ProM 6 Tutorial

H.M.W. (Eric) Verbeek
mailto:h.m.w.verbeek@tue.nl
R. P. Jagadeesh Chandra Bose
mailto:j.c.b.rantham.prabhakara@tue.nl

August 2010

1 Introduction

This document shows how to use ProM 6 to answer some of the common questions that managers have about processes in organizations. The questions are listed in Section 1.1. To answer these questions, we use the process mining plug-ins supported in ProM 6. This tool is open-source and it can be downloaded at www.processmining.org. For the reader unfamiliar with process mining, Section 1.2 provides a concise introduction. All the questions listed in Section 1.1 are answered based on an event log from the running example described in Section 1.3. Finally, we advice you to have ProM 6 at hand while reading this document. This way you can play with the tool while reading the explanations. Section 1.4 explains how to get started with ProM 6.

1.1 Common Questions

Questions that managers usually have about processes in organizations are:

1. What is the average/minimum/maximum throughput time of cases?
2. Which paths take too much time on average? How many cases follow these routings? What are the critical sub-paths for these paths?
3. What is the average service time for each task?
4. How much time was spent between any two tasks in the process model?
5. How are the cases actually being executed?

6. Are the rules indeed being obeyed?
7. How many people are involved in a case?
8. What is the communication structure and dependencies among people?
9. How many transfers happen from one role to another role?
10. Who are important people in the communication flow? (the most frequent flow)
11. Who subcontract work to whom?
12. Who work on the same tasks?

We show how to use ProM 6 to answer these questions in Section 3.

1.2 Process Mining

Nowadays, most organizations use information systems to support the execution of their business processes [8]. Examples of information systems supporting operational processes are Workflow Management Systems (WMS) [5, 6], Customer Relationship Management (CRM) systems, Enterprise Resource Planning (ERP) systems and so on. These information systems may contain an explicit model of the processes (for instance, workflow systems like Staffware [3], COSA [1], etc.), may support the tasks involved in the process without necessarily defining an explicit process model (for instance, ERP systems like SAP R/3 [2]), or may simply keep track (for auditing purposes) of the tasks that have been performed without providing any support for the actual execution of those tasks (for instance, custom-made information systems in hospitals). Either way, these information systems typically support logging capabilities that register what has been executed in the organization. These produced logs usually contain data about cases (i.e. process instances) that have been executed in the organization, the times at which the tasks were executed, the persons or systems that performed these tasks, and other kinds of data. These logs are the starting point for process mining, and are usually called *event logs*. For instance, consider the event log in Table 1. This log contains information about four process instances (cases) of a process that handles fines.

Process mining targets the *automatic* discovery of information from an event log. This discovered information can be used to deploy new systems that support the execution of business processes or as a feedback tool that helps in auditing, analyzing and improving already enacted business processes. The main benefit of process mining techniques is that information is *objectively* compiled. In

Case ID	Task Name	Event Type	Resource	Date	Time	Miscellaneous
1	File Fine	Completed	Anne	20-07-2004	14:00:00	...
2	File Fine	Completed	Anne	20-07-2004	15:00:00	...
1	Send Bill	Completed	system	20-07-2004	15:05:00	...
2	Send Bill	Completed	system	20-07-2004	15:07:00	...
3	File Fine	Completed	Anne	21-07-2004	10:00:00	...
3	Send Bill	Completed	system	21-07-2004	14:00:00	...
4	File Fine	Completed	Anne	22-07-2004	11:00:00	...
4	Send Bill	Completed	system	22-07-2004	11:10:00	...
1	Process Payment	Completed	system	24-07-2004	15:05:00	...
1	Close Case	Completed	system	24-07-2004	15:06:00	...
2	Send Reminder	Completed	Mary	20-08-2004	10:00:00	...
3	Send Reminder	Completed	John	21-08-2004	10:00:00	...
2	Process Payment	Completed	system	22-08-2004	09:05:00	...
2	Close case	Completed	system	22-08-2004	09:06:00	...
4	Send Reminder	Completed	John	22-08-2004	15:10:00	...
4	Send Reminder	Completed	Mary	22-08-2004	17:10:00	...
4	Process Payment	Completed	system	29-08-2004	14:01:00	...
4	Close Case	Completed	system	29-08-2004	17:30:00	...
3	Send Reminder	Completed	John	21-09-2004	10:00:00	...
3	Send Reminder	Completed	John	21-10-2004	10:00:00	...
3	Process Payment	Completed	system	25-10-2004	14:00:00	...
3	Close Case	Completed	system	25-10-2004	14:01:00	...

Table 1: Example of an event log.

other words, process mining techniques are helpful because they gather information about what is *actually* happening according to an event log of a organization, and not what people *think* that is happening in this organization.

The type of data in an event log determines which *perspectives* of process mining can be discovered. If

(i) the log provides the tasks that are executed in the process and

(ii) it is possible to infer their order of execution and link these taks to individual cases (or process instances),

then the *control-flow perspective* can be mined. The log in Table 1 has this data (cf. fields "Case ID", "Task Name", "Date", and "Time"). So, for this log, mining

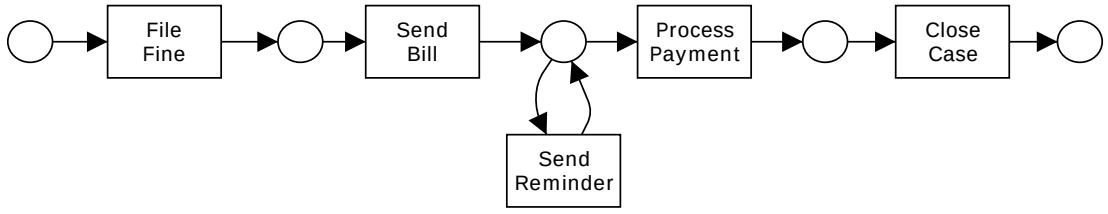

Figure 1: Example process mined from the event log in Table 1.

algorithms could discover the process in Figure 1[1]. Basically, the process describes that after a fine is entered in the system, the bill is sent to the driver. If the driver does not pay the bill within one month, a reminder is sent. When the bill is paid, the case is archived.

If the log provides information about the persons/systems that executed the tasks, the *organizational perspective* can be discovered. The organizational perspective discovers information like the social network in a process, based on transfer of work, or allocation rules linked to organizational entities like roles and units. For instance, the log in Table 1 shows that "Anne" transfers work to both "Mary" (case 2) and "John" (cases 3 and 4), and "John" sometimes transfers work to "Mary" (case 4). Besides, by inspecting the log, the mining algorithm could discover that "Mary" never has to send a reminder more than once, while "John" does not seem to perform as good. The managers could talk to "Mary" and check if she has another approach to send reminders that "John" could benefit from. This can help in making good practices a common knowledge in the organization.

