ProM Framework Tutorial

Authors:

Ana Karla Alves de Medeiros
(a.k.medeiros@tm.tue.nl)

A.J.M.M. (Ton) Weijters
(a.j.m.m.weijters@tm.tue.nl)

Technische Universiteit Eindhoven
Eindhoven, The Netherlands

November 2006

Contents

Chapter 1

Introduction

This document shows how to use the ProM tool 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 the ProM tool. This tool is open-source and 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.

1.1 Common Questions

The questions that managers usually have about processes in organizations are:

1. What is the most frequent path for every process model?
2. How is the distribution of all cases over the different paths through the process?
3. How compliant are the cases (i.e. process instances) with the deployed process models? Where are the problems? How frequent is the (non-) compliance?
4. What are the routing probabilities for each split task (XOR or OR split/join points)?
5. What is the average/minimum/maximum throughput time of cases?
6. Which paths take too much time on average? How many cases follow these routings? What are the critical sub-paths for these paths?
7. What is the average service time for each task?
8. How much time was spent between any two tasks in the process model?

9. How are the cases actually being executed?
10. What are the business rules in the process model?
11. Are the rules indeed being obeyed?
12. How many people are involved in a case?
13. What is the communication structure and dependencies among people?
14. How many transfers happen from one role to another role?
15. Who are important people in the communication flow? (the most frequent flow)
16. Who subcontract work to whom?
17. Who work on the same tasks?

We show how to use ProM to answer these questions in chapters 3 and 4.

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) [4, 5], 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 1.1. 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

Case ID	Task Name	Event Type	Originator	Timestamp	Extra Data
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 1.1: Example of an event log.

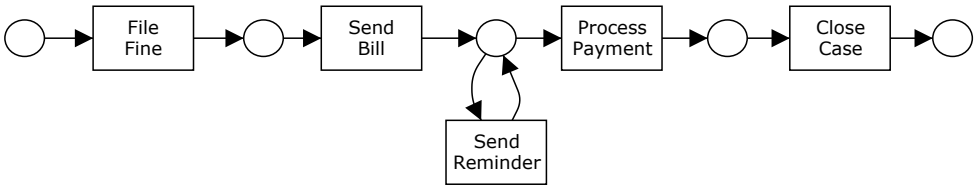


Figure 1.1: Petri net illustrating the control-flow perspective that can be mined from the event log in Table 1.1.

helps in auditing, analyzing and improving already enacted business processes. The main benefit of process mining techniques is that information is *objectively* compiled. In 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 the log (i) 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 1.1 has this data (cf. fields "Case ID", "Task Name" and "Timestamp"). So, for this log, mining algorithms

could discover the process in Figure 1.1[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 1.1 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. When 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 the ProM tool.

The ProM framework [7, 11] is an open-source tool specially tailored to support the development of process mining plug-ins. This tool is currently at version 4.0 and 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 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 1.2 illustrates how we "categorize" these plug-ins. The plug-ins based on data in the event log only are called *discovery* plug-ins because they do not use any existing

[1]The reader unfamiliar with Petri nets is referred to [6, 9, 10].

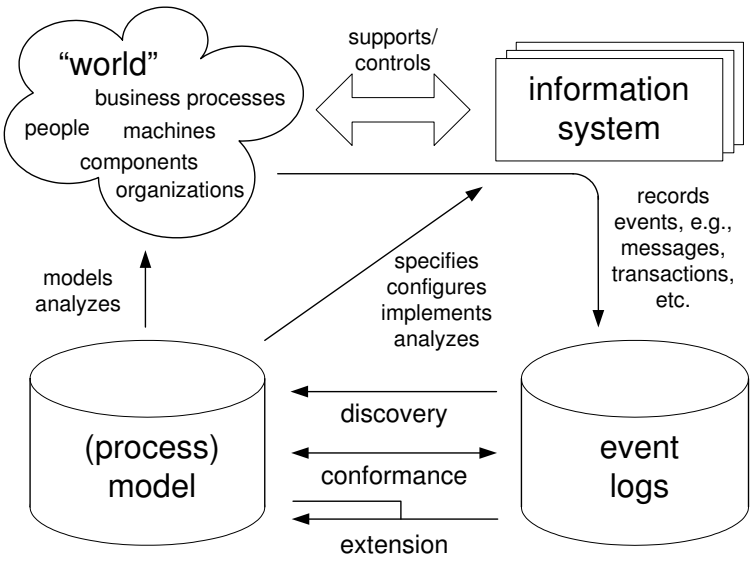


Figure 1.2: Sources of information for process mining. The *discovery* plug-ins use only an event log as input, while the *conformance* and *extension* plug-ins also need a (process) model as input.

information about deployed models. 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. 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 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 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 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.

Chapter 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 times (timestamp) in 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 are 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 chapter we show how you can inspect and clean (or pre-process) an event log in ProM. Furthermore, we show how you can save the results of the cleaned log, so that you avoid redoing work.

The questions answered in this chapter are summarized in Table 2.1. 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 2.1 is not exhaustive, but they are enough to give you an idea of the features offered by ProM 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. In this tutorial we use the log at the location "www.processmining.org/tuto-

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 originators are in the log? Are there running cases in the log? Which originators work in 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 2.1: Log Pre-Processing: questions and pointers to answers.

rial/repairExample.zip". 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 for the running example and save it at your computer.
2. Start the ProM framework. You should get a screen like the one in Figure 2.1. Note that the ProM menus are context sensitive. For instance, since no log has been opened yet, no mining algorithm is available.
3. Open the log via clicking *Open→Open MXML log*, and select your saved copy of the log file for the running example. Once your log is opened, you should get a screen like the one in Figure 2.2. Note that now more menu options are available.

Now that the log is opened, 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 originators are in the log?
4. Are there running cases in the log?
5. Which originators work in which tasks?

The *first four questions* can be answered by the analysis plug-in *Log Summary*. To call this plug-in, choose *Analysis→[log name...]→Log Summary*. 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 104 *running* cases and 1000 *completed* cases. You see that from the information in the table "Ending Log Events" of the log summary (cf. Figure 2.3). Note that only

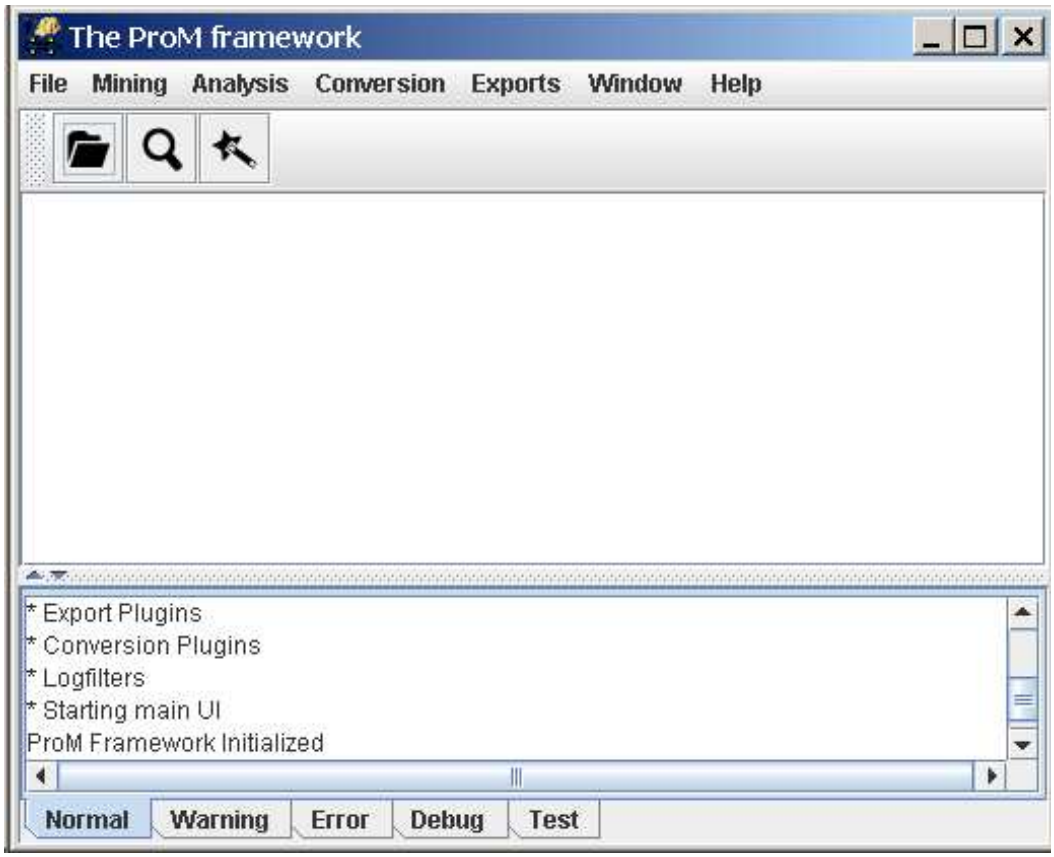


Figure 2.1: Screenshot of the main interface of ProM. The menu *File* allows to open event logs and to import models into ProM. The menu *Mining* provides the mining plug-ins. These mining plug-ins mainly focus on discovering information about the control-flow perspective of process models or the social network in the log. The menu *Analysis* gives access to different kinds of analysis plug-ins for opened logs, imported models and/or mined models. The menu *Conversion* provides the plug-ins that translate between the different notations supported by ProM. The menu *Exports* has the plug-ins to export the mined results, filtered logs etc.

