Process Mining in Model Based Testing

Submitted in partial fulfillment for the degree of master of science

Aswathy George
11782463

Master Information Studies
Data Science
Faculty of Science
University of Amsterdam

2018-07-18

<table>
<thead>
<tr>
<th>Title, Name</th>
<th>Internal Supervisor</th>
<th>External Supervisors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dr Ana Oprescu</td>
<td>Dr Machiel van der Bijl, Drs Taco Witte</td>
<td></td>
</tr>
<tr>
<td>Affiliation</td>
<td>UvA, FNWI, IvI</td>
<td>Axini</td>
</tr>
<tr>
<td>Email</td>
<td><a href="mailto:A.M.Oprescu@uva.nl">A.M.Oprescu@uva.nl</a></td>
<td><a href="mailto:vdbijl@axini.com">vdbijl@axini.com</a>, <a href="mailto:witte@axini.com">witte@axini.com</a></td>
</tr>
</tbody>
</table>

Universiteit van Amsterdam

Amsterdam Data Science
List of Figures

1 Model Based Testing Process 3
2 Labelled Transition System with input and Output 3
3 Prefix and Postfix in transition miner 4
4 Transition Model[9] 5
5 Solution Architecture 5
6 csv Import in ProM 7
7 XES Conversion - Screen 1 7
8 Transition Miner 7
9 tsml Output 8
10 ProM Models Overview 8
11 Model in Axini Modelling Language 9
12 Model Visualized in Test Manager 9
13 Baseline Model in Test Manager 9
14 Unparameterized Stimulus and Response 9
15 Parameterized Stimulus and Response 9

List of Tables

1 Look-up table layout and description 8

Contents

List of Figures 1
List of Tables 1
Contents 1
Acknowledgments 2
Abstract 2
1 Introduction 2
2 Background 3
2.1 Model Based Testing 3
2.2 Process Mining 4
2.3 Process Mining Framework Selection 5
3 Implementation 5
3.1 System Logs 6
3.2 Learning a model with ProM Framework 6
3.3 Conversion of ProM Results to MBT model 7
3.4 Conversion to Axini Modelling Language 8
3.5 Evaluation Approach 8
4 Results 8
4.1 Process Model mined from ProM 8
4.2 Model in Axini Modeling Language 8
4.3 Test Model in Axini Test Manager 9
5 Discussion 9
6 Conclusions and Future Works 10
References 10
A Slides 11
B Models mined in ProM and Test Manager 12
ABSTRACT

Software Testing has evolved from cure-oriented (debugging when an error occurs) to prevention-oriented (structured testing approaches to find faults) in the last few decades[3]. What started as debugging has evolved into a formal and separate process in the field of software engineering. In this era of data, AI and automation, complexity of software systems have increased drastically and along with it the need for efficient testing has increased considerably. This industry is aiming to achieve and improve its various test processes and test automation using evolving technologies. Model Based Testing is one such technology that intends to realize end-to-end test automation. Model Based Testing uses at its base a model encoded with system behavior. The tests are generated and run based on this model. Conventionally the models are created manually from the system specification documents. In this paper we look at a different approach, process mining, to generate the models. Process Mining is a technique which extracts information from system-generated logs and it is applied mainly in the fields of Business Activity Monitoring (BAM), Business Operations Management (BOM), Business Process Intelligence (BPI), data/work flow mining[8]. This research applies the data/work flow mining techniques to mine a model that can be used in Model Based Testing. Towards this, ProM framework was used for process mining and a converter was implemented to convert the design oriented result of ProM to a structure oriented result that can be used in Model Based Testing.

ACKNOWLEDGMENTS

I would first like to thank my thesis supervisor Dr. Ana Oprescu of the FNWI at University of Amsterdam. She was always there whenever I needed her and she pulled me back whenever I steered off the path. I would also like to thank my supervisors at Axini, Dr. Ir. Machiel van der Bijl for providing me the opportunity to do this thesis at Axini under his expert guidance, Drs. Taco Witte for his patience, inputs and support throughout the thesis period and all other colleagues at Axini. I would also like to thank Dr. Carmen Bratosin from TNO for her invaluable and expert advice on Process Mining. Finally I would also like to thank my family, parents and my friends who supported me always and in everything.