If the log contains more details about the tasks, like the values of data fields that the execution of a task modifies, the *case perspective* (i.e. the perspective linking data to cases) can be discovered. So, for instance, a forecast for executing cases can be made based on already completed cases, exceptional situations can be discovered etc. In our particular example, logging information about the profiles of drivers (like age, gender, car etc.) could help in assessing the probability that they would pay their fines on time. Moreover, logging information about the places where the fines were applied could help in improving the traffic measures in these places.

From this explanation, the reader may have already noticed that the control-flow perspective relates to the "How?" question, the organizational perspective to the "Who?" question, and the case perspective to the "What?" question. All these three perspectives are complementary and relevant for process mining, and can be answered by using ProM 6.

ProM 6 [4, 11] is an open-source tool specially tailored to support the development of process mining plug-ins. This tool contains a wide variety of plug-ins. Some of them go beyond process mining (like doing process verification, converting between different modelling notations etc). However, since in this tutorial our focus is to show how to use ProM 6 plug-ins to answer common questions about processes in companies (cf. Section 1.1), we focus on the plug-ins that use as input (i) an event log only or (ii) an event log and a process model. Figure 2 illustrates how these plug-ins can be categorized.

Discovery The plug-ins based on data in the event log only are called *discovery* plug-ins because they do not use any existing information about deployed

[1]The reader unfamiliar with Petri nets is referred to [7, 9, 10].

models.

Conformance The plug-ins that check how much the data in the event log matches the prescribed behavior in the deployed models are called *conformance* plug-ins.

Extension Finally, the plug-ins that need both a model and its logs to discover information that will enhance this model are called *extension* plug-ins.

In the context of our common questions, we use

(i) discovery plug-ins to answer questions like

- How are the cases actually being executed?
- Are the rules indeed being obeyed?,

(ii) conformance plug-ins to questions like

- How compliant are the cases (i.e. process instances) with the deployed process models?
- Where are the problems?
- How frequent is the (non-)compliance?, and

(iii) extension plug-ins to questions like

- What are the business rules in the process model?.

1.3 Running Example

The running example is about a *process to repair telephones in a company.* The company can fix 3 different types of phones ("T1", "T2" and "T3"). The process starts by registering a telephone device sent by a customer. After registration, the telephone is sent to the Problem Detection (PD) department. There it is analyzed and its defect is categorized. In total, there are 10 different categories of defects that the phones fixed by this company can have. Once the problem is identified, the telephone is sent to the Repair department and a letter is sent to the customer to inform him/her about the problem. The Repair (R) department has two teams. One of the teams can fix *simple* defects and the other team can repair *complex* defects. However, some of the defect categories can be repaired by both teams. Once a repair employee finishes working on a phone, this device is sent to the Quality Assurance (QA) department. There it is analyzed by an employee to check if the defect was indeed fixed or not. If the defect is not repaired, the telephone

is again sent to the Repair department. If the telephone is indeed repaired, the case is archived and the telephone is sent to the customer. To save on throughput time, the company only tries to fix a defect a limited number of times. If the defect is not fixed, the case is archived anyway and a brand new device is sent to the customer.

1.4 Getting Started

To prepare for the next sections, you need to do the following:

1. Install ProM 6. See the "ProM 6 Getting Started" document in this booklet.
2. Download a log file and the default LTL model for the running example. These are:
   - repairExample.xes, which is contained in repairExample.xes.zip[2] and
   - standartd.ltl, which is contained in standard.ltl.zip[3]

[2] `http://prom.win.tue.nl/research/wiki/_media/tutorial/repairexample.xes.zip`.
[3] `http://prom.win.tue.nl/research/wiki/_media/tutorial/standard.ltl.zip`.

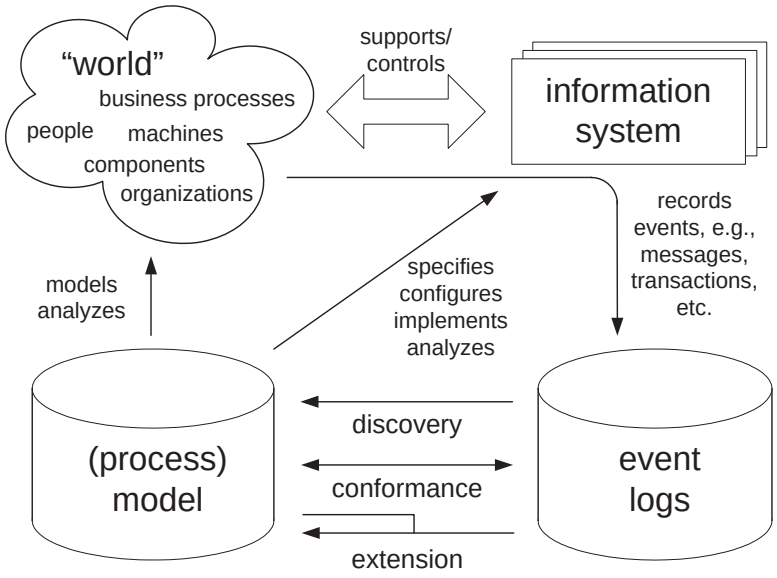

Figure 2: Sources of information for process mining.

Question	Section
How many cases (or process instances) are in the log? How many tasks (or audit trail entries) are in the log? How many resources are in the log? Are there running cases in the log? Which resources work on which tasks?	2.1
How can I filter the log so that only completed cases are kept? How can I see the result of my filtering? How can I save the pre-processed log so that I do not have to redo work?	2.2

Table 2: Log Pre-Processing: questions and pointers to answers.

2 Inspecting and Cleaning an Event Log

Before applying any mining technique to an event log, we recommend you to first get an idea of the information in this event log. The main reason for this is that you can only answer certain questions if the data is in the log. For instance, you cannot calculate the throughput time of cases if the log does not contain information about the dates and times on which tasks were executed. Additionally, you may want to remove unnecessary information from the log before you start the mining. For instance, you may be interested in mining only information about the cases that have completed. For our running example (cf. Section 1.3), all cases without an archiving task as the last one are still running cases and should not be considered. The cleaning step is usually a projection of the log to consider only the data you are interested in. Thus, in this Section we show how you can inspect and clean (or pre-process) an event log in ProM 6. Furthermore, we show how you can save the results of the cleaned log, so that you avoid redoing work.

The questions answered in this Section are summarized in Table 2. As you can see, Section 2.1 shows how to answer questions related to log inspection and Section 2.2 explains how to filter an event log and how to save your work. Note that the list of questions in Table 2 is not exhaustive, but they are enough to give you an idea of the features offered by ProM 6 for log inspection and filtering.

