TECHNISCHE UNIVERSITEIT EINDHOVEN

Department of Mathematics and Computer Science

Performance Analysis of Business Processes through Process Mining

P.T.G. Hornix

Supervisors:
Dr. A.J.M.M. Weijters (TM - IS)
Prof. Dr. P.M.E. De Bra (W&I - DH)
Dr. M. Voorhoeve (W&I - AIS)

Eindhoven, January 2007

Abstract

Process Mining (or *workflow mining*) is an important research area of the section Information Systems (IS/TM) of the department of Technology Management at Eindhoven University of Technology. The starting point of process mining is registered information about tasks that are performed in a process (handling of an insurance claim, treatments of a patient in a hospital etc.). This registered information is typically called an *event log*. The goal of process mining is deducing knowledge (e.g. a process model, average throughput time) from such event logs. The knowledge obtained this way can increase understanding of the processes within an organization and can help with redesigning them if needed.

Within the section IS/TM a process mining framework has been developed, called the *ProM framework*. So far, emphasis within this framework has been on the discovery and validation of process models (workflows). Herein, the ProM framework is practically unique. The analysis functionality that is currently present in the framework mainly consists of qualitative analysis. Main concern of qualitative analysis is the logical correctness of processes (absence of anomalies, like deadlocks and livelocks). For the practical usability of the framework however, functionality which supports analysis of the performance of processes (quantitative analysis) is also required.

The goal of this master's project is to extend the ProM framework with plug-ins that support process performance analysis. In this thesis, the research that was done during the project will be discussed and the plug-ins that have been implemented will be described.

Preface

This master's thesis concludes my study Computer Science at Eindhoven University of Technology. It describes my graduation project, which was issued by, and carried out at the Business Process Management group of the Department of Technology Management at Eindhoven University of Technology.

During my graduation project there have been many people who have supported and assisted me. I would like to take this opportunity to thank them. First of all, I would like to thank my daily supervisor Ton Weijters for giving me the opportunity to work on a very interesting project. Also all other researchers of the BPM group that helped me or presented me with their views and thoughts deserve my gratitude. Furthermore, I would like to thank my other supervisors, Marc Voorhoeve and Paul de Bra, for taking place in my assessment commission. Finally, I would like to thank my parents, brother, sister, and friends for supporting me throughout this project and, of course, also for their support beyond this project.

Peter Hornix
Stramproy, January 2007

Contents

Chapter 1

Research Assignment

1.1 Assignment Background

This assignment forms the graduation project of my study Computer Science at Eindhoven University of Technology. This assignment was issued by the Business Process Management (BPM) group of the section Information Systems (IS/TM) of the department of Technology Management at Eindhoven University of Technology.

1.2 Problem Area

Process Mining (also known as *workflow mining*) is an important, though relatively young research area. The starting point of process mining is registered information about tasks that are performed in a process (e.g. the handling of an insurance claim, the treatments of a patient in a hospital, requesting a building permit). This registered information is typically called an *event log*. The goal of process mining is deducing knowledge (a process model, average throughput times etc.) from such event logs. The knowledge obtained this way can increase understanding of the processes within an organization and can help with redesigning them if needed.

The *ProM framework* is a framework that was developed to support process mining. This framework is used by master Technology Management students to analyze comprehensive event logs of large organizations (e.g. hospitals, government). So far, emphasis within this framework has been on the discovery and validation of process models (workflows). Herein, the ProM framework is practically unique.

1.3 Assignment definition

The analysis functionality contained in the ProM framework at this moment, mainly consists of qualitative analysis. Qualitative analysis focuses on the logical correctness of processes

(absence of anomalies, like deadlocks and livelocks). In practical situations, a quantitative analysis is needed as well to determine how (parts of) business processes are actually performing. For obtaining relatively superficial process characteristics (average throughput time, execution time, etc.) there are commercial tools available (commonly known as *process monitoring tools*). Extending the ProM framework with such functionality is a first step to enhance the practical usability of ProM. The first research question therefore is:

Question 1: Which process mining functionality is available in commercially developed process monitoring tools and which parts of this functionality would be enhancements to the ProM framework?

The ProM framework can be used to derive an explicit process model from an event log. It seems possible to combine this process model with the information that is stored in the event log. For instance, when the process model contains a XOR-split, it could be examined whether there are case properties with which it is possible to predict the branch that is chosen at the XOR-split. The second research question is:

Question 2: Since we have a process model at our disposal within the ProM framework, which additional process mining functionality would qualify for implementation into the ProM framework?

For the answering of the first question, existing tools and functionality in the area of process performance analysis should be examined, as well as earlier master theses within the process mining research area. For answering the second question, discussion with other researchers within the BPM group is eminent. After answering the questions 1 and 2, decisions are to be made based on practical arguments (such as usability, implementation effort etc.) as to which functionality is to be added to the ProM framework (using *Java*). The implementation of the chosen functionality is part of the graduation assignment. User-friendliness and ease of use are of great importance here, because the functionality is to be used by Technology Management students to analyze practical situations.

1.4 Outline

The remainder of this thesis is structured as follows:
In chapter 2, the concept of process mining and the research in this area will be discussed. Thereafter, in chapter 3, the concept of process performance analysis will be addressed. Also the functionality that exists in the process performance analysis area and possible additions to this functionality will be discussed in chapter 3. Chapter 4 describes the requirements of the functionality that was chosen for implementation and in chapter 5, the design and implementation of the plug-ins that contain this functionality will be discussed. Finally, in chapter 6 conclusions will be presented and the results evaluated.

Chapter 2

Process Mining

2.1 Introduction

Since the emergence of disciplines as Business Process Management (BPM) and Business Process Reengineering (BPR) at the end of the 80's, organizations have been focusing more and more on their business processes in order to improve results and reduce costs. A business process can be defined as a related group of tasks that together create value for a (internal or external) customer [6]. Examples of business processes are the processing of orders in a factory or the handling of insurance claims at an insurance company. Before BPR and BPM, hardly any attention was given to these business processes. Because organizations often were (and are) divided into functional areas (e.g. accounting, production, sales), the main focus was on improving and optimizing these individual functions. The relationships between such functions did not get as much attention. With the emergence of BPR and BPM however, organizations started to realize that to strengthen their position and to improve their performance, they would have to improve their cross-functional business processes [8].

To improve and support business processes, many organizations have now installed Process-Aware Information Systems (PAIS). Typical PAIS include Workflow Management (WFM) systems, Enterprise Resource Planning (ERP) systems, Customer Relationship Management (CRM) systems, Software Configuration Management (SCM) systems, and Business to Business (B2B) systems. One of the main problems with these information systems however, is that they require the explicit design of process models that represent the business processes of the organization [24]. Correctly designing such process models is far from easy, because it requires much knowledge of the business processes. Gathering knowledge of business processes can cost much time (to discuss with the participants in the business process) and money. Moreover, in practice, these process models often turn out to be based more on what managers think that should be done, rather than on what actually is done.

This is where process mining first came into play. The goal of process mining is to extract

information about processes (i.e. a functional model) from event logs. Event logs are lists, in which information about processes is stored as they are actually executed. Most PAIS provide a log in which the executions of activities are recorded. In [21, 24, 25, 28], different methods to distil process models from event logs are described. We will not discuss these methods here. Instead we will give an example that illustrates how a process model can be derived from an event log.

Suppose we have the log as displayed in table 2.1. In cases 2, 3, 4, and 5 of this log, the tasks *examine patient*, *examine blood*, *make X-ray*, and *nurse patient* are executed. In case 1, the tasks *examine patient*, *perform surgery*, and *nurse patient* are executed. Clearly, all cases in the log begin with task *examine patient* and end with task *nurse patient*. These two tasks are thus the start and the finish of the process. In between these two tasks, the task *perform surgery* can be executed, or the tasks *make X-ray* and *examine blood* can be executed in arbitrary order. Assuming that the log consists of completed cases only and that the cases are representative and a sufficiently large subset of behaviors is observed, the process model as shown in figure 2.1 can be derived from this log. Since the tasks *examine blood* and *make X-ray* can occur in any order, they are modeled as parallel tasks. The shown process model is depicted in Petri net notation. We will not explain the syntax of Petri nets here (see [2, 12, 16] for a thorough description).

CaseID	**TaskID**	**Originator**	**Date**
4	Examine patient	Paul	07/08/2006 - 09:03:44
2	Examine patient	Steve	07/08/2006 - 09:05:12
3	Examine patient	Paul	07/08/2006 - 09:06:03
4	Examine blood	Nick	07/08/2006 - 09:06:45
1	Examine patient	Luigi	07/08/2006 - 9:07:33
5	Examine patient	Steve	07/08/2006 - 09:08:12
3	Make X-ray	Kevin	07/08/2006 - 09:09:59
2	Make X-ray	Kevin	07/08/2006 - 09:13:14
2	Examine blood	Nick	07/08/2006 - 09:15:11
1	Perform surgery	Luigi	07/08/2006 - 09:15:13
4	Make X-ray	Kevin	07/08/2006 - 09:17:39
5	Examine blood	James	07/08/2006 - 09:17:52
3	Examine blood	James	07/08/2006 - 09:21:24
4	Nurse patient	Kate	07/08/2006 - 09:25:07
2	Nurse patient	Kate	07/08/2006 - 09:27:21
5	Make X-ray	Kevin	07/08/2006 - 09:28:42
1	Nurse patient	Kate	07/08/2006 - 09:28:49
5	Nurse patient	Kate	07/08/2006 - 09:30:01
3	Nurse patient	Kate	07/08/2006 - 09:32:44

Table 2.1: Example of an event log

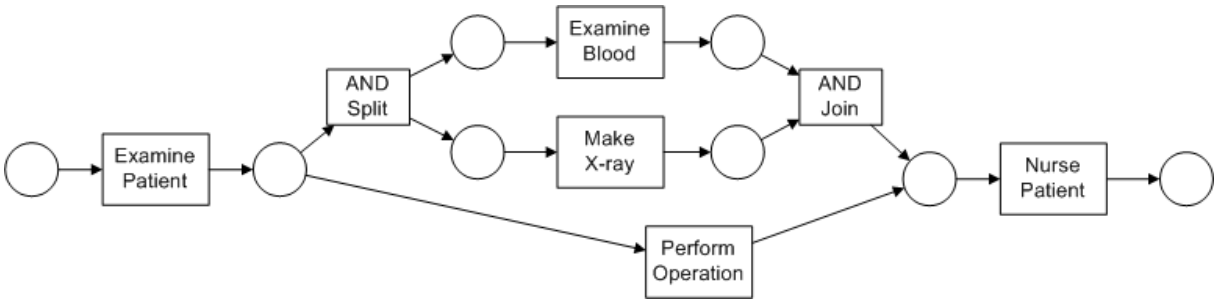

Figure 2.1: The derived process model in Petri net notation

Deriving process models from event logs is not the only possible form of process mining. There is much more information related to processes that can be obtained from event logs. Other possible forms of process mining include conformance testing (described in [17]), social network analysis (described in [22]), and many more. Conformance testing is a method which checks whether, and to what extent, activities actually execute as defined in a process model. Social network analysis focuses on relationships between performers of activities in processes. At [13], more information about the various forms of process mining can be found.

2.2 Requirements

To be able to perform process mining on event logs, some requirements need to be put on the event logs that serve as input. The following requirements are mandatory and are assumed to be met by all event logs that serve as input to process mining [24]:

(i) Each event refers to an activity or task, i.e. a well-defined step in the process (for instance 'contact customer').

(ii) Each event refers to a case, i.e. a process instance (e.g. an insurance claim).

(iii) Events are totally ordered.

Any information system that uses a transactional system will offer this information in some form. However, recorded events often also contain additional data. This can for instance be the resource that handled the activity or the timestamp at which the event occurred. When these data are used, more information can be distilled from event logs than merely a process model. For instance, in order to determine the organizational model or to analyze the social network of a process, the following requirement has to be met as well:

(iv) Each event refers to an originator (a person which executes the task).

It will be clear that to be able to deduce basic performance indicators (e.g. arrival rate of cases, the mean waiting time of an activity, the mean throughput time of a case) from

event logs, a time dimension must be present. Therefore, to be able to derive performance information from event logs, the following requirement will have to be met as well:

(v) Each event refers to a timestamp (i.e. the time at which it was recorded).

The example log in table 2.1 meets all the requirements as described above.

2.3 The ProM framework

The ProM framework is a framework that was developed to support the various forms of process mining. It has been designed to facilitate the implementation of new algorithms and techniques in the field of process mining [26]. New functionality can be added to the framework by means of plug-ins. A plug-in is a piece of software or functionality that can be added or removed without affecting the total framework functionality [14]. Plug-ins can be seen as black boxes with well-defined interfaces.

In figure 2.2 (as presented in [17]), a graphical interpretation of the ProM framework is shown. At start-up, all plug-ins are loaded into the framework after which they are available to the

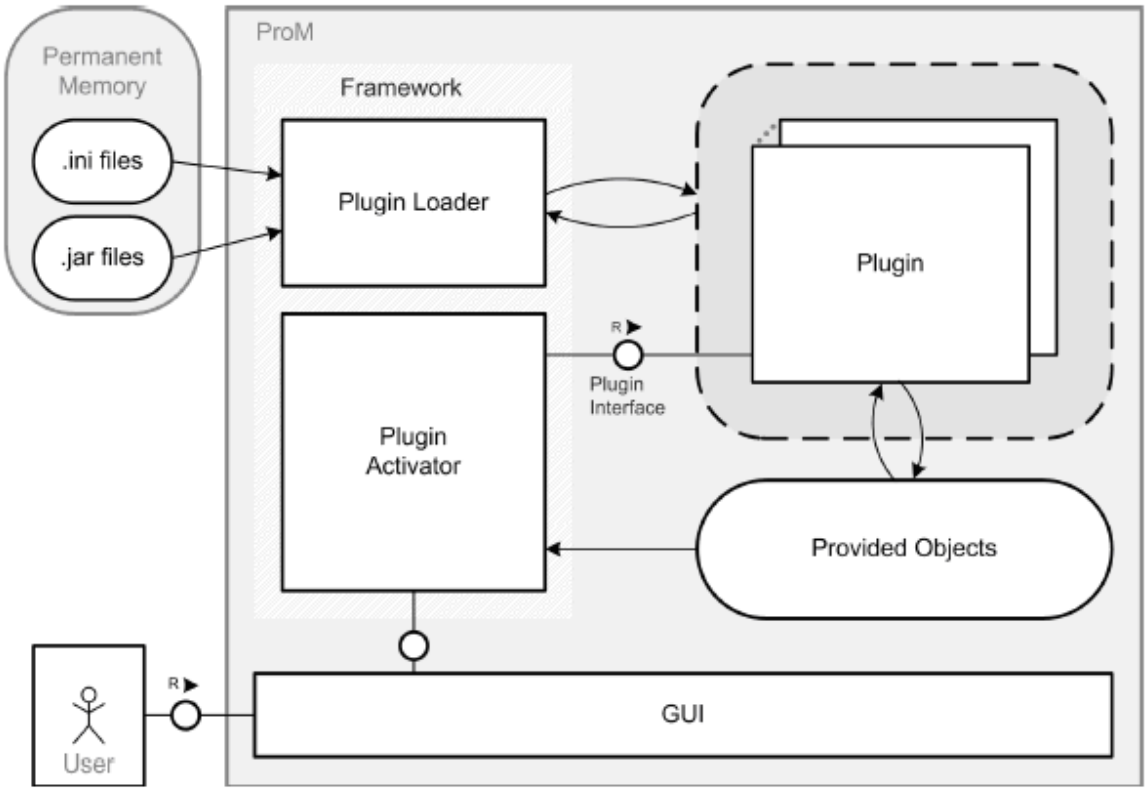

Figure 2.2: The derived process model in Petri net notation

user through a graphical user interface. The ProM framework provides high-level access to so called *Provided Objects*, e.g. log files or graph structures like Petri nets. These provided objects can serve as input and/or output to the plug-ins.