1 INTRODUCTION

This thesis is the outcome of the graduation project for the Master’s Information Studies - Data Science program at the University of Amsterdam and the project was undertaken within Axini, a software company that offers Model Based Testing services. In this thesis, process mining techniques are applied to generate the model for model based testing with the aid of process mining framework ProM. As part of this thesis, a converter that converts the process mined model to a model based testing model is implemented. Software Testing[7] is an integral part of the software engineering process. Testing in this thesis implies a strategic and systematic approach to assess the quality of a software system and improve it. Software Testing can either be done manually or can be automated depending on the project size and complexity. With the revolutionary progress in the industry and technology, the complexity of software systems has also evolved tremendously which in turn demands constant improvement and efficiency in the test automation. Model Based Testing(MBT)[7] is one such technology that tries to meet the growing demands of the testing industry. MBT is a form of formal, specification based, black-box, functional testing. In MBT, an abstraction of the System Under Test(SUT) is created from the specifications, which is nothing but a set of requirements from which the actual system is implemented. This abstraction of the actual system is termed a “model” in MBT. The test suite (a set of test cases to be executed against the actual system) is generated algorithmically from this model.

System specification is usually a document that contains the system requirements. But sometimes this document is inadequately created[5][10] and in such cases it become unfeasible to use it in MBT or rather renders the model created from such a specification inefficient. To get beyond this limitation, in this thesis, we look at an alternative to define the specification, system logs. Already, there are researches going on in the field of model learning with machine learning and active learning techniques[12]. But here we look at process mining as a solution. Almost all software systems are programmed to record the events and actions of the system for the use of debugging or for future process improvements. This recording of events is termed logging and the output document is a log. Mining of the logs to extract information to improve and assess business process work flows has been an ongoing and active research domain and is referred to as process mining. In this project we explore the concept of using logs to arrive at a model to be used in MBT based on process mining techniques.

As an outcome of this research, the project aims to answer the following research questions:

- How to learn a model that can be used for Model Based Testing by applying process mining techniques on system-generated logs?
- Does the machine learned model out-perform the model generated using the traditional methods?

To achieve this, the following steps were undertaken as part of this research:

- Literature study on MBT and Process Mining concepts.
• Compare popular process mining frameworks based on existing research on the process mining framework evaluations and comparisons.
• Implementation of a converter that converts the process mining model (a design output from the Process mining framework) to a structured output that can be used in MBT.
• Evaluation of the resulting MBT model.

2 BACKGROUND
This chapter sets the prerequisites by introducing the concepts used in this thesis. Section 2.1 provides a brief outlining of what MBT is. Section 2.2 offers a detailed introduction to the field of process mining and section 2.3 explains the rationale behind the selection of the ProM framework[1] to perform the process mining in this thesis.

2.1 Model Based Testing
As mentioned in the introduction, Model based Testing[7] makes use of a model of the desired system behavior to generate tests for a software system. A model is an abstraction of the 'real world' representations and in the context of MBT it is an abstraction of the System Under Test. What sets it apart from the regular testing approaches is that it starts with the generation of a large number of test cases (algorithmically generated from the model) as opposed to the generic test automation where the test cases are manually created. So in short it can be said that MBT aims to achieve end-to-end automation of a test cycle. Figure 1 represents Model Based Testing Process.

![Figure 1: Model Based Testing Process](image)

There are several modeling approaches to formally define a model. But in this thesis we look at two of the popular approaches that are used in most of the available MBT frameworks - labelled transition systems and Symbolic Transition Systems[2] (which is an extension of labelled transition systems).

Labelled Transition Systems

The following explanation and the candy machine example illustrated is referenced from Tretmans's paper, “Model Based Testing with Labelled Transition Systems”[7]. Tretmans define[7] labelled transition systems as a structure consisting of states and their transitions labelled with actions. In their representation, states form the nodes and transition form the edges between the states. To formalize it let us consider Q - a countable set of states \( q_0, q_1, q_2, \ldots q_n \) where \( q_0 \) is the start state of the system, \( L \) - a set of labels \( \{ \mu_0, \mu_1, \mu_2, \ldots, \mu_n \} \) and \( T \) - a set of transitions where a transition is defined as the movement from one state to another as a result of an action performed (i.e. \( q_0 \xrightarrow{\mu_0} q_1, q_1 \xrightarrow{\mu_1} q_2, \ldots \)). Another feature to be considered in the explanation of a transition is \( \tau \). While \( \mu \) is an observable action, \( \tau \) represents the unobservable or internal actions within the system. So \( T \) can be defined as \( T \subseteq Q \times (L \cup \{ \tau \}) \times Q \).

