of this research. Therefore, it is best for modelers to simply be aware of the possibility of data limitations and to adjust their modelling practices accordingly.

4.3.2 Process Mining Tools

Process mining is an ever evolving field of study. As the field evolves, the software tools will naturally follow suit to reflect advancements in research. With changes in this field, there is always a risk of obsolescence for older software. While this risk certainly exists, the plugin developed in this paper relies on critical objects within the ProM framework which would cause extreme functionality issues with all existing plugins if removed or altered substantially. Furthermore, the petri net paradigm for modelling process has been in academic use for decades and is unlikely to change. Therefore, the risk of obsolescence for the developed plugin both syntactically and functionally is fairly small.

4.4 Analysis of Results

First and foremost, this study has confirmed that it is indeed possible to use process mining techniques to automate the building of DES models to a certain degree. It further confirms that the paradigm of process modelling using high level Petri nets is not wholly incompatible with the LEAP paradigm commonly used by DES software packages.

The results of this study suggest that modern process mining tools are excellent at organizing large amounts of event data into coherent, basic process models. Information such as location names, time distributions, routing logic, and basic resource usage can all be identified very quickly from real-world data.

As described in Section 4.2 above, the completeness of the generated simulation model depends heavily on the complexity of the real-world process. Processes with fewer complexities are more likely to require fewer alterations by human modelers. Likewise, processes with high complexity are expected to require greater human intervention. This reality is due to inherent limitations in the process mining framework hinder its ability to identify complex processing logic and other more detailed process data. Furthermore, it is important to note that the limitations discussed in Section 4.1 above are the result of these inherent limitations in the process model framework. Once these limitations are recognized at large and addressed within the process mining research community, the ability to automatically create robust models using the methodology described in this study will improve drastically. Therefore, this study has shown that the current generation of process mining tools have the ability to provide a quick baseline with which modelers can then further refine using their knowledge of the process. However, the automated generation of highly complex models relies on fundamental improvements in the high level Petri net framework. Overall, despite these shortcomings, the plugin developed in this study has tremendous potential to greatly reduce modeling lead time in the majority of scenarios

While this result is not the fully automated solution that many have hoped for, this study represents an important first step to recognizing the powerful potential that exists when process mining is applied to advanced DES technology. As described in the preceding sections, the successful development of a LEAP export plugin confers many of the benefits of process mining techniques to the world of discrete-event simulation. By tapping into the vast amount of data potentially available within process event logs, DES modelers can very quickly achieve an accurate baseline of the process which they are trying to model, which allows modelers to more rapidly realize the predictive and prescriptive benefits that DES has to offer. Furthermore, by relying on event logs, DES modelling through process mining techniques allows much of the model building process to be moved off site, opening up the possibility for lower cost outsourcing of simulation modelling exercises. These benefits result in lower costs, quicker lead times, and greater model accuracy.

5 CONCLUSIONS

5.1 Summary

Although both fields focus on helping organizations better understand their processes, process mining and discrete-event simulation represent different types of analysis. Process mining gives organizations insight into the current real-world condition of their processes while discrete-event simulation gives organizations the predictive and prescriptive capabilities to help optimize their processes. Put another way, process mining uses the past to reveal the present, while discrete-event simulation uses the present to help plan for the future. Therefore, there is a powerful synergy that can be achieved by combining the methods of process mining and discrete-event simulation. This study served as an exploratory first step into the possibility of bridging a functional gap between process mining and discrete-event simulation.

The primary purpose of this research was to first determine whether there were enough similarities between process mining models and DES models to use methods from the former to programmatically generate a model for the latter. Once it was determined that this was possible to a certain degree, the research characterized the developed method to discover its benefits and limitations.

To perform the study, a new export plugin named *ProModel Export* was developed for ProM 5.2.[30] The plugin takes as input a mined process model of type HLPetriNet and outputs an Excel workbook containing the critical process data in a format that is more easily read by

[30] Group, "Prom."

commercial DES packages. Through the successful creation of this plugin, the author of this study was able to confirm the possibility of using process mining methods in creating a DES compatible model and was also able to characterize the benefits and limitations thereof.

