# Process Mining towards Semantics

A.K. Alves de Medeiros and W.M.P. van der Aalst

Eindhoven University of Technology,
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
{a.k.medeiros,w.m.p.v.d.aalst}@tue.nl

**Abstract.** Process mining techniques target the *automatic* discovery of information about process models in organizations. The discovery is based on the execution data registered in *event logs*. Current techniques support a variety of practical analysis, but they are somewhat limited because the labels in the log are not linked to any concepts. Thus, in this chapter we show how the analysis provided by current techniques can be improved by including *semantic* data in event logs. Our explanation is divided into two main parts. The first part illustrates the power of current process mining techniques by showing how to use the open source process mining tool ProM to answer concrete questions that managers typically have about business processes. The second part utilizes usage scenarios to motivate how process mining techniques could benefit from *semantic annotated* event logs and defines a concrete semantic log format for ProM. The ProM tool is available at www.processmining.org.

## 1 Introduction

Nowadays, most organizations use information systems to support the execution of their business processes [21]. Examples of information systems supporting operational processes are Workflow Management Systems (WMS) [10,15], 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 [8], 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 [6]), 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

T.S. Dillon et al. (Eds.): Advances in Web Semantics I, LNCS 4891, pp. 35–80, 2008.

**Table 1.** Example of an event log

| 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 | ... |

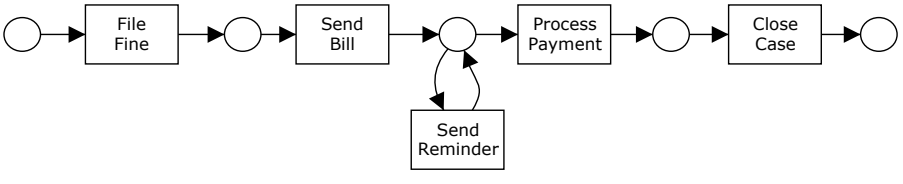


**Fig. 1.** Petri net illustrating the control-flow perspective that can be mined from the event log in Table 1

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 other words, process mining techniques are helpful because they gather information about what is *actually* happening according to an event log of an 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 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]. 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

[1] The reader unfamiliar with Petri nets is referred to [17,30,32].

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. 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.

Current process mining techniques can address all these three perspectives [9,11,12,16,20,22,28,29,31,33,34,35,38,40]. Actually, many of these techniques are implemented in the open-source tool ProM [19,39]. As we show in this chapter, the ProM tool can be used to answer common questions about business processes, like "How is the distribution of all cases over the different paths in through the processes?", "Where are the bottlenecks in the process?" or "Are all the defined rules indeed being obeyed?". Showing how to use the current process mining techniques implemented in ProM to answer these kinds of questions is the first contribution of this chapter.

However, although the process mining techniques implemented in ProM can answer many of the common questions, these techniques are somewhat limited because their analysis is purely based on the *labels* in the log. In other words, the techniques are unable to reason about *concepts* in an event log. For instance, if someone wants to get feedback about *billing* processes in a company or the webservices that provide a certain service, this person has to manually specify all the labels that map to billing processes or webservices providing the given service. Therefore, as supported by [27], we believe that the automatic discovery provided by process mining techniques can be augmented if we include *semantic information* about the elements in an event log. Note that semantic process mining techniques bring the discovery to the conceptual (or semantical)

level. Furthermore, because semantic logs will link to concepts in ontologies, it is possible to embed ontology reasoning in the mining techniques. However, when supporting the links to the ontologies, it is important to make sure that the semantically annotated logs can also be mined by current process mining techniques. This way we avoid recoding of good existing solutions. Thus, the second contribution of this chapter consists of showing (i) how to extend our mining format to support links to ontologies and (ii) providing usage scenarios that illustrate the gains of using such semantic logs.

The remainder of this chapter is organized as follows. Section 2 shows how to use ProM plug-ins to answer common questions that managers have about business processes. Section 3 explains how to extend the current log format used by ProM to support the link to ontologies and discusses usage scenarios based on this format. Section 4 concludes this chapter.

## 2 Process Mining in Action

In this section we show how to use the ProM tool to answer the questions in Table 2. These are common questions that managers typically have about business processes. The ProM framework [19,39] 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 chapter our aim is to show how to use ProM plug-ins to answer *common questions about processes in companies*, 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 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 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?"

The remainder of this section illustrates how to use ProM to answer the questions in Table 2. The provided explanations have a tutorial-like flavor because you should be able to reproduce the results when using ProM over the example in this section or while analyzing other logs. The explanation is supported by the running example in Subsection 2.1. Subsection 2.3 describes how you can inspect and clean an event log before performing any mining. Subsection 2.4 shows how to use *discovery* plug-ins (cf. Figure 2) and Subsection 2.5 describes

**Table 2.** Common questions that managers usually have about processes in organizations

| |
|---|
| 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/join task? |
| 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 process rules/constraints 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? |
| 16. Who subcontract work to whom? |
| 17. Who work on the same tasks? |

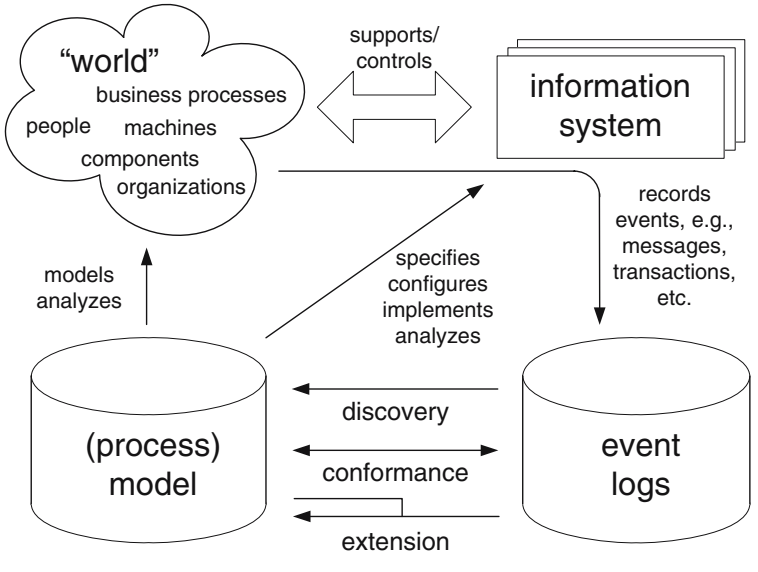


**Fig. 2.** Sources of information for process mining. The *discovery* plug-ins use only an event log as input, while the *conformance* and *extension* plug-ins also need a (process) model as input.

how to mine with *conformance* and *extension* plug-ins. Finally, we advice you to have the ProM tool at hand while reading this section. This way you can play with the tool while reading the explanations. Subsection 2.2 explains how to get started with ProM.

### 2.1 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.

### 2.2 Getting Started

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

1. Install the ProM tool. This tool is freely available at http:// prom.sourceforge. net. Please download and run the installation file for your operating system.
2. Download the two log files for the running example. These logs files are located at (i) http://www.processmining.org/_media/tutorial/repairExample.zip and (ii) http://www.processmining.org/_media/tutorial/repairExampleSample2.zip.

### 2.3 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

**Table 3.** Log Pre-Processing: questions and pointers to answers

| Question | Subsection |
|---|---|
| How many cases (or process instances) are in the log?<br>How many tasks (or audit trail entries) are in the log?<br>How many originators are in the log?<br>Are there running cases in the log?<br>Which originators work in which tasks? | 2.3.1 |
| How can I filter the log so that only completed cases are kept?<br>How can I see the result of my filtering?<br>How can I save the pre-processed log so that I do not have to redo work? | 2.3.2 |

(cf. Section 2.1), all cases without an archiving task as the last one correspond to 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. 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 3. As you can see, Subsection 2.3.1 shows how to answer questions related to log inspection and Subsection 2.3.2 explains how to filter an event log and how to save your work. Note that the list of questions in Table 3 is not exhaustive, but they are enough to give you an idea of the features offered by ProM for log inspection and filtering.

### 2.3.1 Inspecting the Log

The first thing you need to do to inspect or mine a log is to load it into ProM. In this section we use the log at the location *http://www.processmining.org/_media/tutorial/repairExample.zip*. This log has process instances of the running example described in Section 2.1.

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 3. 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 *File→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 4. 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?

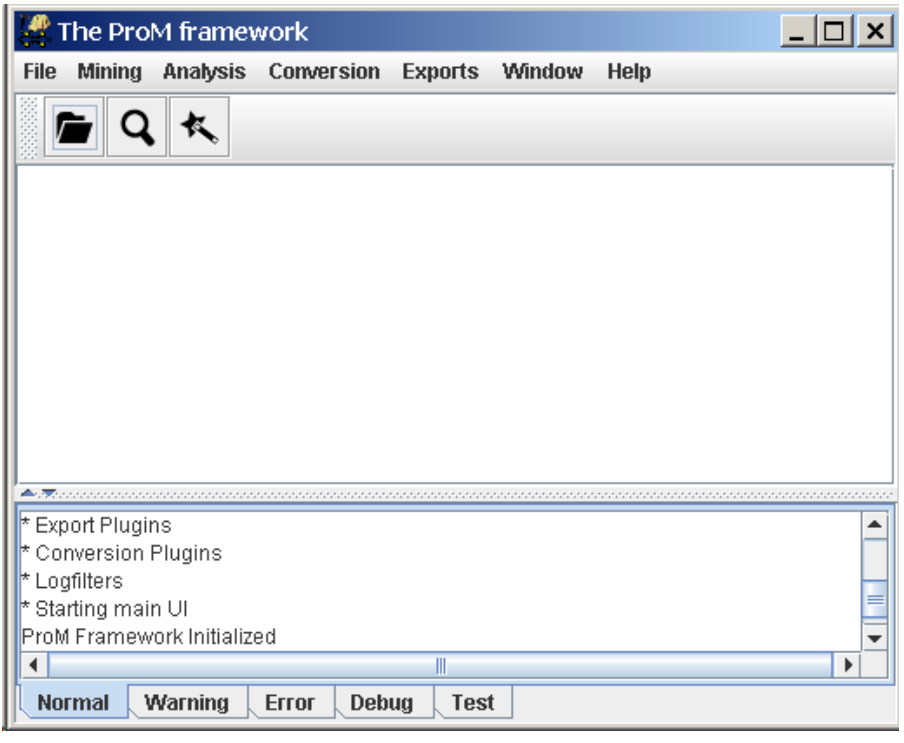


**Fig. 3.** Screenshot of the main interface of ProM. The menu *File* allows to open event logs and to import models into ProM. The menu *Mining* provides the mining plug-ins. These mining plug-ins mainly focus on discovering information about the control-flow perspective of process models or the social 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.

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 above? 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 5). Note that only 1000 cases end with the task "Archive Repair". The *last question* of the list above can be answered by the analysis plug-in *Originator by Task Matrix*. This plug-in can be started by clicking the menu *Analysis→[log*

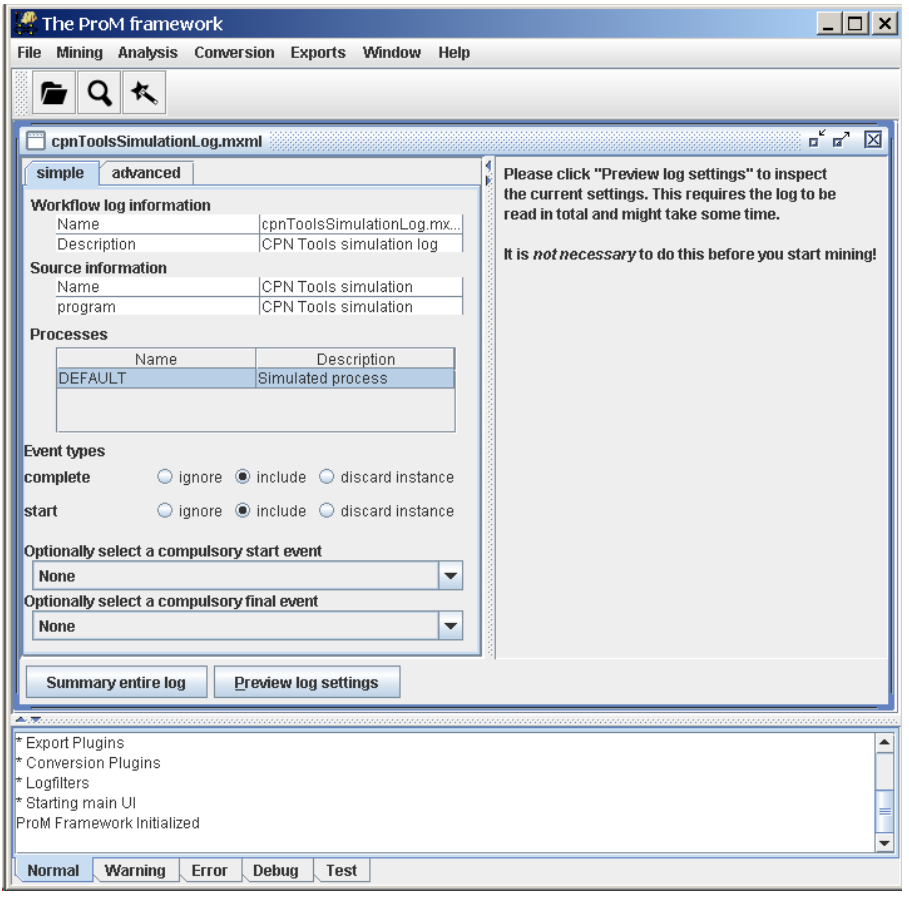


**Fig. 4.** Screenshot of ProM after opening the log of the running example (cf. Section 2.1)

*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 2.1) of the company [2]. 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 4) and then double-clicking on a specific process instance.