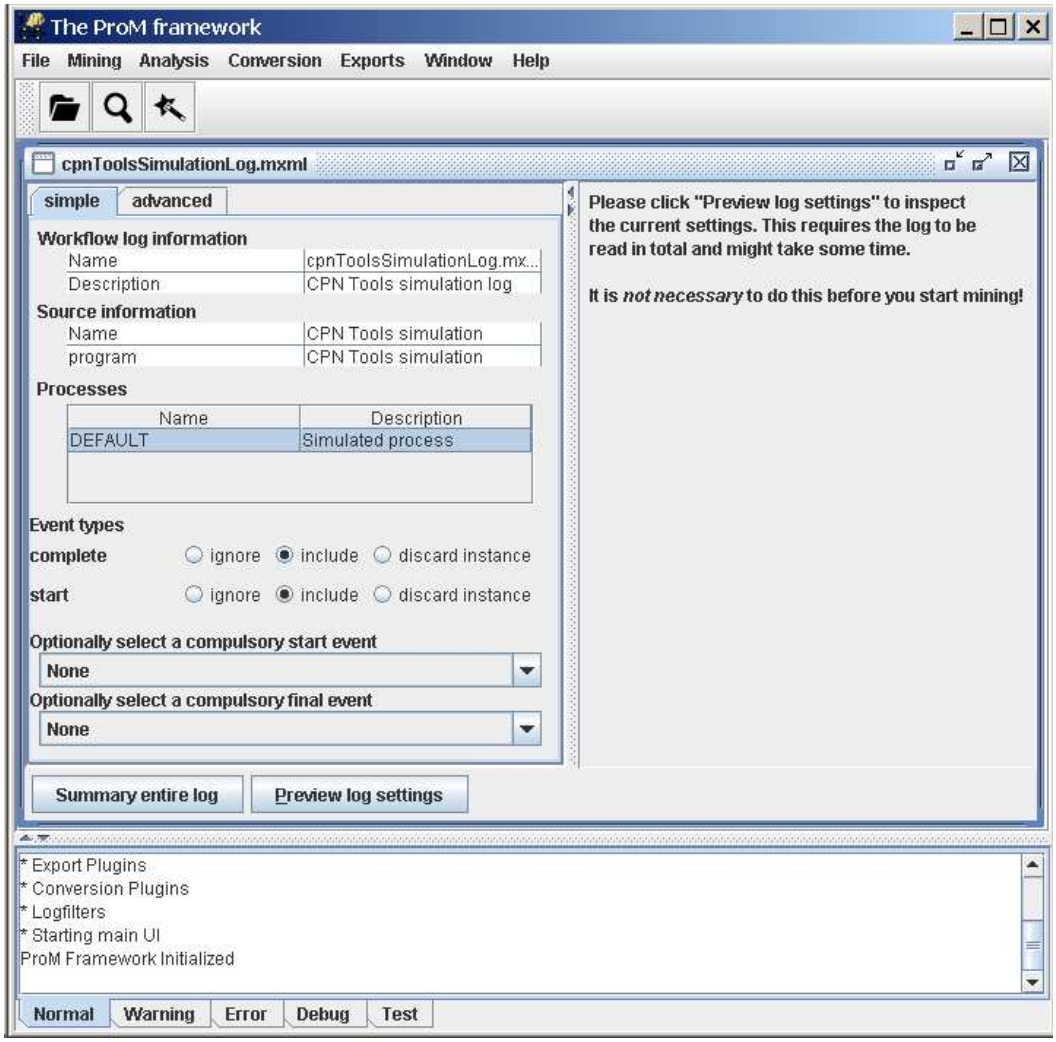


Figure 2.2: Screenshot of the ProM after the log of the running example (cf. Section 1.3) has been opened.

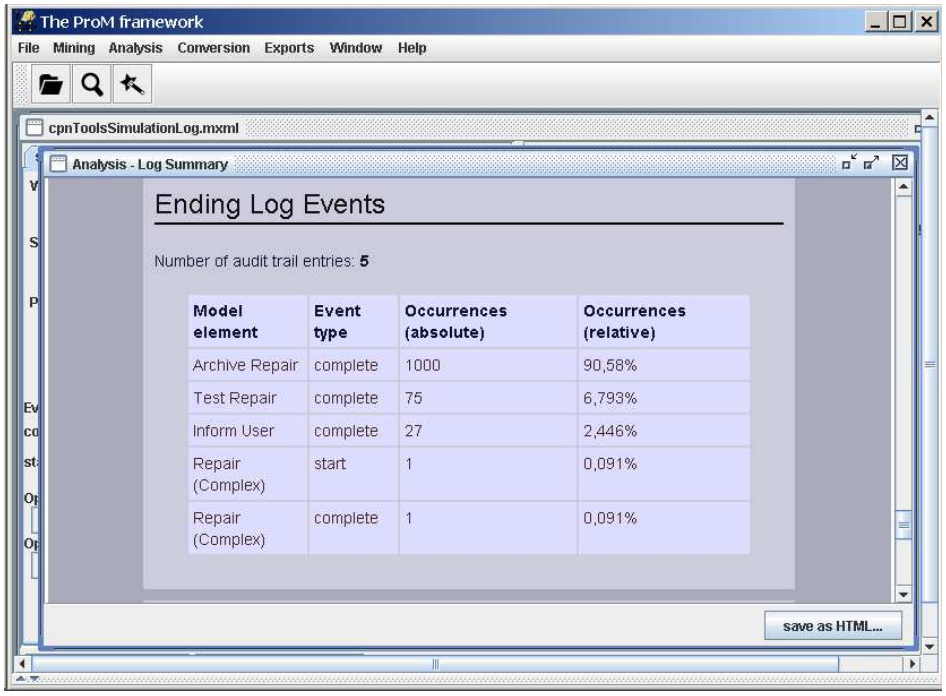
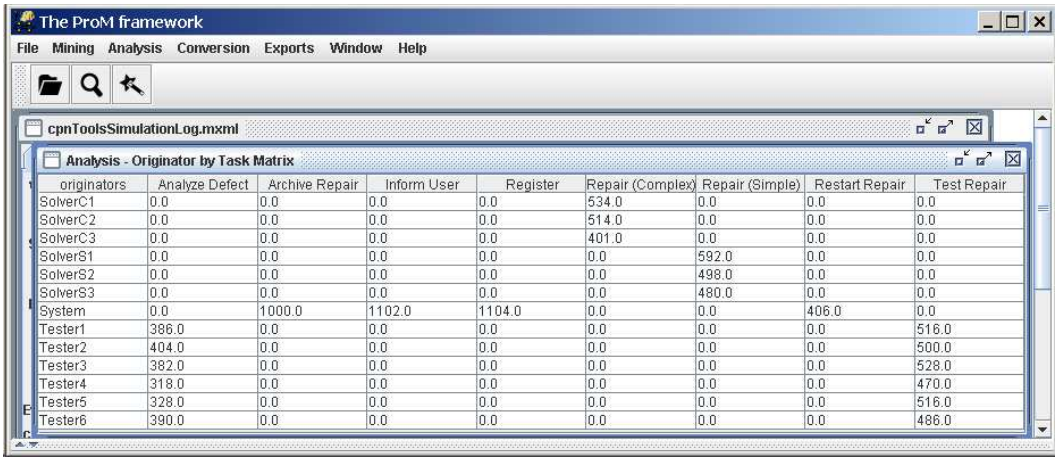


Figure 2.3: Excerpt of the result of the analysis plug-in *Log Summary*.

1000 cases end with the task "Archive Repair". The *last question* of the list on the bottom of page 8 can be answered by the analysis plug-in *Originator by Task Matrix*. This plug-in can be started by clicking the menu *Analysis→[log name...]→Originator by Task Matrix*. Can you identify which originators 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 [1]. 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 with these two analysis plug-ins and find out more information about it. If you like, you can also inspect the *individual* cases by clicking the button *Preview log settings* (cf. bottom of Figure 2.2) and then double-clicking on a specific process instance.

[1] See the originators working on the tasks "Repair (Complex)" and "Repair (Simple)".



originators	Analyze Defect	Archive Repair	Inform User	Register	Repair (Complex)	Repair (Simple)	Restart Repair	Test Repair
SolverC1	0.0	0.0	0.0	0.0	534.0	0.0	0.0	0.0
SolverC2	0.0	0.0	0.0	0.0	514.0	0.0	0.0	0.0
SolverC3	0.0	0.0	0.0	0.0	401.0	0.0	0.0	0.0
SolverS1	0.0	0.0	0.0	0.0	0.0	592.0	0.0	0.0
SolverS2	0.0	0.0	0.0	0.0	0.0	498.0	0.0	0.0
SolverS3	0.0	0.0	0.0	0.0	0.0	480.0	0.0	0.0
System	0.0	1000.0	1102.0	1104.0	0.0	0.0	406.0	0.0
Tester1	386.0	0.0	0.0	0.0	0.0	0.0	0.0	516.0
Tester2	404.0	0.0	0.0	0.0	0.0	0.0	0.0	500.0
Tester3	382.0	0.0	0.0	0.0	0.0	0.0	0.0	528.0
Tester4	318.0	0.0	0.0	0.0	0.0	0.0	0.0	470.0
Tester5	328.0	0.0	0.0	0.0	0.0	0.0	0.0	516.0
Tester6	390.0	0.0	0.0	0.0	0.0	0.0	0.0	486.0

Figure 2.4: Scheenshot with the result of the analysis plug-in *Originator by Task Matrix*.

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, a log can be filtered by applying the provided *Log Filters*. In Figure 2.2 you can see three log filters (see bottom-left of the panel with the log): *Event Types*, *Start Event* and *End Event*. The *Event Types* log filter 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 "include" (as it is in Figure 2.2), (ii) omit the tasks with a certain event type from a trace, select the option "ignore", 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. The *Start Event* filters the log so that only the traces (or cases) that start with the indicated task are kept. The *End Event* works in a similar way, but the filtering is done with respect to the final task in the log trace.

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. Keep the event types selection as in Figure 2.2. I.e., "include" all the *complete* and *start* event types;
2. Select the task "Register (complete)" as the *compulsory start event*;
3. Select the task "Archive Repair (complete)" as the *compulsory final event*.

If you now inspect the log (cf. Section 2.1), for instance, by calling the analysis plug-in *Log Summary*, you will notice that the log contains fewer cases (Can you say how many?) and all the cases indeed start 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 can only specify one task as the start task for cases. It would be handy to have more flexibility, like saying "Filter all the cases that start with task X or task Y". For reasons like that, the *advanced* tab of the panel with the log (cf. Figure 2.5) provides more powerful log filters. Each log filter has a *Help*, so we are not going into details about it. However, we strongly advise you to spent some time trying them on 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. For instance, the log filters *Add Artificial Start Task* and *Add Artificial End Task* support the respective addition of tasks in the beginning and ending of traces. These two log filters are handy when applying process mining algorithms the assume the target model to have a single start/end point.

Once you are done with the filtering, you can save your results in two ways:

1. Export the filtered log by choosing the export plug-in *XML log file*. This will save a copy of the log that contains all the changes made by the application of the log filters.
2. Export the configured log filters themselves by choosing the export plug-in *Log Filter (advanced)*. Exported log filters can be imported into ProM at a later moment and applied to a (same) log. You can import a log filter by selecting *File→[log name...]rightarrowOpen Log Filter (advanced)*.

If you like, you can export the filtered log for our running example. Can you open this exported log into ProM? What do you notice by inspecting this