5.2 Future Work

As industries become more and more data driven, the ability of organizations to optimize their processes will become increasingly relevant. Therefore, the ability to quickly extract process information using process mining methods and the ability to analyze those processes using discrete-event simulation tools will be extremely valuable in the coming years.

Despite being in very early stage development, the *ProModel Export* plugin appears to be very promising in its extensibility. As the field of process mining evolves, so will the capabilities of process mining tools. Therefore, future work in this area should focus on further development of the plugin to improve its functionality by integrating these new tools. Specifically, possible avenues which could be explored include extending the plugin with tools which enhance model logic, using additional real-world event data to further validate and characterize the performance of this plugin, or extending the plugin to output the data in other formats more compatible with other commercial DES packages.

REFERENCES

Corporation, Microsoft. "Microsoft Office Excel 97-2007 Binary File Format (.Xls) Specification." (2007): 349.

Corporation, ProModel. "Proactivex." ProModel Corporation. Last modified 2014. Accessed May 13, 2015.

Corporation, ProModel. *Promodel*. Vol. 2014. 9.1.0.1639 ed. Orem, UT: ProModel Corporation, 2014.

Gehrke, N. and M. Werner. *Process Mining*. Hamburg, Germany: University of Hamburg, 2013.

Group, Process Mining. *Prom*. 5.2 ed. Eindhoven, Netherlands: Eindhoven University of Technology, 2010.

Harrell, C.R. *Simulation Using Promodel*. McGraw-Hill Education, 2011.

Jensen, K. *High-Level Petri Nets*. Springer, 1983.

Onggo, B.S.S. and J. Hill. "Data Identification and Data Collection Methods in Simulation: A Case Study at Orh Ltd." *Journal of Simulation* 8, no. 3 (2014): 11.

Petri, C.A. and W. Reisig. "Petri Net." *Scholarpedia* 3, no. 4 (2008): 6477. http://dx.doi.org/10.4249/scholarpedia.6477.

Rozinat, A., R.S. Mans, M. Song, and W.M.P. van der Aalst. "Discovering Colored Petri Nets from Event Logs." *International Journal on Software Tools for Technology Transfer* 10, no. 1 (January 2008): 18.

Rozinat, A., R.S. Mans, M. Song, and W.M.P. van der Aalst. "Discovering Simulation Models." *Information Systems* 34, no. 3 (2009): 23.

Rozinat, A., R.S. Mans, M. Song, and W.M.P. van der Aalst. "Insurance Company Example." Eindhoven University of Technology. Last modified 2009. Accessed 2015. http://www.processmining.org/_media/documentation/cpnexport/insuranceclaimexample.mxml.gz.

Rozinat, A., R.S. Mans, M. Song, and W.M.P. van der Aalst. "Outpatient Clinic Example." Eindhoven University of Technology. Last modified 2009. Accessed 2015. http://www.processmining.org/_media/documentation/cpnexport/outpatientclinicexample.mxml.gz.

Rozinat, A. and W.M.P. van der Aalst. *Decision Mining in Prom*. Springer, 2006.

Sloot, P. "1 Introduction to Simulation and Modeling." Last modified 2003. Accessed July 13, 2014. http://artemis.wszib.edu.pl/~sloot/1_4.html.

Trybula, W. J. *Building Simulation Models without Data*. Vol. 1. Systems, Man, and Cybernetics, 1994. Humans, Information and Technology., 1994 IEEE International Conference on, 1994.

van den Brand, P. "Implementing and Integrating Plugins in the Process Mining Framework." (2004): 24.

van der Aalst, W.M.P., "Process Mining Manifesto." BPM 2011 Workshops, Clermont-Ferrand, France, 2012.

van der Aalst, W.M.P. "Process Mining - Data Science in Action." Last modified 2014. Accessed March 26, 2015. http://www.tue.nl/en/university/departments/mathematics-and-computer-science/research/research-institutes/data-science-center-eindhoven-dsce/news/14-10-2014-mooc-by-wil-vd-aalst-process-mining-data-science-in-action/.

Weske, M. *Business Process Management: Concepts, Languages, Architectures*. New York: Springer, 2012.