While modeling a labelled transition systems we consider all the possible interactions that a system communicates with its environment without really distinguishing the direction of the interaction. To incorporate that distinction in to the system the labels/actions are split into input labels and output labels where input labels are actions originated at the environment and output labels are actions originated at the system. Input labels are prefixed with a “?” and the output labels are prefixed with a “!”.

Figure 2 represents a simple labelled transition system with inputs and outputs for a candy machine[7]. The initial interaction(\( \mu_0 \)) is a button click represented as “but” to achieve the next state. The new state has 2 possible actions liquorice represented as “liq”(\( \mu_1 \)) and chocolate represented as “choc”(\( \mu_2 \)). The nodes in the graph represent the states and labelled edges represent the labelled transitions.

![Figure 2: Labelled Transition System with input and Output](image)

Symbolic Transition Systems

The explanation of symbolic transition systems is referred from the paper, “Test Generation Based on Symbolic Specifications by Tretmans et al.”[2]. Symbolic Transition Systems are an extension of labelled transition systems. While labelled transition systems use explicit internal representations to define states (which is termed as state oriented), symbolic transition system adds on to this concept, with the concept of location variables and concept of data. In doing so it enhances the
model to exhibit data-dependent behavior[2]. Axini’s MBT framework uses this approach in their modeling. Axini uses their own modelling language termed AML to model the system under test. Axini’s tool traverses the STS (State Transition System) generated from the AML model to generate test cases and the constraints are converted into Prolog and solved by GNU Prolog solver.

2.2 Process Mining

To introduce and explain the basic concepts of process mining and the algorithm used, references and illustrations from the publication, “Process mining : a two-step approach to balance between underfitting and overfitting” written by Aalst et al.[9] is used. As explained briefly in the introduction, process mining is a technique that extracts process related information from the logs. Data mining concepts are applied to the log to identify traces, patterns and details present in the log. Typically there are 3 main classifications in Process Mining - Discovery, Conformance Checking and Performance Mining. This distinction is mainly based on the availability of a "priori" model i.e. a verified process model that is already available for a system. As the name implies "Discovery" discovers a new model from the event log based on the event traces present in the log. A "priori" model is not available or not used in this technique. Conformance Checking compares an existing model ("priori") with the process mined model to identify deviations or to understand the system decisions in a process flow. Performance mining retrieves performance indicators of a system from the event logs to improve performance of the system. This is again based on a "priori" model. In this research we look specifically at process discovery models of process mining as the main goal is to discover the user behavior to build a test model.

There are different algorithms to enable process discovery with most popular ones being, heuristic miner, genetic miner, fuzzy miner and transition system miner. Each of the algorithm differs in their approach and the use of a particular algorithm is decided based on the information available in the system log and project requirement. However in this research, as the basis of a model based testing is a transition system, the project clearly demands the identification of states and transitions. This invariably leads to the use of a transition system miner in this research.

Transition system Miner

Transition system miner as the name indicate mines a transition model out of an event log. The events within the log are identified as transitions and the places in between these two transitions are identified as states. The identification of states is one of the critical steps in this miner. The states can be defined based on the past events (referred to as prefix), future events (referred to as postfix) or a combination of both. This algorithm was intended to create a balance between the overfitting and underfitting issues of process mined model. Overfitting implies a model that does not generalize the system behavior. This happens when the model is tailored to exactly represent a particular log. This results in a model that is exactly like the log without really giving a complete model or overview of the system. Underfitting occurs when the mined model showcases much more than actually present in the log leading to very complex and spaghetti like models. Aalst et al., in their paper[9], describes 5 abstractions to construct a transition model which can be considered as different parameters to generate a model and this in-turn lets the user to control the underfitting/overfitting variables in a model generation. These abstraction techniques are explained briefly in this section. To explain these abstraction lets define a list of sample activities and log. Let’s consider A as a list of activities((A, B, C, D, C, D, C, D, E, F, A, G, H, H, H, I)), σ ∈ A* (where A* is a set of all finite sequence or all cases that forms within A) as a single trace and L ∈ P(A*) as an event log. Lets define the sample log L with cases ABCD,ACBD,ABCD,ABCD,ABCD,ACBD.

Figure 3 explains the prefix/postfix activities[9].

Abstraction 1: Maximal Horizon,h - One of the factors in the state calculation is setting the number of prefixes or postfixes in the identification of states. This is termed maximal horizon(h). “h” can be set to consider all the activities in a trace or a subset of a trace depending on the depth and extent of the log. The state can be identified by the entire sequence of prefix/postfix or a partial sequence. There are several ways to identify the state between E and F in the set of activities, A, mentioned previously. If the model is constructed based on prefix, the prefix set would be {A, B, C, D, C, D, C, D, E} and setting an h=6 would reduce the prefix to {D, C, D, C, D, E} and this would become the basis of identifying a state.