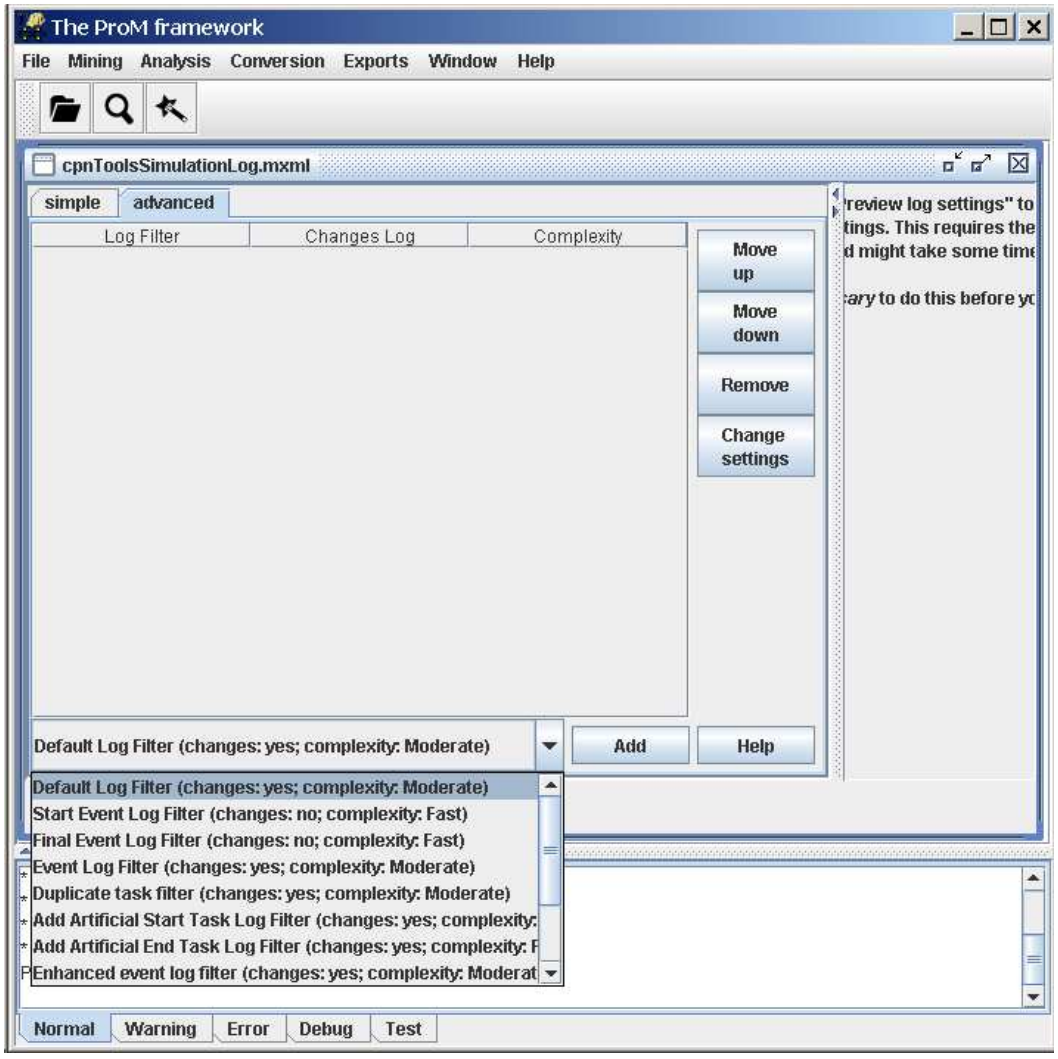


Figure 2.5: Scheenshot of the *Advanced Log Filters* in ProM.

log? Note that your log should only contain 1000 cases and they should all start and end with a single task.

Chapter 3

Questions Answered Based on an Event Log Only

Now that you know how to inspect and pre-process an event log (cf. Chapter 2), we proceed with showing how to answer the questions related to the *discovery* ProM plug-ins (cf. Figure 1.2). Recall that a log is the only input for these kinds of plug-ins.

The questions answered in this chapter are summarized in Table 2.1. 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 [1]. 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 supports various plug-ins to mine the control-flow perspective of process models, as shown in Figure 3.1. In this tutorial, we will use the

[1] 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 3.1: Discovery Plug-ins: questions and pointers to answers.

mining plug-in *Alpha algorithm plugin*. Thus, 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 audit trail entries, 1 start event ("Register"), 1 end event ("Archive Repair"), and 13 originators.
3. Run the *Alpha algorithm plugin* by choosing the menu *Mining→[log name...]→Alpha algorithm plugin* (cf. Figure 3.1).
4. Click the button *Start mining*. The resulting mined model should look like the one in Figure 3.2. Note that the *Alpha algorithm plugin* uses Petri nets[2] as its notation to represent process models. From this mined model, you can observe that:
   - All cases start with the task "Register" and finish with the task "Archive Repair". This is not really surprising since we have filtered the cases in the log.

[2]Different *mining* plug-ins can work with different notations, but the main idea is always the 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 the ProM tool offers *conversion* plug-ins that translate models from one notation to another.

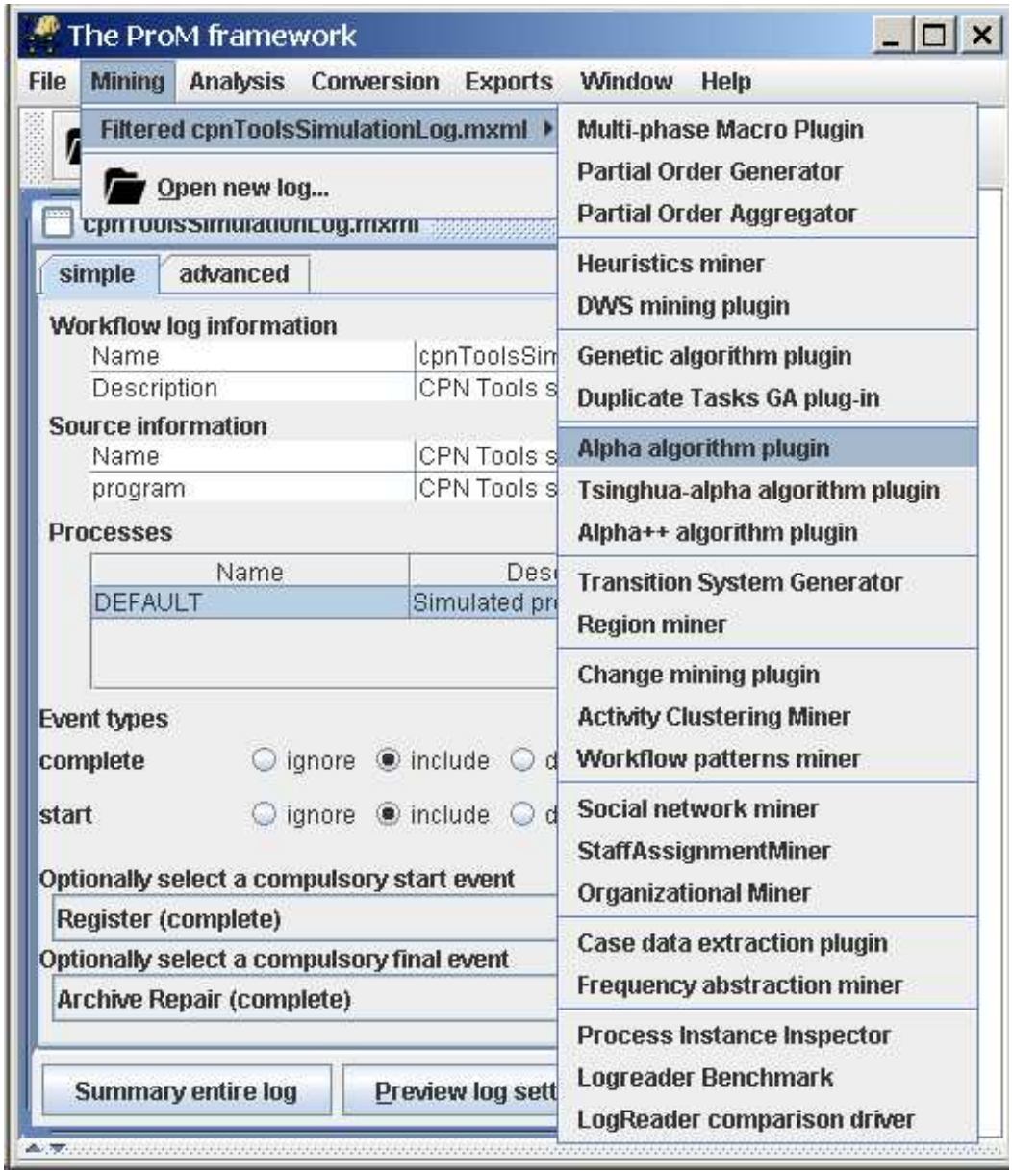


Figure 3.1: Scheenshot of the mining plug-ins in ProM.

- 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.

5. Save the mined model by choosing the menu option *Exports→Selected Petri net→Petri Net Kernel file*. We will need this exported model in Chapter 4.
6. If you prefer to visualize the mined model in another representation, you can convert this model by invoking one of the menu option *Conversion*. As an example, you can convert the mined Petri net to an EPC by choosing the menu option *Conversion→Selected Petri net→Labeled WF net to EPC*.

As a final note, although in this section we mine the log using the *Alpha algorithm plugin*, we strongly recommend you to try other plug-ins as well. The main reason is that the *Alpha algorithm plugin* 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 *Multi-phase Macro plugin*, *Heuristics miner* and *Genetic algorithm plugin* to real-life logs.

3.2 Mining Case-Related Information about a Process

Do you want know the most frequent path for our running example? Or the distribution of all cases over the different paths through the process? Then you should use the analysis plug-in *Performance Sequence Diagram Analysis*. 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 less 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).

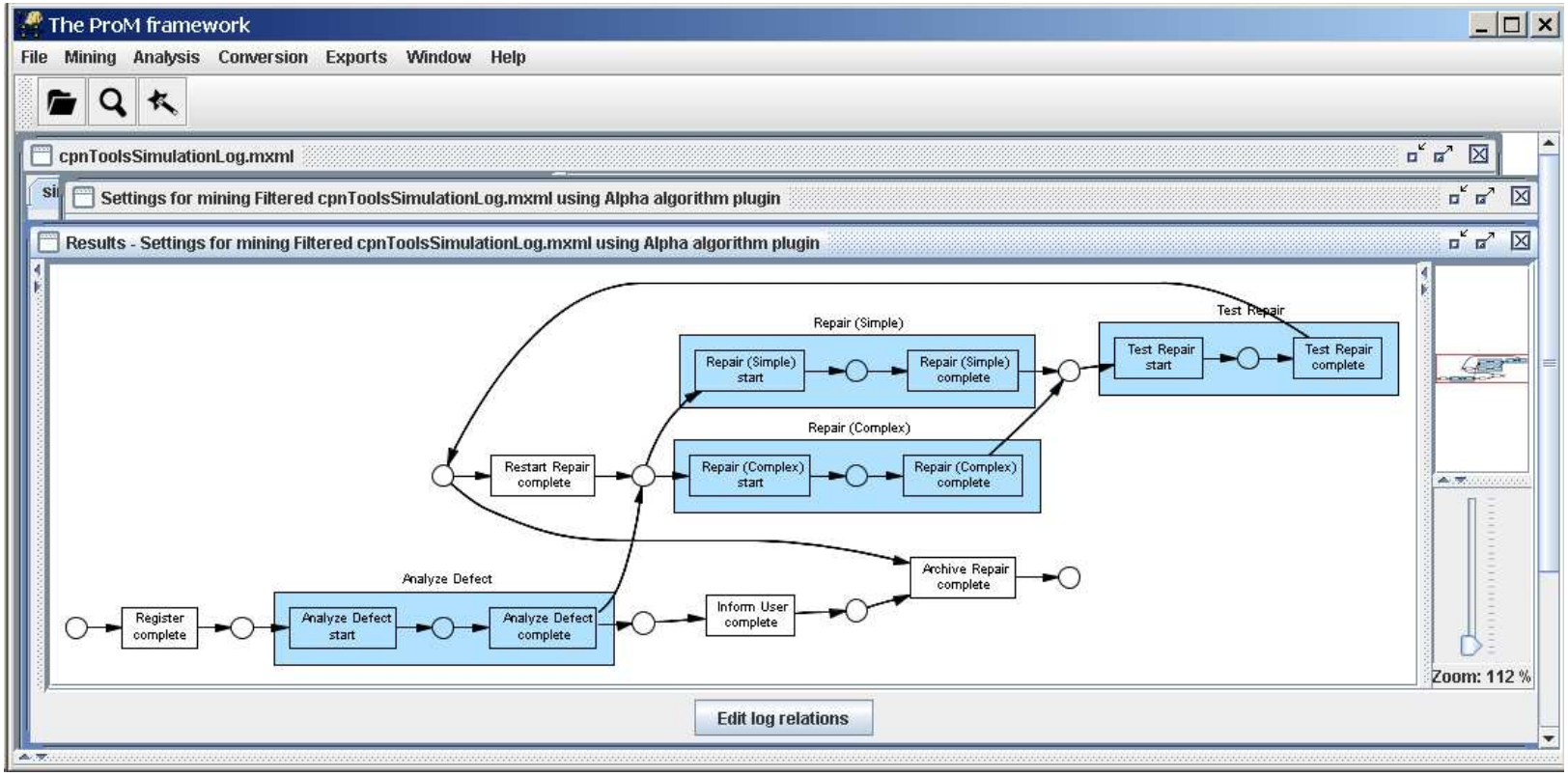


Figure 3.2: Scheenshot of the mined model for the log of the running example.