Westergaard, M. and H.M.W Verbeek. "Cpn Tools Homepage." Eindhoven University of Technology. Last modified 2015. Accessed April 21, 2015. cpntools.org.

APPENDIX A. PROMODEL EXPORT PLUGIN SOURCE

```
package org.processmining.exporting.petrinet;

import java.io.IOException;
import java.io.OutputStream;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Set;
import java.util.StringTokenizer;

import jxl.*;
import jxl.format.UnderlineStyle;
import jxl.write.*;
import jxl.write.biff.RowsExceededException;

import org.processmining.exporting.ExportPlugin;
import org.processmining.framework.models.ModelGraph;
import org.processmining.framework.models.ModelGraphVertex;
import org.processmining.framework.models.hlprocess.HLActivity;
import org.processmining.framework.models.hlprocess.HLAttribute;
import org.processmining.framework.models.hlprocess.HLChoice;
import org.processmining.framework.models.hlprocess.HLCondition;
import org.processmining.framework.models.hlprocess.HLGlobal;
import org.processmining.framework.models.hlprocess.HLGroup;
import org.processmining.framework.models.hlprocess.HLProcess;
import org.processmining.framework.models.hlprocess.HLResource;
import org.processmining.framework.models.hlprocess.att.HLBooleanAttribute;
import org.processmining.framework.models.hlprocess.att.HLBooleanDistribution;
import org.processmining.framework.models.hlprocess.att.HLNominalAttribute;
import org.processmining.framework.models.hlprocess.att.HLNominalDistribution;
import org.processmining.framework.models.hlprocess.att.HLNumericAttribute;
import org.processmining.framework.models.hlprocess.distribution.HLDistribution;
import org.processmining.framework.models.hlprocess.distribution.HLDistribution.DistributionEnum;
import org.processmining.framework.models.hlprocess.distribution.HLGeneralDistribution;
import org.processmining.framework.models.hlprocess.hlmodel.HLPetriNet;
import org.processmining.framework.models.petrinet.PetriNet;
//import org.processmining.framework.models.petrinet.algorithms.CpnWriter;
import org.processmining.framework.plugin.ProvidedObject;

/**
 * Exports a given high level Petri net to an excel file.
 *
 * @see PetriNet
 * @see CustomWriter
 *
 * @author Tony Liu
 */
public class CustomExport implements ExportPlugin {
  /** the mapping from all activity nodes in the model to its belonging high level activity ID */
  private HashMap<ModelGraphVertex,String> vertexMapping;

  /**
   * Default constructor.
   */
  public CustomExport() {
  }

  /**
   * Specifies the name of the plug-in.
   * This is used for, e.g., labeling the corresponding menu item or the
   * user documentation page.
   * @return the name (and the supported version) of the exported file format
   */
  public String getName() {
    return "Excel Export Tools 1.5";
  }

  /**
   * Determines whether a given object can be exported as a Text object.
   * @param object the <code>ProvidedObject</code> which shall be tested for being
   * a valid input to this export plug-in
   * @return <code>true</code> if the given object is a <code>PetriNet</code>,
   * <code>false</code> otherwise
   */
  public boolean accepts(ProvidedObject object) {
    Object[] o = object.getObjects();

    for (int i = 0; i < o.length; i++) {
      if (o[i] instanceof HLPetriNet) {
        return true;
      }
    }
    return false;
  }

  /**
   * Triggers the actual export, that is writes the given object as a XML to
   * the given output stream.
   * @param object the <code>ProvidedObject</code> which shall be exported as a CPN
   * @param output the <code>OutputStream</code> specifying the target of the
   * exported file
   * @throws IOException in the case a problem is encountered while writing
   * the file
   */
  public void export(ProvidedObject object, OutputStream output) throws IOException {
    // Initialize source activity mapping
    vertexMapping = new HashMap<ModelGraphVertex,String>();

    Object[] o = object.getObjects();
    for (int i = 0; i < o.length; i++) {
      if (o[i] instanceof HLPetriNet) {
        // Create a new workbook