Abstraction 2: Filter,F - Once the horizon is set, filter provides an option to further create an abstraction. If a prefix of (C, D, E) is finalized from setting a horizon to identify the state between E and F, defining a filter, F = (C, D) further reduces the prefix to (C, D) from (C, D, E).

Abstraction 3: Maximum Number of Filtered Events,m - This is the 3rd level of abstraction that can be applied in identifying a state in the transition miner algorithm. This number, m, is configured on the filtered prefix or postfix. An example - On a prefix, (A, B, C, D, C, D, C, D, E), if h = 6,F = C,E and m = 2, after h is applied the the prefix becomes (D, C, D, C, D, E). Applying the F will yield a prefix of (C, C,E). The filter,m, is applied on this resulting set and finally we have the prefix, (C, E).

Abstraction 4: Sequence, Multi-set or Set,q - The first 3 abstractions set the sequence of activities to be considered to determine a state. The next one determines the importance of the order or frequency of the identified sequence. The algorithm allows for 3 classifications, sequence (seq) - where the order of the events are maintained which in turn captures the frequency too, multi-set(ms)
- only the frequency is captured and set(set) - just the set of activities are recorded without the order or frequency.

**Abstraction 5: Visible Activities, V** - The final abstraction is identifying a subset of activities to be displayed as the transition labels between the edges.

The Transition miner algorithm uses the above mentioned parameters in the construction of the process model leading to the discovery of multiple versions of the model. The user can decide on these parameter settings based on the degree of generalization that’s required in the project. Figure 4 shows a transition model with the prefix abstractions: $h = \infty$, $F = \text{all activities}$, $m = \infty$, $q = \text{set}$ and $V = \text{all activities}$. Setting these parameters provides a transition model without any abstractions.

![Figure 4: Transition Model[9]](image)

### 2.3 Process Mining Framework Selection

One of the critical steps in this research has been to decide whether to create a process mining solution independently or to use an existing framework. In order to arrive at a decision it was important to conduct a detailed study on the popular frameworks currently available for process mining. There have been 2 masters dissertation works on comparative evaluation of different available process mining frameworks and these have been used as a basis for the implementation decision.

- **Comparative Evaluation of Process Mining tools**, Musie Kebede, University of Tartu [4].

One dissertation[11] performs an exploratory research on the available tools with the aid of surveys and interviews. Their main aim was to address the following questions:

- Which process mining tools are used and by whom?
- What are the important criterion in choosing a process mining tool?

According to this research the most important criteria in choosing the framework was usability, visualization, integration and functionality (including import/export criterion). The paper based their study on the following popular frameworks in process mining - Disco, ProM, Celonis, Perceptive, Aris, Stereologic, XManalyzer, Fujitsu, QPR. The results identified Disco as being the leader in the non functional criterion and ProM to be the leading software in functional aspects.

The second dissertation does the comparative study using an analytical approach. The study proposes a framework to compare the process mining tools on the basis of functional features. In this paper, ProM, Disco and Celonis are compared using the proposed framework. This study identifies some core operations for process mining tool comparison - filtering data, process discovery, conformance checking, social network mining, decision rule mining, process visualization, performance reporting, discriminative rule mining, trace clustering, delta analysis. The results showed that ProM could support all the core operations whereas Disco and Celonis covers only partial operations. The paper advises to use Disco for its ease of use and fast processing and use ProM in case of higher or deeper analysis.

On the basis of the results from both the papers, it was concluded that both Disco and ProM exhibit strong and comparable capabilities. One of them would be an ideal choice in an implementation of process mining. Since both of these tools had the capability of mining a transition system, the decision invariably was to use one of these frameworks instead of reinventing the wheel with an independent solution. Finally, ProM was selected in this study as Disco was not compatible with Linux and the architecture in which the research was conducted used a linux environment. Once ProM was selected further research was done on working with ProM and generating a good model with ProM. For the purpose the paper, “The ProM Framework: A New Era in Process Mining Tool Support” written by F.van Dongen et al.[1] was used.

## 3 IMPLEMENTATION

This section describes the implementation steps done as part of this research to implement the converter which converts a process mined model to a model-based testing model. Figure 5 describes the solution architecture of this implementation. The analysis and preprocessing of the logs, conversion of the process mining model to the test model was implemented in python language using the Data Analysis library, pandas[6].