[2] See the originators working on the tasks "Repair (Complex)" and "Repair (Simple)" in Figure 6.

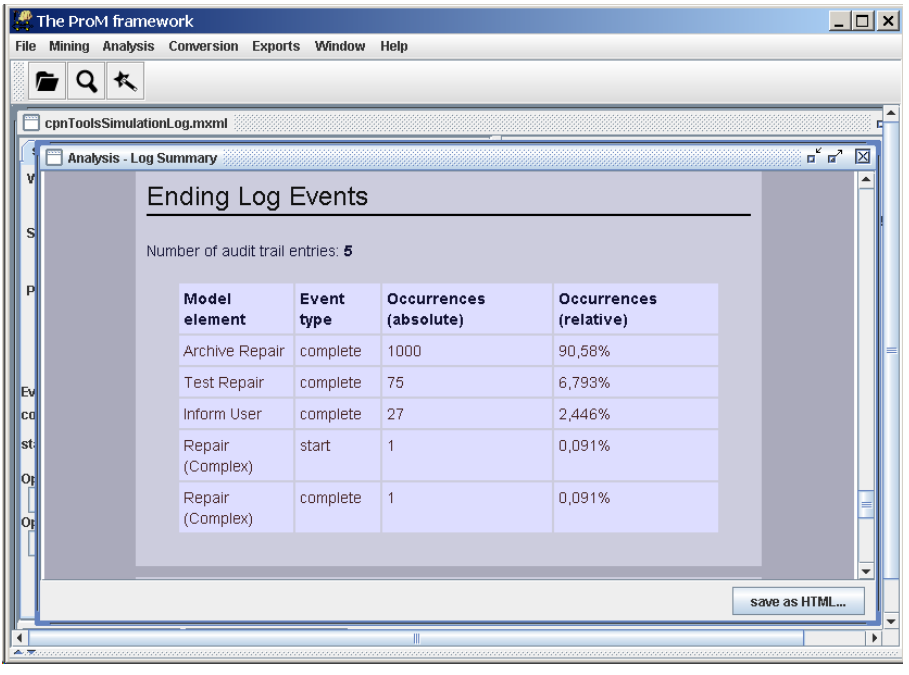


**Fig. 5.** Excerpt of the result of the analysis plug-in *Log Summary*

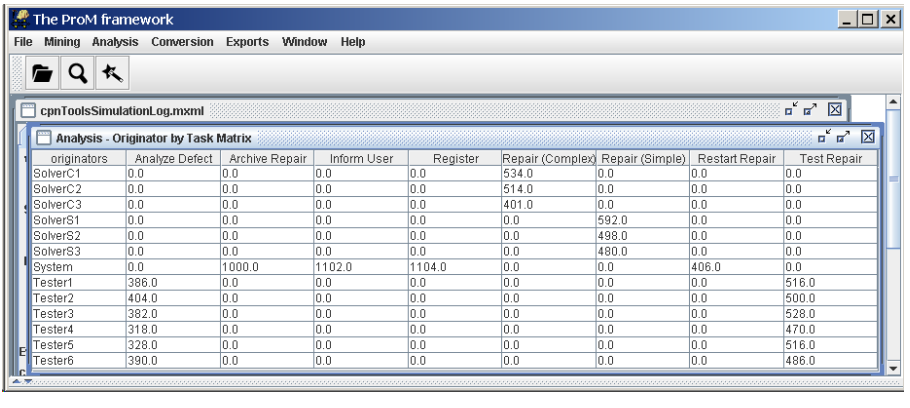


**Fig. 6.** Scheenshot with the result of the analysis plug-in *Originator by Task Matrix*

### 2.3.2 Cleaning the Log

In this chapter we use the process mining techniques to get insight about the process for repairing telephones (cf. Section 2.1). 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 4 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 4), (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 4 (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 2.3.1), 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 7) provides more powerful log filters. Each log filter has a *Help*, so we are not going into details about them. However, we strongly advise you to spend 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. For instance, the log filters *Add Artificial Start Task* and *Add Artificial End Task* support the respective addition of tasks at the begin and end of traces. These two log filters are handy when applying process mining algorithms that assume the target model to have a single start/end point.

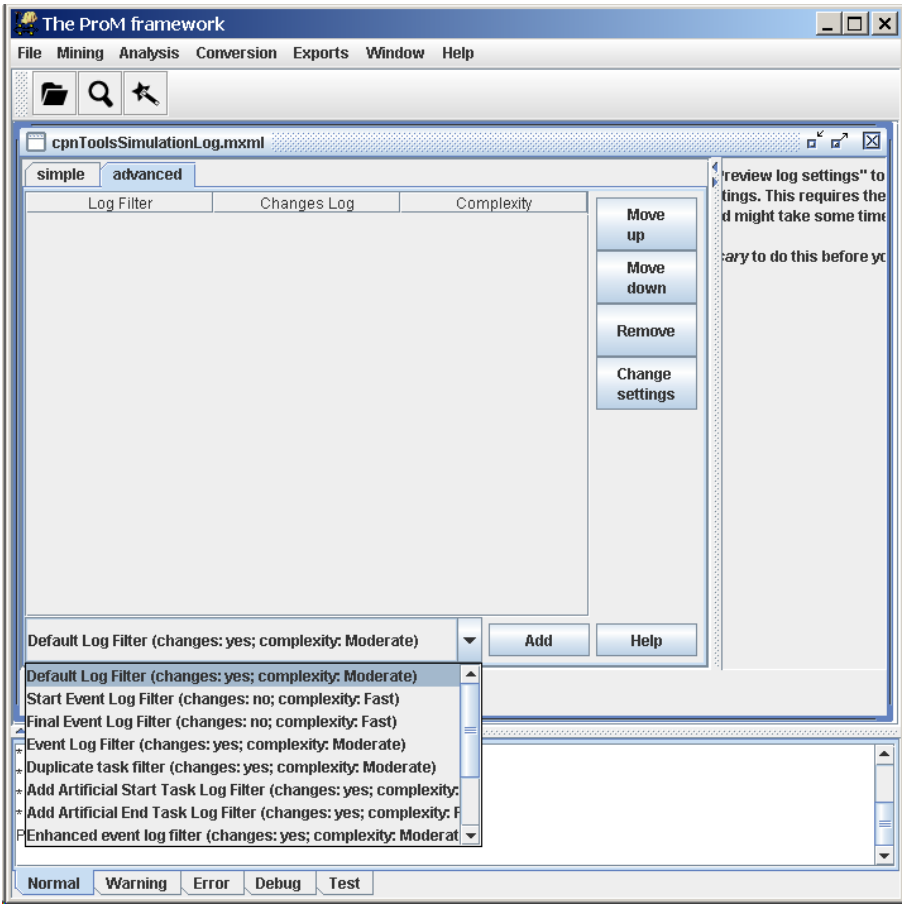


**Fig. 7.** Scheenshot of the *Advanced Log Filters* in ProM

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

**Table 4.** Discovery Plug-ins: questions and pointers to answers

| Question | Subsection |
|---|---|
| How are the cases actually being executed? | 2.4.1 |
| What is the most frequent path for every process model?<br>How is the distribution of all cases over the different paths through the process? | 2.4.2 |
| How many people are involved in a case?<br>What is the communication structure and dependencies among people?<br>How many transfers happen from one role to another role?<br>Who are the important people in the communication flow?<br>Who subcontract work to whom?<br>Who work on the same tasks? | 2.4.3 |
| Are the process rules/constraints indeed being obeyed? | 2.4.4 |

## 2.4 Questions Answered Based on an Event Log Only

Now that you know how to inspect and pre-process an event log (cf. Subsection 2.3), we proceed with showing how to answer the questions related to the *discovery* ProM 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 3. Subsection 2.4.1 shows how to mine the *control-flow* perspective of process models. Subsection 2.4.2 explains how to mine information regarding certain aspects of cases. Subsection 2.4.3 describes how to mine information related to the roles/employees in the event log. Subsection 2.4.4 shows how to use temporal logic to verify if the cases in a log satisfy certain (required) properties.

### 2.4.1 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.

As shown in Figure 8, ProM supports various plug-ins to mine the control-flow perspective of process models. In this section, we will use the 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 2.3.2), 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 8).

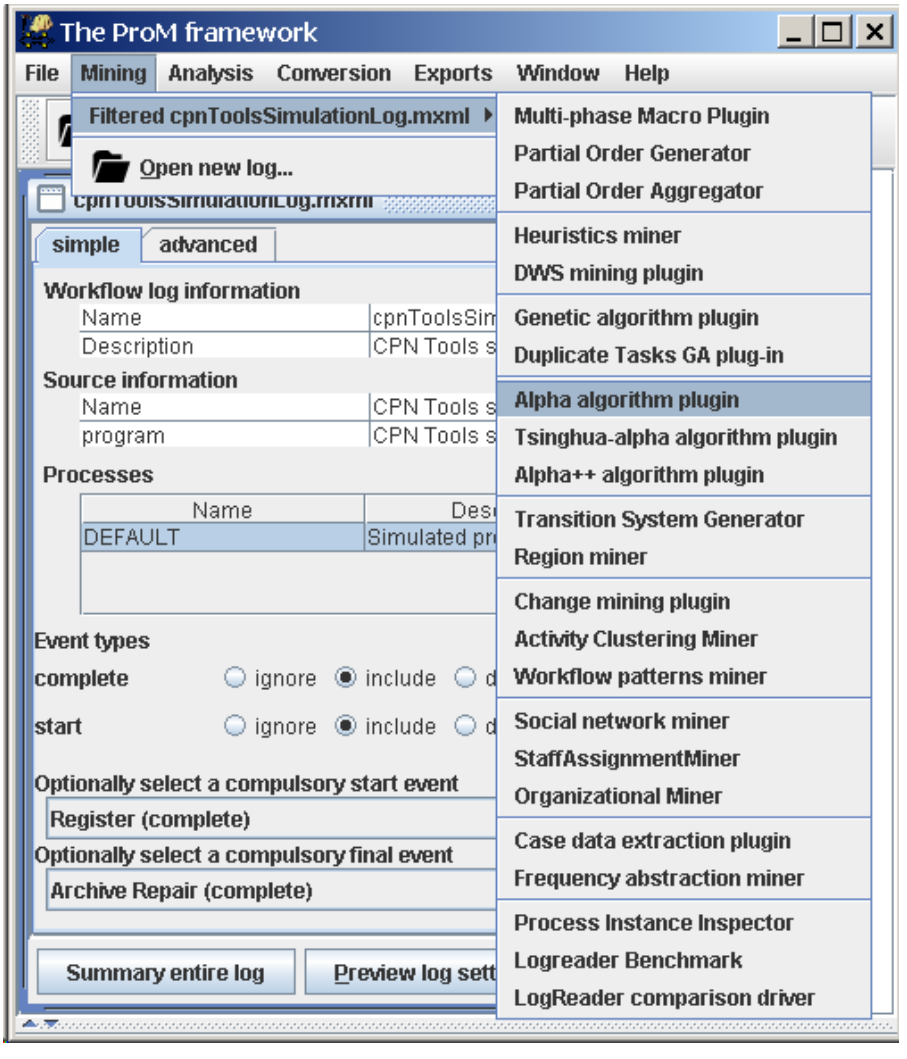


**Fig. 8.** Scheenshot of the mining plug-ins in ProM

4. Click the button *Start mining*. The resulting mined model should look like the one in Figure 9. Note that the *Alpha algorithm plugin* uses Petri nets[3] as its notation to represent process models. From this mined model, you can observe that:

[3] 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 [39].