        WritableWorkbook workbook = Workbook.createWorkbook(output);

        // Write the high level petrinet to the specified excel file
        HLPetriNet net = (HLPetriNet) o[i];
        ModelGraph graph = net.getProcessModel();
        HashSet<ModelGraphVertex> startNodes = graph.getStartNodes();
        int row = 1;

        // Define header format
        WritableFont font = new WritableFont(WritableFont.TIMES, 10, WritableFont.BOLD, false,
UnderlineStyle.SINGLE);
          WritableCellFormat format = new WritableCellFormat(font);

          // Export arrivals
        WritableSheet sheet = workbook.createSheet("Arrivals", 0);
        writeFormat(sheet, 0, 0, "Location", format);
        writeFormat(sheet, 1, 0, "Frequency", format);
        row = writeArrivals(net, startNodes, sheet, row);

        // Export processing information
        sheet = workbook.createSheet("Processing", 1);

        // Write headers
        writeFormat(sheet, 0, 0, "Activities", format);
        writeFormat(sheet, 1, 0, "Processing", format);
        writeFormat(sheet, 2, 0, "Resource", format);
        writeFormat(sheet, 3, 0, "Routing", format);
        writeFormat(sheet, 4, 0, "Routing Rules", format);

        // Get the initial place
        row = 1;
        row = writeDummyEntry(net, startNodes, sheet, row);
        row = writeStartActivites(net, startNodes, sheet, row);

        row = 1;
        sheet = workbook.createSheet("Resources", 2);
        writeFormat(sheet, 0, 0, "Group", format);
        writeFormat(sheet, 1, 0, "Resource", format);
        List<HLGroup> groups = net.getHLProcess().getGroups();
        row = writeGroups(groups, sheet, row);

        row = 1;
        sheet = workbook.createSheet("Attributes", 3);
        writeFormat(sheet, 0, 0, "Attributes", format);
        writeFormat(sheet, 1, 0, "Frequency", format);
        Set<HLAttribute> attrs = net.getHLProcess().getAttributes();
        row = writeAttributes(attrs, sheet, row);

        row = 1;
        sheet = workbook.createSheet("UserDist", 4);
        writeFormat(sheet, 0, 0, "ID", format);
        writeFormat(sheet, 1, 0, "Percentage", format);
        writeFormat(sheet, 2, 0, "Index", format);
        writeFormat(sheet, 3, 0, "Note", format);
        row = writeUserDists(attrs, sheet, row);

        workbook.write();
        try {
          workbook.close();
        } catch (WriteException e) {
          e.printStackTrace();
        }
        return;
      }
    }
  }

  /**
   * Specifies the file extension for the exported file.
   * @return the file extension string, that is the part of a file name which
   * is following the "." (fileName.fileExtension)
   */
  public String getFileExtension() {
    return "xls";
  }

  /**
   * Provides user documentation for the plug-in.
   * @return The Html body of the documentation page.
   */
  public String getHtmlDescription() {
    return "This plug-in exports the High level Petri net model as an Excel file. " +
    "For generating an Excel file including simulation environment, logging monitors, and with various
configuration " +
    "possibilities use the Export plug-in 'Export to Excel Tools 1.0'.";
  }

  /**
   * Writes one cell to the specified work sheet with a format (font).
   *
   * @param sheet the current Excel work sheet
   * @param col column
   * @param row row
   * @param text label text
   * @param format cell format
   */
  private void writeFormat(WritableSheet sheet, int col, int row, String text, WritableCellFormat format) {
    Label label = new Label(col, row, labelHack(text), format);
    try {
      sheet.addCell(label);
    } catch (RowsExceededException ex) {
      ex.printStackTrace();
    } catch (WriteException ex) {
      ex.printStackTrace();
    }
  }

  /**
```

```
 * Writes one label cell to the specified work sheet.
 *
 * @param sheet the current Excel work sheet
 * @param col column
 * @param row row
 * @param text label text
 */
private void write(WritableSheet sheet, int col, int row, String text) {
  Label label = new Label(col, row, labelHack(text));