![Figure 5: Solution Architecture](image)
3.1 System Logs

As mentioned in the earlier sections, the input data for this implementation is a system log. Hence it becomes imperative to describe the log requirements and preprocessing steps necessary for this research. A basic event log for it to be used in process mining should contain a case-id, activity(event) and time stamp. Case id logically groups together all the events that are executed in the same session. Each case would begin with an event that is defined as the initial state. To define a case there should be a clear start or stop event that separates the activities. Activity is the system event recorded and time-stamp is when the event occurred. In this implementation, activity is a combination of system stimulus (communication to the server. Referred as “RECEIVE” in the test model) and response(received from the server. Referred as “SEND” in the test model). Time-stamp is date/time when the activity is recorded. Time-stamp can be omitted if the log is ordered correctly or if sequencing of the event is not a requirement. Time-stamp becomes imperative in case of concurrent behavior in the log or if performance mining is the goal of process mining. The main preprocessing activities for this implementation include:

- Separate cases using unique case identifier. To do this, identify a start or end state in a sequential execution.
- Group stimulus event and the corresponding response event under a single event identification but as separate entities.
- Separate the stimulus constraints and the variable values returned in a response.
- Save the preprocessed log in a csv format.

This study was done with 2 production-like test environment logs as input - Log of an article registration system for billing and a log of Railway tracking system.

**Article Registration System log** This is a log generated by an article registration system for billing and a log of Railway tracking system.

**Railway Tracking system log** This log contained the user behavior of a railway tracking system. A desired model could not be mined from ProM due to missing details in the log. As with the article registration log, this log also contained stimulus and response as separate distinct events. But what really made this distinct events into a limitation is that several stimuli were grouped together in a single row or in consecutive rows and similarly the response for these stimuli were recorded in consecutive rows without any identifier between a stimulus-response which makes it unusable with a transition miner. The author is unaware if this is an anomaly within the system that will be corrected in the future or a system requirement. This limitation is explained with a candy machine log example in the same format as the article registration log. In the log sample below, it can be seen that there is no identifier that can map a particular response to a stimulus and this will result in treating them as independent events. The model generated would be a flat structure with events falling one after the other and not providing any insight into the communication protocol.

### Log Format

<table>
<thead>
<tr>
<th>CASEID</th>
<th>EVENTID</th>
<th>Response</th>
<th>Stimulus</th>
<th>TimeStamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11789</td>
<td>001</td>
<td>HELLO</td>
<td>20110303/132605</td>
</tr>
<tr>
<td>1</td>
<td>11790</td>
<td>678</td>
<td>EVENT1</td>
<td>20110303/132605</td>
</tr>
<tr>
<td>1</td>
<td>11791</td>
<td>239</td>
<td>EVENT3</td>
<td>20110303/132605</td>
</tr>
<tr>
<td>1</td>
<td>11792</td>
<td>001</td>
<td>HELLO</td>
<td>20110303/132605</td>
</tr>
<tr>
<td>2</td>
<td>11793</td>
<td>870</td>
<td>EVENT9</td>
<td>20110303/132605</td>
</tr>
</tbody>
</table>

Log After Preprocessing

### Log Format

<table>
<thead>
<tr>
<th>CASEID</th>
<th>EVENTID</th>
<th>EVENT</th>
<th>TimeStamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1189</td>
<td>HELLO</td>
<td>20110303/132605</td>
</tr>
<tr>
<td>1</td>
<td>1190</td>
<td>CHOC</td>
<td>20110303/132605</td>
</tr>
<tr>
<td>1</td>
<td>1191</td>
<td>LIQ</td>
<td>20110303/132605</td>
</tr>
<tr>
<td>1</td>
<td>1192</td>
<td>GUM</td>
<td>20110303/132605</td>
</tr>
<tr>
<td>1</td>
<td>1193</td>
<td>213</td>
<td>20110303/132605</td>
</tr>
<tr>
<td>1</td>
<td>1194</td>
<td>739</td>
<td>20110303/132605</td>
</tr>
<tr>
<td>1</td>
<td>1195</td>
<td>110</td>
<td>20110303/132605</td>
</tr>
</tbody>
</table>

3.2 Learning a model with ProM Framework

Once the log was preprocessed, the next step was to generate a process model from the ProM framework. XES is the standard logging format accepted for process mining and ProM also uses the same for its modeling. However, ProM also accept logs in csv and MxML format which can be converted to XES format using a plug-in available within the ProM framework. The ProM implementation is explained in steps with the help of few screenshots. The screenshots are provided so that the reader can get an idea on the framework interface.