2.1 Inspecting the Log

The first thing you need to do to inspect or mine a log is to load it into ProM 6. In this tutorial we use the repairExample.xes log. This log has process instances of the running example described in Section 1.3.

To open this log, do the following:

1. Download the log file for the running example and save it at your computer.
2. Start ProM 6. You should get a screen like the one in Figure 3.
3. Import the log via clicking *Import...*, and select your saved copy of the log file for the running example.

Once your file has been imported as an event log, you should get a screen like the one in Figure 4. Now that the log has been imported, we can proceed with the actual log inspection. Recall that we want to answer the following questions:

1. How many cases (or process instances) are in the log?
2. How many tasks (or audit trail entries) are in the log?
3. How many resources are in the log?
4. Are there running cases in the log?
5. Which resources work on which tasks?

The *five questions* can be answered by viewing the log summary. To view this summary, have the "repairExample.xes" entry selected in your workspace (as shown by Figure 4) and select the "Visualize" button in the details view of this entry. This will open a view on the log entry. In this view, select the "Summary" tab. Can you now answer the first four questions of the list on the bottom of page 8? If so, you probably have noticed that this log has 1104 *running* cases and 1000 *completed* cases. You see that from the information in the table "MXML Legacy Classifier / End events" of the log summary (cf. Figure 5). Note that only 1000 cases end with the task "Archive Repair".

The *last question* of the list on the bottom of page 8 can be answered by checking out the "Event Name AND Resource" section of the log summary. Based on this section, can you identify which resources perform the same tasks for the running example log? If so, you probably have also noticed that there are 3 people in each of the teams in the Repair department (cf. Section 1.3) of the company[4]. The employees with login "SolverC..." deal with the complex defects, while the employees with login "SolverS..." handle the simple defects.

Take your time to inspect this log and find out more information about it. If you like, you can also inspect the *individual* cases by first clicking the "Inspector" tab.

[4]See the resources working on the tasks "Repair (Complex)" and "Repair (Simple)".

Figure 3: The workspace of ProM 6.

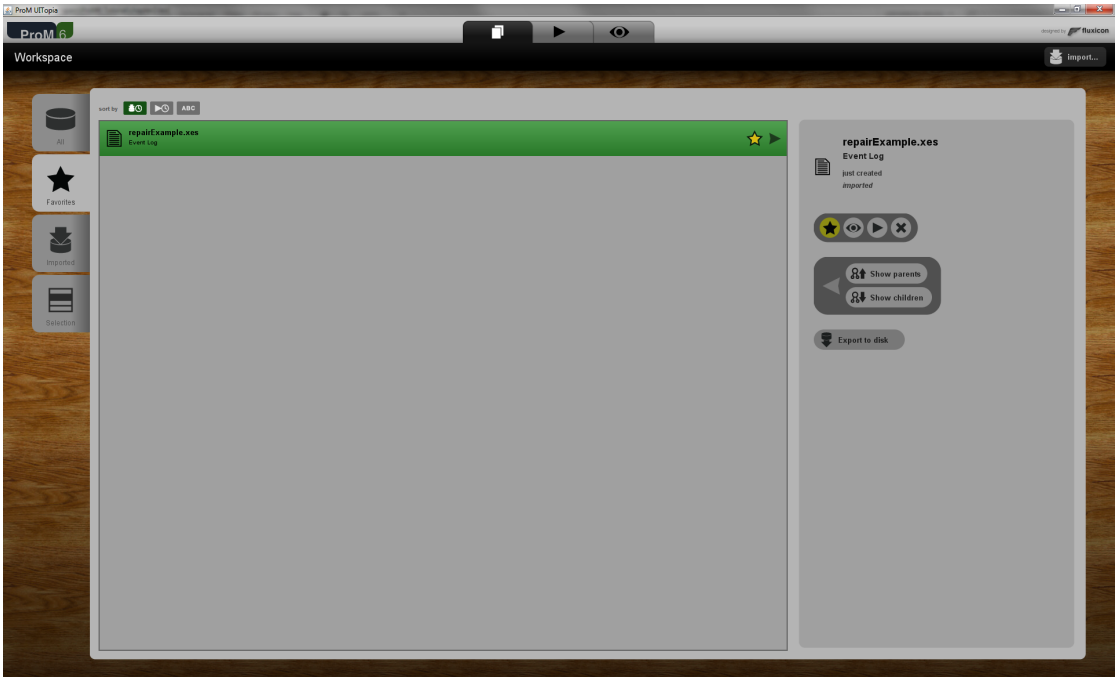

Figure 4: The example log has been imported.

2.2 Cleaning the Log

In this tutorial, we will use the process mining techniques to get insight about the process for repairing telephones (cf. Section 1.3). Since our focus in on the process *as a whole*, we will base our analysis on the *completed* process instances only. Note that it does not make much sense to talk about the most frequent path if it is not complete, or reason about throughput time of cases when some of them are still running. In short, we need to pre-process (or clean or filter) the logs.

In ProM 6, a log can be filtered by applying specific *actions*. From the description of our running example, we know that the completed cases are the ones that start with a task to *register* the phone and end with a task to *archive* the instance. Thus, to filter the completed cases, you need to execute the following procedure:

1. Select the "repairExample.xes" entry in your workspace, and select the "Action" button. This will open the *Action* view (see Figure 6) with the log preselected as an input for the action to perform. In this view, only actions that take a log as input are listed, where actions that only take a log as input are colored green (all inputs are available) and actions that require additional inputs are colored yellow (some but not all inputs are available).