2. Run the *Performance Sequence Diagram Analysis* by choosing the menu *analysis→[log name...]→Performance Sequence Diagram Analysis* (cf. Figure 3.1).
3. Select the tab *Pattern diagram* and click on the button *Show diagram*. You should get a screen like the one in Figure 3.3.
   Take your time to inspect the results (i.e., the sequence patterns and their throughput times). Can you answer our initial questions now? If so, you have probably notice that the 73,5% of the defects could be fixed in the first attempt[3].
4. Now, how about having a look at the resources? Which employees are involved in the most frequent patterns? In which sequence do they interact? To see that, just choose "Originator" as the *Component type* and click on the button *Show diagram*.

Take your time to have a look at the other options provided by this plug-in. For instance, by clicking on the button *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 subcontract work to whom?
- Who work on the same tasks?

These and other related questions can be answered by using the *mining* plug-ins *Social Network Miner* and *Organizational Miner*, and the *analysis* plug-in *Analyze Social Network*. 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 analysis plug-in *Log Summary* (cf. Section 2.1). For

[3]See patterns 0 to 6, notice that the task "Restart Repair" does not belong to these patterns. Furthermore, the sum of the occurrences of these patterns is equal to 735.

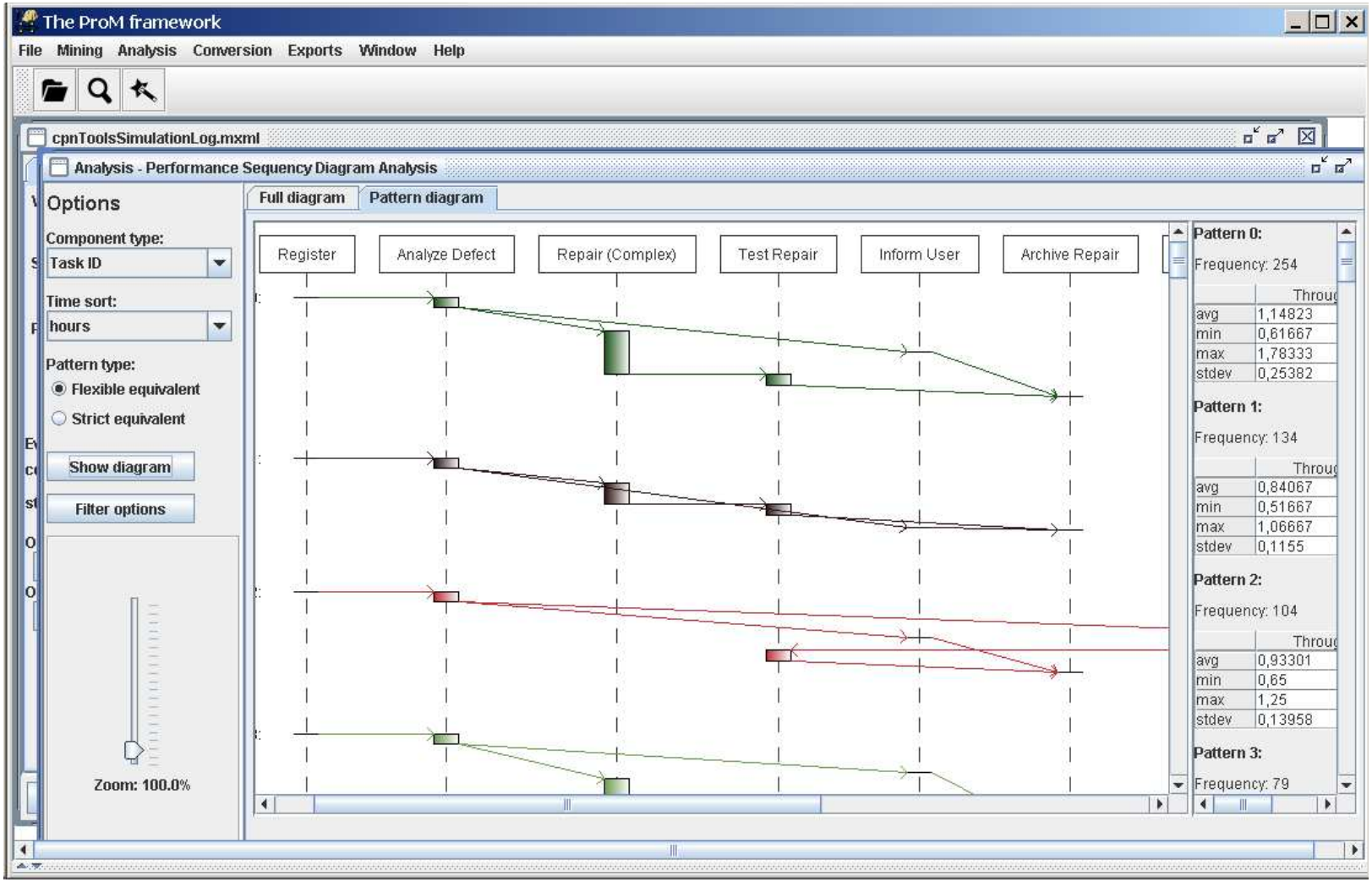


Figure 3.3: Scheenshot of the analysis plug-in *Performance Sequence Diagram Analysis*. The configuration options are on the left side, the sequence diagrams are on the middle and the patterns frequence and throughput times are on the right side.

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. Click the button *Preview log settings*.
3. Right-click on the panel *Process Instance* and click on *Find...*.
4. In the dialog *Find*, field "Text to find", type in *120* and click "OK". This option highlights the process instance in the list.
5. Double-click the process instance *120*.
6. Visualize the log summary for this process instance by choosing the menu option *Analysis→Previewed Selection...→Log Summary*.

You can see *who work on the same tasks* by using the analysis plug-in *Originator by Task Matrix*, or by running the mining plug-in *Organizational Miner*. For instance, to see the roles that work for the same tasks in our example log, you can do the following:

1. Open the filtered log (cf. Section 2.2) for the running example.
2. Select the menu option *Mining→Filtered...→Organizational Miner*, choose the options "Doing Similar Task" and "Correlation Coefficient", and click on *Start mining*.
3. Select the tab *Organizational Model*. You should get a screen like the one in Figure 3.4. Take you time to inspect the information provided at the bottom of this screen. Noticed that the *automatically generated organizational model* shows that the people with the role "Tester..." work on the tasks "Analyze Defect" and "Test Repair", and so on. If you like, you can edit these automatically generated organizational model by using the functionality provided at the other two tabs *Tasks<->Org Entity* and *Org Entity<->Resource*. Note that organizational models can be exported and used as input for other organizational-related mining and analysis plug-ins.

The other remaining questions of the list on page 22 are answered by using the mining plug-in *Social Network* in combination with the analysis plug-in *Analyze Social Network*. 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 this employee 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 3.2, we know that telephones which were not repaired are again sent to the Repair

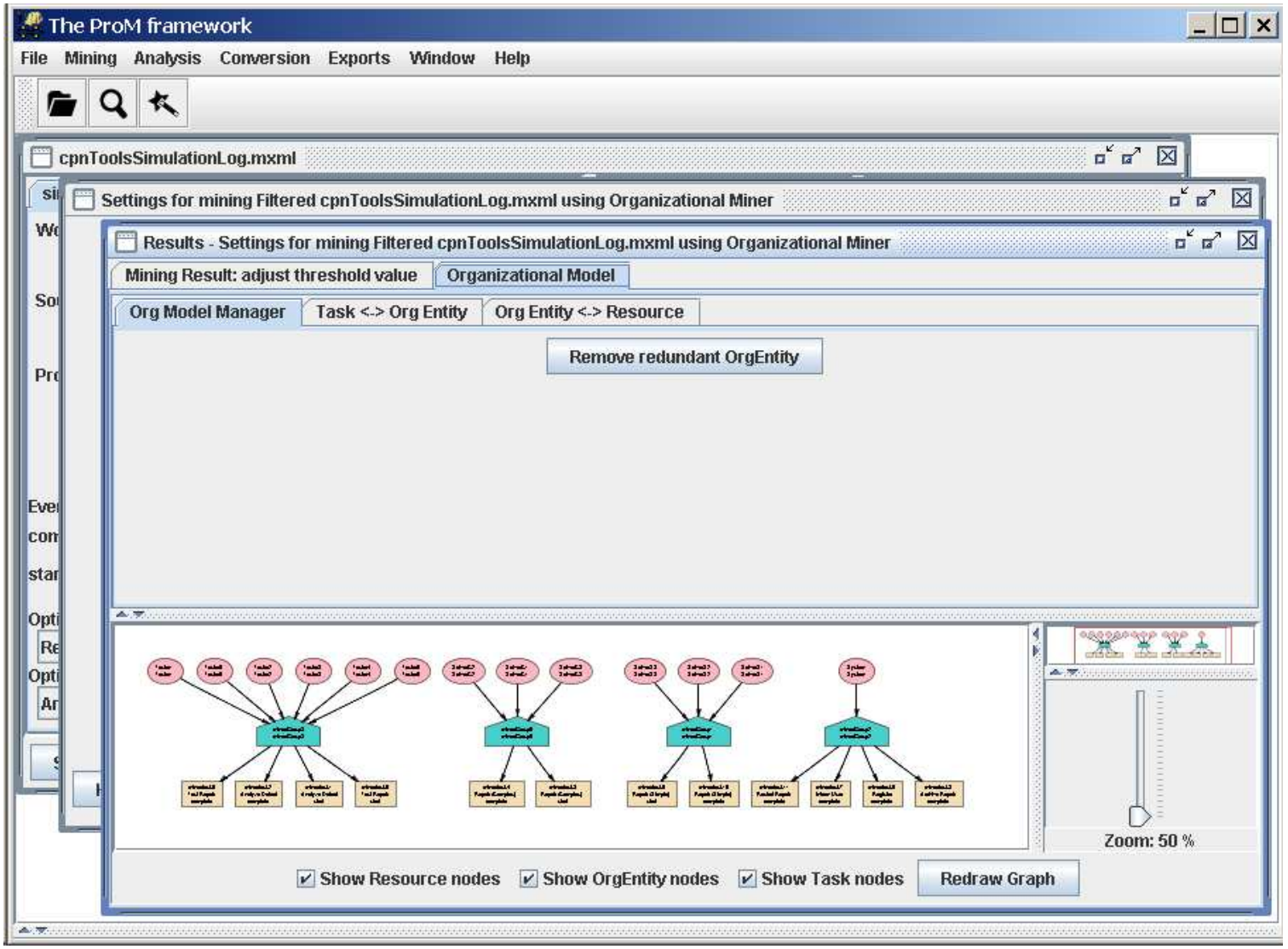


Figure 3.4: Scheenshot of the mining plug-in *Organizational Miner*.