The first step is the import of the preprocessed csv log into ProM framework. Refer figure 6...
Conversion of csv log to XES format is the next step. The screen in Figure 7 shows the list of available plug-ins for the input selected. Here for the csv input, the only plug-in available is the “Convert CSV to XES” plug-in. In the next screen, the user can configure csv parser settings. Since the columns are already parsed in the preprocessing step, the default selections can be kept as it is. In the next screen, the user is prompted to assign the case, event, and timestamp columns based on which the log traces are defined. In this experiment, case id from the preprocessed log is mapped to the case column. Both Stimulus and Response columns are mapped to an event and timestamp is not used in this case as the log is already ordered and a stimulus-response pair can be identified.

Generating a transition model follows XES conversion. Run the transition miner using “Mine Transition System” plug-in on the XES event log (Figure 8). The transition miner parameters and algorithm is explained in detail in section 2.2. The miner allows the user to set the following parameters - maximal horizon(h) and sequence, multi-set or set(q). These parameters were set to build a model with least abstractions as our intention was to mine all the traces available. In this implementation, the states are calculated based on the prefix states and setting the h to “No Limit” ensures that all the prefix states are considered in the calculation of the next state. Also, we set q as a “set” this parameter drops the sequence of events in the state calculation. Based on these parameters, the state transition model that results from the miner differs i.e. each different level of abstraction results in a different model. But to mine a test model, it is better to use a “no abstractions” or least level of abstractions as this ensures that all the available traces are captured from the log.

Next, the generated design-oriented model (in "transition system markup language" format referred to as tsml format) is exported and saved. This saved output is used for the conversion to a structured model that can be used in the Model Based Testing.

### 3.3 Conversion of ProM Results to MBT model

This subsection and the next subsection forms the crux of this experiment i.e. these sections provide a detailed explanation on the implementation of the converter that converts a process mined model to be used in Model Based Testing. Process mined model from any framework which is in a transition system markup language format can be used with this tool. The starting point for the converter is a state-transition model in a tsml format. In this experiment, we use the transition system mined from ProM. The saved output in tsml format is loaded, analysed, and processed in python using pandas library. Figure 9 shows a partial view of the tsml format. This output provides information on the states, transition and their design-oriented specifications i.e. information on each node (state) and their edges (transitions) connecting to a different state. Next step in this implementation was to convert this design-oriented output to a structured output.
For the conversion, firstly, all the states and transitions are extracted. Then an initial look-up table is created. Table 1 provides an explanation on the various fields of the table.

<table>
<thead>
<tr>
<th>Entities</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Unique ID for each state</td>
</tr>
<tr>
<td>State</td>
<td>The state name as in the design output - ex: state1</td>
</tr>
<tr>
<td>StateID</td>
<td>An ID associated with each state in the design output - ex: 131 corresponds to state1</td>
</tr>
<tr>
<td>SucceededBy</td>
<td>StateID that follows a particular state - ex: stateID 132 follows the stateID 131</td>
</tr>
<tr>
<td>Stimulus</td>
<td>The stimulus part in the transition label that leads to a state is captured as that state stimulus.</td>
</tr>
<tr>
<td>Response</td>
<td>The response part in the transition label that leads to a state is captured as that state response.</td>
</tr>
</tbody>
</table>

Table 1: Look-up table layout and description

The ID is a system generated unique identifier and not mapped from the tsml file. The rest of the fields are mapped from the information extracted from the tsml file. The unique states are extracted from the "state" element of the tsml file. Similarly, the transitions are extracted from the "transition" element and they are saved to a dataframe in pandas. The information on the source and target states for each of the transitions are captured in this data frame. This is further analyzed and mapped to create the lookup table. This lookup output can be saved as csv output and used as a base for any test conversions.

3.4 Conversion to Axini Modelling Language

Once the tsml output is mapped to the above mentioned layout, the next step was to generate a script in the Axini Modeling Language to use in the Axini’s Test Manager. The lookup table was mapped to generate a model in the format in Figure 11. All the "stimulus" events were mapped to the "RECEIVE" command and the corresponding "Response" events were mapped to the "SEND" command. The "GOTO" was mapped with the values in the “SUCCEEDED BY” column of the lookup table. The generated model in the AML format was loaded to Axini’s test framework, Test Manager. The visualized representation of the model was considered the final output and was used in the evaluation of this experiment. The resulting model of this final mapping is presented in the Results section 4.3.