The ProM framework distinguishes five different groups of process mining plug-ins [26]:

Mining plug-ins which implement algorithms that mine models from event logs, e.g. the α-algorithm that distils a process model in Petri net notation from event logs.

Analysis plug-ins which typically implement some property analysis on some mining result. For example, for Petri nets there is a plug-in which constructs place invariants, transition invariants, and a coverability graph. However, there are also analysis plug-ins that can compare a log and a model (i.e. conformance testing) or a log and an LTL formula.

Import plug-ins which implement functionality that makes it possible to import ("open") objects of a certain data format, e.g. load instance-EPCs from ARIS PPM.

Export plug-ins which implement functionality that makes it possible to export ("save") objects to a certain data format, e.g. save EPCs, Petri nets, spreadsheets, etc.

Conversion plug-ins which implement conversions between different data formats, e.g. from EPCs to Petri nets.

The log files that can be accessed through the ProM framework need to be stored in the standard Mining XML (*MXML*) log format. This format is a generic XML format that has been developed for the ProM framework. It was based on a thorough comparison of the required input of various existing (ad-hoc) process mining tools (e.g. *EMiT*, *MinSoN*, *Little Thumb*) and the information typically contained in the event logs of complex information systems (e.g. ERP or WFM systems) [26]. In appendix A this MXML log format is described in more detail. At this moment, what is important to know is that a log in MXML format is a list of process instances (i.e. cases). Each process instance in turn is a list that contains the *audit trail entries* that belong to the process instance. An audit trail entry is a data structure in which information about one event is stored. This data structure encompasses:

- *WorkflowModelElement*: the name of the task that was involved in the event (TaskID).
- *Event type*: for each task, different types of events can occur:
  - *schedule*: occurs when the task is scheduled, i.e. is ready to be executed.
  - *(re)assign*: occurs when the task is (re)assigned to a resource.
  - *start*: occurs when the task actually starts execution.
  - *complete*: occurs when the task is completed, i.e. the execution of the task ends.
  - *suspend*: occurs when the task is suspended, i.e. execution is halted.

  - *resume*: occurs when a suspended task starts executing again, i.e. is resumed.
  - *skip*: occurs when a task is skipped (manually or automatically).
  - *withdraw*: occurs when a task is withdrawn, i.e. decision was made that the task will not be executed.
  - *abort*: occurs when a task is aborted, i.e. execution of the task is stopped and will not be continued.
  - *normal*: Some information systems do not keep track of the type of event that occurs, so when the event type is unknown, it is said to be *normal*.

- *Originator* (optional): the person or resource that handled the event.
- *Timestamp* (optional): the moment at which the event occurred.
- *Other data* (optional): here any other data that is known about the event can be stored, e.g. the department at which it was handled.

To bridge the gap between the log formats of the various information systems and the standard MXML log format, another tool: the *ProM-import framework* [5], was developed. This tool makes it possible to convert the logs of the various types of information systems to the standard MXML log format.

Chapter 3

Process Performance Analysis

3.1 Introduction

Process Performance Analysis (PPA) is closely related to disciplines as *Performance Measurement*, *Business Performance Management*, *Process Monitoring* and *Business Activity Monitoring (BAM)*. PPA supports organizations in measuring and analyzing the performance of their business processes. Although the term *performance* is commonly used in literature, few people agree on what it actually means. Definitions of the term "performance" have appeared e.g. in [3, 4, 11]. We adhere to the definition from [3], namely "the degree in which an organization carries its objectives into effect".

The performance of an organization can be measured through quantitative, measurable indicators. These are commonly known as *Performance Indicators* (*PIs*). There exist many different performance indicators, which can be time-related (e.g. throughput time of a process, service time of an activity), cost-related (e.g. process costs, material costs) or quality-related (e.g. visiting frequencies, error rates). For every different kind of organization, different performance indicators are of importance. A restaurant that has Michelin stars, for instance, will most likely find the quality of the food it serves of greater importance than the costs. A fast-food restaurant on the other hand will focus more on the costs and less on the quality of the food. The indicators that best represent the mission and strategy of an organization are called its *Key Performance Indicators* (*KPIs*). They reflect the critical success factors of the organization and help the organization to define and measure progress towards its organizational goals [15]. The well-known Balanced Scorecard (BSC), as described in [10], is often used to translate the mission and strategy of an organization into a comprehensive set of KPIs. The BSC results in a set of planned values for the KPIs of the organization. The actual values of these KPIs can then be monitored and analyzed to evaluate the performance status of the organization. The BSC divides KPIs over four perspectives: *Financial, Learning and Growth, Customer* and *(Internal) Business Processes*. PPA focuses on the *Business Process*

perspective of the organization. KPIs based on this perspective help to determine how well business processes are contributing to achieving the goals of the organization.

Process monitoring tools are tools that support PPA by monitoring the actual values of the KPIs of business processes. In case large deviations occur between the planned and actual values of these KPIs, process monitoring tools help in finding the causes of these deviations. Reviewing the actual values of KPIs against planned values of the same indicators may then lead to corrective actions in order to increase the likelihood of achieving the goals. However, it may also lead to challenging and adjusting these goals. PPA enhances processes by creating better feedback loops. This continuous assessment helps to identify and eliminate problems before they grow. Therefore, PPA can be useful in risk analysis, prediction of outcomes of various scenarios and making plans to overcome potential problems [29].

In the remainder of this chapter, first the PPA functionality present in commercial process monitoring tools will be discussed in section 3.2. After that, in section 3.3, the academic process monitoring tool *EMiT* will be discussed and finally, in section 3.4, functionality derived from other (not process monitoring) tools and completely new functionality that supports PPA will be discussed.

3.2 Commercial Process Monitoring Tools

During our research, the following commercial process monitoring tools and/or their documentation have been examined:

- *Aris Process Performance Manager (Aris PPM)*
- *Business Objects*
- *HP Business Process Cockpit (HP BPC)*
- *Hyperion System 9*
- *ILOG JViews*

These tools are among the market leaders in the area of process monitoring. In the remainder of this section, the functionality that was found during research of these tools will be addressed.

3.2.1 Graphs and tables

All investigated commercial monitoring tools can monitor the values of process-related KPIs (often in real time), and these tools support many different KPIs (over one hundred in some). Some process monitoring tools even allow the user to create his own KPIs.

In most commercial process monitoring tools, the values of the monitored KPIs can be viewed in tables and graphs. In its simplest form, a graph contains only the current value of a KPI compared against planned values. In figure 3.1 for instance, the *average process cycle time* of the business process under investigation is displayed in a 'speedometer'. In the example, the *average process cycle time* is in the critical area, i.e. it is above a certain bound value. More detailed analysis is needed to derive the causes of such an extreme value.

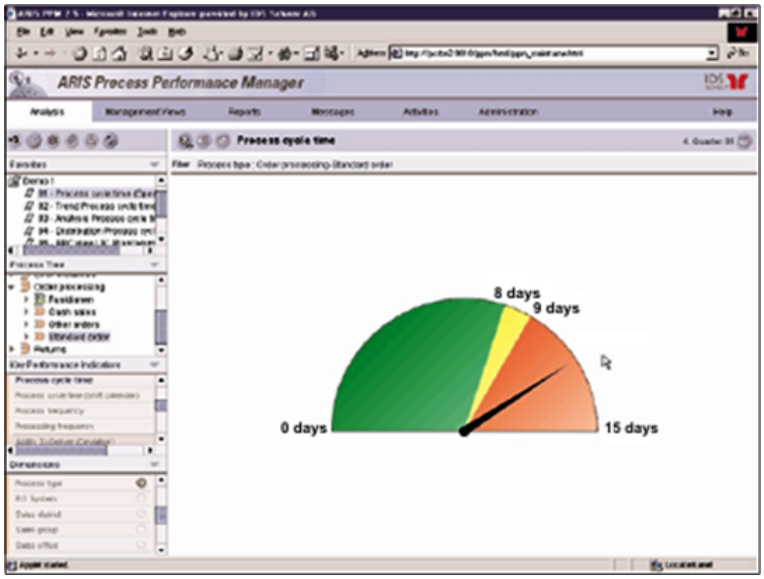

Figure 3.1: Speedometer containing the *average process cycle time* [19]

One way to perform a more detailed analysis is by using dimensions or perspectives against which KPIs can be measured. Possible dimensions include *time*, *task*, *originator*, *department*, *country* and *sold-to party*, but there are many more. In this manner it is possible to view a KPI, such as the *total number of cases in the process*, in a graph (or table) measured against a dimension. If for example the *time* dimension is used, it is possible to analyze the trend of the KPI (is it going up or down?) and to see if, and when extreme values have occurred. Such information can help with identifying the causes of deviations of certain KPIs. Suppose for instance that there is an upward trend for the *total number of cases in the process*. The increase in cases to handle may well be one of the causes of an increase in the *average process cycle time*.

Many monitoring tools also allow the user to measure a KPI against more than one dimension at the same time. In the example in figure 3.2 for instance, the KPI *average waiting time* is displayed in a 3D-graph, measured against the dimensions *task* and *originator*. From such a graph, various sorts of information can be derived, e.g. the user can quickly see when (i.e. under the influence of which values for the dimensions) extreme values have occurred, but also information about relations between dimensions can be derived. In the example in

figure 3.2, an extreme value for the *average waiting time* occurs in those process instances were originator *John* is involved in task *Sell*. At the same time one can see that there are no process instances in which the task *Reject order* is performed by originator *Clara*, since there is no relation between these elements visible in the graph.

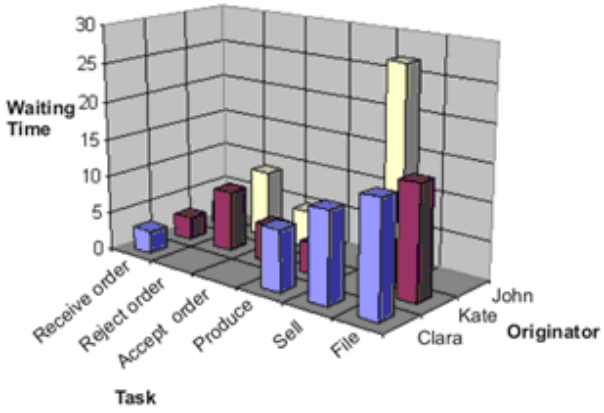

Figure 3.2: 3D-graph of average waiting time measured against dimensions *originator* and *task*

Process monitoring tools often start by displaying average values of KPIs (e.g. average process cycle time, average number of cases in the process), but they also allow the user to view the statistical distribution of these KPIs. An example of a statistical distribution is shown in figure 3.3. Using such statistical distributions, the user can easily see how many (and in some tools also which) cases result in an extreme value for the KPI.

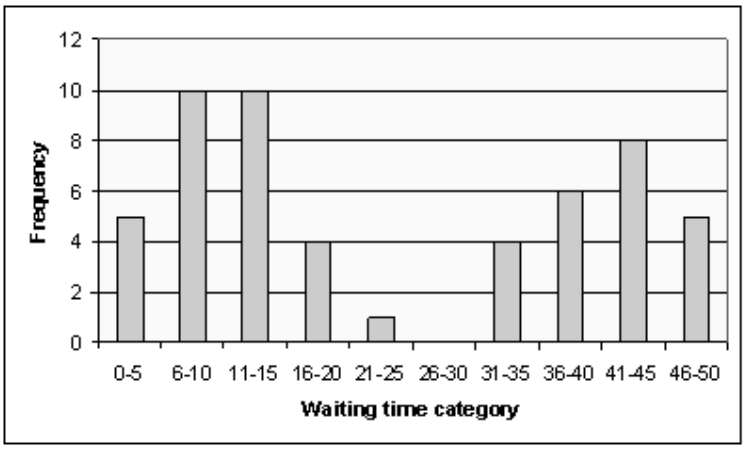

Figure 3.3: Statistical distribution of the waiting time

In Aris PPM the user has the possibility to select a group from such a statistical distribution. After selection of a group, the routings that the selected cases (i.e. cases that belong to the selected group) follow within the process definition of the business process can be displayed.

Furthermore, the user can compare these routings with the routings that other cases follow. This can be useful for detecting whether odd behavior has occurred during processing of the selected cases, e.g. a task is performed a number of times during processing of the selected cases.

Let us consider an example that illustrates how the described functionality can assist the user in finding the causes of extreme values for a KPI. Suppose the user wants more details about the high value for the *average waiting time* of task *Sell* when it is performed by originator *John*. The user can select the corresponding bar (the highest one) in the diagram of figure 3.2, and choose to view the statistical distribution instead of the average value for this KPI. After this, the user can select the cases for which the waiting time (when task *Sell* is performed by *John*) is the highest, and view the routings of these cases. If the user compares these routings, with the routings that cases with normal values for this KPI follow, then he can easily see if odd behavior has occurred (e.g. strange routings followed), and perhaps prevent such behavior from occurring in the future.

3.2.2 Process model

In some tools, such as ILOG JViews [9], performance information concerning a business process can be accessed through a (user-defined) process model that represents the business process. In such a process model, the user can select objects, i.e. a specific task in the model or the whole model. After selection of an object, performance information that is related to that particular object is displayed. In the example in figure 3.4, the user has selected the task *1st Level Support*. The shown information is the change in the *number of pending work items* over time for this task, displayed in a graph.

Using the process model as a user-interface makes it easy to spot bottlenecks within a business process and to detect the causes thereof. A bottlenecks is a point of congestion in a system that occurs when workloads arrive at that point more quickly than the point can handle them. The inefficiencies brought about by the bottleneck often create a queue and a longer overall cycle time [7]. In JViews a task is colored red if it is considered to be a bottleneck, else it is colored yellow (perhaps a bottleneck), blue (probably not a bottleneck) or white (certainly not a bottleneck). In case the user selects a task that is in fact a bottleneck, the shown information that belongs to the task may well assist in detecting what causes the task to be a bottleneck.

3.2.3 Views

Many people are involved in monitoring the performance of business processes, this includes:

- Senior management

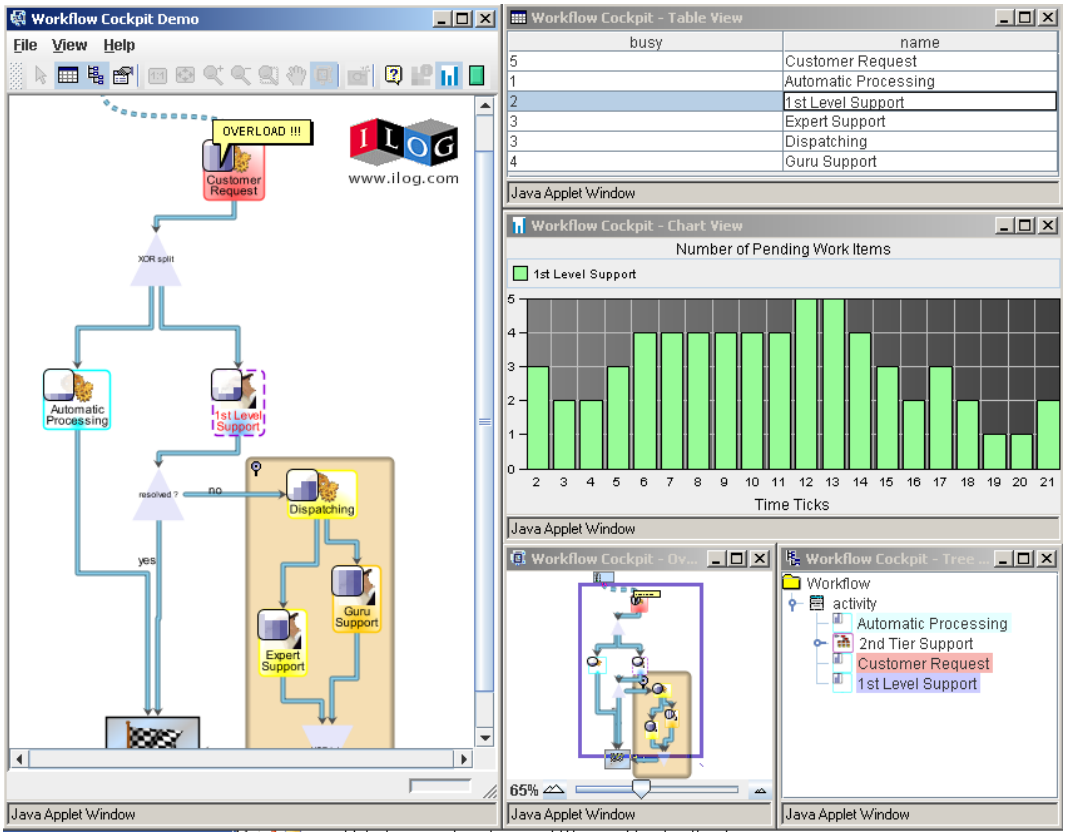

Figure 3.4: Example of ILOG JViews in which a process model is used as UI

- Middle management
- Participants (in the business process)

Each of these function roles has its own requirements for the provision and preparation of information. Senior Management for instance, is more interested in global information such as the evolution of the performance of one of the business processes. Participants on the other hand need information to be highly detailed, i.e. participants need information concerning the specific task(s) in which they are involved. Some monitoring tools, such as ILOG JViews, support this by providing different views for different users.