2. Select the "Simple log filter" action (should be green) and select the "Start"

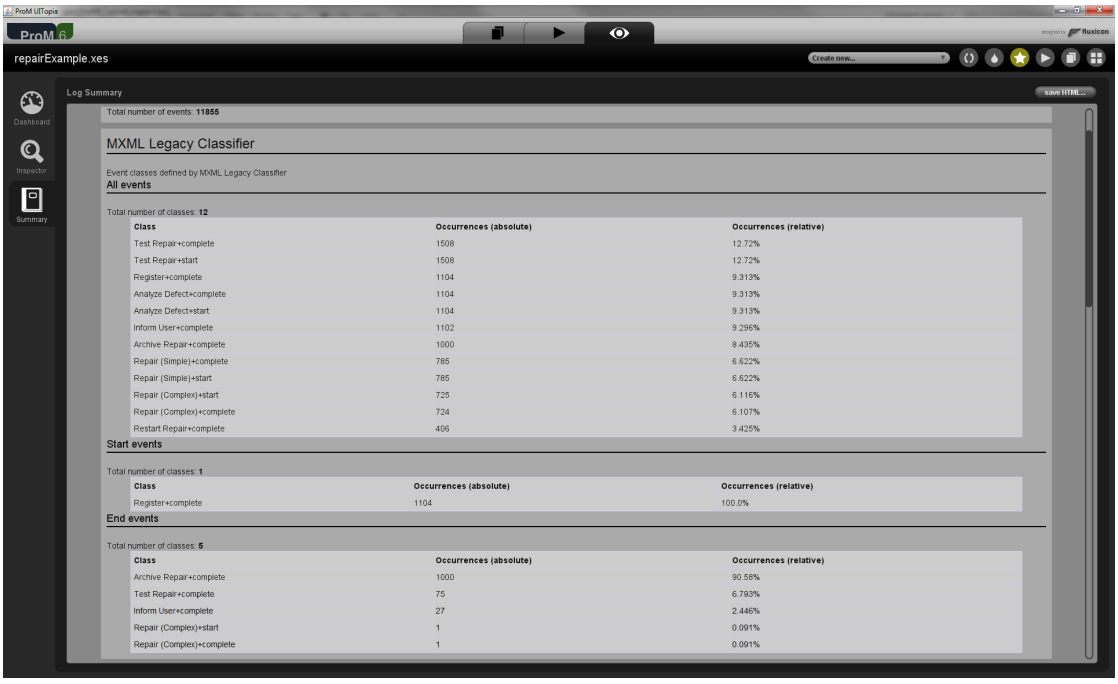

Figure 5: Summary view on the example log.

button. This will start the "Simple log filter" action on the example log. This action combines a number of log filters, that can be configured individually using a wizard.

3. The first log filter to configure is the event type filter, which allows us to select the type of events (or tasks or audit trail entries) that we want to consider while mining the log. For our running example, the log has tasks with two event types: *complete* and *start*. If you want to:

   (i) keep all tasks of a certain event, you should select the option "keep" (as it is in Figure 4),

   (ii) omit the tasks with a certain event type from a trace, select the option "remove", and

   (iii) discard all traces with a certain event type, select the option "discard instance". This last option may be useful when you have aborted cases etc.

   Options can be selected by clicking on an event type. When done, select the "Next" button.

4. The second filter is the start event filter, which filters the log in such a way that only the traces (or cases) that start with the indicated tasks are kept. The slider at the bottom allows us to select the most frequent start events. For example, if this slider is set to "80%", then the most frequent start events will be selected until at least 80% of the traces is covered. As all traces start with "Register+complete", this step is straightforward. When done, select the "Next" button.

5. The third filter is the end event filter, which filters (see Figure 7) the log in such a way that only the traces (or cases) that end with the indicated tasks are kept. The slider at the bottom allows us to select the most frequent end events. Figure 7 shows that over 80% of the traces end with "Archive repair+complete". If you want to select more end events, you can either select them manually, or use the slider. When done, select the "Next" button.

6. The fourth filter is the event filter, which filters all unselected events from the log. The slider at the bottom allows us to select the most frequent events. If, as a result, all events are removed from a trace, then the entire trace will be removed (no empty traces remain). When done, select the "Finish" button.

If you now inspect the resulting log (cf. Section 2.1), you will notice that the log contains fewer cases (Can you say how many?) and all the cases indeed start

Figure 6: Action view with the example log selected.

Figure 7: Selecting end events.

with the task "Register (complete)" and finish with the task "Archive Repair (complete)".

Although the log filters we have presented so far are very useful, they have some limitations. For instance, you cannot rename tasks (or events) in a log. For reasons like this, ProM 6 provides more powerful log filters. We strongly advise you to spent some time trying them out and getting more feeling about how they work. Our experience shows that the advanced log filters are especially useful when handling real-life logs. These filters not only allow for projecting data in the log, but also for adding data to the log.

Once you are done with the filtering, you can export the filtered log by selecting it in your workspace and selecting the "Export to disk" button. If you like, you can export the filtered log for our running example. Can you open this exported log into ProM 6? What do you notice by inspecting this log? Note that your log should only contain 1000 cases and they should all start and end with a single task.

3 Questions Answered Based on an Event Log Only

Now that you know how to inspect and pre-process an event log (cf. Section 2), we proceed with showing how to answer the questions related to the *discovery* plug-ins (cf. Figure 2). Recall that a log is the only input for these kinds of plug-ins.

The questions answered in this Section are summarized in Table 2. Section 3.1 shows how to mine the *control-flow* perspective of process models. Section 3.2 explains how to mine information regarding certain aspects of cases. Section 3.3 describes how to mine information related to the roles/employees in the event log [5]. Section 3.4 shows how to use temporal logic to verify if the cases in a log satisfy certain (required) properties.

3.1 Mining the Control-Flow Perspective of a Process

The control-flow perspective of a process establishes the dependencies among its tasks. Which tasks precede which other ones? Are there concurrent tasks? Are there loops? In short, what is the process model that summarizes the flow followed by most/all cases in the log? This information is important because it gives you feedback about how *cases are actually being executed* in the organization.

ProM 6 supports various actions to mine the control-flow perspective of process models. In this tutorial, we will use the "α-algorithm" plug-in, see Figure 8.

[5] More technically, these plug-ins require the *originator* field to be present in the event log.

Question	Section
How are the cases actually being executed?	3.1
What is the most frequent path for every process model? How is the distribution of all cases over the different paths through the process?	3.2
How many people are involved in a case? What is the communication structure and dependencies among people? How many transfers happen from one role to another role? Who are the important people in the communication flow? Who subcontract work to whom? Who work on the same tasks?	3.3
Are the rules indeed being obeyed?	3.4

Table 3: Discovery plug-ins: questions and pointers to answers.

Figure 8 shows that this plug-in takes an "Event Log" as input, and it produces a "Petri net" and a "Marking" (the initial marking of the Petri net) as output. To mine the log of our running example, you should perform the following steps:

1. Open the filtered log that contains only the completed cases (cf. Section 2.2), or redo the filtering for the original log of the running example.

2. Verify with the analysis plug-in *Log Summary* if the log is correctly filtered. If so, this log should contain 1000 process instances, 12 event classes, 1 start event ("Register+complete"), 1 end event ("Archive Repair+complete"), and 13 originators.

3. Run the "α-algorithm" plug-in:

   (a) Select the filtered log in the workspace view.

   (b) Select the "Action" button. This will open the action view with the filtered log selected as input.

   (c) Select the "α-algorithm" plug-in.

   (d) Select the "Start" button.