3.5 Evaluation Approach

The logs used for process mining had a corresponding verified test model. The output test model was evaluated against this baseline model for evaluation. Manual comparison of models were performed under the supervision of two of the MBT experts at Axini. One of the experts, Machiel holds a PhD in Model Based Testing and the other expert, Taco has made several complex models in the last 7 years to enable model based testing for different industrial sectors.

4 RESULTS

This section presents the result from ProM and the converters without any evaluations. The evaluation and the analysis of the results are presented in the section 5.

4.1 Process Model mined from ProM

Figure 10 is an image of the design output generated from ProM for both the article registration log and the railway tracking system log (masked with candy machine example). Since the railway tracking system log did not provide a good model (as response-stimulus pair was not identifiable from the log), rest of the results are presented with the model generated from article registration log. Figure 9 shows the results for article registration log in mark-up format of the model. A detailed and zoomed in model visualization for article registration model is available in the Appendix section. 10 traces are discovered from the log with 201 distinct states.

4.2 Model in Axini Modeling Language

The ProM model is converted to a model in Axini’s modeling language. Figure 11 shows a partial model sample.
4.3 Test Model in Axini Test Manager
The Final step was to use the AML model in the test framework of Axini. Figure 12 represents the visualization in Test Manager. A detailed and zoomed in model visualization is available in the Appendix section.

Figure 12: Model Visualized in Test Manager

5 DISCUSSION
This section provides a detailed evaluation and analysis of the process mined model results in comparison with the baseline test model. The evaluation presented below is based on the article registration log. Figure 13 represents the baseline model and as mentioned in results, figure 12 is the ProM generated model converted to Axini test model.

Figure 13: Baseline Model in Test Manager

At a glance, the process mined model from ProM could identify the traces presented in the log. The resulting model was a labeled transition system model. So a direct comparison with the baseline model was not possible as Test Manager works with and generates a symbolic transition system model. A symbolic transition system model could not be generated as the log did not contain information on the datatypes and constraints defining the parameters that were passed. However, the workflows and traces identified can be compared manually to validate if the model generated is accurate. Both the baseline model and the process mined model is built on the stimuli and responses in the communication protocol defined for the article registration system. The figure 14, 15, represents the comparison results of the unparameterized and parameterized stimulus and response between the baseline and test model from process mining. The corresponding blank spaces in the column, Stimulus-ProM Model and Response-ProM Model represents the unidentified stimuli and responses. This is due to the fact the log does not contain the events that lead to the traces with these stimulus or response.

Figure 14: Unparameterized Stimulus and Response

Also, on deeper analysis, we can see that the traces are not merged when there are identical inflows/outflows and they are considered as separate traces. This is due to the abstraction parameter settings while generating a model with the ProM. This also explains...
the 201 distinct states from the small number of stimulus. There are parameters that can be set to merge the identical inflows and outflows. The converter was implemented with a "no abstraction" decision as this enabled identifying detailed and exact flows and complies with "completeness" factor which is one of the important requirements and challenges in software testing. There are totally 10 traces captured all starting with the same state and is similar to the baseline model. From the log perspective we can say that the recall is 100% but from system perspective that cannot be said as the used log does not represent or contain traces for all the system functionality and this invariably forms a limitation in the use of process mining to mine a test model. Since the model was generated from a test environment log, we can see some bad weather behavior where the registrations are attempted even after signoff and these are captured as individual traces in itself. This can be considered one of the advantages of using production-like test environment logs. The log contains both good weather and bad weather behavior adding on to the completeness factor. When looking at the ProM result for the railway tracking system, it can be seen that the ProM produces a flat structured model (Figure 10) and the communication protocol was not captured correctly. This clearly shows that currently the transition miner algorithm can only mine a good model if the stimulus-response pair can be identified from the log.

Hence, in this analysis, it is imperative that we also set the basic requirements for a log to be suitable to be used for generating a test model. From this study it was clear that only logs that visibly defines a stimulus-response pair produces a good model. Also, the timestamp field is important if the sequence of the event is not defined. If there is a clear start or end event and stimulus-response pair, a correct but basic workflow can be mined. But the model can be improved tremendously if data types or constraints defining the parameters are also included in the log.