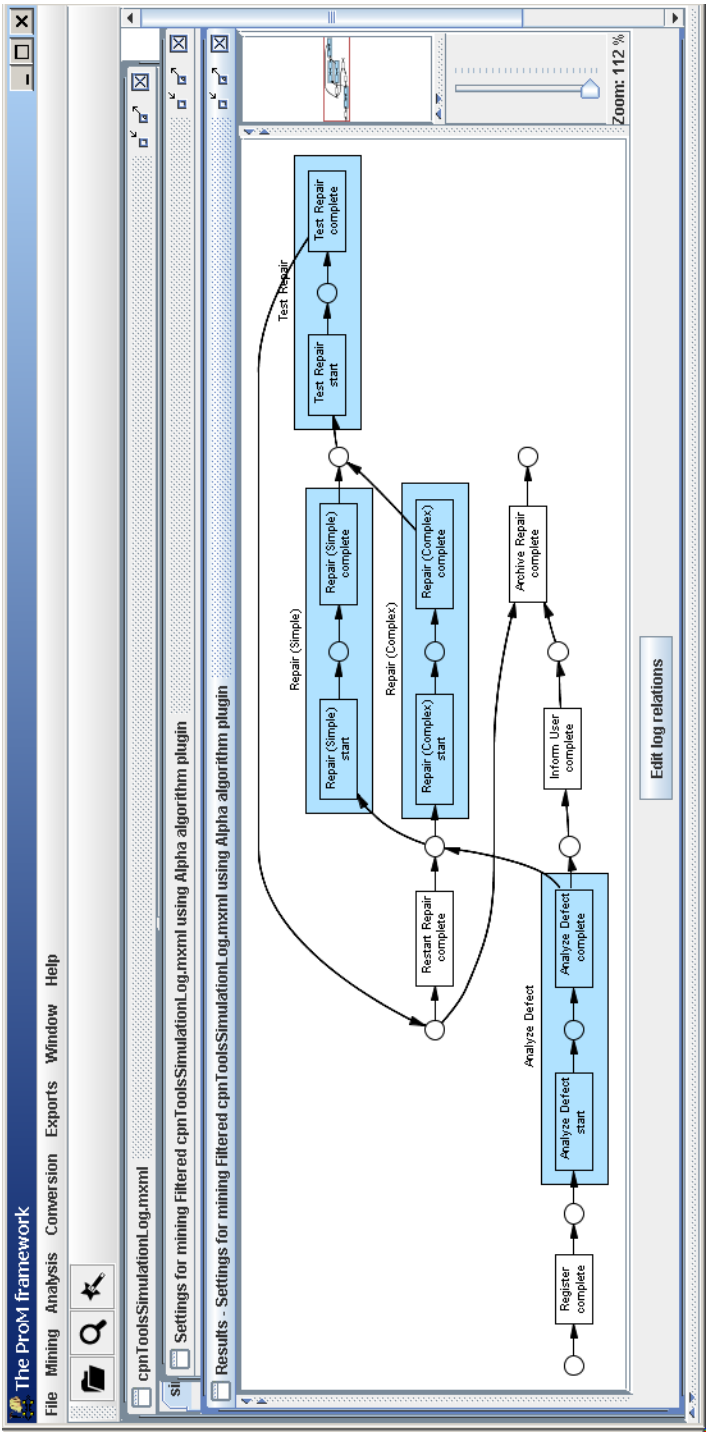


**Fig. 9.** Scheenshot of the mined model for the log of the running example

- 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.
- 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 2.1.

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 Subsection 2.5.
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.

### 2.4.2 Mining Case-Related Information about a Process

Do you want to 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 attempt) 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 2.3.2).
2. Run the *Performance Sequence Diagram Analysis* by choosing the menu *Analysis→[log name...]→Performance Sequence Diagram Analysis*.
3. Select the tab *Pattern diagram* and click on the button *Show diagram*. You should get a screen like the one in Figure 10.

   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

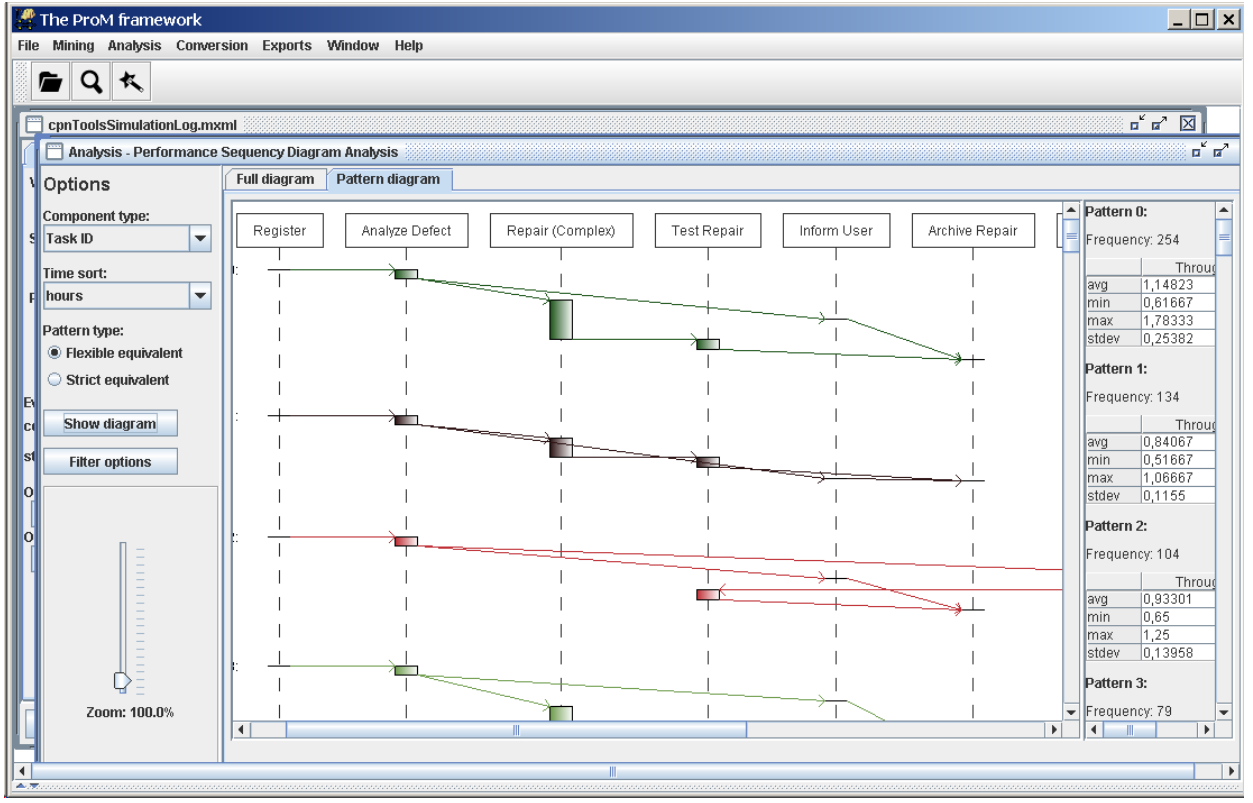


**Fig. 10.** Scheenshot of the analysis plug-in *Performance Sequence Diagram Analysis*. The configuration options are on the left side, the sequence diagrams are on the middle and the patterns frequence and throughput times are on the right side.

have probably notice that the 73,5% of the defects could be fixed in the first attempt[4].

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.

### 2.4.3 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?
- 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 2.3.1). 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 2.3.2) 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:

[4] 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.

1. Open the filtered log (cf. Section 2.3.2) 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 11. 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 52 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 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 2.1) 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 at the *handover of work* for the tasks performed by the people in this department. In other words, we can have a look at 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 2.3.2) to filter the log so that only the four tasks "Repair (Simple) (start)", "Repair (Simple) (complete)", "Repair (Complex) (start)" 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* (cf. Figure 8).
4. Select the tab *Handover of work*, and click the button *Start mining*. You should get a result like the one in Figure 12. 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

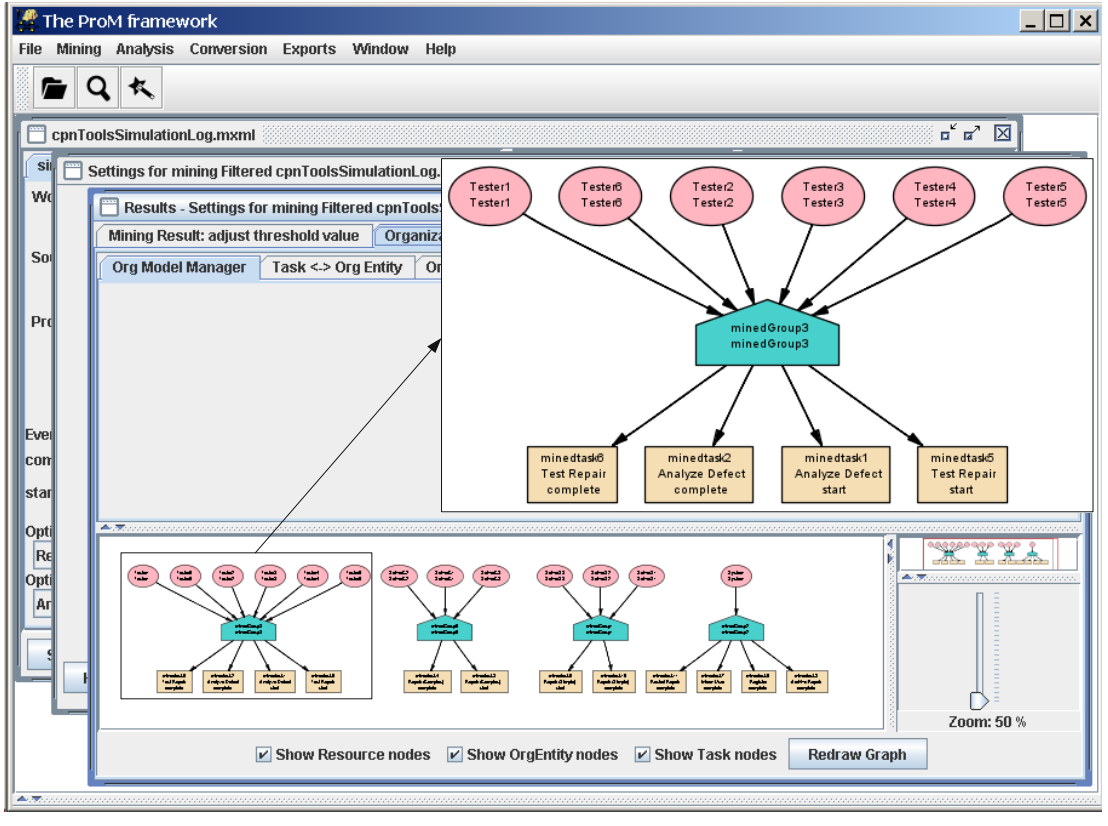


**Fig. 11.** Scheenshot of the mining plug-in *Organizational Miner*

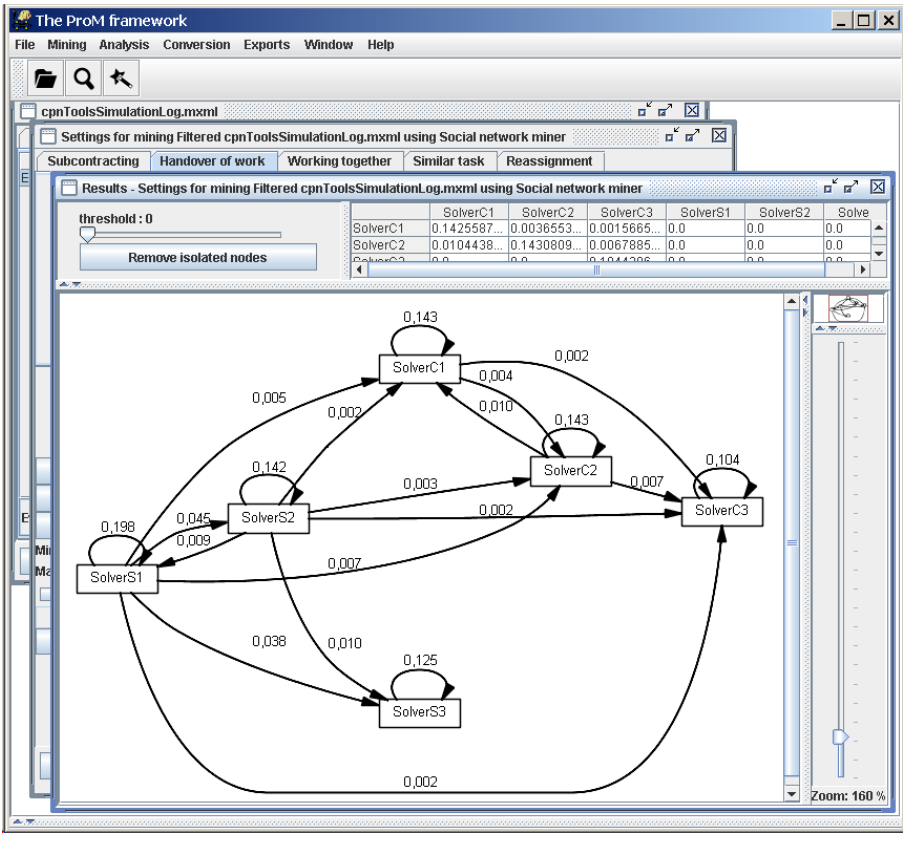


**Fig. 12.** Scheenshot of the mining plug-in *Social Network Miner*

mouse to re-arrange the nodes in the graph). The resulting graph (cf. Figure 13) 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 "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?