4. The resulting mined model should look like the one in Figure 9. Note that the "α-algorithm" plug-in uses Petri nets[6] as its notation to represent process models. From this mined model, you can observe that:

[6]Different *mining* plug-ins can work with different notations, but the main idea is always the

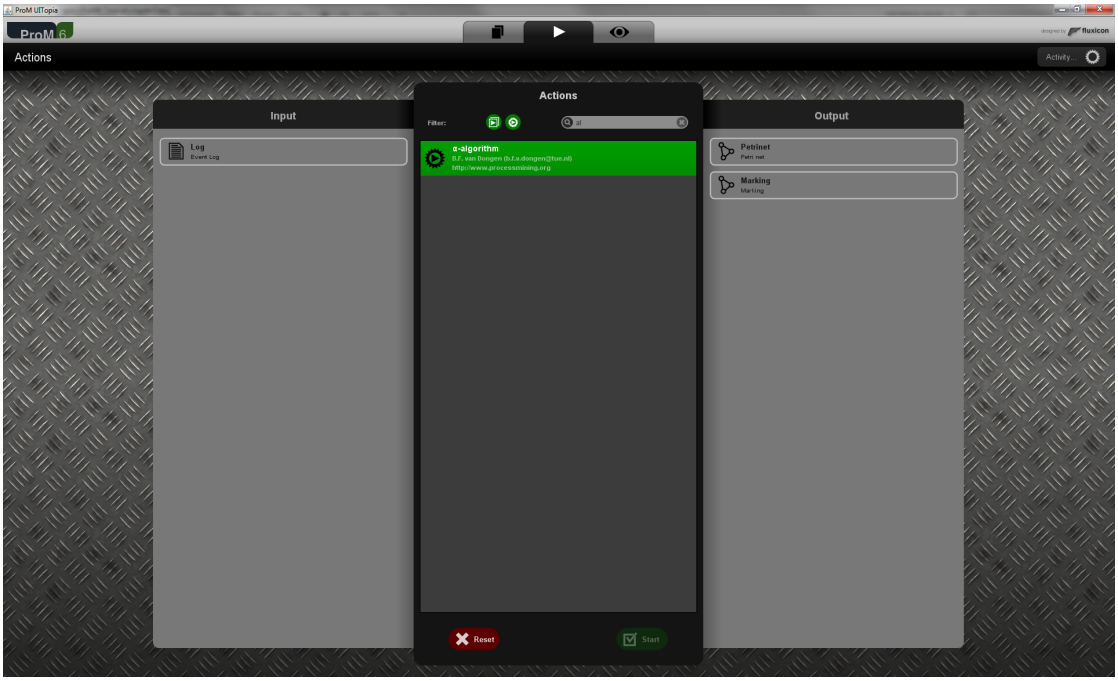

Figure 8: "α-algorithm" action.

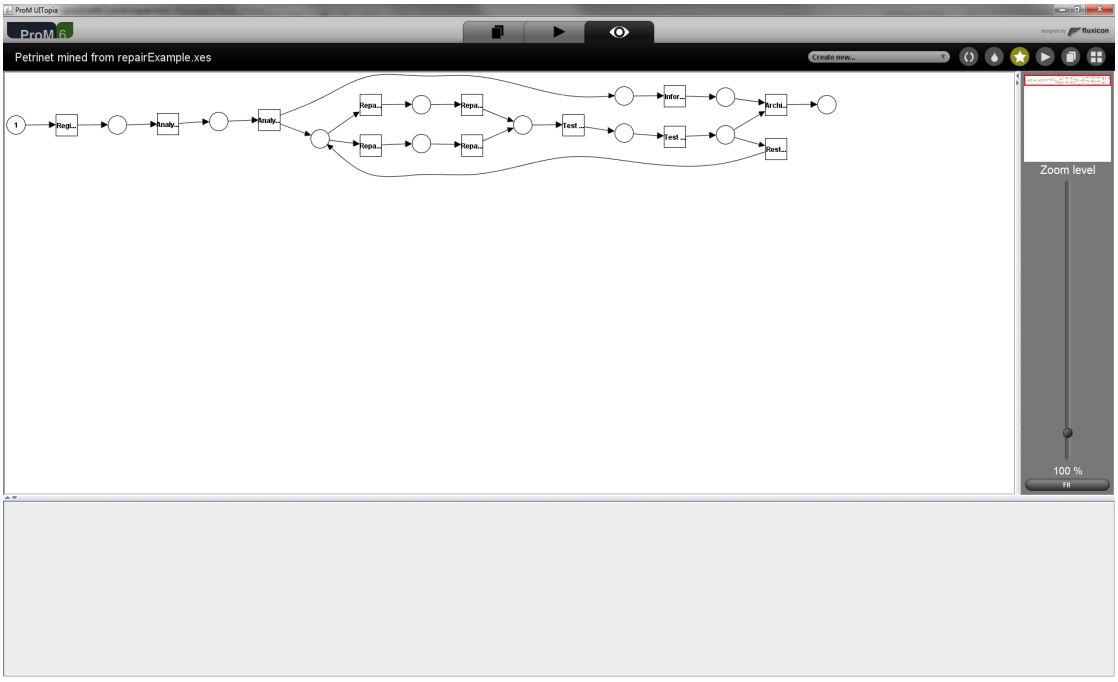

Figure 9: Mined model for the log of the running example.

- All cases start with the task "Register" and end with the task "Archive Repair". This is not really surprising since we have filtered the cases in the log.
- After the task *Analyze Defect* completes, some tasks can occur in parallel:
  (i) the client can be informed about the defect (see task "Inform User"), and
  (ii) the actual fix of the defect can be started by executing the task *Repair (Complete)* or *Repair (Simple)*.
- The model has a loop construct involving the repair tasks.

Based on these remarks, we can conclude that the cases in our running example log have indeed been executed as described in Section 1.3.

As a final note, although in this section we mine the log using the "α-algorithm" plug-in, we strongly recommend you to try other plug-ins as well. The main reason is that the "α-algorithm" plug-in is not robust to logs that contain noisy data (like real-life logs typically do). Thus, we suggest you have a look at the help of the other plug-ins before choosing for a specific one. In our case, we can hint that we have had good experience while using the mining plug-ins "Flexible Heuristics Miner", "Genetic Miner", and "Fuzzy Miner" to real-life logs.

3.2 Mining Case-Related Information about a Process

In this section we answer questions regarding the execution patterns in the event log. A few questions are:

- What are the most frequent paths in the process?
- Are there any loop patterns in the process?
- What is the distribution of all cases over the different paths through the process?
- Can I select a subset of traces where particular paths were executed?
- Can I simplify the log by abstracting the most frequent paths?

---

same: portray the dependencies between tasks in a model. Furthermore, the fact that different mining plug-ins may work with different notations does not prevent the interoperability between these representations because ProM 6 offers actions that translate models from one notation to another.