3.2.4 Reports

Commercial monitoring tools often offer extensive reporting functionality. With this functionality, the user can easily create reports that contain relevant performance information and distribute these reports to others. Some tools can even generate reports automatically, by creating them at predefined moments (e.g. on a daily or weekly basis). Possible output formats for process monitoring reports include:

- Internet-compatible pages in HTML.

- Platform independent PDF-documents.
- XML files, which allow for further processing.
- Comma-delimited text files, which can be used by other programs, e.g. MS Excel, StatGraphics.

3.3 Academic Process Monitoring Tool

There also exist academic tools that support PPA. *EMiT*, developed at Eindhoven University of Technology is such a tool. EMiT is a tool which can derive a process model from an event log by using a mining technique: the α-algorithm (as presented in [25]). The process model that results from this mining effort is a workflow net. A workflow net is a Petri net which has exactly one start place and one end place, where every other place or transition in the net is on a directed path from the start place to the end place [20]. As noted in section 2.3, in event logs different types of events can occur. In workflow nets resulting from mining such event logs, tasks are often not represented by one transition, but by multiple transitions with places in between. One transition then represents one event of the activity, e.g. a start, or a complete event. In the example in figure 3.5 for instance, task D has been split up in transitions *D-Schedule* and *D-Complete* with a place in between these transitions.

EMiT uses the mined workflow net as an intuitive interface through which the values of KPIs can be accessed, but also as a means to derive the values of these KPIs. EMiT derives these values by simulating the cases that exist in the event log within the workflow net. During simulation of a case, the case is represented by tokens in the Petri net. Each event that refers to the case is then considered in chronological order. When an event that refers to the case is considered, the transition to which the event corresponds is fired. As argued in [23], firing of transitions in mined workflow nets is atomic, because each event has a timestamp instead of a duration attached to it. Tokens thus spend time at places in between transitions, instead of at the transitions themselves. In EMiT, time-related KPIs are therefore associated with places instead of with transitions. Of every place in the Petri net, EMiT calculates the **Sojourn time**. The sojourn time of place p is the (average, minimum, maximum, and variance in) time tokens spend in p, between the moment they are created and the moment they are consumed. EMiT further divides the sojourn time in:

Synchronization time : Time that passes from the partial enabling of a transition (i.e. at least one input place marked) until full enabling (i.e. all input places are marked). In other words: the time that a token spends in a place, between the moment of its creation and the moment at which the transition that is going to consume the token is fully enabled (i.e. has a token in each input place). Synchronization times are only greater than 0 when the transition to fire is an AND-join.

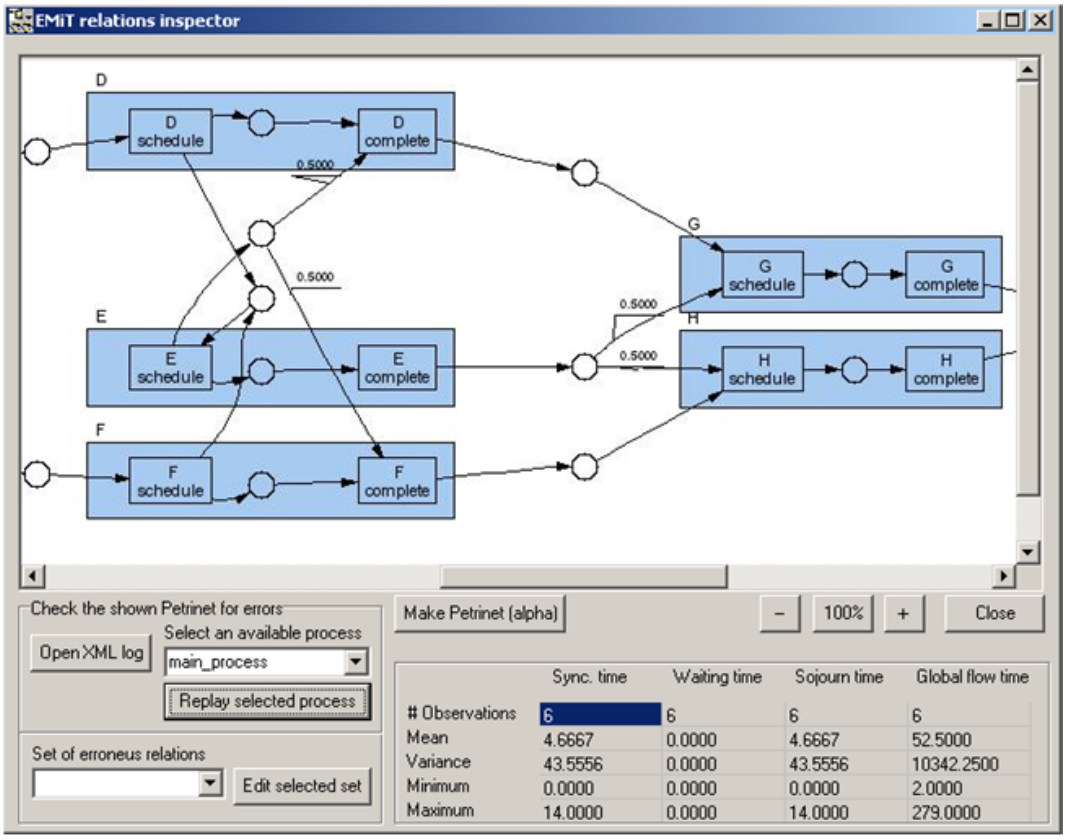

Figure 3.5: Screenshot of the EMiT tool

Waiting time : Time that passes from the enabling of a transition until its firing, i.e. the time between the moment at which the transition that is going to consume the token is enabled, and the moment at which this transition actually fires and consumes the token.

To clarify this, let us look at an example. Suppose we have the WF net of figure 3.5 and during replay of a case in this net, transition *D-Complete* first fires, then transition *E-Complete*, and after that transition *G-Schedule*. For p, the place that is located between *D-Complete* and *G-Schedule*, the sojourn time in this example is the time between the firing of *D-Complete* and the firing of *G-Schedule*. The synchronization time of place p is then the time between the firing of *D-Complete* and the firing of *E-Complete*. The waiting time of place p is the time between the firing of *E-Complete* and the firing of *G-Schedule*.

In EMiT, the user can summon the values of the KPIs that are related to a place, by selecting the place within the Petri net. In the example in figure 3.5, the user has selected the place between *D-Complete* and *G-Schedule*. The KPIs are displayed below the workflow net. Next to the already described KPIs, also the following KPIs are available in the EMiT tool:

- **Process flow time**: Throughput time of the process, i.e. (average, minimum, maxi-

mum, variance in) time that cases spend in the process.

- **Probabilities**: At each outgoing arc of XOR-splits (i.e. places that are input to more than one transition): the probability that a token in the XOR-split is consumed by the transition to which the arc leads. Suppose for instance that place p' is input to two transitions: A and B. The probability that a token in place p' is consumed by transition A is for instance 0.20 and the probability that the token is consumed by transition B is 0.80.
- **Number of observations**: The number of recorded values of each time-related KPI.

3.4 Other Process Performance Functionality

Next to the functionality available in process monitoring tools, also functionality that may assist in PPA and is contained in other tools was found. Furthermore, some completely new functionality that supports PPA was discovered. In this section we will describe this additional functionality.

3.4.1 Sequence diagrams

IBM Web Services Navigator(*WSN*), as described in [1], is a tool that was created to foster a better understanding of service-oriented architecture (SOA) applications. These days, systems often consist of components (e.g. servers), which communicate with each other. WSN is a tool that mines communication between servers from logs and displays this communication within sequence diagrams (see appendix B for more information about sequence diagrams).

In these sequence diagrams, the components that communicate (services) are put along the horizontal axis as classifier roles. Along the vertical axis time is measured, increasing from top to bottom. The time can be displayed as conventional time (seconds, minutes etc.) or as logic steps. In a sequence diagram, a set of *transactions* or *scenarios* – which are separate flows of connected arrows and vertical blocks – is displayed. An arrow in a transaction represents communication between services, i.e. a web service request or respond, and a block represents a period of activity of a certain service. Each transaction in a sequence diagram represents a communication sequence and is drawn in a unique color.

Next to the basic sequence diagrams, WSN also provides the user with a *Flow Patterns* view, which classifies similar transaction flows into flow patterns, i.e. transactions that represent similar behavior are classified in the same pattern. By doing such, the information that the user must digest is decreased dramatically. In figure 3.6 an example of a flow patterns view is shown. In this view the flow patterns are displayed sorted descendingly on the frequency of their appearance in the sequence diagram. WSN helps to solve problems, ranging from

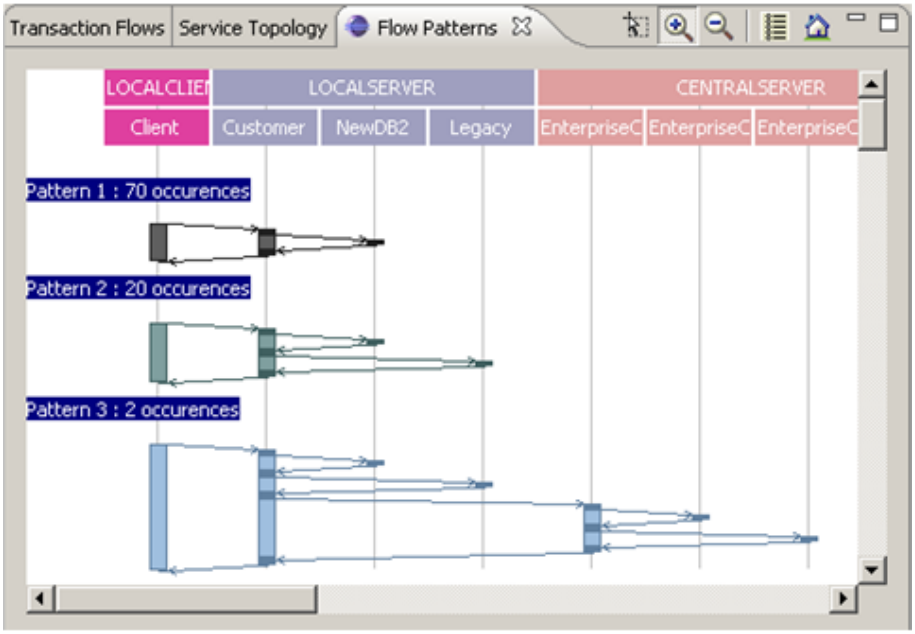

Figure 3.6: IBM Websight Flow Patterns view

business logic misunderstandings to performance bottlenecks, to semantic and syntactic errors, by visualizing how applications really execute. This enables businesses to understand how their applications actually behave.

WSN led to the idea to use sequence and pattern diagrams for process performance analysis in ProM. An event log in MXML format is taken as starting point of this type of performance analysis. Each event log consists of a number of process instances or cases. Let us suppose each such process instance describes one transaction flow. To be able to display process instances in sequence diagrams however, we also need classification roles or components. Good options for components can be tasks (how is work transferred between tasks), originators (how do originators communicate, i.e. how is work transferred between originators), or departments (how is work transferred between departments). The typical place where such information is stored in logs is within audit trail entries. In an audit trail entry, information about one event is stored, e.g. who performed it, which task is involved. The communication that should be mined is the transfer of work between instances of data-elements. For instance when the data-element (i.e. the component) is *Originator*, then instances thereof can be *Jack* or *Paul* or another person. These persons are then the 'components' that communicate.

Let us look at an example. Suppose we have a log as in table 3.1. When we examine the communication between departments for these traces, it may be possible to derive a sequence diagram as the one displayed in figure 3.7. The sequences that visualize *case 0* and *case 2* bare strong resemblance. The sequence that visualizes *case 1* is different from the beginning

Case 0:

TaskID	Originator	Department	Timestamp
Receive order	Nick	Sales	1/1/2006 - 0:15:00
Send production request	Nick	Sales	1/1/2006 - 1:00:00
Calculate number of needed goods	James	Production	1/1/2006 - 1:00:00
Request purchase of goods	James	Production	1/1/2006 - 1:25:00
Order goods	Fred	Purchase	1/1/2006 - 1:25:00
Deliver goods	Fred	Purchase	1/1/2006 - 2:20:00
Create product	James	Production	1/1/2006 - 2:20:00
Finish product	James	Production	1/1/2006 - 3:05:00
Sell product	Nick	Sales	1/1/2006 - 3:05:00
File order	Nick	Sales	1/1/2006 - 4:03:00

Case 1:

TaskID	Originator	Department	Timestamp
Order goods	David	Purchase	1/1/2006 - 3:00:00
Deliver goods	David	Purchase	1/1/2006 - 3:51:00
Create product	John	Production	1/1/2006 - 3:51:00
Finish product	John	Production	1/1/2006 - 4:57:00
Use product	David	Purchase	1/1/2006 - 4:57:00
Request product sell	David	Purchase	1/1/2006 - 5:30:00
Sell product	Nick	Sales	1/1/2006 - 5:30:00
File order	Nick	Sales	1/1/2006 - 6:22:00

Case 2:

TaskID	Originator	Department	Timestamp
Receive Order	Jim	Sales	1/1/2006 - 4:17:00
Send Production Request	Jim	Sales	1/1/2006 - 5:01:00
Calculate number of needed goods	James	Production	1/1/2006 - 5:01:00
Request purchase of goods	James	Production	1/1/2006 - 5:44:00
Order goods	Fred	Purchase	1/1/2006 - 5:44:00
Deliver goods	Fred	Purchase	1/1/2006 - 6:51:00
Create product	John	Production	1/1/2006 - 6:51:00
Finish product	John	Production	1/1/2006 - 7:42:00
Sell product	Jim	Sales	1/1/2006 - 7:42:00
File Order	Jim	Sales	1/1/2006 - 8:50:00

Table 3.1: Example event log

however, since it starts with the purchase department being active. It is thus clear that there are two different patterns in this diagram. These patterns are displayed separately in the diagram in figure 3.8 along with their corresponding frequency, i.e. the number of sequences that follow the pattern.

The communication patterns between data-elements can be useful for the analysis of processes. With these patterns, common behavior can easily be distinguished from rare behavior. If also additional performance information is included, such as the average throughput time, then wanted behavior (patterns with a low average throughput time) can be distinguished from unwanted behavior (patterns with a high average throughput time) as well. The discovered patterns can also assist in finding the causes of certain behavior (e.g. a large period in which a specific data-element instance is active). Measures may then be taken to prevent unwanted behavior from occurring in the future and to stimulate wanted behavior. For example, suppose there is a pattern with a high average throughput time in which work is transferred from originator *John* to *Ben*. If there is also a pattern where instead work is transferred from *John* to *Mike*, and which results in a low average throughput time, then it may be worthwhile to request *John* to transfer work to *Mike* rather than to *Ben*, if possible.