    jxl.write.Number number  =   new  jxl.write.Number( 1 ,  0 ,  555.12541 );
    try {
     sheet.addCell(label);
  } catch (RowsExceededException ex) {
     ex.printStackTrace();
  } catch (WriteException ex) {
     ex.printStackTrace();
  }
}

/**
 * Writes one integer cell to the specified work sheet.
 *
 * @param sheet the current Excel work sheet
 * @param col column
 * @param row row
 * @param text label text
 */
private void writeInteger(WritableSheet sheet, int col, int row, double val) {
  WritableCellFormat integerFormat = new WritableCellFormat (NumberFormats.INTEGER);
    jxl.write.Number number = new jxl.write.Number( col, row, val, integerFormat);
    try {
     sheet.addCell(number);
  } catch (RowsExceededException ex) {
     ex.printStackTrace();
  } catch (WriteException ex) {
     ex.printStackTrace();
  }
}

/**
 * Writes one integer cell to the specified work sheet.
 *
 * @param sheet the current Excel work sheet
 * @param col column
 * @param row row
 * @param text label text
 */
private void writePercentage(WritableSheet sheet, int col, int row, double val) {
  WritableCellFormat pctFormat = new WritableCellFormat (NumberFormats.PERCENT_FLOAT);
    jxl.write.Number number = new jxl.write.Number(col, row, val, pctFormat);
    try {
     sheet.addCell(number);
  } catch (RowsExceededException ex) {
     ex.printStackTrace();
  } catch (WriteException ex) {
     ex.printStackTrace();
    }
  }

  /**
   * Exports the specified destination activity.
   *
   * @param node the designated transition (activity) node to be exported
   * @param expr the associated routing rule
   * @param sheet the current Excel work sheet
   * @param i starting row in the Excel work sheet
   * @return the new row
   * @throws IOException
   */
  private int writeOneDestActivity(ModelGraphVertex node, String expr, WritableSheet sheet,
      int i) throws IOException
  {
    write(sheet, 3, i, node.getIdentifier());

    if (expr != null) {
      write(sheet, 4, i, expr);
    }

    i++;
    return i;
  }

  /**
   * Returns a string that represents a formula for the specified distribution.
   * @param the HLDistribution object
   * @return the string representation of the formula
   */
  private String distributionFormula(HLDistribution dist) {
    HLGeneralDistribution exec = (HLGeneralDistribution)dist;
    DistributionEnum distType = exec.getBestDistributionType();
    String formula = "";
    double v1, v2;
    switch (distType)
    {
    case CONSTANT_DISTRIBUTION:
      formula = String.format("%.4f", exec.getConstant());
      break;
    case NORMAL_DISTRIBUTION:
      formula = String.format("%c(%.4f,%.4f)", distType.toString().charAt(0), exec.getMean(),
Math.sqrt(exec.getVariance()));
      break;
    case UNIFORM_DISTRIBUTION:
      v1 = exec.getMin();
      v2 = exec.getMax();
      formula = String.format("%c(%.4f,%.4f)", distType.toString().charAt(0), (v1 + v2) / 2.0, (v2 - v1) / 2.0);
      break;
    case DISCRETE_DISTRIBUTION:
      formula = String.format("If BI(1, .5) = 1 Then { %.4f } Else { %.4f }", exec.getMin(), exec.getMax());
      break;
    case EXPONENTIAL_DISTRIBUTION:
      formula = String.format("%c(%.4f)", distType.toString().charAt(0), exec.getIntensity());
      break;
  case BERNOULLI_DISTRIBUTION:
    formula = String.format("BI(%d,%.4f)", 1, exec.getProbability());
    break;
  case BINOMIAL_DISTRIBUTION:
    formula = String.format("BI(%d,%.4f)", exec.getNumberExperiments(), exec.getProbability());
    break;
  case STUDENT_DISTRIBUTION:
    v1 = (double)exec.getDegreesOfFreedom();
    formula = String.format("N(0,%.4f)", Math.sqrt(v1 / (v1 - 2.0)));
    break;
  case CHISQUARE_DISTRIBUTION:
    formula = String.format("G(%.4f,2)", (double)exec.getDegreesOfFreedom() / 2.0);
    break;
  case ERLANG_DISTRIBUTION:
    formula = String.format("ER(%.4f,%d)", exec.getIntensity(), exec.getEmergenceofEvents());
    break;
  case POISSON_DISTRIBUTION:
    formula = String.format("%c(%.4f)", distType.toString().charAt(0), exec.getIntensity());
    break;
  default:
    break;
  }

  return formula;
}

/**