These and other related questions can be answered by using the *Pattern Abstractions* visualization. As an illustration, in the context of our running example, one would expect that paths without the task *Restart Repair* (i.e., situations in which the defect could not be fixed in the first try) should be more frequent than the ones with this task. But is this indeed the current situation? Questions like this will be answered while executing the following procedure:

1. Open the filtered log that contains only the completed cases (cf. Section 2.2), or redo the filtering for the original log of the running example.

2. Run the *Pattern Abstractions* visualizer:

   - Select the filtered log in the workspace view.
   - Select the "Visualizer" button. By default, you will notice the "Log summary" visualizer. Change the visualizer to "Pattern Abstractions" as shown in Figure 10.

   You should get a screen like the one in Figure 11.

3. Configure the parameters and execute the actions for the visualizer as mentioned below

   - Choose the "loop patterns" and click "Find Patterns". In order to find frequent paths not involving loops, choose "(Maximal) Repeat Patterns".
   - Click the "Find Pattern Frequency" button
   - Uncheck the "Ignore Duplicate Traces" checkbox in the "Filter Patterns" panel

   You should get a screen like the one in Figure 12. You can observe that the activity *Restart Repair* is involved in two loop patterns, one with "Repair (Simple)" and the other with "Repair (Complex)" procedures in conjunction with "Test Repair". The value in the instance count (%) column gives the distribution of a particular pattern. For example, you can notice that in 20% of the cases, the *Restart Repair* is required following a "Repair (Simple)" procedure and in 7% of the cases where "Repair (Complex)" procedure was performed, a *Restart Repair* was required. This also implies that in 73% of the cases, the defects could be fixed in the first attempt.

4. Suppose, you want to extract the cases where a *Restart Repair* was required after a "Repair (Complex)" procedure was performed. You can extract those cases by doing the following

Figure 10: Selecting Pattern Abstractions Visualizer

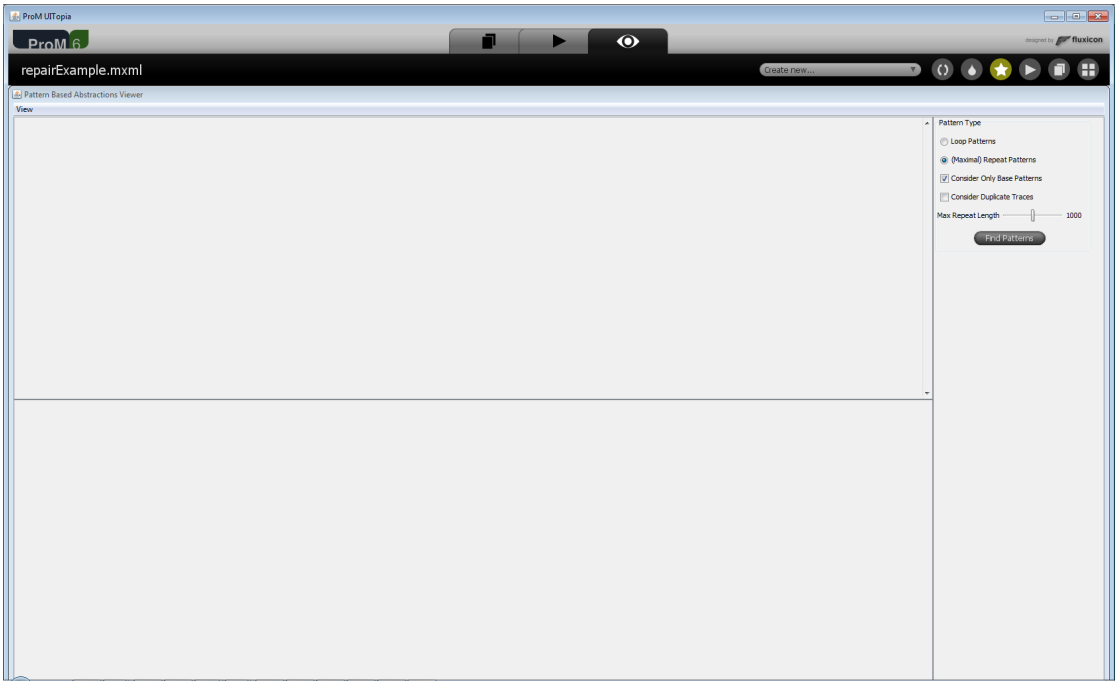

Figure 11: Pattern Abstractions Visualizer

- Click “Find Abstractions” Button. You should get a screen like the one in Figure 13.
- In the top left panel, select the check box against the pattern (path) of your interest. In this case, against “Repair (Complex)”, “Test Repair” and “Restart Repair”.
- Click “Export Selected (Log)” button. An event log containing only the cases where this particular path is executed will be pushed onto the workspace. You should be able to see the log with the name “AbstractionLog_[Repair (Complex), Test Repair]”.

5. You can also use this visualizer to abstract frequent paths. In order to do this, define meaningful names for the patterns in the top left panel obtained after clicking “Find Abstractions”. Then click “Transform Log” button. This also creates sub-logs that capture the manifestation of the frequent paths. The fuzzy miner plug-in can make use of these abstractions to mine hierarchical process models. You can iterate over this transformation process to any number of steps thereby simplifying the log to a desired granularity.

Take your time to have a look at the other options provided by this visualizer. For instance, by configuring the *Filter options* you can select specific mined patterns etc.

3.3 Mining Organizational-Related Information about a Process

In this section we answer questions regarding the social (or organizational) aspect of a company. The questions are:

- How many people are involved in a specific case?
- What is the communication structure and dependencies among people?
- How many transfers happen from one role to another role?
- Who are important people in the communication flow? (the most frequent flow)
- Who subcontracts work to whom?
- Who work on the same tasks?