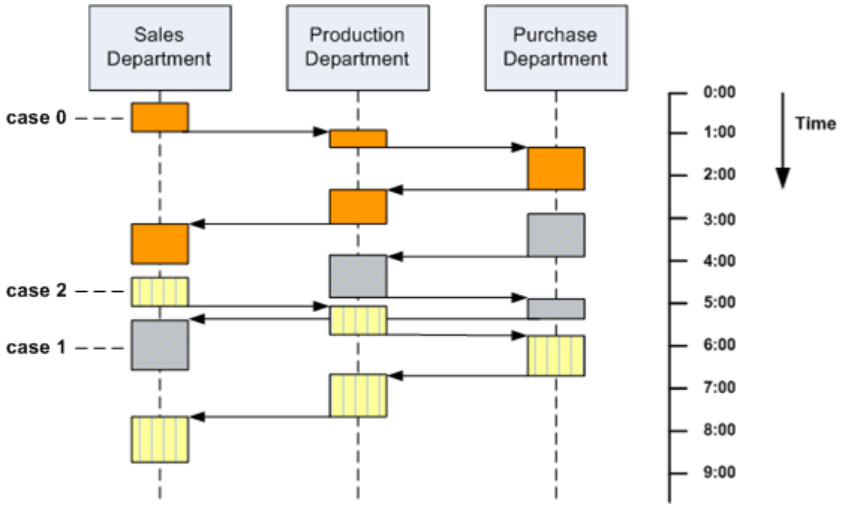

Figure 3.7: Sequence diagram of communication between departments

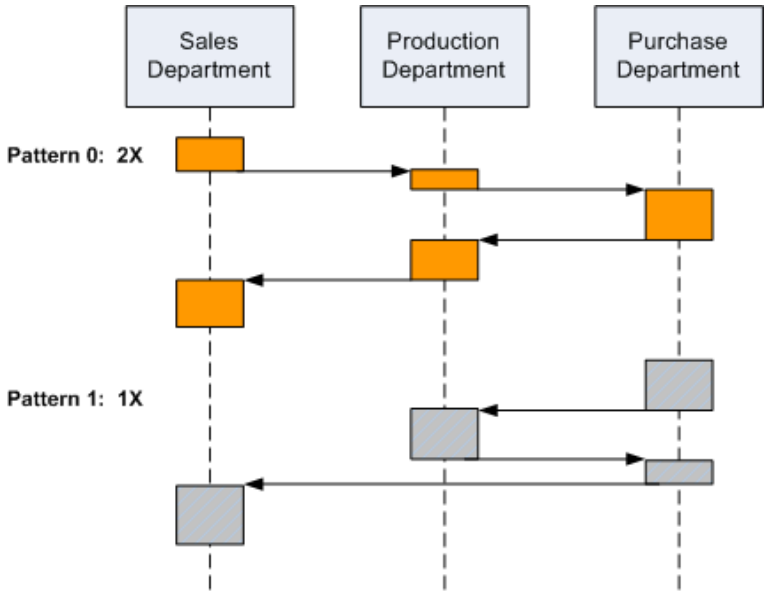

Figure 3.8: Sequence diagram of communication patterns between departments

3.4.2 Process model related functionality

When we have a process model at our disposal, there is some additional functionality that can be useful, next to the functionality already discussed.
One of these additions involves XOR-splits. A XOR-split in a process model is a decision point. At such a location it is decided if a certain task if going to be performed on a case or another task (but never both). Often, choice at these XOR-splits is not done randomly, but is dependent on certain case properties. Suppose for instance that the cases in the business process under investigation are in fact orders. A case property can then be the size of the order, the client who made the order, or the employee that approved the order. At a certain XOR-split, choice can be based on such case properties. It is for instance possible that orders bigger than a thousand pieces follow a different routing within a process than orders smaller than a thousand pieces. For those event logs in which such case properties are registered, it may be possible to determine the case properties that are responsible for choice at each XOR-split. When the case properties on which choice is dependent can be summoned for each XOR-split, this can be of great use when trying to detect the causes of extreme values of certain KPIs, e.g. all orders bigger than a thousand pieces follow a certain route in the process model, which always results in a high throughput time.

In case Petri nets are used as process models (as in EMiT), a feature may be added that allows users to select two transitions in the Petri net. The time (average, minimum, maximum, and variance) that elapses between the firing of the one transition and the firing of the other transition can then be displayed along with the number of observations, i.e. the number of times both transitions fire. The user can then derive time-related KPIs himself. For instance, in the situation where in the Petri net a task is represented by a *start* transition and a *complete* transition, the user can select these two transitions and the time between the firing of these two transitions will be given. This time in fact is the service time of the corresponding task, where the service time of a task is the time in which the task is actually being executed. Another option here is to allow the user to fill in bound values for the time that may elapse at most, and the time that should elapse at least between the firing of the two transitions. After filling in these bound values, the number of cases, the average (and the min, max, var) time between the transitions, and the distinguishing case properties (e.g. size of all cases in the range is bigger than 1000) of those cases that fall in the range, can be displayed.

3.4.3 Dotted charts

Another type of functionality that is not contained in any other monitoring tool, is a chart similar to a Gantt chart. This *dotted chart* has two dimensions: time and component. The time should be measured along the horizontal axis of the chart. Along the vertical axis, in boxes, components should be put. The components that we will focus on are again the

instances of data-elements that can be found at the audit trail entry level of logs.

The user should select a data-element (e.g. *Task*, *Originator*, *Department*) and an event-type (e.g. *complete*, *schedule*, *normal*). A marking (i.e. a dot) should then be drawn in the chart for each event in the log that refers to an instance of the selected data-element and to the selected event-type. This marking should be positioned at the data-element instance to which the event refers, at the timestamp (relative to the first event of the process instance) to which the event refers. Suppose for instance that the user has selected event-type *complete* and data-element *task*. For every event that refers to the completion of a task, a dot should then be drawn in the chart, positioned at the task in question and at the (relative) timestamp to which the event refers. For a certain log, this may result in the dotted chart as depicted at the top of figure 3.9. Dotted charts like this one can be used for purpose of analysis. From the top chart of figure 3.9 one can for instance derive that task *Buy* always is completed before task *Sell*. It becomes more interesting when you compare a dotted chart with other dotted charts that are derived from the same event log and with the same event-type, but with another data-element selected. See for instance the other dotted charts in figure 3.9. From these charts one can derive that *Mike* always performs task *Sell*, and that *Jack* and *Mike* both can handle task *Buy*, but that *Jack* seems to be faster in completing it. Furthermore, one can derive that *Jack* and *Mike* both work at the *Production* department. Dotted charts can help in finding the causes of certain performance problems (e.g. all cases where *Mike* performs task *Buy* results in a high average completion time of that task).

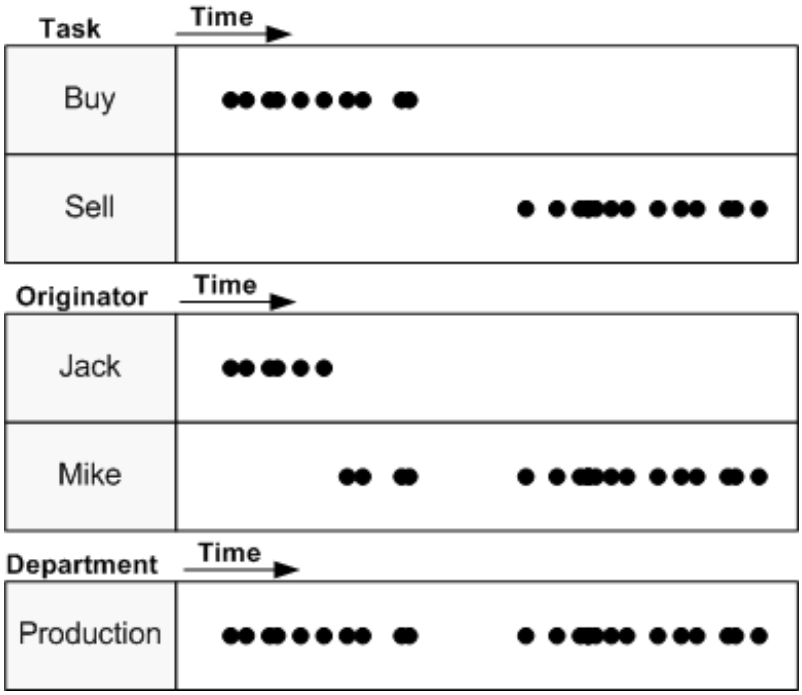

Figure 3.9: examples of dotted charts

Chapter 4

Requirements

4.1 Introduction

Although the second research question (as defined in section 1.3) leaves room for functionality that supports other forms of process mining, the main focus of this graduation project has been on functionality that supports Process Performance Analysis (PPA). As seen in the previous chapter, there is plenty of PPA functionality available that could be great additions to the ProM framework. If all this functionality would be implemented, the user could do a more than excellent analysis. Unfortunately however, the ProM-group does not have the resources available to complete such a large task. Therefore, decisions had to be made about which functionality should be added to the ProM framework.

The chosen approach was to start by implementing the functionality that is the most essential for analyzing the performance of the business processes of an organization. Such functionality should help to detect possible problems within the (performance of) business processes, and should assist in finding the causes of these problems.

In this chapter, the selected functionality and requirements placed thereon will be described. The functionality has been divided over two plug-ins. In section 4.2 the functionality that makes use of a process model, next to an event log in the standard MXML-format, is described. In section 4.3 the other part, that consists of the functionality that only makes use of an event log, is described. Both types of functionality, require the input log to meet (at least) requirements (i), (ii), (iii) and (v) as described in section 2.2.

Motivations for choosing to implement, or dropping certain functionality are given in appendix C. Explicit requirements are listed in appendix D.

4.2 Performance analysis with process model

The main purpose of the 'performance analysis with process model' functionality is to provide the user with a means to obtain the values of a selected group of key performance indicators (KPIs) out of an event log. The values of the selected KPIs are obtained by means of the log replay method. When defining the selected KPIs, we first need to consider this method.

4.2.1 Log replay

The input of the log replay method is a Petri net model and an event log in MXML format. In addition, a relation R must be given between the event classes that are registered in the log and the transitions in the Petri net. An event class encloses all events that refer to the same *WorkflowModelElement* (i.e. task, activity) and the same *Event type* (e.g. *start*, *complete*). The relation R is needed to be able to replay the cases that are registered in the log, in the Petri net. The ProM framework provides two ways to establish such a relation:

- The Petri net is mined out of the event log by a mining plug-in of the framework. When this is the case, a relation between the mined transitions and the event classes in the log is automatically established by the mining algorithm.
- The Petri net is loaded into the framework in combination with the event log. When this is the case, the user has to establish which transitions in the Petri net should be related to which event classes in the log.

An event class can be related to zero or more transitions. Likewise, a transition can be related to zero or more event classes. Transitions that are related to no event classes are said to be *invisible*. Such transitions can for instance be used for routing purposes. Events in class e in the log may correspond to firing any transition t in the net satisfying $(e, t) \in R$.

When the required input is present, log replay can start. The log replay method simulates cases within the Petri net. We refer to the set of all cases in the log as C. Replaying a case $c \in C$ starts with setting the Petri net to its initial marking M_0. After this, the events that refer to case c are replayed in the Petri net in chronological order. When an event of class e is replayed and e is related to at least one transition in the net, then one of the related transitions in $\{t \in T \mid (e, t) \in R\}$ should fire. When a transition fires, a token is removed from each of its input places and a token is added to each of its output places. To determine which transition in $\{t \in T \mid (e, t) \in R\}$ should fire, we need to establish which transitions in this set can fire. A transition t can only fire when it is enabled, i.e. when there is a token in each of its input places. However, the Petri net may contain (enabled) invisible transitions which can fire in such a sequence that t becomes enabled, when it was not before. See for instance the Petri net in figure 4.1, where the current marking is $M = [p3, p4]$. Suppose that

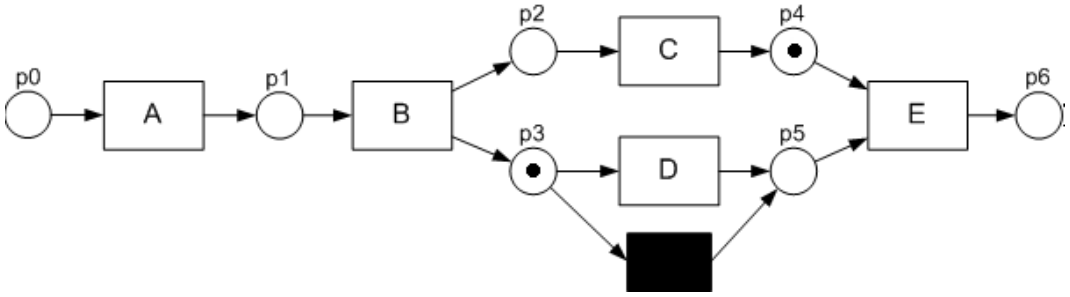

Figure 4.1: log replay example

the class of the event that is currently replayed is only related to transition E in the Petri net. This transition is not fully enabled in this example, because place $p5$ does not contain a token. E can be enabled however by firing the invisible, black transition.

We refer to the set of transitions in $\{t \in T \mid (e,t) \in R\}$ that are enabled or can be enabled by firing a sequence of invisible transitions as T'. One of the following three propositions holds for T':

- $\#_{T'} = 0$: none of the related transitions is or can be enabled.
  When this occurs, a random transition t' is chosen out of the set $\{t \in T \mid (e,t) \in R\}$. The chosen transition t' is said to fail execution and is artificially enabled, i.e. a token is added to each of its input places that did not contain a token yet. After this, t' is fired.

- $\#_{T'} = 1$, one of the related transitions is or can be enabled. If this transition t' is enabled, it is fired. Else the transitions, that are in the shortest sequence of invisible transitions that can enable t', are fired first. In [17], the author describes how to find this shortest sequence of invisible transitions. After the transitions in the sequence have fired, t' is enabled and is fired.

- $\#_{T'} > 1$, more than one of the related transitions is or can be enabled. The most suitable candidate $t' \in T'$ is selected. In [17], the author describes how the most suitable candidate is selected. If t' is enabled, it is fired. Else the transitions, that are in the shortest sequence of invisible transitions that enables t', are fired first. After the transitions in the sequence have fired, t' is enabled and is fired.

At each transition firing that takes place during log replay, the following information about the firing is stored:

trans	:	Identifier of the transition that fired.
case	:	Identifier of the case that was being replayed when the transition fired.
time	:	The timestamp of the event that was being replayed when the transition fired.

act : Identifier of the activity to which the event, that is related to and replayed by the firing transition, refers.
type : The event-type of the event that is related to, and replayed by the firing transition.
failed : Boolean, set to true if *trans* failed execution for this firing.

Note that when an invisible transition fires, no value is stored for *act* or *type*. The *time* that is stored when an invisible transition fires is the same as the *time* stored for the transition that is enabled by the (sequence of) invisible transition(s). We thus assume that invisible transitions are not eager to fire, i.e. they fire at the latest possible moment.

The log replay method has as output a list L, that contains the information of each transition firing that took place during log replay. The list L can be used for the calculation of the values of the selected KPIs.

4.2.2 Process-related KPIs

In this section the KPIs that are related to the performance of the entire process will be described.