6 CONCLUSIONS AND FUTURE WORKS

This section describes the conclusion derived out of this study and lay ground for some future researches. As a concluding remark, it can be said that application of process mining is a promising yet not a complete solution for the creation of a test model in the software testing domain. One of the main supporting argument to this conclusion is regarding completeness. In Software testing, test coverage/completeness is one of the important deciding factors on the success of testing or in other words, quality of a software system. Specification derived from a log can only provide completeness locally i.e. the log is complete only with respect to the system functionality from which the logs were generated and hence the model is also limited. The process mined model cannot identify or generate traces with stimulus that are not present in the input log. To achieve the desired test coverage, many logs from the same system needs to be mined and merged. Due to the limited accessible logs and time limitations, this is not a capability of the current implementation. However, this can be looked in to as a future study given that one has as input, multiple and diverse functionality logs from the same system or a single log that contains all the traces of a system.

Another proposal for the future work is regarding the mining of a symbolic transition system. There were 2 main hindrances to mine a symbolic transition system in this research. One was that the log did not contain constraints/data types defining the parameters passed and the second was currently there is no plug-in to mine a symbolic transition system in ProM. Identifying and defining a set of log standards to be used in Model Based Testing can be the basis to achieve symbolic transition mining. In other words, a log specification can be manually crafted to define the specific requirements for model based testing. Configuring the logs in line with this specification and creation of a new plug-in to mine a symbolic transition system in ProM could be a way forward to enable symbolic transition mining. Another limitation that can be observed in mining with ProM is its incapability of handling parallelism. With the example in Railway Tracking System, we can observe that when various stimulus and responses are executed together and not in a stimulus-response pair, a good model cannot be mined.

As a conclusion, it can be said that the current implementation in its maturity and with the limitations mentioned above is not a fully reliable method in model learning and cannot yet mine a test model that is better than the traditionally mined model.

REFERENCES

A SLIDES

Introduction

Process Mining
- Log mining
- Classifications – Process Discovery, Conformance Checking, Performance Mining

Problem statement
- Black box testing – building tests from specifications
- Model Based Testing – Type of testing that aims to automate end-to-end test cycle, based on a test model from which the tests are generated
- Can tests be used to build the test model? Will it be better than building a test model manually?

Research Question

- Research Questions
  - RQ1: How to learn a model that can be used for model-based testing by applying process mining techniques on system generated logs?
  - RQ2: Does the process mined model outperforms the model generated using the traditional methods?

- Research Approach
  - Literature study – Model Based Testing, Process Mining
  - Process mining from get go or use an existing framework?
  - How to learn a test model from process mined model? – RQ1
  - Evaluate the model – RQ2

Approach - How to process mine??

Framework comparison with results from 2 master’s thesis
- Dieter Erben, Process Mining in Practice: Comparative Study of Process Mining Software, University of Ghent

- ProfM

- Model Based Testing
  - Jan Verbraak, 2006 Model Based Testing with Labelled Transition Systems.
  - M. Oud, J. Verbraak, and H. van Doorn. 2006. Test Generation based on Symbolic Specifications

Approach - Literature Study/Related Works

- Process Mining

- ProfM
  - W. van der Aalst, W.H. van der Aalst, and M.W. van der Aalst. 2011: The ProfM framework. A new tool for process mining that supports

- Model Based Testing
  - Jan Verbraak, 2006 Model Based Testing with Labelled Transition Systems.
  - M. Oud, J. Verbraak, and H. van Doorn. 2006. Test Generation based on Symbolic Specifications

Approach - How to learn a test model using process mining approach

How to learn a test model using process mining approach – cont..

- Input – csv format system logs containing or pre-processed to include
  - Cases (activity denoting case start or case ends is a read)
  - Event = stimulus = response
  - Time stamps (millisecond)

- Mining with ProfM
  - Log conversion to XES format (standard EML format with mark ups)
  - Transition miners: F
t
  - No abstractions model
  - Design model exported as a txml format

- Converting
  - Txml mappings are converted to Arno’s AML script
Evaluation

- Model from ProM - Labelled Transition system.
  - Log does not always capture the datatypes and the factors defining the data input
  - Number input might be a string in the system
  - Input data, a number between 5 and 30
- Workflow identified correctly from the log but is it complete?
  - Local optimum
  - Is global optimum a feasibility

Conclusion and Future works

- Log learning – an evolving field.
  - Process mining moving to machine learning
- Current implementation with its limitation is not fully reliable. However,
  - Multiple logs of the same system covering diverse functionality – global optimum?
  - Logs configured to achieve the desired depth

B MODELS MINED IN PROM AND TEST MANAGER
Model Mined from ProM