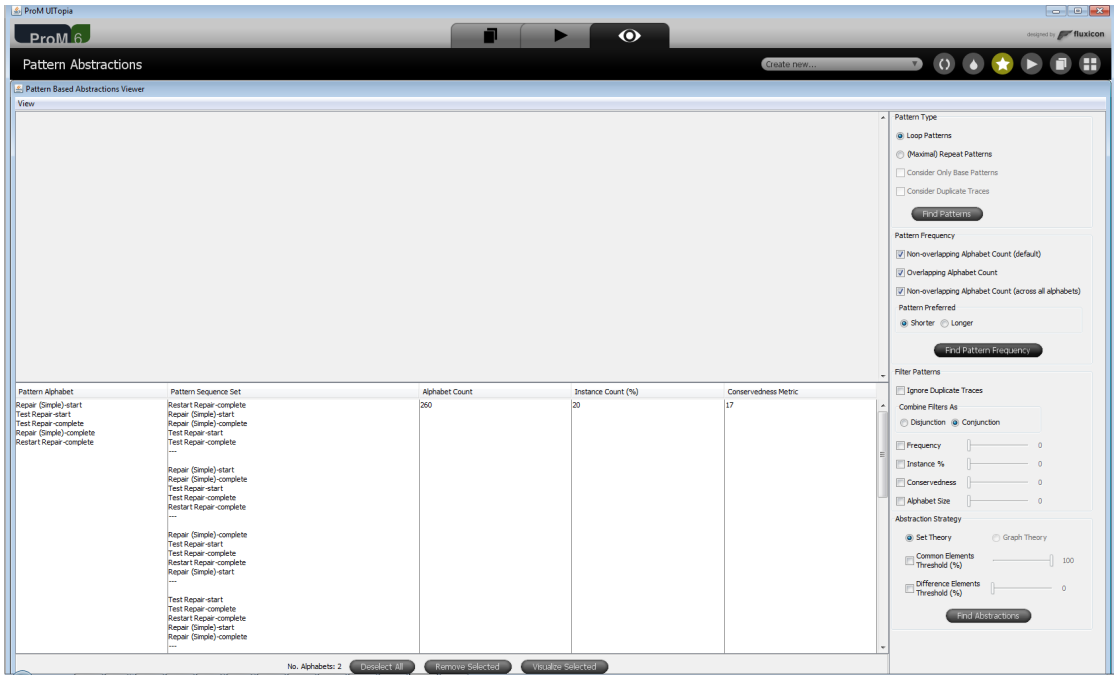

Figure 12: Loop patterns in the running example

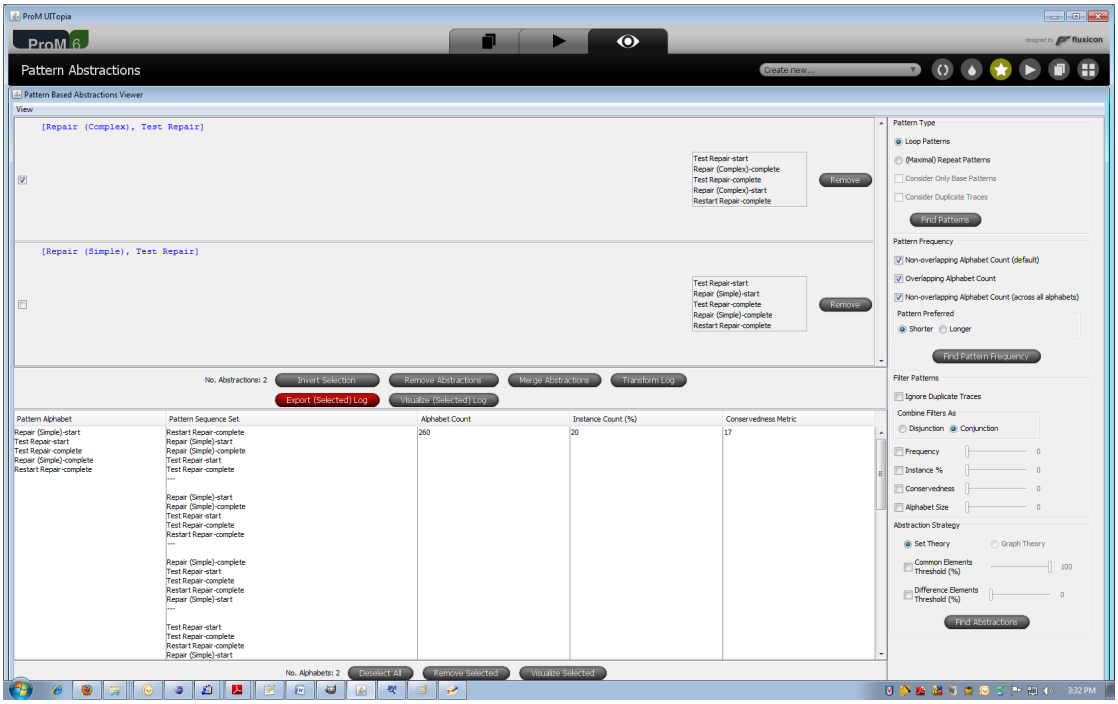

Figure 13: Selecting the cases where a particular path is executed

These and other related questions can be answered by using the *Social Network Miner* plug-ins. In the following we explain how to answer each question in the context of our running example.

To know the *people that are involved in a specific case or all the cases in the log*, you can use the *Log Summary* (cf. Section 2.1). For instance, to check which people are involved in the process instance *120* of our example log, you can do the following:

1. Open the filtered log (cf. Section 2.2) for the running example.
2. Select the "Inspector" tab.
3. Select the "Browser" tab.
4. In the "Instances" pane, locate the instance with ID 120, and select it.

The remaining questions of the list on page 20 are answered by using the *Social Network* plug-ins. For instance, in the context of our running example, we would like to check if there are employees that outperform others. By identifying these employees, one can try to make the good practices (or way of working) of these employees a common knowledge in the company, so that peer employees also benefit from that. In the context of our running example, we could find out which employees are better at fixing defects. From the process description (cf. Section 1.3) and from the mined model in Figure 9, we know that telephones which were not repaired are again sent to the Repair Department. So, we can have a look on the *handover of work* for the task performed by the people in this department. In other words, we can have a look on the handover of work for the tasks *Repair (Simple)* and *Repair (Complex)*. One possible way to do so is to perform the following steps:

1. Open the log for the running example.
2. Use the "Simple log filter" plug-in (cf. Section 2.2) to filter the log so that only the task "Repair (Simple)+start", "Repair (Simple)+complete", "Repair (Complex)+start" and "Repair (Complex)+complete" are kept. (Hint: Use the *Log Summary* to check if the log is correctly filtered!).
3. Run the "Social Network (HoW) miner" (HoW is a shorthand for handover of work) on the filtered log:
   (a) Select the filtered log in the workspace view.
   (b) Select the "Action" button. This opens the action view with the filtered log selected as input.

(c) Select the “Social Network (HoW) miner” plug-in.

(d) Select the “Start” button.

4. Keep the default settings and select ”Continue”.

5. Select the options “size by ranking”, “stretch by degree ratio”, and set *Mouse Mode* to “Picking” (so you can use the mouse to re-arrange the nodes in the graph). You should get a result like the one in Figure 14. The resulting graph shows which employees handed over work to other employees in the process instances of our running example. By looking at this graph, we can see that the employees with roles “SolverS3” (bottom-left node) and “SolverC3” (top-right node) outperform the other employees because the telephones these two employees fix always pass the test checks and, therefore, are not re-sent to the Repair Department (since no other employee has to work on the cases involving “SolverS3” and “SolverC3”). The oval shape of the nodes in the graph visually expresses the relation between the *in* and *out* degree of the connections (arrows) between these nodes. A higher proportion of ingoing arcs lead to more vertical oval shapes while higher proportions of outgoing arcs produce more horizontal oval shapes. From this remark, can you tell which employee has more problems to fix the defects?