Throughput time of the process

The throughput time of a case is the total time that the case spends in the process. Given replay output L and case c, the definition of the throughput time $T(c)$ of case c is equal to:

$MAX_{(x \in L \ \wedge \ case(x)=c)} \ time(x) \ - \ MIN_{(x \in L \ \wedge \ case(x)=c)} \ time(x)$

For the throughput time of the process the following statistical values should be calculated:

- $AVG_{c \in C} \ T(c)$ the average throughput time.
- $MIN_{c \in C} \ T(c)$ the minimum throughput time.
- $MAX_{c \in C} \ T(c)$ the maximum throughput time.
- $STDEV_{c \in C} \ T(c)$ the standard deviation in throughput time.
- $AVG_{low}(x, T(c)) = AVG_{c \in C_{low}(x)} \ T(c)$, where $C_{low}(x)$ is defined by:
  $c \in C_{low}(x) \iff \frac{\#_{\{d \in C| \ T(d) \leq T(c)\}}}{\#_C} \leq \frac{x}{100}$
  $C_{low}(x)$ contains the $x\%$ cases in C that have the lowest throughput times.
  The user should have the possibility to set the value of x, where $x \leq 100$.
- $AVG_{high}(y, T(c)) = AVG_{c \in C_{high}(y)} \ T(c)$, where $C_{high}(y)$ is defined by:
  $c \in C_{high}(y) \iff \frac{\#_{\{d \in C| \ T(d) \geq T(c)\}}}{\#_C} \leq \frac{y}{100}$
  $C_{high}(y)$ contains the $y\%$ cases in C that have the highest throughput times.
  The user should have the possibility to set the value of y, where $y \leq 100 - x$.
- $AVG_{normal}(x, y, T(c)) = AVG_{c \in C_{normal}(x,y)} \ T(c)$, where $C_{normal}(x, y)$ is defined by:
  $C_{normal}(x, y) = C \backslash (C_{low}(x) \bigcup C_{high}(y))$
  $C_{normal}(x, y)$ contains the cases in C that are fast nor slow.

Furthermore, the user should have the possibility to export the values in $\{T(c) \mid c \in C\}$ to a comma-separated text-file.

Arrival rate of cases
The arrival rate of cases is equal to the number of cases that arrive to the process per time unit. Given replay output L and case c, the arrival time $T'(c)$ of case c is:
$MIN_{(x \in L \ \wedge \ case(x)=c)} \ time(x)$

The (average) arrival rate of cases to the process then is:
$$\frac{\#_C}{MAX_{c \in C} \ T'(c) \ - \ MIN_{c \in C} \ T'(c)}$$

Number of cases
The number of cases in the log should also be available to the user. This number is equal to $\#_C$.

Number of fitting cases
To give the user an idea about the conformance between the Petri net and the event log, the number of cases that 'fit' in the Petri net should be given as well. The fitting cases are those cases in which no transition fails execution during log replay. The number of fitting cases can be defined as: $\#_{\{c \in C \mid \ \nexists_{x \in L}(case(x)=c \wedge failed(x)=true)\}}$
For more detailed information about the conformance between the used Petri net and event log, the *Conformance Checker* plug-in of the framework should be used.

4.2.3 Place-related KPIs

Firing of transitions during log replay is atomic, since events have a timestamp and not a duration attached to them. Therefore we can say that the tokens, that represent a case during log replay, spend time in places. In this section, the KPIs that are related to places will be discussed.

Since the timestamp at which transitions fire is known, also the creation timestamp of each token that is added to an output place of the transition is known. For each token that is consumed by the firing transition, the consumption timestamp is also known. However, when an input place of a firing transition contains multiple tokens, then we cannot tell which token is removed exactly. A pseudo-random function with equal weight is used to select a token from each input place. Note that this may affect the KPIs defined below.

Sojourn time of place p
The sojourn time of a token in place p is equal to the total time a token spends in p, i.e. the time between creation and consumption of the token. Statistical values should be calculated for the sojourn time of place p, based on the sojourn times of all tokens that are added to place p during log replay. This calculation can be done in a similar manner as for the throughput

time of the process. Note that all statistical values besides the average sojourn time of p may be influenced by the random choice when p contains multiple tokens at firing of a transition. This also holds for the synchronization and waiting time of p in which the sojourn time can be divided.

Synchronization time of place p
The synchronization time of a token is the time between the moment at which the token is created and the moment at which the transition that consumes the token is fully enabled (i.e. there is a token in each input place). Statistical values should be calculated for the synchronization time of place p, based on the synchronization times of all tokens that are added to place p during log replay.

Waiting time of place p
The waiting time of a token is the time between the moment at which the transition that consumes the token is fully enabled and the moment at which the token is actually consumed. Statistical values should be calculated for the waiting time of place p, based on the waiting times of all tokens that are added to place p during log replay.

Arrival rate of tokens to place p
The average arrival rate of tokens to place p should be calculated, based on the creation times of the tokens that are added to p. The arrival rate can then be calculated in a similar manner as the arrival rate of cases.

Frequency of tokens in place p
The number of tokens that are added to place p during log replay should be available to the user.

Probabilities at XOR-splits
A special KPI should be depicted at places with multiple outgoing arcs, more commonly known as XOR-splits. Each outgoing arc of a place p ends in a transition t. We define the probability for each outgoing arc (p, t) of XOR-split p as:
$\frac{\#_{\{x \in L |\ trans(x)=t\}}}{\#_{\{x \in L |\ trans(x) \in \{t' |\ (p,t') \in A)\}\}}}$, where A is the set of all arcs in the Petri net.

4.2.4 Time between two transitions

Another KPI that should be calculated is the time between two transitions, where the user should have the possibility to select the two transitions. Given replay output L and case c, the definition of the (absolute) time between transition u and t, $T(c, u, t)$, is equal to:
$|MIN_{(x \in L \ \wedge \ case(x)=c \ \wedge \ trans(x)=u)}\ time(x)\ -\ MIN_{(x \in L \ \wedge \ case(x)=c \ \wedge \ trans(x)=t)}\ time(x)|$
Based on the time between u and t for all cases in which both transitions fire at least once,

statistical values for the time between the transitions should be calculated. Furthermore, the number of cases in which both transitions u and t fire at least once during log replay should be available to the user. Note that for each case, at most one time between two transitions is calculated.

4.2.5 Activity-related KPIs

In event logs, an activity is often represented by multiple events with different event-types. Because of this, various activity-related KPIs can be discovered.

Waiting time of activity a
The time between *schedule* and *start* events of activity a. For each activity a, the waiting time of all registered executions of a should be calculated. This is done as follows:
Each element $x \in L$, where $act(x) = a$ and $type(x) = schedule$, is compared with the nearest following element $x' \in L$ for which $act(x') = a$ and $type(x') = start$. When in list L, in between x and x', no element $x'' \in L$ is registered for which $act(x'') = a$ and $type(x'') = schedule$ holds, then the waiting time of this 'execution' is defined to be: $time(x') - time(x)$. Note that we assume here that an activity cannot be executed in parallel with the same activity.

Statistical values should be calculated for the waiting time of a, based on the waiting times of all registered executions of a. Next to these values, the number of waiting time measurements for activity a should also be available to the user.

Execution time of activity a
The time between *start* and *complete* events of activity a, without the time that a is suspended in between (i.e. without the time between the *suspend* and *resume* events of a that occurred in between). For each activity a, the execution time of all registered executions of a should be calculated. This is done as follows:
Each element $x \in L$, where $act(x) = a$ and $type(x) = start$, is compared with the nearest following element $x' \in L$ for which $act(x') = a$ and $type(x') = complete$. When in list L, in between x and x', no element $x'' \in L$ is registered for which holds: $act(x'') = a$ and $type(x'') = start$, then the execution time with suspended time of the execution is defined to be: $time(x') - time(x)$. The total suspended time in between x and x', $T_{susp}(x, x')$ needs to be deducted from this. All elements $y \in L$ located in between x and x' in the list, for which $act(y) = a$ and $type(y) = suspend$ holds, should be compared against the nearest following $y' \in L$ (registered before x' in L) for which $act(y') = a$ and $type(y') = resume$ holds. When there is no $y'' \in L$ located between y and y' for which $act(y'') = a$ and $type(y'') = suspend$ holds, then $T_{susp}(x, x')$ should be raised with $time(y') - time(y)$. The execution time (without suspended time) of the execution is defined as: $time(x') - time(x) - T_{susp}(x, x')$

Statistical values should be calculated for the execution time of a, based on the execution times of all registered executions of a. Next to these values, the number of execution time measurements for activity a should also be available to the user.

Sojourn time of activity a
The time between *schedule* and *complete* events of activity a. For each activity a, the sojourn time of all registered executions of a should be calculated. This is done as follows:
Each element $x \in L$, where $act(x) = a$ and $type(x) = schedule$, is compared with the nearest following element $x' \in L$ for which $act(x') = a$ and $type(x') = complete$. When in list L, in between x and x', no element $x'' \in L$ is registered for which $act(x'') = a$ and $type(x'') = schedule$ holds, then the sojourn time of the execution is defined to be: $time(x') - time(x)$.

Statistical values should be calculated for the sojourn time of a, based on the sojourn times of all registered executions of a. Next to these vales, the number of sojourn time measurements for activity a should be also available to the user.

Arrival rate of a
The number of times activity a is scheduled (on average) per time unit should be calculated.

Upper bound values
Not in all event logs *schedule*, *start*, and *complete* events will be present for each execution of an activity. In some event logs for instance, only *complete* events are registered. In such case, the waiting time, execution time, sojourn time and arrival rate of the activities cannot be calculated. However, the Petri net defines how transitions, and thus also activities, are interrelated. That is why an upper bound for the value of the sojourn time can be calculated in this case. This bound is equal to the time between the enabling and the firing of the transition that is related to the *complete* event of the activity. In table 4.1 is depicted which values should be calculated for an activity, dependent on the event-types that are registered in the log for that activity. When a bound value has been calculated for a certain KPI, then it should be made clear to the user that this is a bound value and not an exact value.

4.2.6 Failing transitions

When during log replay a transition is not (and cannot be) enabled when it should fire, we say it fails execution. When this occurs, tokens are created artificially to enable the transition and log replay continues. The creation timestamp of the artificially created tokens is set to the same timestamp as the firing timestamp of the transition that consumes the tokens. This clearly has its effect on the measurements taken. To cope with this, the user should be presented with options with which (s)he can set and adjust the manner in which the plug-in deals with these failing transitions. At least the following options should be available:

Event-types in log:	Exact values for:	Upper bound values for:
schedule, start, complete	sojourn time waiting time execution time	-
schedule, start	waiting time	sojourn time execution time
schedule, complete	sojourn time	-
start, complete	execution time	sojourn time waiting time
schedule	-	sojourn time
start	-	sojourn time execution time waiting time
complete	-	sojourn time

Table 4.1: Values of activity-related KPIs

- All measurements are used for calculation of values of KPIs.
- Only measurements taken in 'fitting' cases are used for calculation of the values of KPIs. A case 'fits' if no transition fails execution during log replay of the case.
- All measurements taken in 'fitting' cases, as well as all measurements taken for 'non-fitting' cases, that are taken before a transition fails execution during replay of the case, are used for calculation of the values of KPIs.

4.2.7 User-interface

User-friendliness is of high importance for this plug-in. The plug-in should help the user to spot problems within processes in an intuitive manner. One way in which this is established is by using the input Petri net as an interface.

The plug-in should extend the input Petri net by giving the places in the net a color that corresponds to the average waiting time of the place. At least three waiting time levels should be distinguished: low, medium, and high. The user should have the possibility to change the bounds of these levels, as well as the color that corresponds to each of these levels. Coloring places can help the user to discover bottlenecks, because bottlenecks are typically locations in the process where a lot of time is spend.

Next to this, at each outgoing arc of places that are XOR-splits, the probability should be displayed that a token in the place is consumed by the transition to which the arc leads.

This can help to determine the causes of certain behavior, because the probabilities give an indication of the routings that cases follow within the process model.

The Petri net should also be used as a means to access the values of KPIs. It should be possible to access the values of the place-related KPIs of a certain place, by selecting the place within the Petri net. The time between two transitions should be accessible by selecting the transitions in the Petri net. When a transition is selected, the activity-related KPIs of the activities to which related event classes of the transition refer, should be displayed.

Furthermore, the user should also have the possibility to select and deselect process instances. All KPIs should be based only on the selected process instances.

4.3 Sequence Diagram functionality

The sequence and pattern diagrams, as described in section 3.4.1, should be implemented in a separate plug-in. This type of functionality appears to be very promising, since communication patterns can give the user a clear view of what is normal and what is exceptional behavior within a business process. It can also assist in detecting the causes of certain behavior. The plug-in requires an event log in standard MXML format as input. To be able to define requirements on this plug-in, we need to understand how sequence and pattern diagrams can be derived from an event log.

4.3.1 Deriving Sequence Diagrams

In a sequence diagram the process instances that occur in a log can be displayed as transactions (i.e. sequences), which represent the transfer of work between data-element instances over time. Such sequences can of course only be derived after a data-element has been selected. The user should have the possibility to select this data-element out of a list that contains all data-elements that appear at the audit trail entry level in the log. Examples of data-elements are *task*, *originator*, *department* etc.

To be able to determine when during an execution of a process (i.e. when in a process instance) there is a transfer of work from one instance of a data-element to another, the relations between the events in that process instance are needed. Such relations can be obtained in a similar manner as is done by another plug-in of the ProM framework: the multi-phase miner. This plug-in establishes an *instance ordering* for each process instance in the log. In [27], the authors describe how to obtain instance orderings. Therefore, we will not dig into this here. What is important for us to know is that for a process instance σ, where σ is a (chronologically ordered) trail of events $e_1...e_n$, an instance ordering $\succ_\sigma$ on the domain of σ, $D_\sigma = \{1...n\}$ can be obtained. This ordering is irreflexive ($\forall_{a \in D_\sigma}(a \not\succ_\sigma a)$), asymmetric ($\forall_{a_0,a_1 \in D_\sigma}(a_0 \succ_\sigma a_1 \Rightarrow a_1 \not\succ_\sigma a_0)$) and acyclic ($\forall_{a_0,a_1 \in D_\sigma}(a_0 \succ_\sigma a_1 \Rightarrow \nexists_{a_2,...,a_n \in D_\sigma}(a_1 \succ a_2 \succ$

$... \succ a_n \succ a_0)))$ and defines how events in σ are interrelated. This information can be used to determine how work is transferred between data-element instances in σ. When there are two events e_a and e_b in σ and in the instance ordering $a \succ_\sigma b$ appears, then e_a is said to be a closest causal predecessor of e_b. An event can only appear in a process instance, after all its closest causal predecessors have appeared.

When determining how work is transferred between data-element instances in a process instance, each event in the process instance should be compared against its closest causal predecessors. When event e_b is compared against closest causal predecessor e_a ($a \succ_\sigma b$), two situations can occur:

- e_a and e_b refer to the same data-element instance. If this is the case, we know that the data-element instance to which both refer, is active at least from the timestamp to which e_a refers, until the timestamp to which e_b refers.
- e_a and e_b refer to different data-element instances. If this is the case, we know that there is an arrow, i.e. a transfer of work from the data-element instance to which e_a refers to the data-element instance to which e_b refers. This arrow starts at the timestamp to which e_a refers and ends at the timestamp to which e_b refers.

The comparison of all events of a process instance against all their closest causal predecessors results in a collection of arrows and periods of activity. These arrows and blocks (i.e. periods) together form a sequence. It is possible that the periods in which a certain data-element instance is active in a sequence overlap. In a department for instance, more than one activity for the same case can be executed at the same time. When overlap occurs, the blocks that overlap should be combined, because a data-element instance is active or not and cannot be active a number of times. Now we can define:

Definition Sequence : A sequence s_σ represents the transfer of work between the set of data-element instances I_{s_σ}, that occur in process instance σ, over time. The sequence consists of a set of blocks B_{s_σ} and a set of arrows AR_{s_σ}, where:

B_{s_σ} : a set of blocks or periods in which the data-element instances in I_{s_σ} are active. Each block $b \in B_{s_\sigma}$ has a begin timestamp, $T_{begin}(b)$ an end timestamp $T_{end}(b)$, and a data-element instance $di(b)$, associated with it.