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 (Complete)*. One possible way to do so is to perform the following steps:

1. Open the log for the running example.
2. Use the *advanced* log filter *Event Log Filter* (cf. Section 2.2) to filter the log so that only the task "Repair (Simple) (complete)" and "Repair (Complex) (complete)" are kept. (Hint: Use the analysis plug-in *Log Summary* to check if the log is correctly filtered!).
3. Run the *Social Network Miner* by choosing the menu option *Mining*→ *Filtered...*→*Social network miner*.
4. Select the tab *Handover of work*, and click the button *Start mining*. You should get a result like the one in Figure 3.5. We could already analyze this result, but we will use the analysis plug-in *Analyze Social Network* to do so. This analysis plug-in provides a more intuitive user interface. This is done on the next step.
5. Run the *Analyze Social Network* by choosing the menu option *Analysis*→ *SNA*→*Analyze Social Network*. Select the options "Vertex size", "Vertex degree ratio stretch" and set *Mouse Mode* to "Picking" (so you can use the mouse to re-arrange the nodes in the graph). The resulting graph (cf. Figure 3.6) 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" and "SolverC3" 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 by "SolverS3" and "SolverC4"). The oval shape of the nodes in the graph visually express 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?

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

As a final remark, we point out that the results produced by the *Social Network* mining plug-in can be exported to more powerful tools like AGNA[4]

[4]http://agna.gq.nu

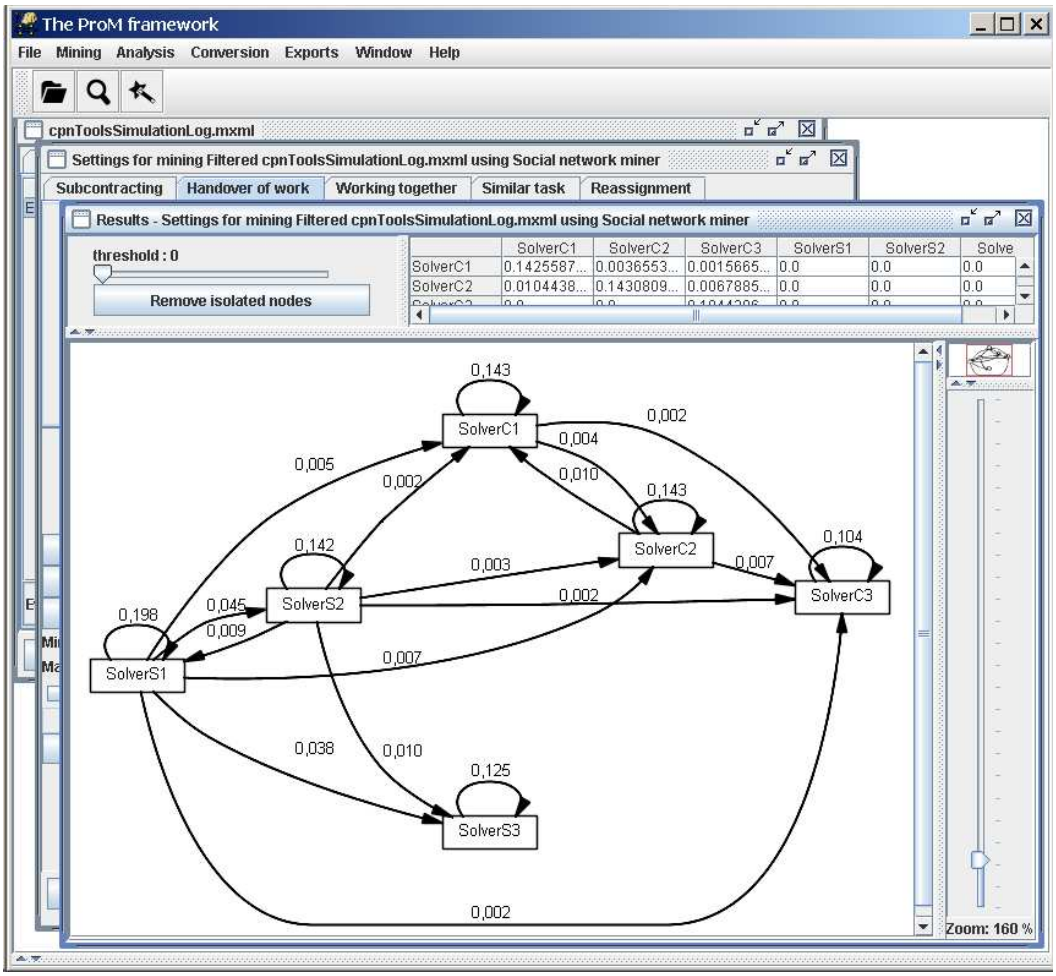


Figure 3.5: Scheenshot of the mining plug-in *Social Network Miner*.

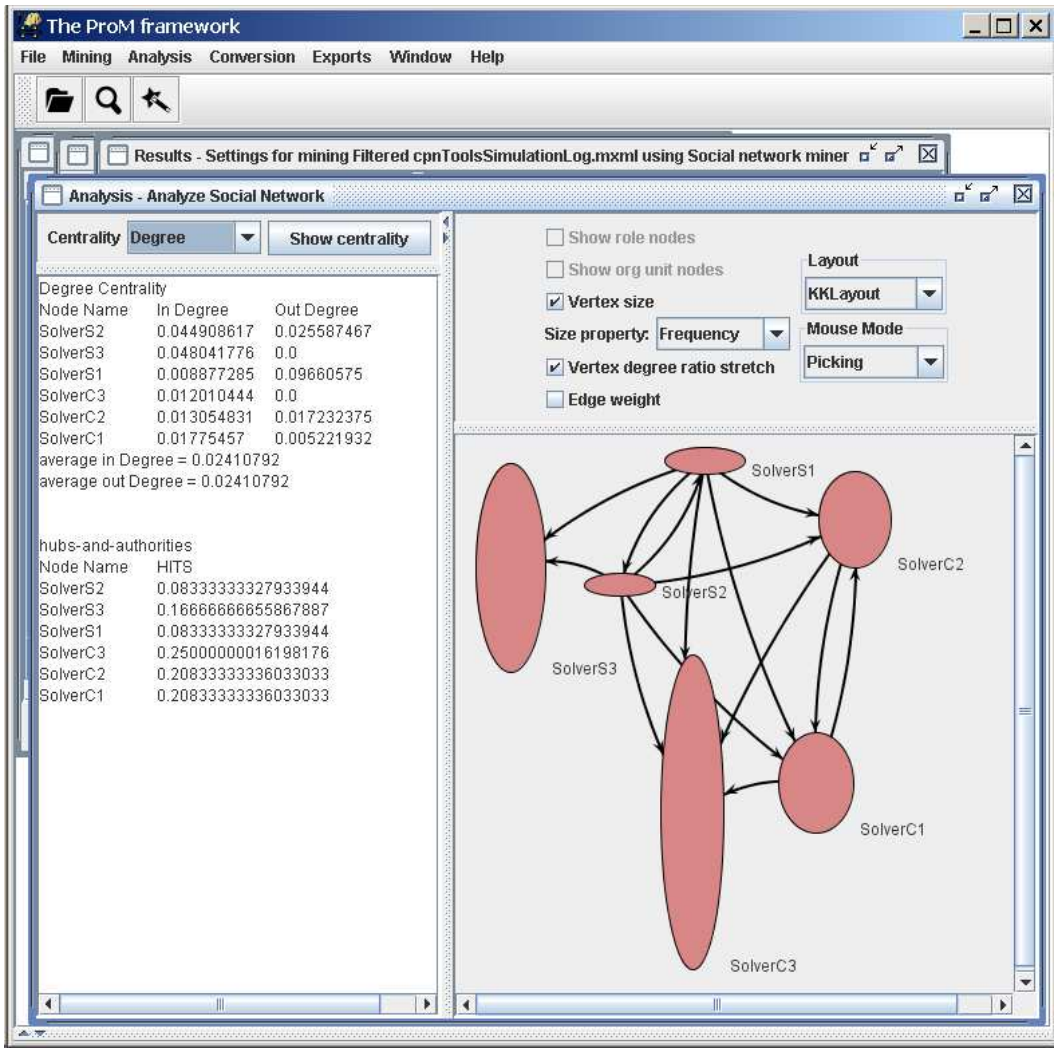


Figure 3.6: Scheenshot of the mining plug-in *Analyzer Social Network*.

and NetMiner[5], which are especially tailored to analyze social networks and provide more powerful user interfaces.

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 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 register what has happened in a organization. In ProM, auditing of a log is provided by the analysis plug-in *Default LTL Checker Plugin*[6].

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 *Default LTL Checker Plugin* to verify the property: *Does the task "Test Repair" always happen* after *the tasks "Repair (Simple)" or "Repair (Complex)" 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. Run the *Default LTL Checker Plugin* by selecting the menu option *Analysis→[log name...]→Default LTL Checker Plugin*. You should get a screen like the one in Figure 3.7.
3. Select the formula "eventually_activity_A_then_B_then_C".
4. Give as values: (i) activity A = *Repair (Simple)*, (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.
5. Click on the button *Check*. The resulting screen 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. Note that the menu options now also allow you to do mining, analysis etc. over the split log. For instance, you can apply again the LTL plug-in over the incorrect process instances to check if the remaining instances refer to situations in which the task "Repair (Complex)" was executed. Actually, this is what we do in the next step.

[5]http://www.netminer.com/
[6]LTL stands for Linear Temporal Logic.

6. Run the *Default LTL Checker Plugin* over the *Incorrect Process Instances* by choosing *Analysis→Incorrect Instances (573)→Default LTL Checker Plugin*.
7. Select the same formula and give the same input as in steps 3 and 4 above. However, this time use activity A = *Repair (Complex)*.
8. Click on the button *Check*. Note that all 573 cases satisfy the formula. So, there are not situations in which a test does not occur after a repair.

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"?

In this section we have shown how to use the pre-defined formulae of the LTL analysis plug-in to verify properties in a log. However, you can also *define your own formulae and import them into ProM*. The tutorial that explains how to do so is provided together with the documentation for the ProM tool[7].

[7]For Windows users, please see *Start→Programs→Documentation→All Documentation→LTLChecker-Manual.pdf*.