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

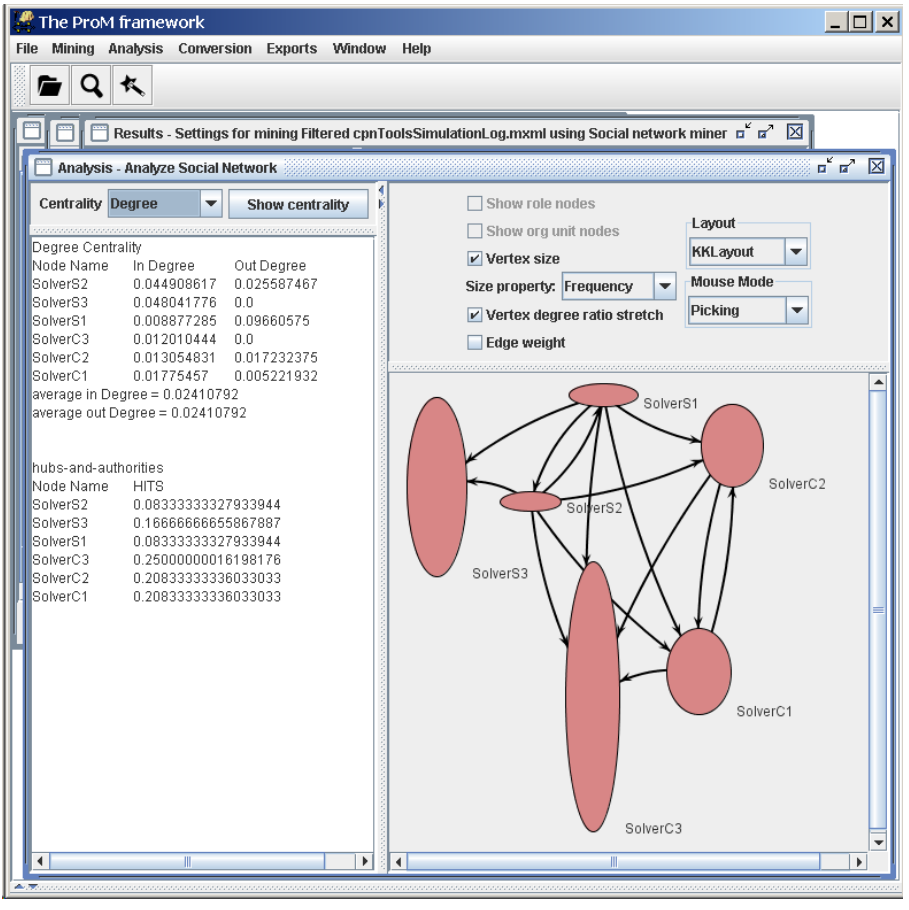


**Fig. 13.** Scheenshot of the mining plug-in *Analyzer Social Network*

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[5] and NetMiner[6], which are especially tailored to analyze social networks and provide more powerful user interfaces.

### 2.4.4 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 some tasks should not be executed by a same person within a process instance. 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

[5] http://agna.gq.nu
[6] http://www.netminer.com/

are indeed being obeyed is to audit the log with data about what has happened in an organization. In ProM, auditing is provided by the analysis plug-in *Default LTL Checker Plugin*[7].

From the description of our running example (cf. Section 2.1), 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 2.3.2) 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 14.
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.
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, for this log, 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[8].

[7] LTL stands for Linear Temporal Logic.

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

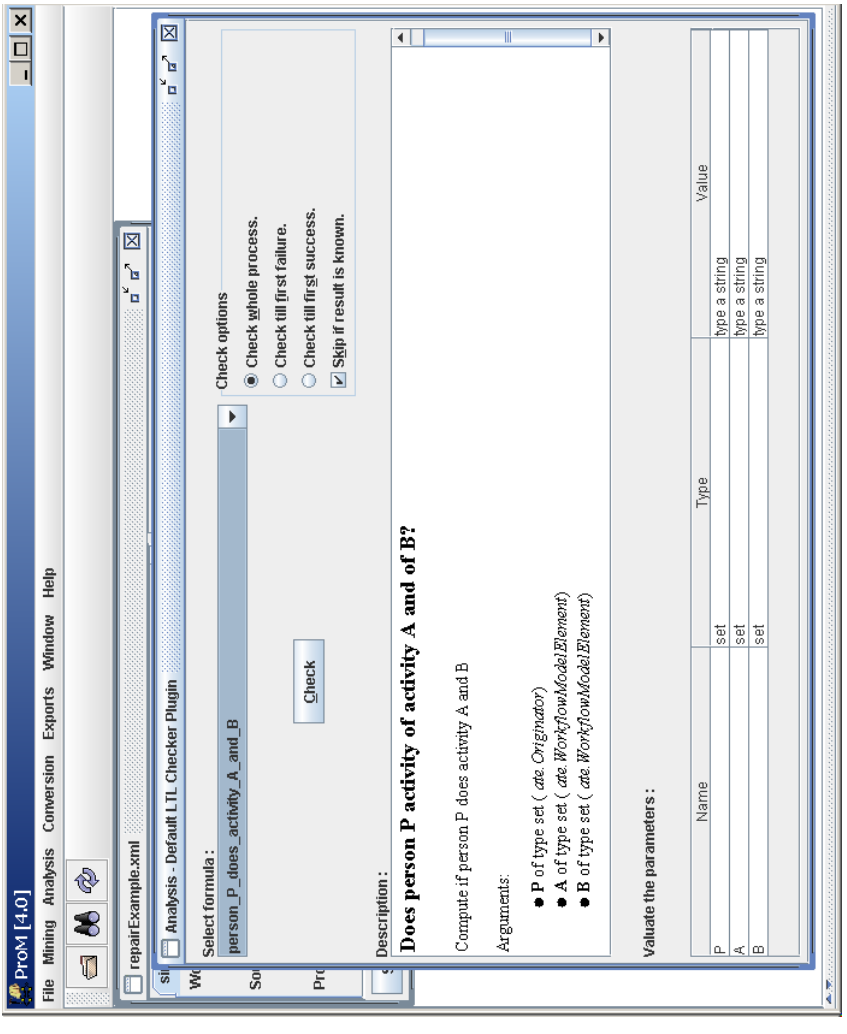


**Fig. 14.** Scheenshot of the analysis plug-in *Default LTL Checker Plugin*

**Table 5.** Conformance and Extension Plug-ins: questions and pointers to answers

| Question | Subsection |
|---|---|
| How compliant are the cases (i.e. process instances) with the deployed process models? Where are the problems? How frequent is the (non-) compliance? | 2.5.1 |
| What are the routing probabilities for each slipt/join task?<br>What is the average/minimum/maximum throughput time of cases?<br>Which paths take too much time on average? How many cases follow these routings? What are the critical sub-paths for these routes?<br>What is the average service time for each task?<br>How much time was spent between any two tasks in the process model? | 2.5.2 |
| What are the business rules in the process model? | 2.5.3 |

## 2.5 Questions Answered Based on a Process Model Plus an Event Log

In this section we explain the ProM analysis plug-ins that are used to answer the questions in Table 5. These plug-ins differ from the ones in Section 2.4 because they require a log *and* a (process) model as input (cf. Figure 2). Subsection 2.5.1 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. Subsections 2.5.2 and 2.5.3 describe *extension* ProM plug-ins that respectively extend the models with performance characteristics and business rules.

### 2.5.1 Conformance Checking

Nowadays, companies usually have some process-aware information system [21] 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 2.4.1, page 50) 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 is with the prescribed model. The procedure is the following:

1. Open the log "repairExampleSample2.zip". This log can be downloaded from *http://www.processmining.org/_media/tutorial/repairExampleSample2.zip*.
2. Open the exported PNML model that you created while executing the procedure on page 50.
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"[9], and click the button *Start analysis*. You should get results like the ones shown in figures 15 and 16, 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 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?

### 2.5.2 Performance Analysis

Like the *Conformance Checker* (cf. Section 2.5.1), the plug-in *Perfomance Analysis with Petri net* also requires a log and a Petri net as input[10]. 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 execute the *Perfomance Analysis with Petri net* analysis plug-in over the log of our running example, perform the following steps:

1. Open the filtered log (cf. Section 2.3.2) for the running example.
2. Open the exported PNML model that you created while executing the procedure on page 50.
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*.

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

[10] 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-ins to translate your model to a Petri net.