$AR_{s_\sigma} \subseteq B_{s_\sigma} \times B_{s_\sigma}$: a set of arrows that connect the blocks and represent the transfer of work between data-element instances. Each arrow $ar \in AR_{s_\sigma}$ has associated with it a begin timestamp $T_{begin}(ar)$, an end timestamp $T_{end}(ar)$, a source block $b_{src}(ar)$ from which it originates, and a destination block $b_{dest}(ar)$ in which it ends.

Let us consider an example that illustrates how a sequence can be derived from a process instance by means of an instance ordering. Suppose we have the process instance σ and the

TaskID	Department	Timestamp	Instance Ordering
A	Production	1:00	$1 \succ_\sigma 2$
B	Production	2:00	$2 \succ_\sigma 3$
C	Sales	3:00	$2 \succ_\sigma 4$
D	Purchase	3:00	$3 \succ_\sigma 5$
E	Sales	4:00	$4 \succ_\sigma 6$
F	Purchase	4:00	$5 \succ_\sigma 7$
G	Sales	5:00	$6 \succ_\sigma 7$
H	Production	5:00	$7 \succ_\sigma 8$

Table 4.2: Process instance σ and its instance ordering

instance ordering of this process instance, as displayed in table 4.2. Suppose also that we want to study how work is transferred between departments.
We go through the events in the process instance in chronological order, starting with the first event (event 1). This event does not have a closest causal predecessor. However, from this event we can deduce that the production department is active at least for a moment at t_{e_1}, where t_{e_1} is the timestamp associated with the first event (in this case timestamp 1:00). Therefore, a block b_0 is created and we set:

$$\begin{aligned} di(b_0) &= \text{Production} \\ T_{begin}(b_0) &= t_{e_1} \\ T_{end}(b_0) &= t_{e_1} \end{aligned}$$

Since we have the relation $1 \succ_\sigma 2$, we compare the department of the first event with the department of the second event. These are the same, so we now know that the production department is active at least from t_{e_1} to t_{e_2}. This overlaps with b_0 however, so we combine these periods by setting $T_{end}(b_0)$ to t_{e_2}. We then move on to the third event. For this event the following relation can be found: $2 \succ_\sigma 3$. The third event does not refer to the production department, but to the sales department. The sales department is thus active for at least a moment at t_{e_2}. Therefore we create block b_1, where:

$$\begin{aligned} di(b_1) &= \text{Sales} \\ T_{begin}(b_1) &= t_{e_2} \\ T_{end}(b_1) &= t_{e_2} \end{aligned}$$

Because the second and third event refer to different departments, also an arrow ar_0 is created, where we set:

$$\begin{aligned} T_{begin}(ar_0) &= t_{e_1} \\ T_{end}(ar_0) &= t_{e2} \\ b_{src}(ar_0) &= b_0 \\ b_{dest}(ar_0) &= b_1 \end{aligned}$$

If we continue with the other events in this same manner, the result will be the sequence

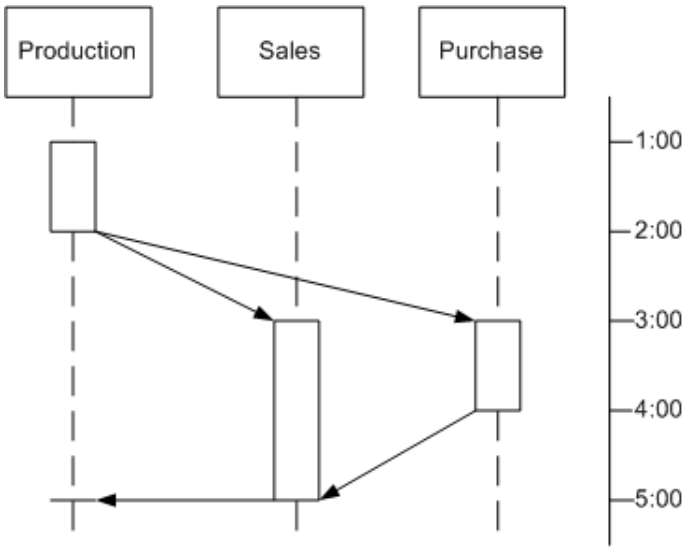

Figure 4.2: The derived sequence

as depicted in figure 4.2. This sequence contains four periods (blocks) in which one of the departments is active and four arrows which represent a transfer of work from one department to another. In figure 4.2, the sequence is displayed in a sequence diagram. We define:

Definition (Performance) Sequence Diagram : a sequence diagram is a diagram in which a set of sequences S is displayed, which represent the transfer of work between data-element instances over time. Each sequence in a sequence diagram is displayed in a different color and sequences can overlap, i.e. a data-element instance can be active in more than one sequence at the same time. A sequence diagram has two dimensions:

I : the set of all data-element instances in S: $\{i \in I_{s_\sigma} |\ s_\sigma \in S\}$.
The data-element instances that occur in this set are displayed as lifelines along the horizontal axis.

T : the time that passes from the beginning of the first sequence in S until the end of the last sequence in S. The time dimension is displayed along the vertical axis. The time axis begins at $MIN_{(b \in B_{s_\sigma} \wedge s_\sigma \in S)}\ T_{begin}(b)$ and ends at $MAX_{(b \in B_{s_\sigma} \wedge s_\sigma \in S)}\ T_{end}(b)$

It should be possible for the user to view a sequence diagram that contains all sequences that can be derived from the input event log. Furthermore, for each sequence $s_\sigma \in S$, the following information should be accessible:

- **Throughput time of** s_σ : Time between the beginning and end of the sequence s_σ. This can be defined as:
  $T(s_\sigma) = MAX_{b \in B_{s_\sigma}}\ T_{end}(b) - MIN_{b \in B_{s_\sigma}}\ T_{begin}(b)$

For each block $b \in B_{s_\sigma}$, also the following should be available:

- **Time in block** b
  Time between begin and end of block b, this can be defined as:
  $T_{end}(b) - T_{begin}(b)$

For each arrow $ar \in AR_{s_\sigma}$, also the following should be available:

- **Time in arrow** ar
  Time between begin and end of arrow ar. This can be defined as:
  $T_{end}(ar) - T_{begin}(ar)$

4.3.2 Deriving Pattern Diagrams

To diminish the load of information that the user has to search through, pattern diagrams should be available to the user. In pattern diagrams, sequences that follow similar behavior are represented by one pattern. To be able to group sequences in this way however, it first has to be clear how sequences can be distinguished from other sequences.

Comparing two sequences starts with checking whether the number of blocks in the two sequences is equal, and if the number of arrows in the sequences is equal as well. When this is the case, it should be possible to compare each block in the one sequence with a block in the other sequence, and an arrow in the one sequence with an arrow in the other sequence. That is why the blocks and the arrows in these sequences should be sorted. For each sequence s_σ, blocks should be sorted first alphabetically on the name of the data-element instance that it is part of. If there is more than one block active in the same data-element instance, the blocks should further be sorted (ascendingly) on their begin timestamp. Since there are no two blocks active in the same data-element instance at the same time in the same sequence, the blocks do not have to be sorted any further. By sorting, each block $b \in B_\sigma$ gets a unique index. When comparing two sequences, each block in the one sequence should be compared against the block in the other sequence that has the same index. For a block $b \in B_\sigma$, we refer to its peer, i.e. block with the same index, in another sequence $s_{\sigma'}$ as $sim(b, s_{\sigma'})$.

In a similar manner the arrows should be sorted. First on the name of the data-element instance in which the source block is active, then on the name of the destination, then on the begin timestamp and finally on the end timestamp. However, sometimes it can occur that arrows have the same values for all these attributes. If this is the case, a random order is chosen for these arrows. This can be done, because the arrows are equal anyway. After the sorting, each arrow in AR_σ has a unique index, which should be used to compare the arrows (with the same index) of different sequences. For an arrow $ar \in AR_\sigma$, we refer to its peer, i.e. arrow with the same index, in sequence $s_{\sigma'}$ as $sim(ar, s_{\sigma'})$.

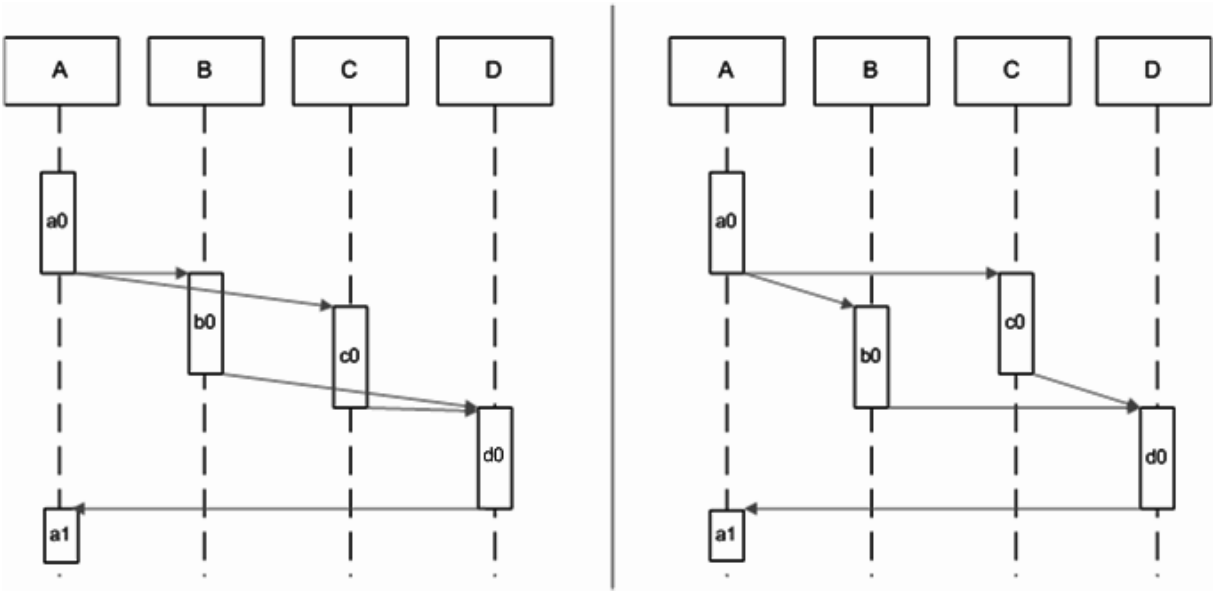

Figure 4.3: Example of sequences

When the orderings of the blocks and the arrows are known, it is possible to compare sequences. There are still different ways to do this however. Suppose we have the sequences as depicted in figure 4.3. If we compare these two sequences in a strict manner, these sequences are different, because in the left sequence block b_0 begins (and ends) before block c_0, while in the right sequence block c_0 begins (and ends) before block b_0. However, suppose that what these sequences represent is the transfer of work between tasks. It is easy to imagine that after task A, task B and C are executed in parallel, so it can happen that at one time task B begins execution first, and the other time task C begins execution first. It is a valid option then to place these sequences in the same pattern. Therefore two different ways of comparing sequences should be distinguished: one using *strict-equivalence*, and one using *flexible-equivalence*. The user should have the possibility to select which of both equivalence types should be used.

Definition Sequence Equivalence:
Two sequences s_σ and $s_{\sigma'}$ are equivalent, i.e. follow the same pattern, if and only if the following requirements are met:

(i) The sequences contain the same data-element instances: $I_{s_\sigma} \equiv I_{s_{\sigma'}}$

(ii) Equivalent blocks are present in the sequences. This is dependent on the type of equivalence (strict or flexible) used though. For both the equivalence types, the data-element instance of each block in the one sequence must be equal to the data-element instance of the block in the other sequence that has the same index. The following function evaluates to true if this requirement is met:

(a) $\#_{B_{s_\sigma}} \equiv \#_{B_{s_{\sigma'}}} \ \wedge \ \forall_{b \in B_{s_\sigma}}(di(b) \equiv di(sim(b, s_{\sigma'})))$

Strict-equivalence furthermore requires the blocks in both sequences to begin and end in exactly the same order. The following function evaluates to true if this requirement is met:

(b) $\forall_{b_0,b_1 \in B_{s_\sigma}}((T_{begin}(b_0) \geq T_{begin}(b_1) \Rightarrow T_{begin}(sim(b_0, s_{\sigma'})) \geq T_{begin}(sim(b_1, s_{\sigma'})))$
$\wedge \ (T_{end}(b_0) \geq T_{end}(b_1) \Rightarrow T_{end}(sim(b_0, s_{\sigma'})) \geq T_{end}(sim(b_1, s_{\sigma'}))))$

Flexible-equivalence instead adds the requirement that for each block b_1 in sequence s_σ that begins after block b_0 ends, its peer in $s_{\sigma'}$, $sim(b_1, s_{\sigma'})$, should begin after or at the same time the peer of b_0, $sim(b_0, s_{\sigma'})$, ends. Or formally:

(c) $\forall_{b_0,b_1 \in B_\sigma}(T_{begin}(b_1) > T_{end}(b_0) \Rightarrow T_{begin}(sim(b_1, s_{\sigma'})) \geq T_{end}(sim(b_0, s_{\sigma'})))$

(iii) Equivalent arrows should exist. This again is dependent on the type of equivalence used. At least the source block of each arrow in the one sequence must be equal to (i.e. have the same index as) the source block of the arrow with the same index in the other sequence, and the same for the destination block. The following function evaluates to true if this requirement is met:

(a) $\#_{AR_{s_\sigma}} \equiv \#_{AR_{s_{\sigma'}}} \ \wedge \ \forall_{ar \in AR_{s_\sigma}}(b_{src}(ar) \equiv sim(b_{src}(sim(ar, s_{\sigma'})), s_\sigma)$
$\wedge \ b_{dest}(ar) \equiv sim(b_{dest}(sim(ar, s_{\sigma'})), s_\sigma))$

This is the only requirement placed on arrows for **flexible-equivalence**.

Strict-equivalence also requires arrows to start and end in exactly the same order. The following function evaluates to true if this is the case:

(b) $\forall_{ar_0,ar_1 \in AR_{s_\sigma}}((T_{begin}(ar_0) \geq T_{begin}(ar_1) \Rightarrow T_{begin}(sim(ar_0, s_{\sigma'})) \geq T_{begin}(sim(ar_1, s_{\sigma'})))$
$\wedge \ (T_{end}(ar_0) \geq T_{end}(ar_1) \Rightarrow T_{end}(sim(ar_0, s_{\sigma'})) \geq T_{end}(sim(ar_1, s_{\sigma'}))))$

In the example in figure 4.3, the requirements (i), (iia), (iic) and (iiia) are met, but requirements (iib) and (iiib) are not. The sequences are thus flexible-equivalent, but not strict-equivalent.

Using sequence equivalence, all sequences derived from the input log, should be grouped in patterns, i.e. groups of sequences that follow similar behavior. This should be done by examining the sequences one by one and comparing each sequence with the patterns that were found thus far. If the sequence is sequence equivalent with the sequences that follow a certain pattern, then the sequence should be added to the list of sequences that follow that pattern. If the sequence is not sequence equivalent with any of the (sequences in the) existing patterns, a new pattern should be created of which the sequence becomes the initial sequence. Each pattern thus consists of at least one (initial) sequence and possibly of a group of sequences that follow the same behavior as this first sequence. Furthermore, each sequence is part of exactly one pattern.