 * Exports the current node (transitions).
 *
 * @param net the set of attributes
 * @param node the designated transition (activity) node
 * @param sheet the current Excel work sheet
 * @param i starting row in the Excel work sheet
 * @return the new row
 * @throws IOException
 */
private int writeOneSourceActivity(HLPetriNet net, ModelGraphVertex node, WritableSheet sheet,
    int i) throws IOException
{
  // Check to see if the source activity is already exported
  if (vertexMapping.get(node) != null)
    return i;

  // Write the activity to the file as the source activity
  write(sheet, 0, i, node.getIdentifier());

  // Write the resource group this activity belongs
  HLActivity activity = net.findActivity(node);
  HLGroup group = activity.getGroup();
  if (group != null && !emptyGroup(group)) {
    String groupName = group.getName().replaceAll(":", " OR ");
    write(sheet, 2, i, groupName);
  }

  // Write processing (distribution)
  HLDistribution dist = activity.getExecutionTime();
```

```
    if (dist != null) {
      if (dist instanceof HLGeneralDistribution)
      {
        String distText = distributionFormula(dist);
        write(sheet, 1, i, distText);
      }
    }

    // Add to the already exported list
    vertexMapping.put(node, node.getIdentifier());

    // Export all immediate successors as target activities (routing destinations)
    Iterator iter = node.getSuccessors().iterator();

    while (iter.hasNext()) {
      i = writeSuccessors(net, (ModelGraphVertex)iter.next(), sheet, i);
    }

    // Recursively export all successors as source activities
    i = writeStartActivites(net, node.getSuccessors(), sheet, i);

    return i;
  }

  /**
   * Exports all successors (transitions) as the target transitions from the current place node.
   *
   * @param net the set of attributes
   * @param place the designated place node
   * @param sheet the current Excel work sheet
   * @param i starting row in the Excel work sheet
   * @return the new row
   * @throws IOException
   */
  private int writeSuccessors(HLPetriNet net, ModelGraphVertex place, WritableSheet sheet,
       int i) throws IOException
  {
    Iterator iter = place.getSuccessors().iterator();

    // If it's the end place, write EXIT as the target and increment the row counter
    if (!iter.hasNext()) {
      write(sheet, 3, i++, "EXIT");
    }

    while (iter.hasNext()) {
      ModelGraphVertex transition = (ModelGraphVertex)iter.next();
      HLActivity activity = net.findActivity(transition);
      HLChoice choice = net.findChoice(place);
      String expr = null;

      if (choice != null) {
        HLCondition cond = choice.getCondition(activity.getID());
        expr = cond.getExpression().toString();
      }
```

```
        expr = hackExpr(net, expr);
        i = writeOneDestActivity(transition, expr, sheet, i);
    }

    return i;
}

private String hackExpr(HLPetriNet net, String expr) {
    if (expr == null)
        return null;

    StringTokenizer tokens = new StringTokenizer(expr, "()<>= ", false);

    while(tokens.hasMoreTokens()) {
        String token = tokens.nextToken();
        Integer index = valueToIndex(net, token);

        if (index > 0) {
            expr = expr.replaceAll(token, index.toString());
        }
    }

    return expr;
}

private Integer valueToIndex(HLPetriNet net, String token) {
    Set<HLAttribute> attrs = net.getHLProcess().getAttributes();
    Iterator<HLAttribute> e = attrs.iterator();

    while (e.hasNext()) {
        HLAttribute el = (HLAttribute) e.next();

        if (el instanceof HLNominalAttribute) {
            HLNominalAttribute att = (HLNominalAttribute)el;
            HLNominalDistribution dist = att.getPossibleValues();
            HashMap<String, Integer> map = dist.getValuesAndFrequencies();
            int index = 1;
            for (String val : map.keySet()) {
                if (val.equals(token)) {
                    return index;
                }

                index++;
            }
        }
    }

    return 0;
}

/**
 * Exports all attributes of the high level Petri net.