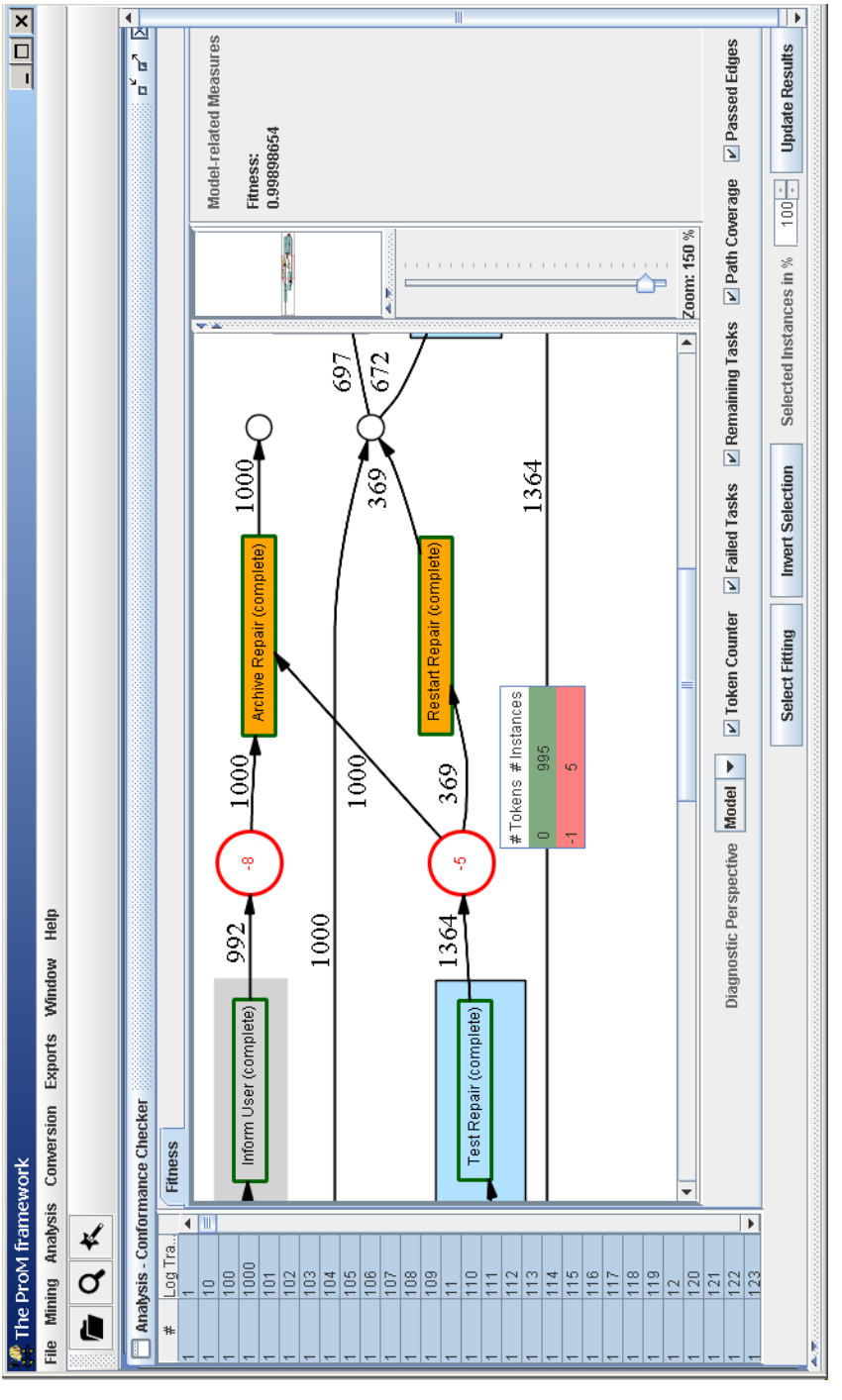


**Fig. 15.** Scheenshot of the analysis plug-in *Conformance Checker*: Model view

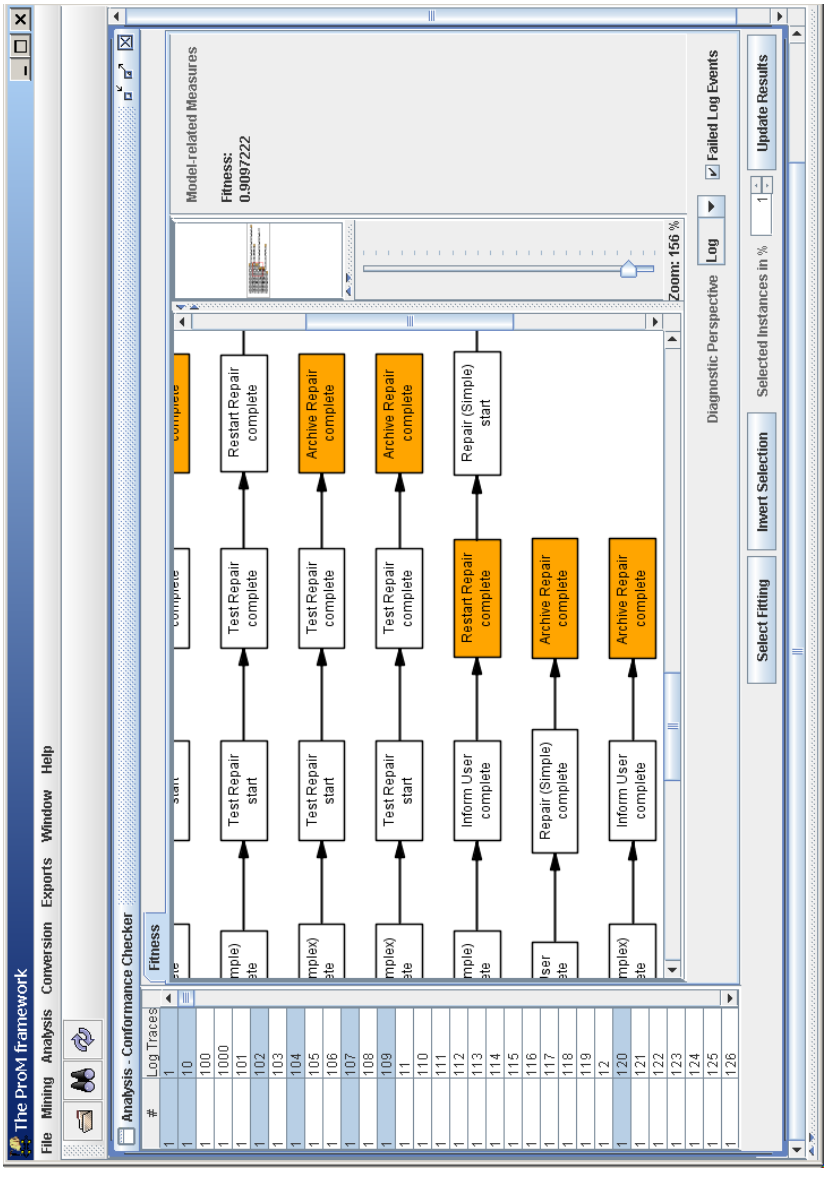


**Fig. 16.** Scheenshot of the analysis plug-in *Conformance Checker*: Log view

4. Set the field *Times measured in* to "hours" and click on the button *Start analysis*. The plug-in will start replaying the log and computing the time-related values. When it is ready, you should see a screen like the one in Figure 17.
5. Take your time to have a look at these results. Note that the right-side panel provides information about the *average/minimum/maximum throughput* times. The central panel (the one with the Petri net model) 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 2.4.2) in combination with the *Performance Analysis with Petri net*. In the context of our example, since the results in Figure 17 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 50.
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 can check for *critical sub-paths* by doing the remaining steps in this procedure.

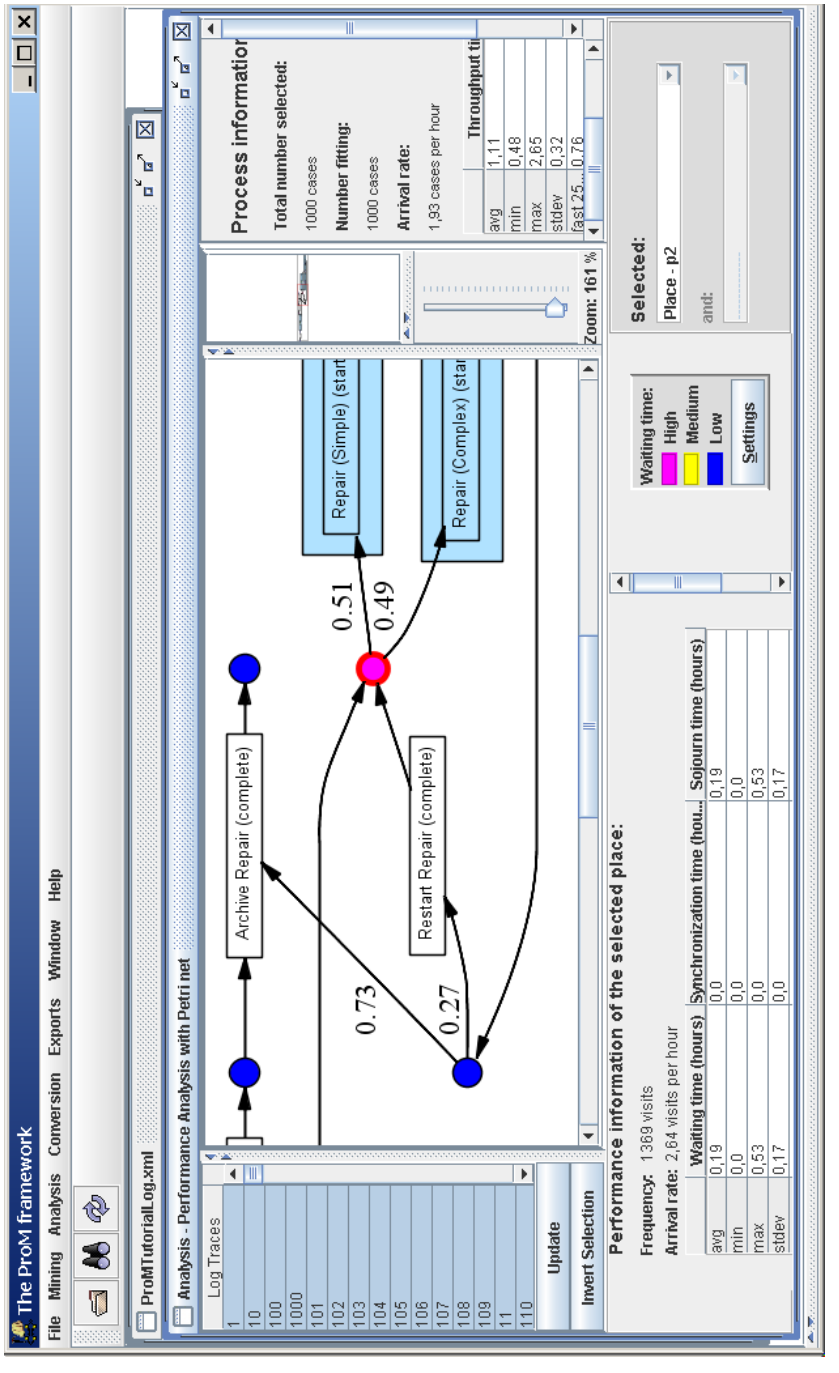


**Fig. 17.** Scheenshot of the analysis plug-in *Performance Analysis with Petri net*

5. Open exported PNML model that you created while executing the procedure on page 50, 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 analyze 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.

### 2.5.3 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 2.1) are fixed by which team. The procedure to do so has the following steps:

1. Open the filtered log (cf. Section 2.3.2) for the running example.
2. Open the exported PNML model that you created while executing the procedure on page 50.
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)"[11].
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*. 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 18). Note that cases with a defect types[12] from 1 to 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

[11] **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.

[12] From the problem description (cf. Section 2.1), we know that there are 10 types of defects.