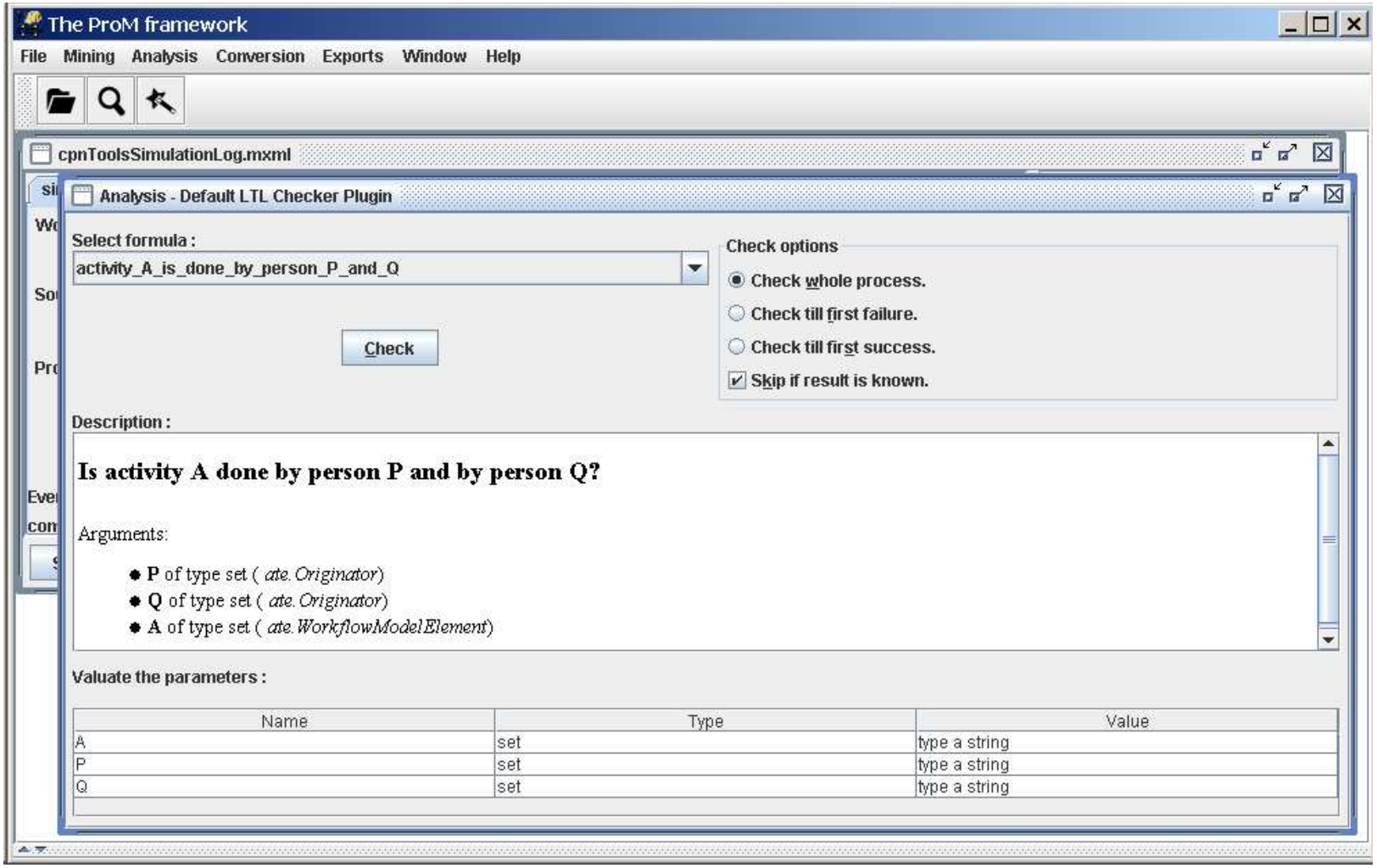


Figure 3.7: Scheenshot of the analysis plug-in *Default LTL Checker Plugin*.

Chapter 4

Questions Answered Based on a Process Model Plus an Event Log

In this section we explain the ProM analysis plug-in that are used to answer the questions in Table 4.1. These plug-ins differ from the ones in Chapter 3 because they require a log *and* a (process) model as input (cf. Figure 1.2). Section 4.1 explains a *conformance* ProM plug-in that detects discrepancies between the flows prescribed in a model and the actual process instances (flows) in a log. Sections 4.2 and 4.3 describe *extension* ProM plug-ins that respectively extend the models with performance characteristics and business rules.

4.1 Conformance Checking

Nowadays, companies usually have some process-aware information system (PAIS) [8] to support their business process. However, these process models may be incomplete because of reasons like: one could not think of all possible scenarios while deploying the model; the world is dynamic and the way employees work may change but the prescribed process models are not updated accordingly; and so on. Either way, it is always useful to have a tool that provides feedback about this.

The ProM analysis plug-in that checks *how much process instances in a log match a model and highlights discrepancies* is the *Conformance Checker*. As an illustration, we are going to check the exported mined model (cf. Section 3.1, page 20) for the log of the running example against a new log provided by the company. Our aim is to check how compliant this new log

Question	Section
How compliant are the cases (i.e. process instances) with the deployed process models? Where are the problems? How frequent is the (non-) compliance?	4.1
What are the routing probabilities for each slipt/join task? What is the average/minimum/maximum throughput time of cases? Which paths take too much time on average? How many cases follow these routings? What are the critical sub-paths for these routes? What is the average service time for each task? How much time was spent between any two tasks in the process model?	4.2
What the the business rules in the process model?	4.3

Table 4.1: Conformance and Extension Plug-ins: questions and pointers to answers.

is with the prescribed model. The procedure is the following:

1. Open the log "PromTutorialSample2.zip". This log can be downloaded from *www.processmining.org/tutorial/repairExampleSample2.zip*.
2. Open the exported PNML model that you created while executing the procedure on page 20.
3. Check if the automatically suggested mapping from the tasks in the log to the tasks in the model is correct. If not, change the mapping accordingly.
4. Run the *Conformance Checker* plug-in by choosing the menu option *Analysis→Selected Petri net→Conformance Checker*.
5. Deselect the options "Precision" and "Structure"[1], and click the button *Start analysis*. You should get results like the ones shown in figures 4.1 and 4.2, which respectively show screenshots of the *model and log diagnostic perspective* of the Conformance Checker plug-in. These two perspectives provide detailed information about the problems encountered during the log replay. The *model perspective* diagnoses information about *token counter* (number of missing/left tokens), *failed tasks* (tasks that were not enabled), *remaining tasks* (tasks that remained enabled), *path coverage* (the tasks and arcs that were used during the log replay) and *passed edges* (how often every arc in the model was

[1]These are more advanced features that we do not need while checking for compliance. Thus, we will turn them off for now.

used during the log replay). The *log perspective* indicates the points of non-compliant behavior for every case in the log.

Take your time to have a look at the results. Can you tell how many traces are not compliant with the log? What are the problems? Have all the devices been tested after the repair took places? Is the client always being informed?

4.2 Performance Analysis

Like the *Conformance Checker* (cf. Section 4.1), the plug-in *Perfomance Analysis with Petri net* also requires a log and a Petri net as input[2]. The main difference is that this plug-in focuses on analyzing *time-related* aspects of the process instances. In other words, this plug-in can answer the questions:

- What are the routing probabilities for each slipt/join task?
- What is the average/minimum/maximum throughput time of cases?
- Which paths take too much time on average? How many cases follow these routings? What are the critical sub-paths for these routes?
- What is the average service time for each task?
- How much time was spent between any two tasks in the process model?

To run the *Perfomance Analysis with Petri net* analysis plug-in for the log of our running example, perform the following steps:

1. Open the filtered log (cf. Section 2.2) for the running example.
2. Open the exported PNML model that you created while executing the procedure on page 20.
3. Run the *Perfomance Analysis with Petri net* analysis plug-in by selecting the menu option *Analysis→Selected Petri net→Performance Analysis with Petri net*.
4. Set the field *Times measured in* to "hours" and click on the button *Start analysis*. The plug-in will start replaying and log and computing the time-related values. When it is ready, you should see a screen like the one in Figure 4.3.
5. Take your time to have a look at these results. Note that the right-side panel provides information about the *average/minimum/maximum throughtput* times. The central panel (the one with the Petri net model)

[2]If the model you have is not a Petri net but another one of the supported formats, you can always use one of the provided *conversion* plug-in to translate your model to a Petri net.

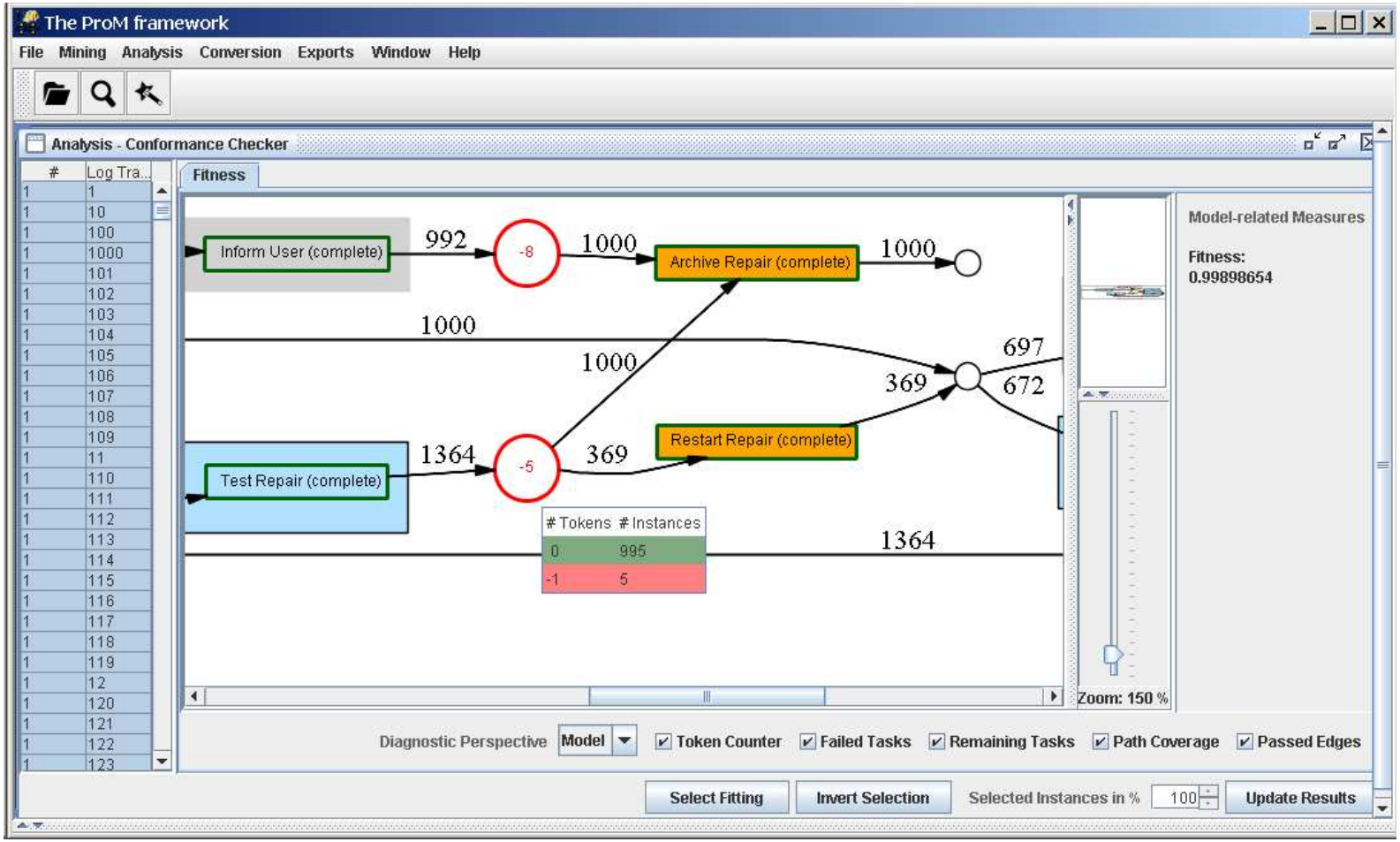


Figure 4.1: Scheenshot of the analysis plug-in *Conformance Checker*: Model view.