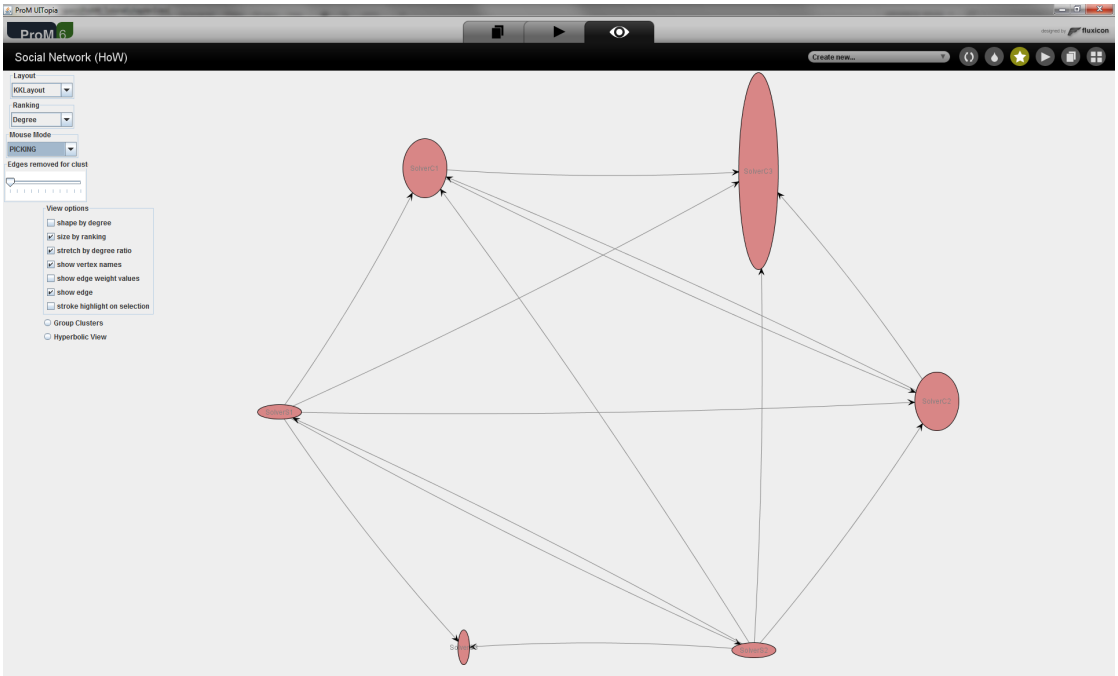

Figure 14: Handover of work for Solvers.

Take you time to experiment with the plug-ins explained in the procedure above. Can you now answer the other remaining questions?

3.4 Verifying Properties in an Event Log

It is often the case that processes in organizations should obey certain rules or principles. One common example is the "Four-eyes principle" which determines that a same person cannot execute certain tasks (e.g., make the budget and approve it). These kinds of principles or rules are often used to ensure quality of the delivered products and/or to avoid frauds. One way to check if these rules are indeed being obeyed is to audit the log that registers what has happened in a organization. In ProM 6, auditing of a log is provided by the "LTL Checker" plug-in[7].

From the description of our running example (cf. Section 1.3), we know that after a try to fix the defect, the telephone should be tested to check if it is indeed repaired. Thus, we could use the "LTL Checker" plug-in to verify the property: *Does the task "Test Repair" always happen* after *the tasks "Repair (Simple)" and* before *the task "Archive Repair"?* We do so by executing the following procedure:

1. Open the filtered log (cf. Section 2.2) for the running example.
2. Open the default LTL model (that is, import "standard.ltl").
3. Run the "LTL Checker" plug-in on the filtered log and the default LTL model:
   (a) Select the default LTL model in the workspace view.
   (b) Select the "Action" button. This opens the action view with the default LTL model selected as input.
   (c) Select the "LTL Checker" plug-in.
   (d) Select the "Event Log" input of the "LTL Checker" plug-in, and select the filtered log for this.
   (e) Select the "Start" button.

   You should get a screen like the one in Figure 15.
4. Select the formula "eventually_activity_A_then_B_then_C" (see also Figure 15).
5. Give as values (see also Figure 15):
   (i) activity A = *Repair (Simple)*,

[7]LTL stands for Linear Temporal Logic.

(ii) activity B = *Test Repair* and

(iii) activity C = *Archive Repair*.

Note that the LTL plug-in is case sensitive. So, make sure you type in the task names as they appear in the log.

6. Select the "Finish" button.

7. Select the "Instances" tab. This tab should show the log split into two parts: one with the cases that satisfy the property (or formula) and another with the cases that do not satisfy the property.

Take your time to experiment with the LTL plug-in. Can you identify which pre-defined formula you could use to check for the "Four-eyes principle"?

4 Conclusions

This tutorial showed how to use the different ProM 6 plug-ins to answer common questions about process models (cf. Section 1.1). Since our focus was on this set of questions, we have not covered many of the other plug-ins that are in ProM 6.

Figure 15: The "LTL Checker" plug-in.

We hope that the subset we have shown in this tutorial will help you in finding your way in ProM 6. ProM 6 can be downloaded at www.processmining.org.

References

[1] COSA Business Process Management. *http://www.cosa-bpm.com/*.

[2] SAP. *http://www.sap.com/*.

[3] Staffware Process Suite. *http://www.staffware.com/*.

[4] W.M.P. van der Aalst, B.F. van Dongen, C.W. Günther, R.S. Mans, A.K. Alves de Medeiros, A. Rozinat, V. Rubin, M. Song, H.M.W. Verbeek, and A.J.M.M. Weijters. ProM 4.0: Comprehensive Support for Real Process Analysis. In J. Kleijn and A. Yakovlev, editors, *Application and Theory of Petri Nets and Other Models of Concurrency (ICATPN 2007)*, volume 4546 of *LNCS*, pages 484–494. Springer-Verlag, Berlin, 2007.

[5] W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2002.

[6] Workflow Management Coalition. WFMC Home Page. http://www.wfmc.org.

[7] J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1995.

[8] M. Dumas, W.M.P. van der Aalst, and A.H. ter Hofstede, editors. *Process-Aware Information Systems: Bridging People and Software Through Process Technology*. John Wiley & Sons Inc, 2005.

[9] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[10] W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.

[11] H.M.W. Verbeek, B.F. van Dongen, J. Mendling, and W.M.P. van der Aalst. Interoperability in the ProM Framework. In T. Latour and M. Petit, editors, *Proceedings of the CAiSE'06 Workshops and Doctoral Consortium*, pages 619–630, Luxembourg, June 2006. Presses Universitaires de Namur.