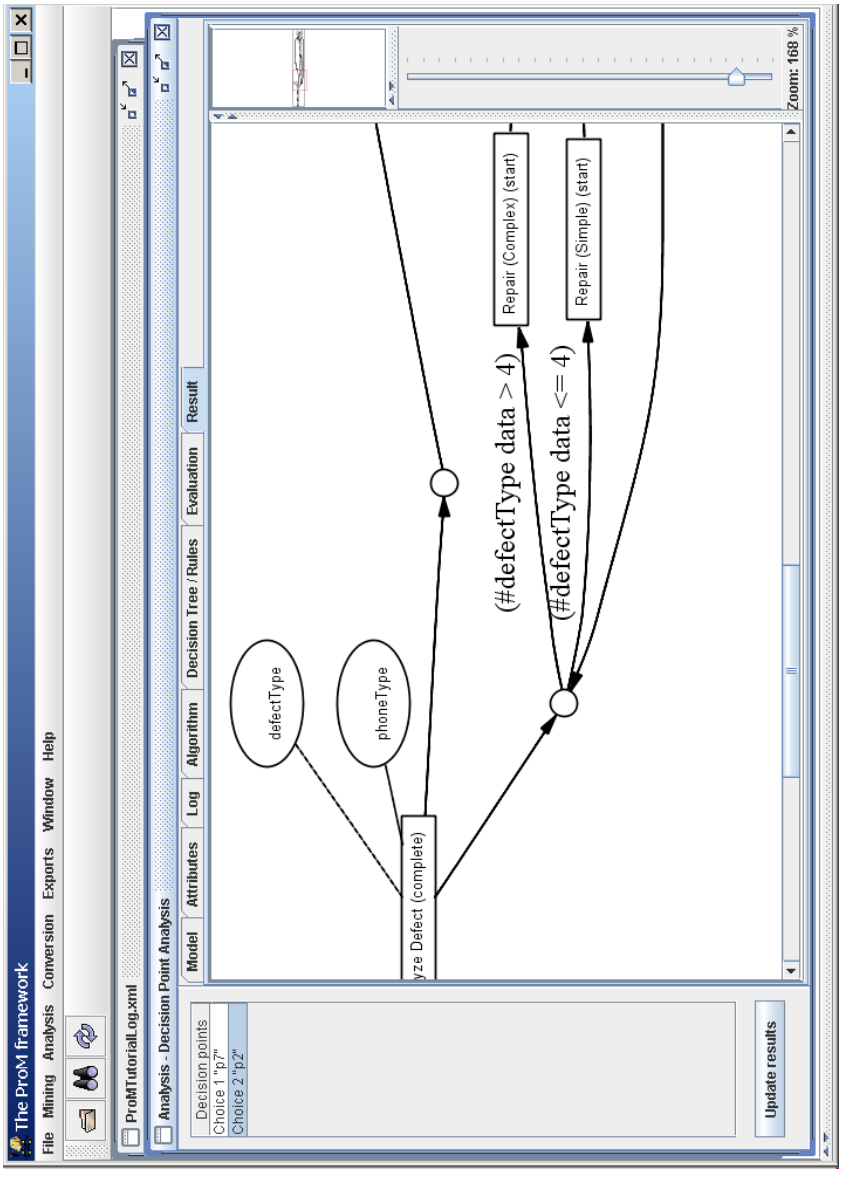


**Fig. 18.** Scheenshot of the analysis plug-in *Decision Point Analysis*: *Result* tab

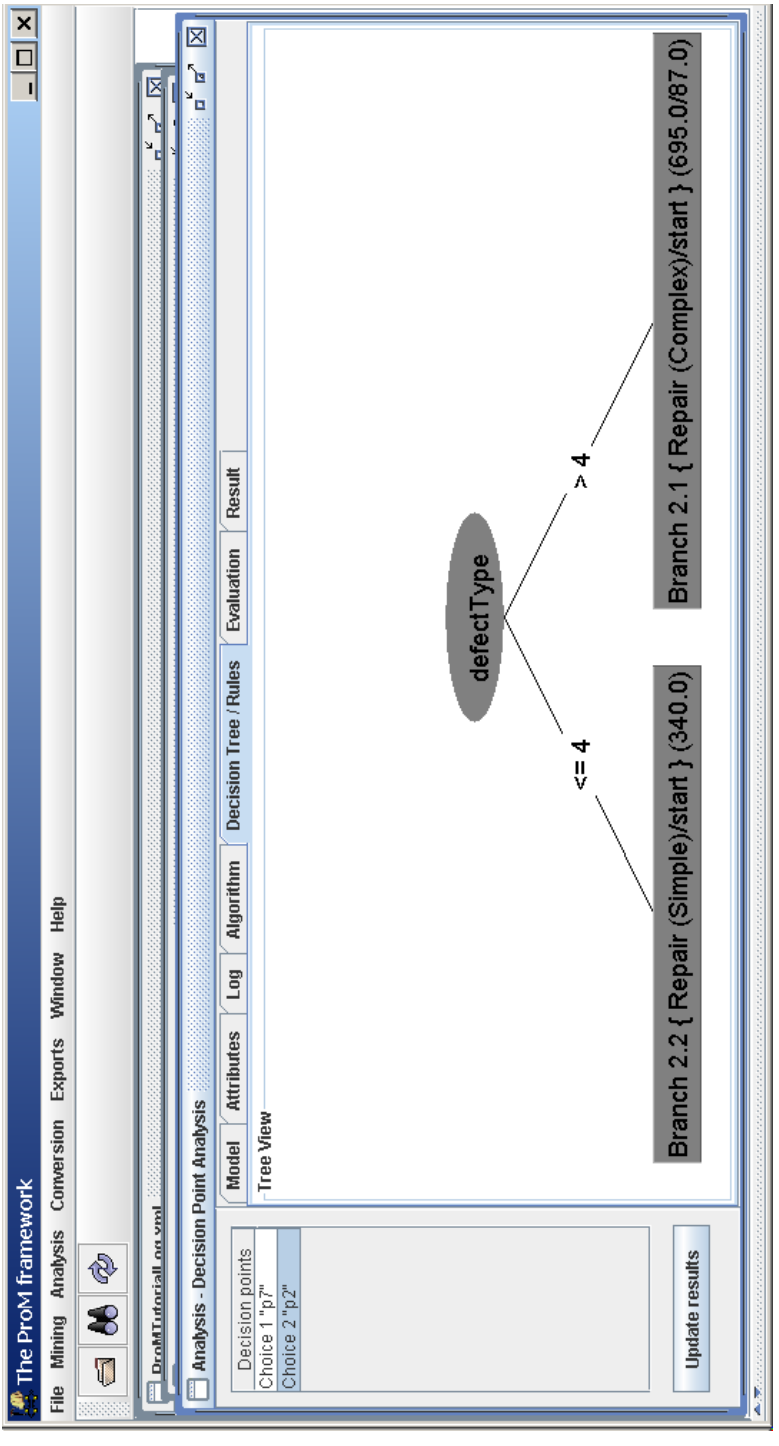


**Fig. 19.** Scheenshot of the analysis plug-in *Decision Point Analysis*: *Decision Tree/Rules* tab

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 19). 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 2.1), 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.

The explanations in this section show that the current process mining techniques can be used to answer the set of typical common questions (cf. Table 2) for the analysis of business processes. However, the level of abstraction and reuse provided by these techniques for this analysis is quite limited because all the queries are based on a label-level. For instance, it is not possible to define a generic property that checks for the "four-eyes principle". Therefore, the next section motivates which benefits the use of semantics could bring to current process mining techniques.

## 3 Semantic Process Mining

Semantic process mining aims at bringing the current process mining techniques from the level of *label-based* analysis to the level of *concept-based* analysis. Recall that the starting point of any mining algorithm is a log and that some techniques also use a model as input (cf. Section 2, Figure 2). Thus, the core idea in semantic process mining is to explicitly relate (or annotate) elements in a log with the concepts that they represent. This can be achieved by linking these elements to concepts in *ontologies*.

As explained in [25], "an ontology is a formal explicit specification of a shared conceptualization". Therefore, ontologies define (i) a set of concepts used by (a group of) people to refer to things in the world and (ii) the relationships among these concepts. Furthermore, because the concepts and relationships are *formally* defined, it is possible to *automatically reason or infer other relationships* among these concepts. In fact, ontologies are currently been used for modelling and for performing certain types of analysis in business processes [14,23,24,37]. Thus, it is realistic to base semantic process mining on ontologies.

To illustrate how ontologies can enhance the analysis provided by process mining tools, let us return to our running example (cf. Section 2.1). For this example, one could use the ontologies in Figure 20 to express the concepts in the business process for repairing the telephones. This figure shows three ontologies: *TaskOntology*, *RoleOntology* and *PerformerOntology*. In this figure, the concepts are modelled by ellipses and the instances of the concepts by rectangles. Additionally, the arrows define the relationships between the concepts and the instances. The arrows go from a concept to a superconcept, or from an instance to a concept. By looking at these ontologies, it is possible to infer subsumption relations among the concepts and instances. For instance, it is possible to identify that the elements "Repair (Complex)" and"Repair (Simple)" are *tasks* for *repairing* purposes. The use of ontologies (and the automatic reasoning they support) enables process mining techniques to analyze at different abstraction levels (i.e., at the level of instances and/or concepts) and, therefore, promotes re-use. Remark that current process mining techniques only provide for analysis based on the "instance" level. As an illustration, consider the last procedure on page 53. In this procedure we had to filter the log so that only the tasks "Repair (Simple)" and "Repair (Complex)" would be kept in the log. Now, assume that the elements in the log would link to the ontologies as illustrated in Figure 20. In this setting, the filtering performed on page 53 (cf. Step 2) could be simplified **from** *keep all the tasks "Repair (Simple)" and "Repair (Complex)"* **to** *keep all the tasks linking to the concept "TaskOntology:Repair"* [13]. Note that the filtering now specifies a *set of concepts* to maintain in the log. Therefore, it is up to the process mining log filter to analyze the concepts in the log and automatically infer that the tasks "Repair (Simple)" and "Repair (Complex)" should be kept in the log because these two tasks link to *subconcepts* of the concept "TaskOntology:Repair".

The remainder of this section provides an outlook on the possibilities for semantic process mining. Our aim is not to describe concrete semantic process mining algorithms, but rather motivate the opportunities and identify the elements that are needed to proceed towards these semantic algorithms. This is done in Subsection 3.1. Additionally, as a first step for realizing semantic process mining, Subsection 3.2 explains a concrete semantically annotated extension of the input format for logs mined by the ProM tool. The elements in this semantically annotated format support the kind of analysis discussed in the usage scenarios.

### 3.1 Usage Scenarios

The usage scenarios presented in this section illustrate the benefits of bringing the analysis discussed in Section 2 to the *semantic* level. While describing these scenarios, we assume that (i) the elements in event logs and models given as input link to concepts in ontologies, and (ii) the mining algorithms are able to load and infer subsumption relationships about the concepts that are referenced

[13] In this section, we use the notation "<ontology name>:<ontology concept>" when referring to concepts of the ontologies illustrated in Figure 20.

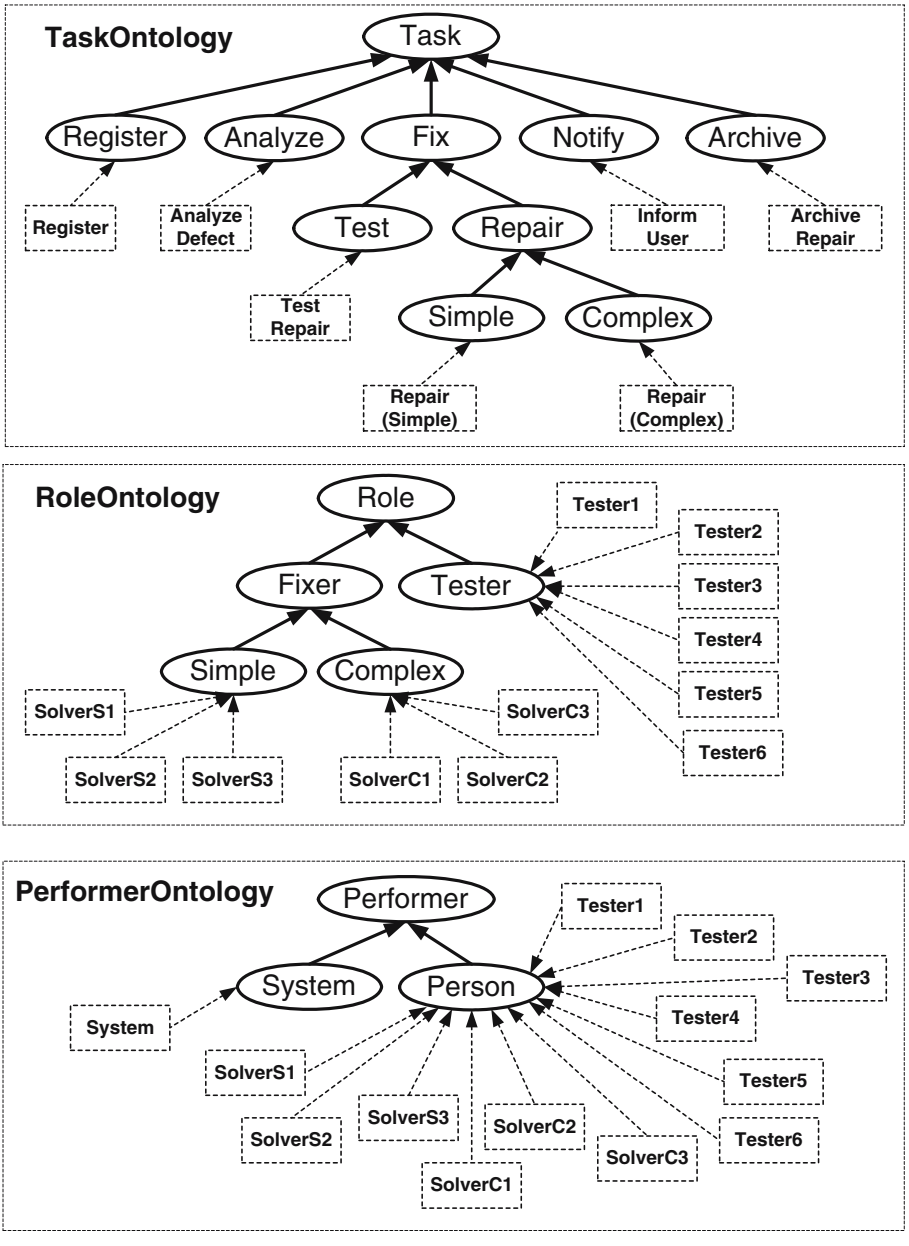


**Fig. 20.** Example of ontologies that could be defined for the running example in Section 2.1. Concepts are represented by ellipses and instances by rectangles. The root ellipses specify the most generic concepts and the leaf ellipses the most specific subconcepts.

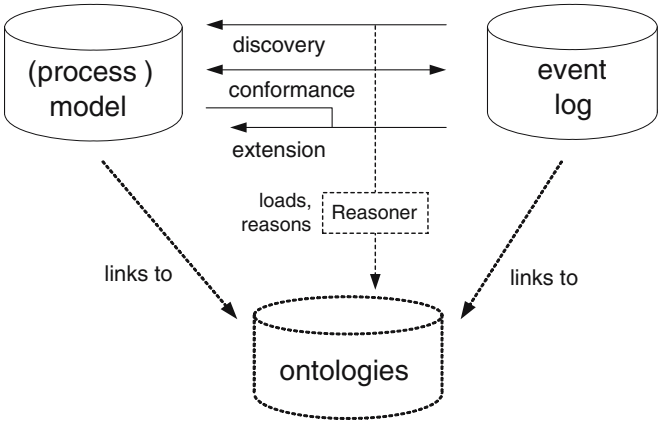


**Fig. 21.** Sources of information for *semantic* process mining. The additional elements (with dashed lines) are necessary to support the mining at the conceptual level(cf. Figure 2).

in these logs/models. Both the loading of the ontologies and the respective subsumption inferences are provided by ontology reasoners [3,4,5,26,36]. Figure 21 describes the sources of information for semantic process mining. In the following we elaborate on the usage scenarios. All the scenarios are based on the ontologies in Figure 20 and the running example in Subsection 2.1.

**Scenarios for Log Inspection.** Log inspection provides an overview of the elements in the log (cf. Subsection 2.3.1). Thus, when a log has links to concepts in ontologies, the log inspection techniques could also give an overview of different semantic perspectives. As a minimal requirement, these techniques should support the visualization of the concepts that are referenced in the log, their instances (i.e. the actual elements), and all the superconcepts of these concepts. This is necessary because the user needs to know which concepts he/she can use while performing the mining. Other possibilities would be to automatically find out (i) completed cases (or process instances) in the log (e.g., the ontologies could contain axioms that would define when instances of a given process were completed), (ii) which roles are in the log and which tasks were executed by originators with these roles, (iii) which concepts link to which labels, (iv) which tasks are executed by people and which by systems, and so on.