Definition Pattern A pattern p consists of a set of sequences S_p that are *sequence equivalent*. A pattern, just like a sequence represents the transfer of work between a set of

data-element instances I_p ($= \{i \in I_{s_\sigma} |\ s_\sigma \in S_p\}$), and consists of a set of blocks B_p, and a set of arrows AR_p, where:

B_p : a set of blocks or periods in which the data-element instances in I_p are active. B_p contains the same blocks as each of the sequences in S_p. For each block $b \in B_p$, $sim(b, s)$ refers to the block in sequence $s \in S_p$ that has the same index as b. Then di_b is of course equal to $di_{sim(b,s)}$. Each block $b \in B_p$, does not have a begin (or end) timestamp associated with it, but an average begin (and end) time. The average begin time is the average time between the begin timestamp of a sequence and the begin timestamp of the block $sim(b, s)$ in the sequence. For each $b \in B_p$, we define:
$\bar{T}_{begin}(b) = AVG_{s \in S_p}\ (T_{begin}(sim(b, s)) - MIN_{b \in B_s}\ T_{begin}(b))$
$\bar{T}_{end}(b) = AVG_{s \in S_p}\ (T_{end}(sim(b, s)) - MIN_{b \in B_s}\ T_{begin}(b))$

$AR_p \subseteq B_p \times B_p$: a set of arrows that connect the blocks and represent the transfer of work. AR_p contains the same arrows as each of the sequences in S_p. Each arrow $ar \in AR_p$ has an arrow $sim(ar, s)$ in each sequence $s \in S_p$, that has the same index as ar and the 'same' source block, $sim(b_{src}(ar), s) \equiv b_{src}(sim(ar, s))$ and destination block $sim(b_{dest}(ar), s) \equiv b_{dest}(sim(ar, s))$ Just like with blocks, the average time from the begin timestamp of each sequence is calculated for the begin and end of each arrow $ar \in AR_p$:
$\bar{T}_{begin}(ar) = AVG_{s \in S_p}\ (T_{begin}(sim(ar, s)) - MIN_{b \in B_s}\ T_{begin}(b))$
$\bar{T}_{end}(ar) = AVG_{s \in S_p}\ (T_{end}(sim(ar, s)) - MIN_{b \in B_s}\ T_{begin}(b))$

The patterns obtained this way should be placed in a pattern diagram. In such a diagram, each pattern that occurs in the input log is displayed separately. In the diagram the patterns are sorted descendingly based on the frequency. The frequency of a pattern p is the number of sequences that it contains ($\#_{S_p}$).

Definition Pattern Diagram : a pattern diagram is a diagram in which a set of patterns P is displayed. Each pattern is displayed separately from the other patterns. The pattern with the highest frequency is displayed at the top of the diagram, and the one with the lowest frequency at the bottom. A pattern diagram has two dimensions:

I : the set of all data-element instances in P: $\{i \in I_p |\ p \in P\}$.
The data-element instances that occur in this set are displayed as lifelines along the horizontal axis.

T : the time dimension. For each pattern p the time starts at 0, and ends at:
$MAX_{b \in B_p}\ \bar{T}_{end}(b) - MIN_{b \in B_p}\ \bar{T}_{begin}(b)$

The user should have the possibility to view the pattern diagram, which contains all different patterns that occur in the sequence diagram. Furthermore, for each pattern the following information should be available:

- **Throughput time of pattern** p: The time between the begin and end of sequences that belong to p. The following statistical values are calculated for the throughput time of pattern p:
  - $AVG_{s \in S_p}\ T(s)$ the average throughput time.
  - $MIN_{s \in S_p}\ T(s)$ the minimum throughput time.
  - $MAX_{s \in S_p}\ T(s)$ the maximum throughput time.
  - $STDEV_{s \in S_p}\ T(s)$ the standard deviation in throughput time.

For each block $b \in B_p$ the following information should be available:

- **Time in block** b: average duration of block b:
  $\bar{T}_{end}(b) - \bar{T}_{begin}(b)$

For each arrow $ar \in AR_p$ the following information should be available:

- **Time in arrow** ar: average time between the begin and end of arrow ar:
  $\bar{T}_{end}(ar) - \bar{T}_{begin}(ar)$

Chapter 5

Design and Implementation

5.1 Introduction

In this chapter, the plug-ins that have been implemented during this graduation project and their design will be discussed.

This graduation project has resulted in the implementation of two separate analysis plug-ins, which have been added to the ProM framework:

1. Performance Analysis with Petri net (described in section 5.2)
2. Performance Sequence Diagram Analysis (described in section 5.3)

5.2 Performance Analysis with Petri net

The purpose of the *Performance Analysis with Petri net* plug-in is to provide the user with a means to assess the performance of business processes. Main focus here is to provide the user with the values of key performance indicators in an intuitive manner.

A design of the plug-in was made, based on the requirements that were established in section 4.2. Design of the plug-in started with setting up use cases (described in appendix E.1) and creating prototypes (described in appendix F.1). After this, UML class diagrams were designed that describe the structure and interfaces of the plug-in. In class diagrams the classes with their most important methods and the relationships between these classes are depicted. Figures 5.1 and 5.2 depict the class diagrams of the *Performance analysis with Petri net* plug-in. The classes located in the lower part of the diagrams are actually part of the plug-in. The classes in the upper part of the diagrams are not part of the plug-in, but can be accessed by the plug-in through the ProM framework.

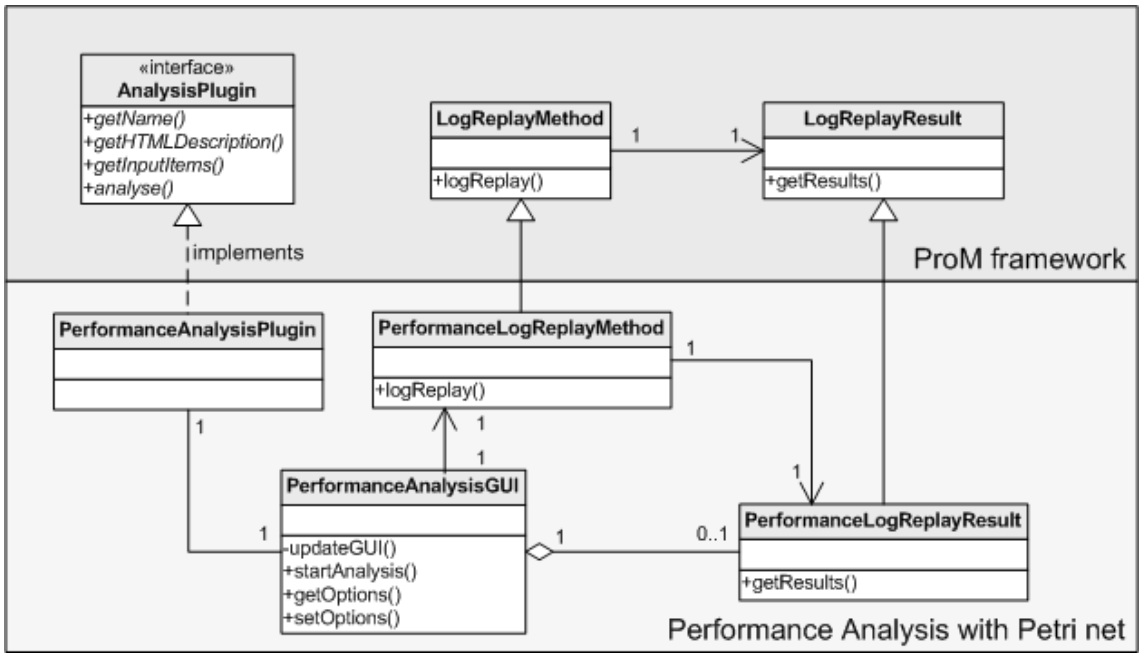

Figure 5.1: Class diagram *Performance Analysis with Petri net* plug-in

All analysis plug-ins of the ProM framework need to implement the `AnalysisPlugin` interface. As can be seen in 5.1, the `PerformanceAnalysisPlugin` class implements this interface for the *Performance Analysis with Petri net* plug-in. The interface requires four methods: one specifying the name of the plug-in, one which returns the user documentation in HTML, one which specifies the required input items, and finally the `analyse()` method, which is called as soon as the plug-in is invoked. When the `analyse()` method of the `PerformanceAnalysisPlugin` is called, first the `PerformanceAnalysisGUI`, i.e. the GUI of the plug-in, is created. Through this GUI the user can set certain options before starting the actual analysis. When the analysis is started, the `logReplay()` method of the `PerformanceLogReplayMethod` is called. The `PerformanceLogReplayMethod` extends the `LogReplayMethod` with performance analysis functionality. The results of the `logReplay()` method are obtained as a `PerformanceLogReplayResult`, which provides access to the data that has been collected.

As can be seen in figure 5.2, the `PerformanceLogReplayResult` consists of an extension of the input log and an extension of the input Petri net with performance information. Methods that start with `getValues` return the statistical values of the time-related KPI in question. Note that activity-related KPIs can only be accessed through an `ExtendedTransition`. Each `ExtendedTransition` is related to zero or more `Activity` objects. A visualization of the Petri net is available through the `ExtendedPetriNet` class and can be displayed in the GUI of the plug-in.

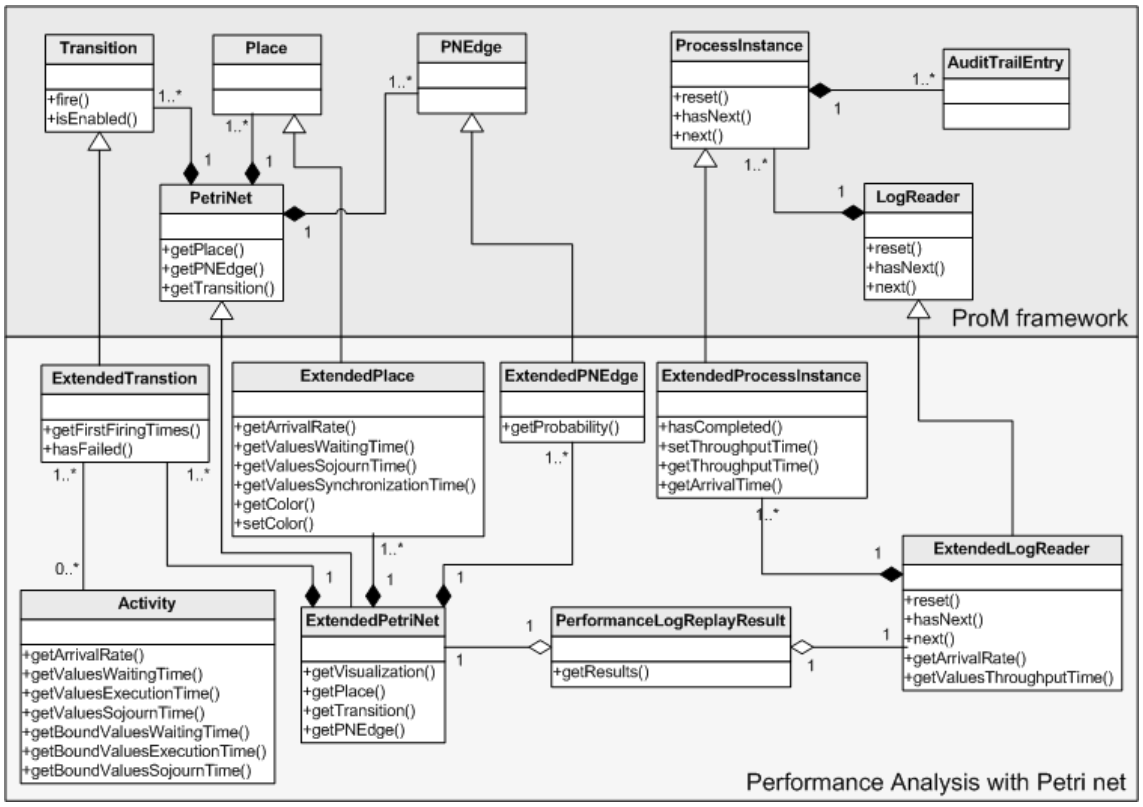

Figure 5.2: Class diagram Results of performance log replay

Based on the design, the plug-in was implemented and added to the ProM framework after thorough testing. In figure 5.3 the main graphical user interface of the plug-in is shown. In the example shown here, a place has been selected and the KPIs related to this place are depicted below the Petri net. More details on how to use the plug-in can be found in the user manual as presented in appendix H.1.

During the tests, the use cases that were defined in E.1 were performed with the plug-in. A description of these test cases can be found in appendix G.1. The test cases were performed with various logs and Petri nets as input. In general the plug-in performed well during these tests and results where obtained within reasonable time. However, in some situations problems arose. In the cases where the input Petri net was large and contained many invisible transitions, the log replay method took much time. Research learned that the building of coverability graphs during log replay was the main cause of this. Coverability graphs are built during log replay to obtain the shortest sequence of invisible transitions at every time the 'transition to fire' is not immediately enabled. The building of a coverability graph takes much time in those cases where many markings can be reached by firing invisible transitions. This typically appears in large Petri nets that contain many invisible transitions. To be able to get results from the plug-in within reasonable time in all cases, an option was added which,

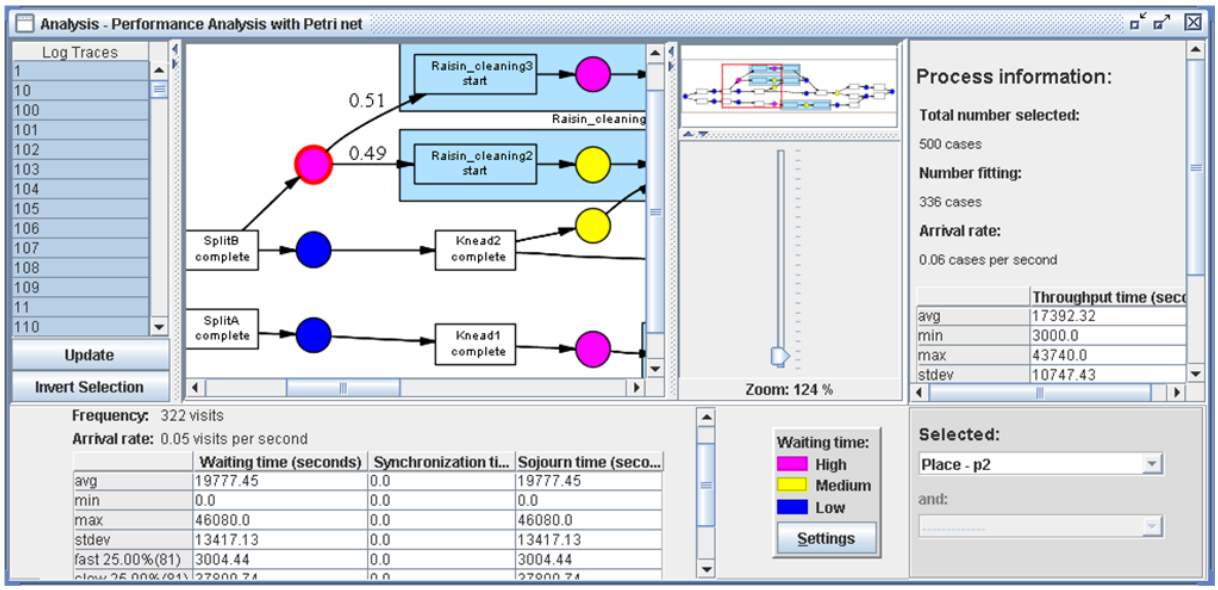

Figure 5.3: Screenshot of the *Performance Analysis with Petri net* plug-in, where a place is selected

if enabled, makes sure that the log replay method does not build coverability graphs. Instead, all the 'transitions to fire' that are not immediately enabled fail execution. This clearly has its effect on the obtained results though, because cases that actually fit in the process model can become non-fitting. Especially the values of the KPIs that are calculated based on the structure of the Petri net, such as place-related KPIs and bound values for activity-related KPIs, are affected by this. A possible solution may be to perform log replay in heuristic nets instead of in Petri nets. Heuristic nets are process models as well, but they do not contain places nor 'invisible' transitions or tasks. Because the concept of places does not exist, the place-related KPIs will be lost when using heuristic nets and 'bottleneck coloring' cannot be done then. On the plus side, the bound values of activity-related KPIs can be calculated more accurately when using heuristic nets and results will likely be found faster. Further research will be needed to see if using heuristic nets is feasible and useful.

Another issue that requires attention is the consumption of a token from each input place at firing of transitions during log replay. When an input place contains multiple tokens, the log replay method selects a token from this place (pseudo-)randomly. In real life situations, selection often is not done randomly at all. The First In First Out (FIFO) principle is for instance frequently used instead. Certain statistical values of KPIs can be affected by the used selection method, e.g. the minimum, maximum, and standard deviation value of the sojourn time of a place. It may be worthwhile to extend the log replay method with support for other selection methods, such as FIFO. The user can then be given the choice on which principle to use for each place.