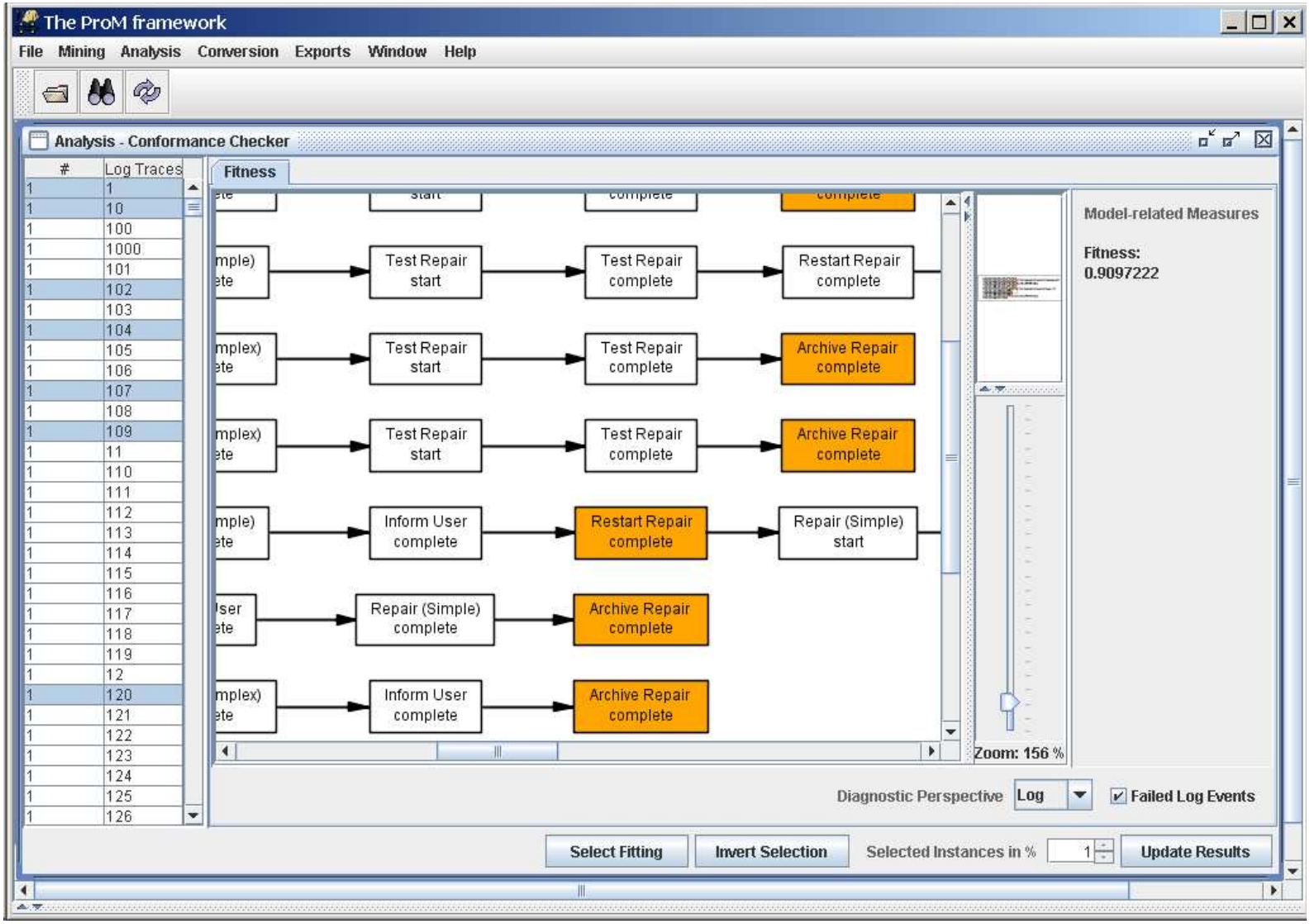


Figure 4.2: Scheenshot of the analysis plug-in *Conformance Checker*: Log view.

shows (i) the *bottlenecks* (notice the different colors for the places) and (ii) the *routing probabilities* for each split/join tasks (for instance, note that only in 27% of the cases the defect could not the fixed on the first attempt). The bottom panel show information about the *waiting times* in the places. You can also select one or two tasks to respectively check for *average service times* and *the time spent between any two tasks in the process model*. If you like, you can also change the *settings* for the waiting time (small window at the bottom with *High, Medium* and *Low*).

The previous procedure showed how to answer all the questions listed in the beginning of this section, except for one: *Which paths take too much time on average? How many cases follow these routings? What are the critical sub-paths for these routes?* To answer this last question, we have to use the analysis plug-in *Performance Sequence Diagram Analysis* (cf. Section 3.2) in combination with the *Performance Analysis with Petri net*. In the context of our example, since the results in Figure 4.3 indicate that the cases take on average *1.11 hours* to be completed, it would be interesting to analyze what happens for the cases that take longer than that. The procedure to do so has the following steps:

1. If the screen with the results of the *Performance Analysis with Petri net* plug-in is still open, just choose the menu option *Analysis→Whole Log→Performance Sequence Diagram Analysis*. Otherwise, just open the filtered log for the running example and run Step 2 described on page 22.
2. In the resulting screen, select the tab *Pattern Diagram*, set *Time sort* to hours, and click on the button *Show diagram*.
3. Now, click on the button *Filter Options* to filter the log so that only cases with throughput time superior to 1.11 hours are kept.
4. Select the option *Sequences with throughput time*, choose "above" and type in "1.1" in the field "hours". Afterwards, click first on the button *Update* and then on button *Use Selected Instances*. Take your time to analyze the results. You can also use the *Log Summary* analysis plug-ins to inspect the *Log Selection*. Can you tell how many cases have throughput time superior to 1.1 hours? Note that the *Performance Sequence Diagram Analysis* plug-in shows how often each cases happened in the log. Try playing with the provided options. For instance, what happens if you now select the *Component Type* as "Originator"? Can you see how well the employees are doing? Once you have the log selection with the cases with throughput time superior to 1.1 hour, you

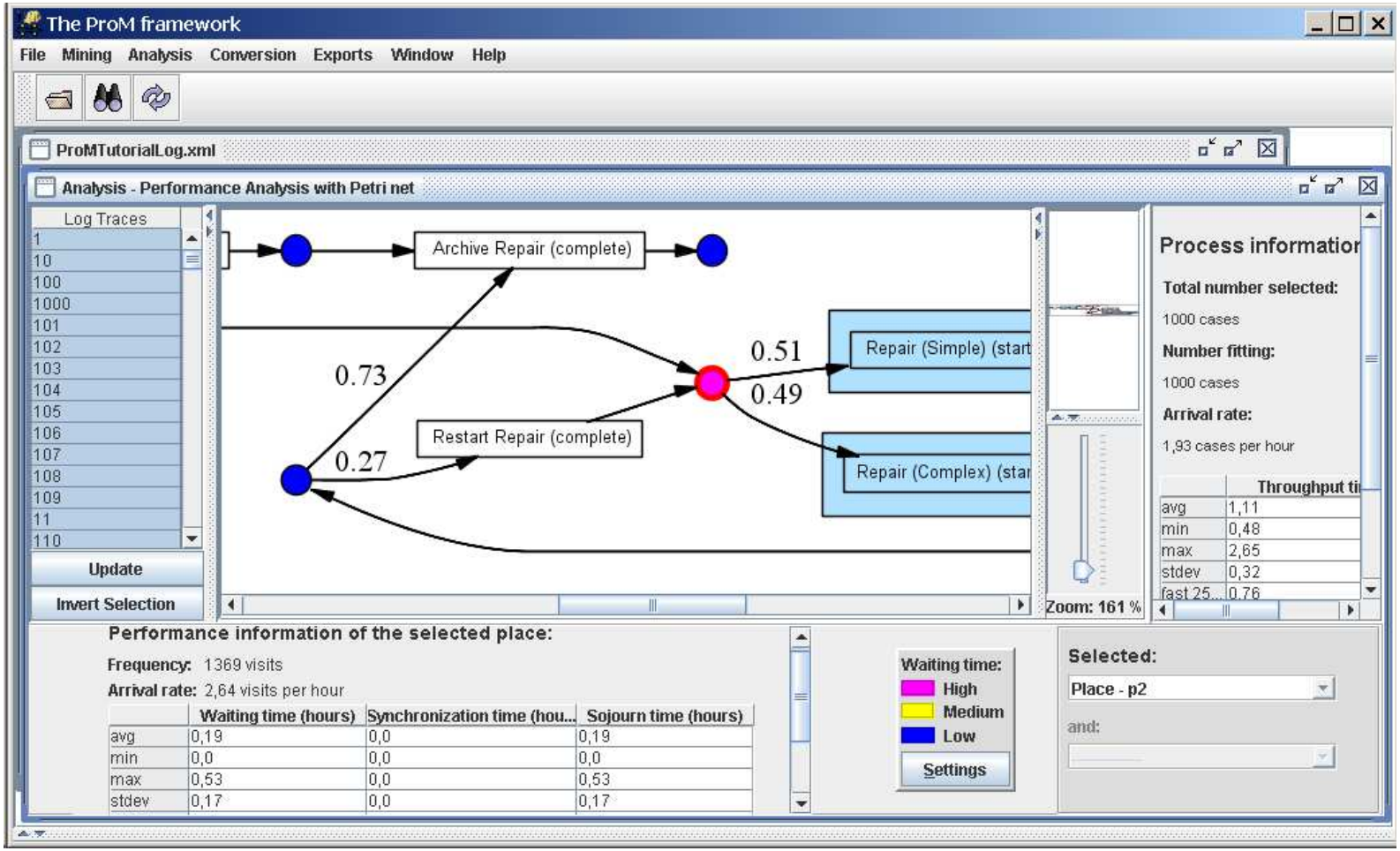


Figure 4.3: Scheenshot of the analysis plug-in *Performance Analysis with Petri net*.

can check for *critical sub-paths* by doing the remaining steps in this procedure.

5. Open exported PNML model that you created while executing the procedure on page 20, but this time link it to the *selected cases* by choosing the menu option *File→Open PNML file→With:Log Selection*. If necessary, change the automatically suggested mappings and click on the button *OK*. Now the imported PNML model is linked to the process instances with throughput times superior to 1.1 hours.
6. Run the analysis plug-in *Performance Analysis with Petri net* to discover the *critical sub-paths* for these cases. Take your time to analyse the results. For instance, can you see that now 43% of the defects could not be fixed on the first attempt?

Finally, we suggest you spend some time reading the Help documentation of this plug-in because it provides additional information to what we have explained in this section. Note that the results of this plug-in can also be exported.