**Scenarios for Cleaning the Log.** Log cleaning allows for projecting and/or adding data to a log (cf. Subsection 2.3.2). Thus, a semantically annotated log could also be cleaned based on the concepts its elements link to. For instance, keep in the log only the tasks that are instances (i.e., link to) concepts X, W and Z or any of their subconcepts. Note that log filters defined over concepts are more generic because they can be reused over multiple logs involving the same concepts. At the current situation, the reuse of log filters

only makes sense when the logs have elements with identical labels. Thus, the use of concepts would boost the (re-)use of log filters.

**Scenarios for Discovery Plug-ins.** The analysis performed by discovery plug-ins is based on the log only (cf. Subsection 2.4). The *control-flow mining plug-ins* (cf. Subsection 2.4.1) discover a process model by inferring ordering relations between tasks in a log. When these tasks link to concepts in ontologies, these algorithms can mine process models at different levels of abstraction by inferring ordering relations between these concepts. The higher the considered concepts are in the subsumption trees derived from ontologies, the higher the level of abstraction of the mined models. For instance, for the ontology "TaskOntology", if a control-flow mining algorithm would use only the concepts at level 1 of its tree (i.e., use only the concepts "Register", "Analyze", "Fix", "Notify" and "Archive"), a process model like the one in Figure 22 could be discovered. In a similar way, if this same algorithm would use the instances of the concepts in this ontology, the mined model could be just like the one in Figure 9. Note that the mined model in Figure 22 is more compact (i.e., has a higher abstraction level) than the one in Figure 9. In as similar way, the *case-related information* plug-ins (cf. Subsection 2.4.2) could show most frequent paths with respect to concepts in the log. For the plug-ins that mine *organizational-related information* (cf. Subsection 2.4.3), the ontological concepts linked to the originator and tasks in the log would allow for a more precise analysis of the organizational model expressed in the log. For instance, consider the automatically discovered organizational model for the running example (cf. Figure 11). In this case, the tasks "Analyze Defect..." and "Test Repair..." are grouped together because they are executed by the same originators. However, if the link to ontologies would be present in the log, the groups in Figure 23 could be automatically inferred. Note that the use of ontological concepts would make it possible to (i) distinguish between the tasks "Analyze Defect..." and "Test Repair..." and (ii) identify that all originator "Tester..." have two roles: "RoleOntology:Classifier" and "RoleOntology:Tester". Finally, mining plug-ins for *verification of properties in the log* (cf. Subsection 2.4.4) could also benefit from a reasoning at the concept level. For instance, in the procedure on page 57, we have checked if *all "repaired" devices would always be tested before archiving*. To do so, we had to explicitly inform the *Default LTL Checker Plugin* the labels for the two repair tasks "Repair (Simple)" and "Repair (Complex)". If there were concepts in the log, this verification could be simplified to the formula "eventually_concept_A_then_B_then_C", where A = "TaskOntology:Repair", B = "TaskOntology:Test" and C = "TaskOntology:Archive". By using ontology reasoners, the plug-in would automatically find out the appropriate task labels to verify. Furthermore, like it happens for the log filters, LTL formulae defined over concepts can be more easily reused than the ones defined over labels.

**Scenarios for Conformance and Extension Plug-ins.** These plug-in enhance existing models by adding to them extra information discovered from

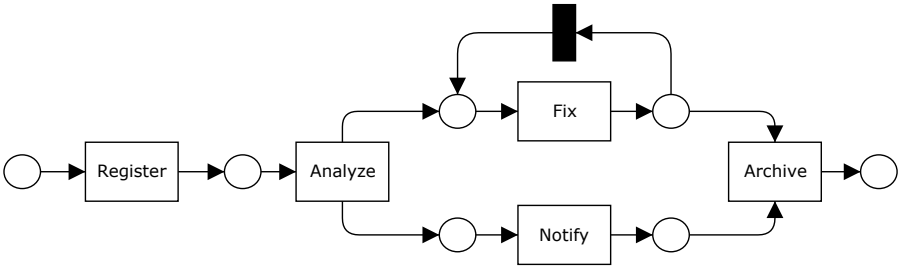


**Fig. 22.** Example of a mined model for the running example when only the concepts at the level 1 of the tree for the ontology "TaskOntology" (cf. Figure 20) are considered. Note that this mined model is more compact than the one in Figure 9.

logs. Because they need a log and a model while executing their analysis, these plug-ins would require the provided (process) models to also contain links to concepts in ontologies. This way they can find out relations between elements in models and logs the surpass the string matching level. Therefore: (i) the *conformance checking* plug-ins (cf. Subsection 2.5.1) would be performed at the conceptual level (with subsumption inferences taken into account), as well as the automatically suggested mapping between tasks in the log and in the model (cf. Step 3, on page 59); (ii) the *performance analysis* plug-ins (cf. Subsection 2.5.2) would be able to answer questions like "What are the routing probabilities of split/joint points of a *certain concept*?" or "Given tasks of a certain concept, which subconcepts outperform/underperform others in terms of service times?"; and (iii) *decision point analysis* plug-ins (cf. Subsection 2.5.3) would be able to automatically infer if a data value is *nominal* or *numeric*.

The starting point to realize the semantic process mining techniques illustrated in this subsection is to define a *semantic annotated format for event logs*. This format is the subject of the next subsection.

### 3.2 Semantic Annotated Mining XML Format

The Semantic Annotated Mining eXtensible Markup Language (SA-MXML) format is a *semantic annotated version* of the MXML format used by the ProM framework. In short, the SA-MXML incorporates the *model references* (between elements in logs and concepts in ontologies) that are necessary to implement our approach. However, before explaining the SA-MXML, let us first introduce the MXML format.

The Mining XML format (MXML) started as an initiative to share a common input format among different mining tools [11]. This way, event logs could be shared among different mining tools. The schema for the MXML format (depicted in Figure 24) is available at is.tm.tue.nl/research/processmining/Workflow-Log.xsd.

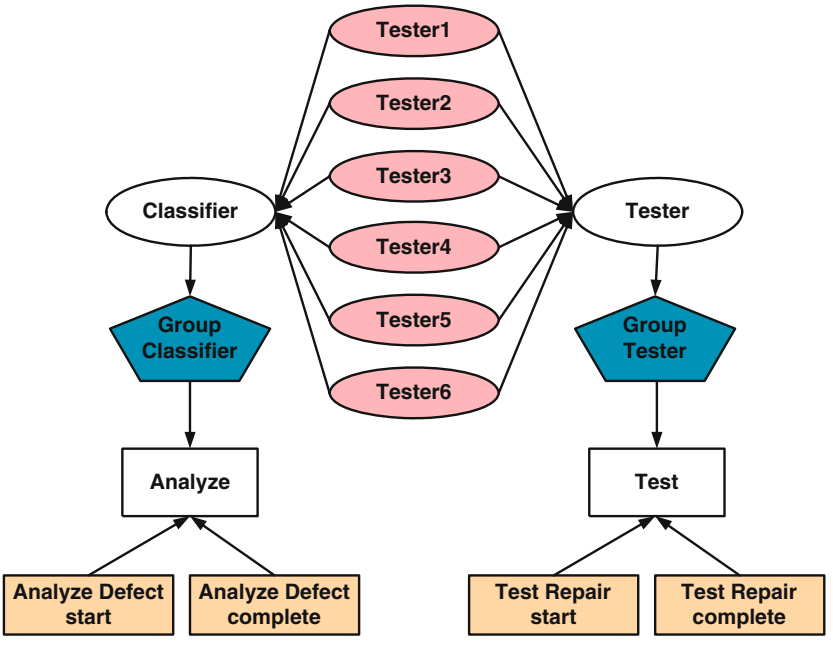


**Fig. 23.** Example of an automatically inferred organizational model based on a semantically annotated log for the running example. In this example we assume that in the log (i) the tasks "Analyze Defect..." link to the concept "TaskOntology:Analyze" and the tasks "Test Repair..." to "TaskOntology:Repair"; and (ii) the originators "Tester..." link to the concept "RoleOntology:Tester" when executing the tasks "Test Repair..." and to the concept "RoleOntology:Classifier" while performing the task "Analyze Defect...". Note that this model more precisely identifies the groups in the log than the one in Figure 11.

As can be seen in Figure 24, an event log (element *WorkflowLog*) contains the execution of one or more processes (element *Process*), and optional information about the source program that generated the log (element *Source*) and additional data elements (element *Data*). Every process (element *Process*) has zero or more cases or process instances (element *ProcessInstance).* Similarly, every process instance has zero or more tasks (element *AuditTrailEntry*). Every task or audit trail entry (ATE) must *at least* have a name (element *WorkflowModelElement*) and an event type (element *EventType*). The event type determines the state of the tasks. There are 13 supported event types: schedule, assign, reassign, start, resume, suspend, autoskip, manualskip, withdraw, complete, ate_abort, pi_abort and unknown. The other task elements are optional. The *Timestamp* element supports the logging of time for the task. The *Originator* element records the person/system that performed the task. The *Data* element allows for the logging of additional information. Figure 25 shows an excerpt of the running example