 *
 * @param net the set of attributes
 * @param nodes the set of initial source nodes (places)
 * @param sheet the current Excel work sheet
 * @param i starting row in the Excel work sheet
 * @return the new row
 * @throws IOException
 */
private int writeStartActivites(HLPetriNet net, HashSet<ModelGraphVertex> nodes, WritableSheet sheet,
     int i) throws IOException
{
  // Iterator to enumerate all starting places
  Iterator<ModelGraphVertex> it = nodes.iterator();

  // first, enumerate the nodes on this level
  while (it.hasNext()) {
    ModelGraphVertex n = it.next();

    // Get actual transitions from the current place
    Iterator iter = n.getSuccessors().iterator();

    while (iter.hasNext()) {
      i = writeOneSourceActivity(net, (ModelGraphVertex)iter.next(), sheet, i);
    }
  }

  return i;
}

private boolean emptyGroup(HLGroup group) {
  List<HLResource> resources = group.getResources();
  Iterator<HLResource> r = resources.iterator();
  if (!r.hasNext())
    return true;

  boolean empty = false, onlyOne = true;
  while (r.hasNext()) {
    String name = r.next().getName();

    if (!onlyOne)
      empty = false;

    if (name.equalsIgnoreCase("nobody") && onlyOne)
      empty = true;

    onlyOne = false;
  }

  return empty;
}

/**
 * Exports all resource groups of the high level Petri net.
 *
 * @param groups the list of resource groups
 * @param sheet the current Excel work sheet
 * @param i starting row in the Excel work sheet
 * @return the new row after all groups are exported
 * @throws IOException
 */
```

```
private int writeGroups(List<HLGroup> groups, WritableSheet sheet,
    int i) throws IOException
{
  Iterator<HLGroup> e = groups.iterator();

  while (e.hasNext()) {
    HLGroup el = (HLGroup) e.next();
    String group = el.getName();
    // If the group contains any :, don't write it
    if (group.indexOf(":") >= 0 || emptyGroup(el))
      continue;

    write(sheet, 0, i, el.getName());

    // Write resources assigned to this group
    List<HLResource> resources = el.getResources();
    Iterator<HLResource> r = resources.iterator();
    if (!r.hasNext())
      i++;
    while (r.hasNext()) {
      write(sheet, 1, i++, r.next().getName());
    }
  }

  return i;
}

/**
 * Exports all attributes of the high level Petri net.
 *
 * @param attrs the set of attributes
 * @param sheet the current Excel work sheet
 * @param i starting row in the Excel work sheet
 * @return the new row after all attributes are exported
 * @throws IOException
 */
private int writeAttributes(Set<HLAttribute> attrs, WritableSheet sheet,
    int i) throws IOException
{
  Iterator<HLAttribute> e = attrs.iterator();

  while (e.hasNext()) {
    HLAttribute el = (HLAttribute) e.next();
    write(sheet, 0, i, el.getName());

    if (el instanceof HLNominalAttribute) {
      HLNominalAttribute att = (HLNominalAttribute)el;
      write(sheet, 1, i++, att.getName() + "Dist()");
    }
    else if (el instanceof HLNumericAttribute) {
      HLNumericAttribute att = (HLNumericAttribute)el;
      HLDistribution dist = att.getPossibleValuesNumeric();
      write(sheet, 1, i++, this.distributionFormula(dist));
    }
    else if (el instanceof HLBooleanAttribute) {
      HLBooleanAttribute att = (HLBooleanAttribute)el;
```

```
        HLBooleanDistribution dist = att.getPossibleValues();
        double prob = dist.getProbability();
        String formula = String.format("%.4f", prob);
        write(sheet, 1, i++, formula);
      }
      else
        i++;
    }

    return i;
  }

  /**
   * Exports all nominal attributes, their values and frequencies of the high level Petri net.
   *
   * @param attrs the set of attributes
   * @param sheet the current Excel work sheet
   * @param i starting row in the Excel work sheet
   * @return the new row after all nominal attributes are exported
   * @throws IOException
   */
  private int writeUserDists(Set<HLAttribute> attrs, WritableSheet sheet,
      int i) throws IOException
  {
    Iterator<HLAttribute> e = attrs.iterator();

    while (e.hasNext()) {