4.3 Decision Point Analysis

To discover the *business rules* (i.e. the conditions) that influence the points of choice in a model, you can use the *Decision Point Analysis* plug-in. For instance, in the context of our running example, we could investigate which defect types (cf. Section 1.3) are fixed by which team. The procedure to do so has the following steps:

1. Open the filtered log (cf. Section 2.2) for the running example.
2. Open the exported PNML model that you created while executing the procedure on page 20.
3. Run the *Decision Point Analysis* plug-in by selecting the menu option *Analysis→Selected Petri net→Decision Point Analysis*.
4. Double-click the option "Choice 4 p2". This will select the point of choice between execution the task "Repair (Complex)" or "Repair (Simple)"[3].
5. Select the tab *Attributes* and set the options: (i) *Attribute selection scope* = "just before", (ii) change the *Attribute type* of the field *defectType* to "numeric". Afterwards, click on the button *Update results*.

[3]**Note:** If your option "Choice 4 p2" does not correspond to the point of choice we refer to in the model, please identify the correct option on your list. The important thing in this step is to select the correct point of choice in the model.

This analysis plug-in will now invoke a *data mining* algorithm (called J48) that will discover which fields in the log determine the choice between the different branches in the model.

6. Select the tab *Result* to visualize the mined rules (cf. Figure 4.4). Note that cases with a defect types from 1 to 4 [4] are routed to the task "Repair (Simple)" and the ones with defect type bigger than 4 are routed to the task "Repair (Complex)". This is the rule that covers the *majority* of the cases in the log. However, it does not mean that all the cases follow this rule. To check for this, you have to perform the next step.
7. Select the tab *Decision Tree/Rules* (cf. Figure 4.5). Remark the text *(695.0/87.0)* inside the box for the task "Repair (Complex)". This means that *according to the discovered business rules, 695 cases should have been routed to the task "Repair (Complex)" because their defect type was bigger than 4. However, 87 of these cases are* misclassified *because they were routed to the task "Repair (Simple)".* Thus, the automatically discovered business rules describe the conditions that apply to the majority of the cases, but it does not mean that all the cases will fit these rules. Therefore, we recommend you to always check for the results in the tab *Decision Tree/Rules* as well. In our case, these result makes sense because, from the description of the running example (cf. Section 1.3), we know that some defect types can be fixed by both teams.

To get more insight about the *Decision Point Analysis*, we suggest you spend some time reading its Help documentation because it provides additional information that was not covered in this section. As for the many ProM plug-ins, the mined results (the discovered rules) can also be exported.

[4] From the problem description (cf. Section 1.3), we know that there are 10 types of defects.

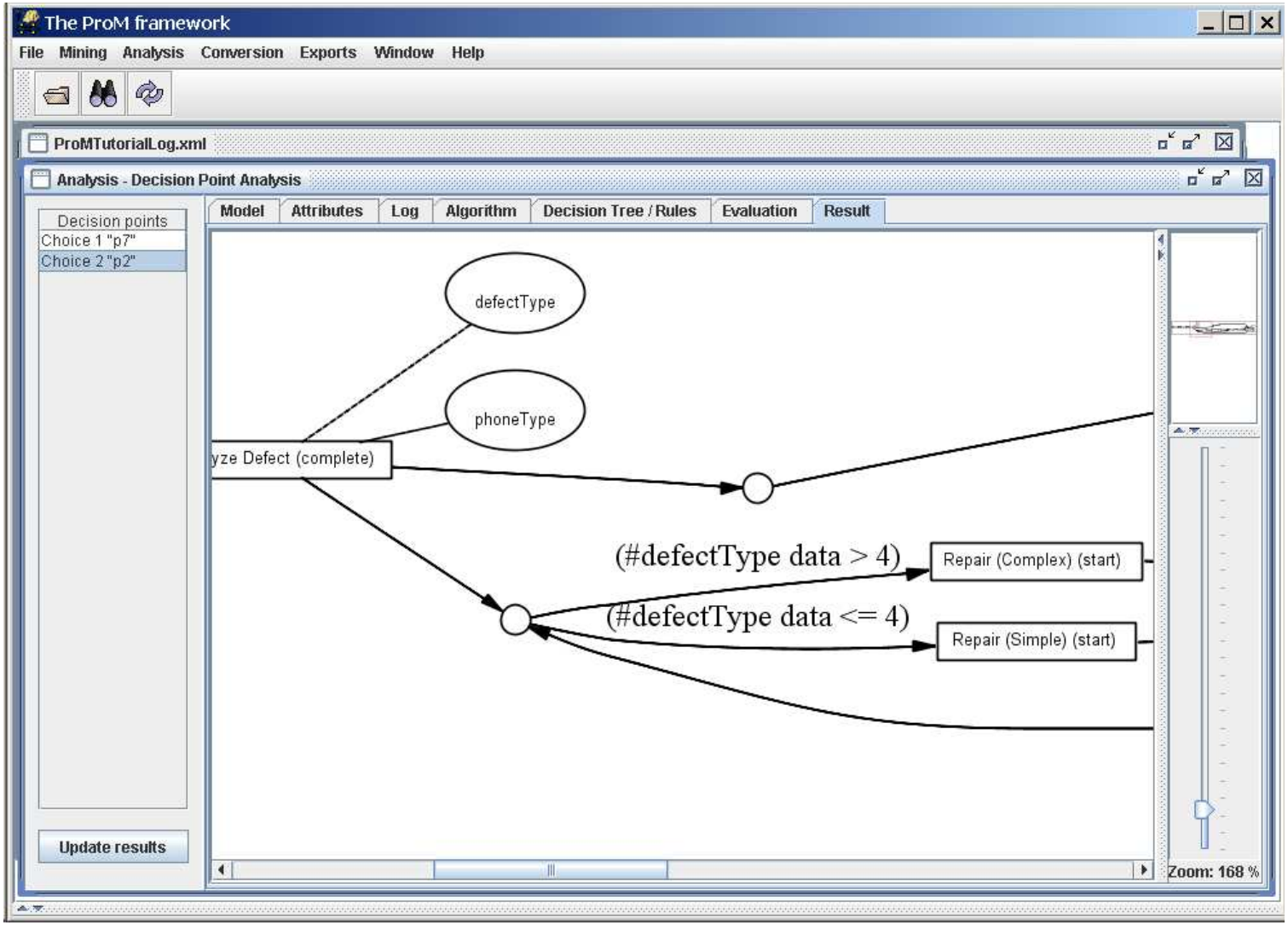


Figure 4.4: Scheenshot of the analysis plug-in *Decision Point Analysis*: *Result* tab.

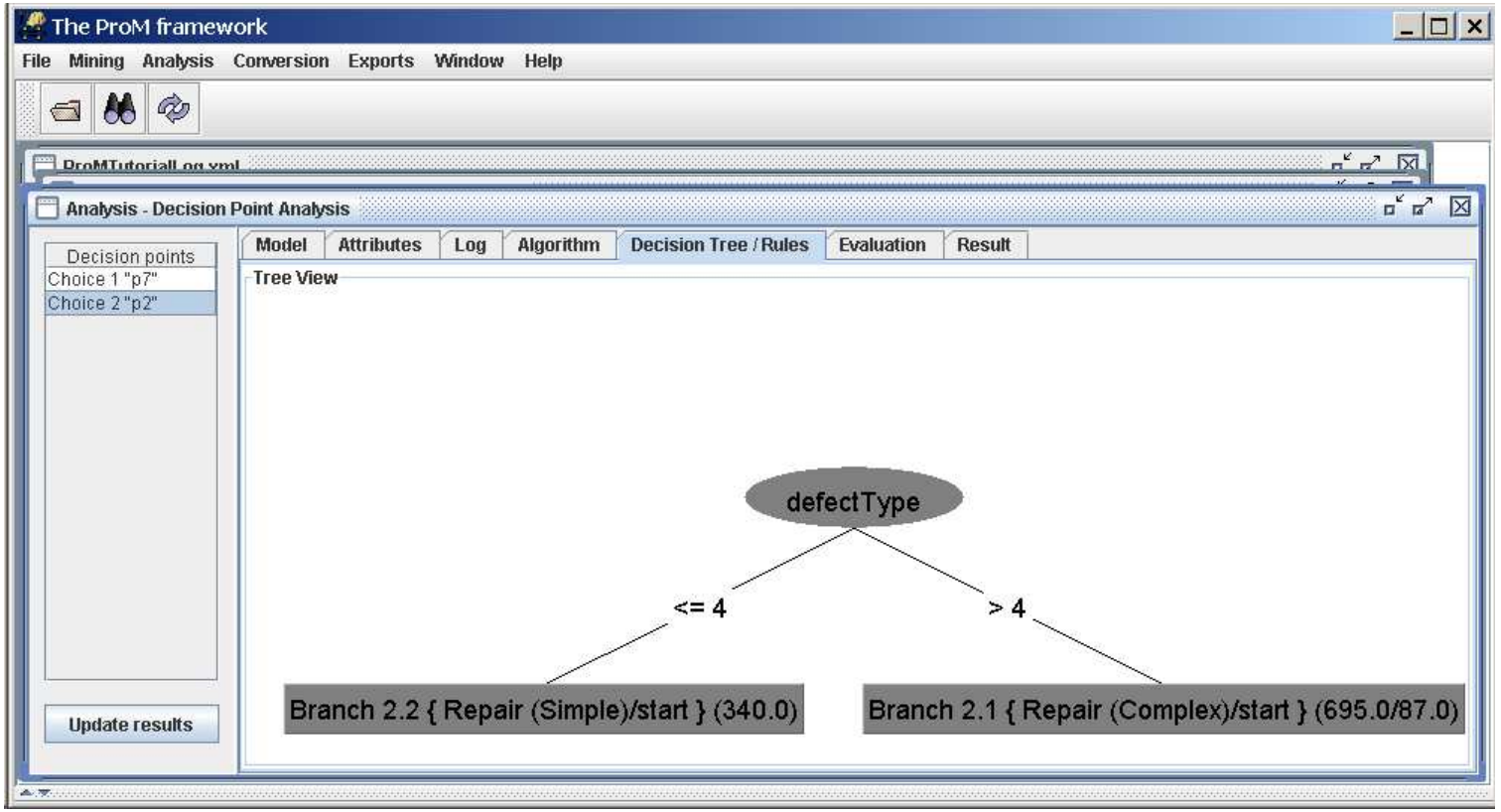


Figure 4.5: Scheenshot of the analysis plug-in *Decision Point Analysis*: *Decision Tree/Rules* tab.

Chapter 5

Conclusions

This tutorial showed how to use the different ProM plug-ins to answer common questions about process models (cf. Section 1.1). Since our focus was on this set of question, we have not covered many of the other plug-ins that are in ProM. We hope that the subset we have shown in this tutorial will help you in finding your way in ProM. However, if you are interested, you could have a further look in plug-ins to *verify (process) models and detect potential problems* (by using the analysis plug-ins *Check correctness of EPC*, *Woflan Analysis* or *Petri net Analysis*), *quantify (from 0% until 100%) how much behavior two process models have in common with respect to a given even log* (by using the analysis plug-in *Behavioral Precision/Recall*), *create simulation models with the different mined perspectives* (by using the export plug-in *CPN Tools*) etc. The ProM tool can be downloaded at www.processmining.org.

Bibliography

[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 and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2002.

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

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

[7] B.F. van Dongen, A.K. Alves de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM Framework: A New Era in Process Mining Tool Support. In G. C. and P. Darondeau, editors, *ICATPN*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer, 2005.

[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.