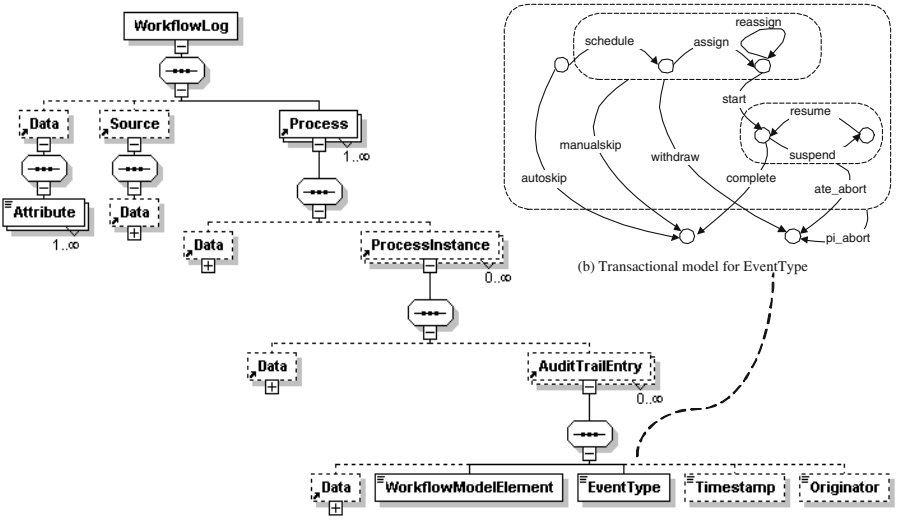


**Fig. 24.** The visual description of the schema for the Mining XML (MXML) format

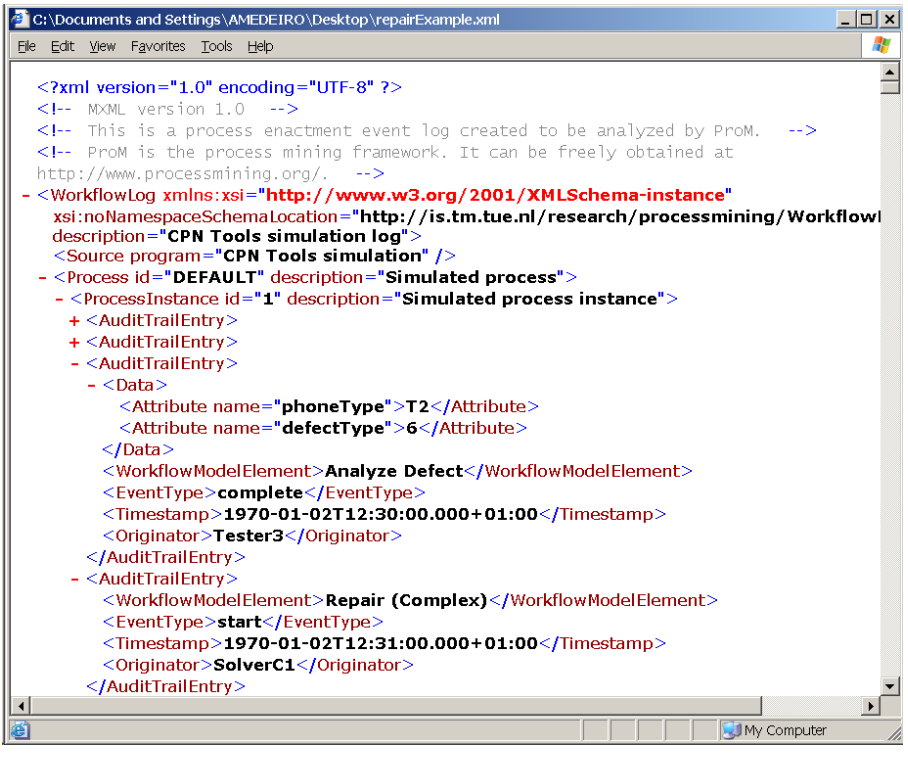


**Fig. 25.** Excerpt of the running example log (cf. Section 2.1) in the the MXML format

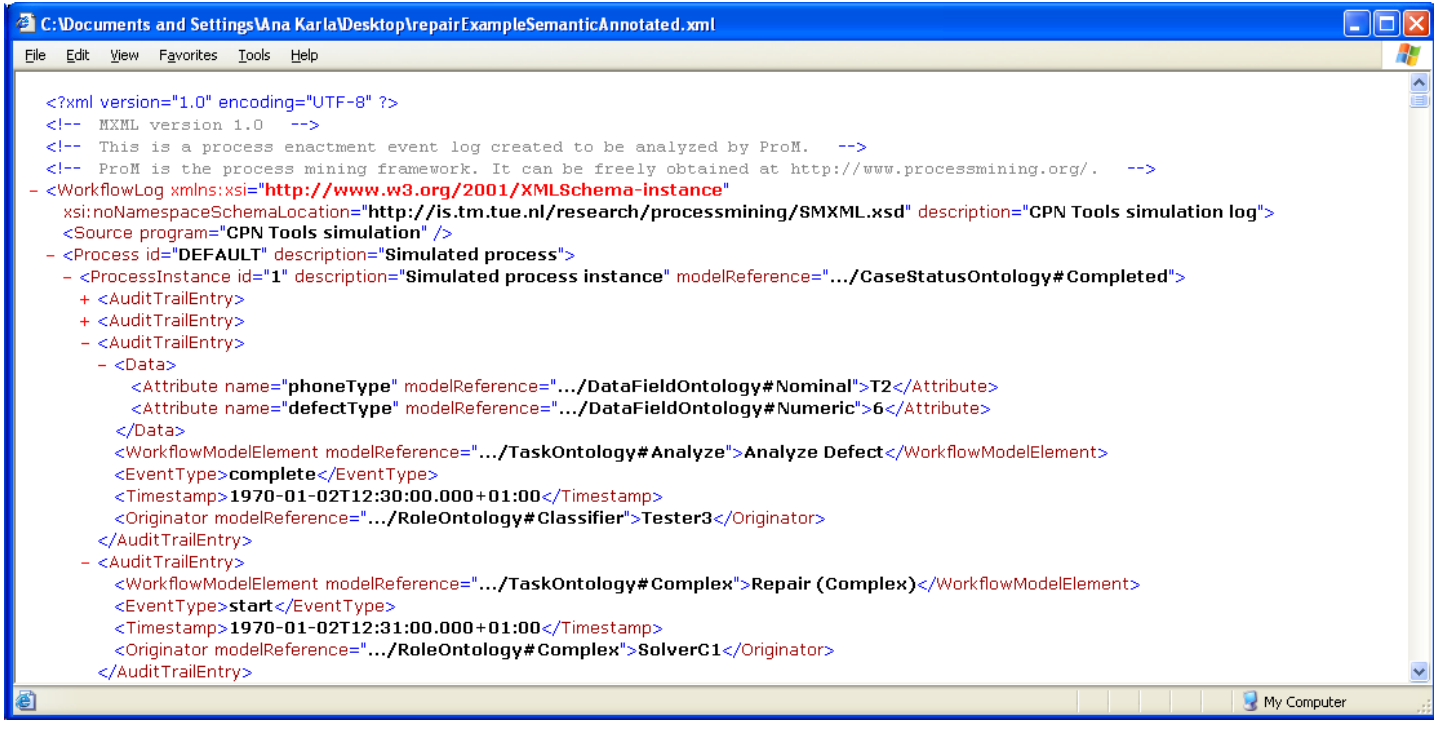


**Fig. 26.** Excerpt of the running example log (cf. Section 2.1) in the SA-MXML format. Note that the optional field *modelReference* is used to link the elements of the MXML format to concepts in ontologies.

(cf. Subsection 2.1) log in the MXML format. More details about the MXML format can be found in [18,19].

The SA-MXML format is just like the MXML format plus the addition that *all elements (except for AuditTrailEntry and Timestamp) have an optional extra attribute* called *modelReference*. This attribute links to a *list of concepts* in ontologies and, therefore, support the necessary *model references* for our approach. The concepts are expressed as URIs and the elements in the list are separated by blank spaces. Actually, the use of *modelReference* in the SA-MXML format is based on the work for the semantic annotations provided by SAWSDL (Semantically Annotated Web Service Definition Language) [7]. The schema for the SA-MXML format is available at is.tm.tue.nl/research/processmining/SAMXML.xsd. Figure 3.2 shows an example of a log with semantic annotations for the log of our running example with respect to the ontologies in Figure 20. Note that the fields "ProcessInstance", "Data", "WorkflowModelElement" and "Originator" link to concepts in the ontologies. Furthermore, note that the SA-MXML format is *backwards compatible* with MXML format. This way the process mining techniques that do not support a semantic treatment can also be directly applied to logs in SA-MXML.

## 4 Conclusions

In this chapter we have shown (i) how to use the open source process mining tool ProM to get useful feedback about processes, (ii) how the inclusion of semantic information in event logs can empower process mining techniques, and (iii) we have defined a concrete semantic log format, the SA-MXML format.

Since our focus when illustrating the power of current process mining techniques was on answering a set of common questions that managers usually have about processes, we have not covered many of the other plug-ins that are in ProM. We hope that the subset we have shown in this chapter 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.

Embedding semantic information in event logs brings the process mining techniques from the level of label-based analysis to the concept-based one. This allows for working with different levels of abstractions while getting feedback about processes and properties in a log. Furthermore, it also supports easier reuse of queries defined over logs.

The SA-MXML format is the first step towards creating semantic process mining algorithms. This format extends the current MXML format by specifying that any element present in the MXML format may link to a set of concepts in

ontologies. Following steps will focus on developing semantic process mining algorithms to implement the usage scenarios described in this chapter.

## Acknowledgements

The authors would like to thank Ton Weijters, Boudewijn van Dongen, Anne Rozinat, Christian Günther, Minseok Song, Ronny Mans, Huub de Beer, Laura Maruster and Peter van den Brand for their on-going work on process mining techniques and tools at Eindhoven University of Technology. Additionally, the authors would like to thank Monique Jansen-Vullers, Mariska Netjes, Irene Vanderfeesten and Hajo Reijers for their input while selecting the common questions used in Section 2. The work presented in this chapter was partially funded by the European Commission under the project SUPER (FP6-026850).

## References

1. COSA Business Process Management, http://www.cosa-bpm.com/
2. Extensible Markup Language (XML), http://www.w3.org/XML/
3. KAON2, http://kaon2.semanticweb.org/
4. Pellet, http://pellet.owldl.com/
5. Racer, http://www.racer-systems.com/
6. SAP, http://www.sap.com/
7. Semantic Annotations for Web Service Description Language (SA-WSDL), http://www.w3.org/TR/2006/WD-sawsdl-20060630/
8. Staffware Process Suite, http://www.staffware.com/
9. van der Aalst, W.M.P., de Beer, H.T., van Dongen, B.F.: Process Mining and Verification of Properties: An Approach Based on Temporal Logic. In: Meersman, R., Tari, Z. (eds.) OTM 2005. LNCS, vol. 3760, pp. 130–147. Springer, Heidelberg (2005)
10. van der Aalst, W.M.P., van Hee, K.M.: Workflow Management: Models, Methods, and Systems. MIT Press, Cambridge (2002)
11. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow Mining: A Survey of Issues and Approaches. Data and Knowledge Engineering 47(2), 237–267 (2003)
12. van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow Mining: Discovering Process Models from Event Logs. IEEE Transactions on Knowledge and Data Engineering 16(9), 1128–1142 (2004)
13. Bussler, C., Haller, A. (eds.): BPM 2005. LNCS, vol. 3812. Springer, Heidelberg (2006)
14. Casati, F., Shan, M.C.: Semantic Analysis of Business Process Executions. In: Jensen, C.S., Jeffery, K., Pokorný, J., Šaltenis, S., Bertino, E., Böhm, K., Jarke, M. (eds.) EDBT 2002. LNCS, vol. 2287, pp. 287–296. Springer, Heidelberg (2002)
15. Workflow Management Coalition. WFMC Home Page, http://www.wfmc.org
16. Cook, J.E., Du, Z., Liu, C., Wolf, A.L.: Discovering Models of Behavior for Concurrent Workflows. Computers in Industry 53(3), 297–319 (2004)
17. Desel, J., Esparza, J.: Free Choice Petri Nets. Cambridge Tracts in Theoretical Computer Science, vol. 40. Cambridge University Press, Cambridge (1995)

18. van Dongen, B.F., van der Aalst, W.M.P.: A Meta Model for Process Mining Data. In: Proceedings of the CAiSE 2005 WORKSHOPS, vol. 2. FEUP (2005)
19. van Dongen, B.F., Alves de Medeiros, A.K., Verbeek, H.M.W., Weijters, A.J.M.M., van der Aalst, W.M.P.: The ProM Framework: A New Era in Process Mining Tool Support. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 444–454. Springer, Heidelberg (2005)
20. van Dongen, B.F., van der Aalst, W.M.P.: Multi-phase Process mining: Aggregating Instance Graphs into EPCs and Petri Nets. In: Proceedings of the Second International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management (PNCWB) (2005)
21. Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H. (eds.): Process-Aware Information Systems: Bridging People and Software Through Process Technology. John Wiley & Sons Inc., Chichester (2005)
22. Greco, G., Guzzo, A., Pontieri, L.: Mining Hierarchies of Models: From Abstract Views to Concrete Specifications. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 32–47. Springer, Heidelberg (2005)
23. Green, P., Rosemann, M.: Business Systems Analysis with Ontologies. Idea Group Publishing (2005)
24. Grigori, D., Casati, F., Castellanos, M., Dayal, U., Sayal, M., Shan, M.C.: Mining Exact Models of Concurrent Workflows 53(3), 321–343 (2004)
25. Gruninger, M., Lee, J.: Special Issue on Ontology applications and design - Introduction. Communications of ACM 45(2), 39–41 (2002)
26. Haarslev, V., Möller, R.: Racer: A core inference engine for the semantic web. In: Sure, Y., Corcho, Ó. (eds.) EON. CEUR Workshop Proceedings, vol. 87 (2003) CEUR-WS.org
27. Hepp, M., Leymann, F., Domingue, J., Wahler, A., Fensel, D.: Semantic Business Process Management: a Vision Towards Using Semantic Web services for Business Process Management. In: IEEE International Conference on e-Business Engineering (ICEBE 2005), pp. 535–540 (2005)
28. Herbst, J., Karagiannis, D.: Workflow Mining with InWoLvE. Computers in Industry 53(3), 245–264 (2004)
29. Thao Ly, L., Rinderle, S., Dadam, P., Reichert, M.: Mining staff assignment rules from event-based data. In: Bussler and Haller [13], pp. 177–190
30. Murata, T.: Petri Nets: Properties, Analysis and Applications. Proceedings of the IEEE 77(4), 541–580 (1989)
31. Pinter, S.S., Golani, M.: Discovering Workflow Models from Activities Lifespans. Computers in Industry 53(3), 283–296 (2004)
32. Reisig, W., Rozenberg, G. (eds.): APN 1998. LNCS, vol. 1491. Springer, Heidelberg (1998)
33. Rozinat, A.: Decision mining in prom. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 420–425. Springer, Heidelberg (2006)
34. Rozinat, A., van der Aalst, W.M.P.: Conformance Testing: Measuring the Fit and Appropriateness of Event Logs and Process Models. In: Bussler and Haller [13], pp. 163–176
35. Schimm, G.: Mining Exact Models of Concurrent Workflows. Computers in Industry 53(3), 265–281 (2004)
36. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. Web Semantics: Science, Services and Agents on the World Wide Web 5(2), 51–53 (2007)

37. Thomas, M., Redmond, R., Yoon, V., Singh, R.: A semantic approach to monitor business process. Communications of ACM 48(12), 55–59 (2005)
38. van der Aalst, W.M.P., Reijers, H.A., Song, M.: Discovering Social Networks from Event Logs. Computer Supported Cooperative Work 14(6), 549–593 (2005)
39. Verbeek, H.M.W., van Dongen, B.F., Mendling, J., van der Aalst, W.M.P.: Interoperability in the ProM Framework. In: Latour, T., Petit, M. (eds.) Proceedings of the CAiSE 2006 Workshops and Doctoral Consortium, pp. 619–630. Presses Universitaires de Namur (June 2006)
40. Wen, L., Wang, J., Sun, J.: Detecting Implicit Dependencies Between Tasks from Event Logs. In: Zhou, X., Li, J., Shen, H.T., Kitsuregawa, M., Zhang, Y. (eds.) APWeb 2006. LNCS, vol. 3841, pp. 297–306. Springer, Heidelberg (2006)