      HLAttribute el = (HLAttribute) e.next();

      if (el instanceof HLNominalAttribute) {
        HLNominalAttribute att = (HLNominalAttribute)el;
        write(sheet, 0, i, att.getName() + "Dist");

        HLNominalDistribution dist = att.getPossibleValues();
        int totalFreq = dist.getSumOfAllFrequencies();
        HashMap<String, Integer> map = dist.getValuesAndFrequencies();
        int index = 1;
        for (String val : map.keySet()) {
          Integer freq = dist.getFrequencyPossibleValueNominal(val);
          writePercentage(sheet, 1, i, (double)freq / (double)totalFreq);
          writeInteger(sheet, 2, i, index++);
          write(sheet, 3, i++, val);
        }
      }
    }

    return i;
  }

  /**
   * Exports arrivals and case generating scheme of the high level Petri net.
   *
   * @param net the set of attributes
   * @param nodes the set of initial source nodes (places)
   * @param sheet the current Excel work sheet
   * @param i starting row in the Excel work sheet
```

```
 * @return the new row
 * @throws IOException
 */
private int writeArrivals(HLPetriNet net, HashSet<ModelGraphVertex> nodes, WritableSheet sheet,
    int i) throws IOException
{
  HLProcess process = net.getHLProcess();
  HLGlobal global = process.getGlobalInfo();
  HLDistribution dist = global.getCaseGenerationScheme();

  if (dist != null) {
    if (dist instanceof HLGeneralDistribution)
    {
      String distText = distributionFormula(dist);
      write(sheet, 1, i, distText);
    }
  }

  // Iterator to enumerate all starting places
  Iterator<ModelGraphVertex> it = nodes.iterator();

  // first, enumerate the nodes on this level
  while (it.hasNext()) {
    ModelGraphVertex n = it.next();

    // Count number of initial activities
    int count = n.getSuccessors().size();

    // Output entryDummy as the arrival if more than one starting activities exist
    if (count > 1) {
      write(sheet, 0, i++, "entryDummy");
      break;
    }

    // Get actual transitions from the current place
    Iterator iter = n.getSuccessors().iterator();

    while (iter.hasNext()) {
      ModelGraphVertex node = (ModelGraphVertex) iter.next();
      write(sheet, 0, i++, node.getIdentifier());
    }
  }

  return i;
}

/**
 * Exports the dummy entry (when more than one startiving activites exist).
 *
 * @param net the high level Petrinet object
 * @param nodes the set of initial source nodes (places)
 * @param sheet the current Excel work sheet
 * @param i starting row in the Excel work sheet
 * @return the new row
 * @throws IOException
 */
```

```
  private int writeDummyEntry(HLPetriNet net, HashSet<ModelGraphVertex> nodes, WritableSheet sheet,
      int i) throws IOException
  {
    // Iterator to enumerate all starting places
    Iterator<ModelGraphVertex> it = nodes.iterator();

    // first, enumerate the nodes on this level
    while (it.hasNext()) {
      ModelGraphVertex n = it.next();

      // Count number of initial activities
      int count = n.getSuccessors().size();

      // Output entryDummy as the arrival if more than one starting activities exist
      if (count <= 1) {
        continue;
      }

      write(sheet, 0, i, "entryDummy");

      // Get actual transitions from the current place
      Iterator iter = n.getSuccessors().iterator();

      while (iter.hasNext()) {
        ModelGraphVertex node = (ModelGraphVertex) iter.next();

        HLActivity activity = net.findActivity(node);
        HLChoice choice = net.findChoice(n);
        String expr = null;

        if (choice != null) {
          HLCondition cond = choice.getCondition(activity.getID());
          expr = cond.getExpression().toString();
        }

        expr = hackExpr(net, expr);
        i = writeOneDestActivity(node, expr, sheet, i);
      }
    }

    return i;
  }

  /**
   * Static utility function to remove line feed/carriage return from the input string.
   *
   * @param input the input string
   * @return string with lf/cr removed if any
   */
  private static String labelHack(String input) {
    return input.replaceAll("\\\\n", " ");
  }

}
```