Furthermore, we discovered that the log replay method in some (exceptional) situations may cause a transition to fail when it should not. Take for instance the Petri net in figure 5.4 in which trace *abc* is replayed. Using the current log replay method, first transition a fires, and then b. Transition c is not enabled after this and cannot be enabled by firing a sequence of invisible transitions, so c fails execution. However, when after firing transition a, the invisible transition t is fired before transition b is, then after firing b, c is enabled and can fire normally. Note that this a very specific situation, which does not appear very often in practice. Still, research should be done to find a solution for this.

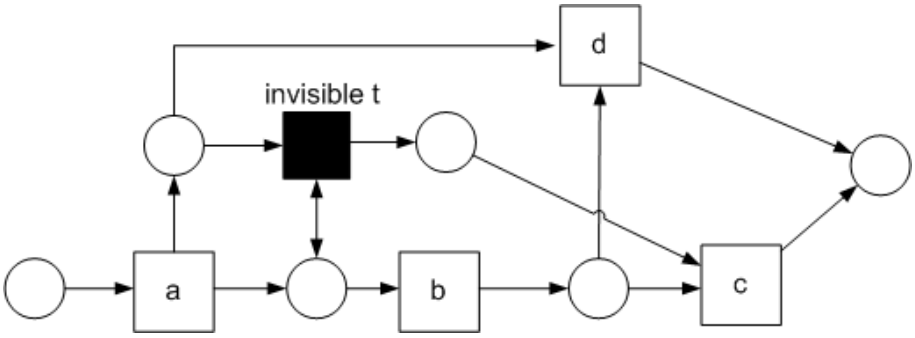

Figure 5.4: Exception

5.3 Performance Sequence Diagram Analysis

The purpose of the *Performance Sequence Diagram analysis* plug-in is to provide the user with a means to assess the performance of business processes. The main focus here is to assist in finding the causes of certain (e.g. extreme or common) behavior. This is done by means of diagrams in which the hand-over of work between data-element instances is displayed.

A design of this plug-in was made, based on the requirements that were established in section 4.3. Again, design started with creating use cases (described in appendix E.2) and prototypes (described in appendix F.2) for the plug-in. In figure 5.5 the class diagram that resulted from the design effort is shown. The classes in the lower part of the diagram are actually part of the *Performance Sequence Diagram Analysis* plug-in. The classes in the upper part of the diagram are not part of the plug-in, but can be accessed by the plug-in through the ProM framework. The `PerformanceSequenceAnalysisPlugin` class is the implementation of the `AnalysisPlugin` interface for this analysis plug-in. When the `analyse()` method of this plug-in is invoked, the GUI of the plug-in (`PerformanceSequenceAnalysisGUI`) is built. Through this GUI initial options (data-element, equivalence type, time unit) can be set. When the `deriveDiagrams()` method is called, a sequence and a pattern diagram are derived from the input event log, using the options and the `InstanceMiner`. The `InstanceMiner` is a plug-in of the ProM framework which can derive log relations from a log. These log relations can

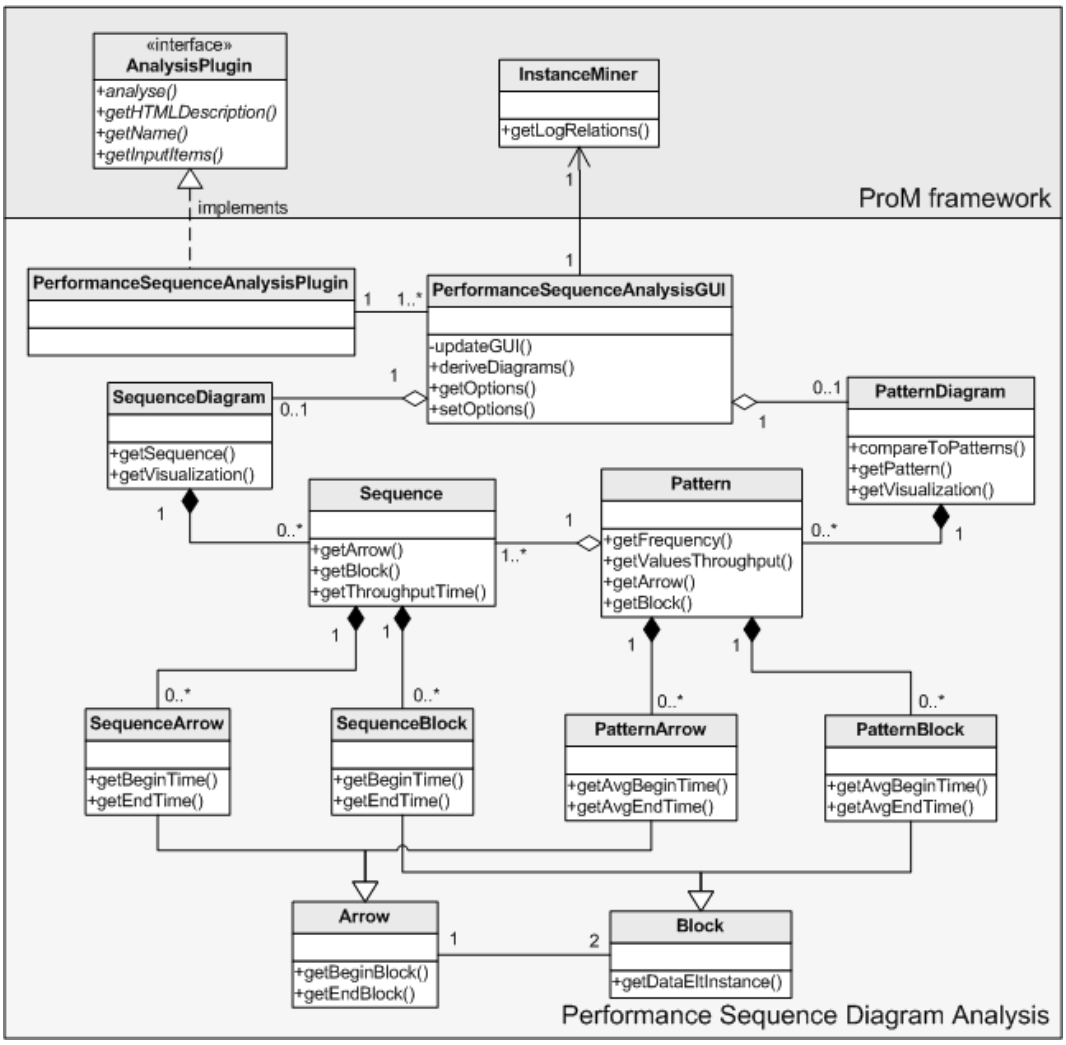

Figure 5.5: Class diagram *Performance Sequence Diagram* plug-in

be used to obtain instance orderings, which are needed to be able to derive sequences. The resulting `SequenceDiagram` contains the derived `Sequences` which consist of `SequenceBlocks` and `SequenceArrows`. The `PatternDiagram` contains `Patterns`, which represent a number of `Sequences`. `Patterns` consist of `PatternBlocks` and `PatternArrows`.

Based on the design, the plug-in was implemented and added to the ProM framework after thorough testing. In figure 5.6 the GUI of the plug-in is shown. In the depicted example the pattern diagram is shown. For more details on how to use the plug-in, see the user manual as presented in appendix H.2.

During the tests, the use cases that were defined in E.2 were performed with the plug-in. A description of these test cases can be found in appendix G.2. The test cases were performed with various logs. The plug-in performed well during the tests and results were obtained

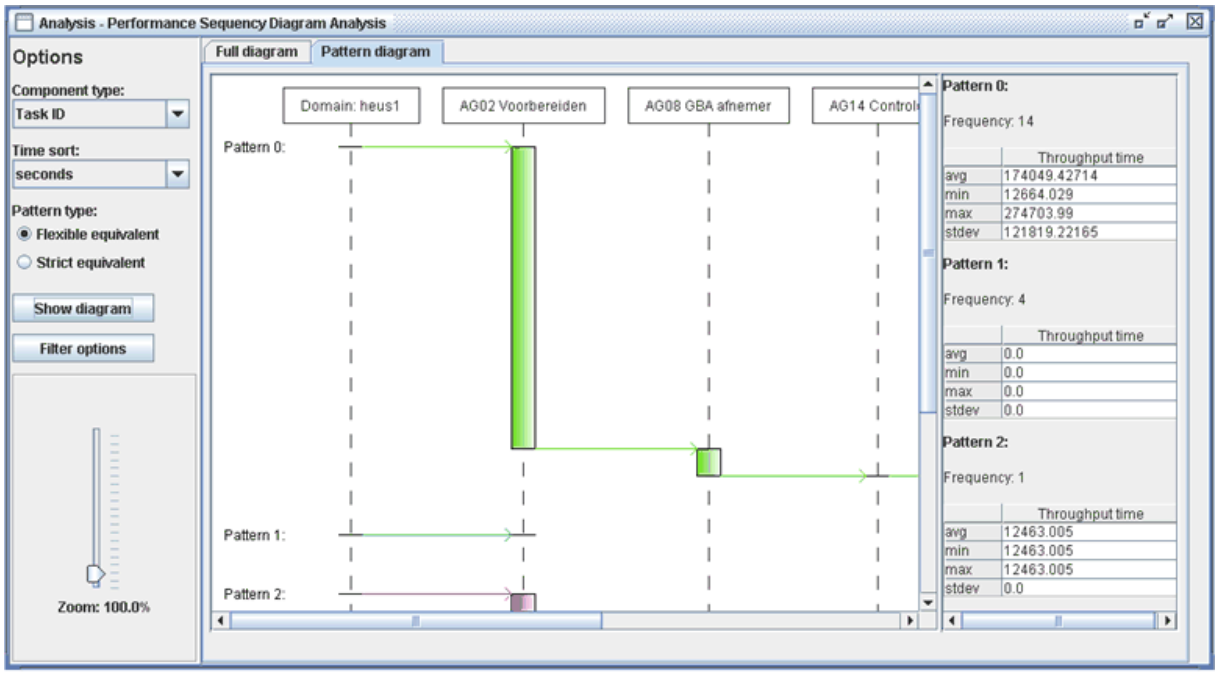

Figure 5.6: Screenshot of *Performance Sequence Analysis* plug-in (Pattern diagram view)

within reasonable time. One of the things that we learned from the tests was that some of the pattern diagrams that were derived contained many different patterns. A downside of having many patterns in a diagram is that the user still has to search through a lot of information to determine the causes of certain behavior. To diminish the load of information that the user has to consider, filtering options were added to the plug-in. Filtering allows the user to examine only those sequences and/or patterns that are relevant to him. The filtering options that were introduced use process instance selection. The sequence and the pattern diagram are derived from the selected process instances only. The following options to select process instances were made available to the user:

- Select process instances by selecting their names in a table.
- Select all process instances that are represented by sequences that have a throughput time above/below a certain bound value.
- Select all process instances that are represented by sequences that follow one of the patterns that have an average throughput time above/below a certain bound value.
- Select all process instances that are represented by sequences that follow a certain pattern (or one of group of patterns), by selecting the pattern (or a group of patterns).

Chapter 6

Evaluation

Throughout this graduation project, the emphasis of our research has been on the analysis of the performance of business processes. In chapter 1 of this document, two research questions were defined, which aim was to guide the research in the right direction. During our research we have tried to answer these questions. The first research question was defined as:

Which process mining functionality is available in commercially developed process monitoring tools and which parts of this functionality would be enhancements to the ProM framework?

In order to answer this research question, tools were investigated that are among the market leaders in the field of process performance analysis: *Aris Process Performance Manager*, *Business Objects*, *HP Business Process Cockpit*, *Hyperion System 9*, and *ILOG JViews*. An overview of the most important functionality present in these tools was presented in section 3.2 of this thesis. Next to the investigation of these commercial tools, our research aimed at answering the second research question, which was defined as:

Since we have a process model at our disposal within the ProM framework, which additional process mining functionality would qualify for implementation into the ProM framework?

Although this question left room for investigation of functionality in other process mining areas, our main focus here was also on the analysis of the performance of business processes. The reason for this was that the need for performance analysis functionality in the ProM framework at that moment was particularly high. During research we detected that the second question partly overlaps with the first, since some commercial process monitoring tools, such as *ILOG JViews*, do make use of process models. In these tools a user-defined process model is used as an interface through which performance information can be accessed. An advantage of the ProM framework is that next to user-defined process models, also process models that are mined from event logs can be obtained and used for this purpose. Discussion with other researchers and investigation of the academic tool *EMiT* led us to believe that process models, next to being used as a user interface, can also be used as a means to derive certain performance information. Functionality that supports process performance analysis

and that makes use of a process model, was presented in sections 3.2.2, 3.3, and 3.4.2 of this thesis.

Next to answering the research questions, our research aimed at finding more functionality that could assist in analyzing the performance of business processes. Research of related tools (*IBM Web Services Navigator*) and discussion with other researchers at the BPM section led to new ideas for functionality in this area. The sequence and pattern diagrams (as presented in section 3.4.1) and the dotted charts (as presented in section 3.4.3) were for instance found this way.

The main conclusion that we drew from our research was that tools that support process performance analysis, and thus also our plug-ins, should mainly serve the following purposes:

- Monitor the performance status of processes, by keeping track of the values of KPIs.
- Assist in detecting the causes of possible deviations in the values of the monitored KPIs.

After discussion with other researchers at the BPM section, the functionality that was found most suitable for achieving this was selected. The selected functionality and the requirements placed thereon are discussed in chapter 4 of this document. Eventually, our research has led to the implementation of two plug-ins, which have been added to the ProM framework and can be used for analyzing the performance of business processes.

The *Performance Analysis with Petri net* plug-in, is a plug-in which derives the values of certain KPIs from event logs by using a process model (a Petri net). This process model is also used as an intuitive interface in which problem areas can easily be spotted. In section 5.2 of this document, the design and implementation of the plug-in was discussed.

The *Performance Sequence Diagram Analysis* plug-in assists in Process Performance Analysis by visualizing the transfer of work between the components that collaborate in a business process. This can be particularly useful for detecting the causes of certain rare or unwanted behavior. In section 5.3 of this document, the design and implementation of the plug-in was discussed.

The overall conclusion that we can draw is that this project has resulted in the extension of the ProM framework with plug-ins that enable the analysis of the performance of business processes. Therefore we can establish that the goals that were set at the beginning of the project have been largely achieved.

Bibliography

[1] M. de Pauw, W.and van Lei, E. Pring, L. Villard, M. Arnold, and J.F. Morar. Web services navigator: Visualizing the execution of web services. *IBM Systems Journal*, 2005.

[2] J. Desel and J. Esparza. Free choice petri nets. *Cambridge Tracts in Theoretical Computer Science*, 40, 1995.

[3] S.D.P. Flapper, L. Fortuin, and P.P.M. Stoop. Towards consistent performance management systems. *International Journal of Operations And Production Management*, 16:27–37, 1996.

[4] Nonprofit Good Practice Guide. Performance. `http://www.npgoodpractice.org/CompleteGlossary.aspx?ID=-1&curLetter=P`, 18-8-2006.

[5] C.W. Gunther and W.M.P. van der Aalst. A generic import framework for process event logs. 2006.

[6] M. Hammer and J. Champy. *Reengineering the Corporation - A Manifesto for Business Revolution.* Harper Collins Publishers, 1993.

[7] Investopedia.com. Bottleneck. `http://www.investopedia.com/terms/b/bottleneck.asp`, 22-8-2006.

[8] J. Jeston and J. Nelis. *Business Process Management: Practical Guidelines to Successful Implementations.* Elsevier, 1st edition, 2006.

[9] ILOG JViews. Business process management visualization with ilog jviews. technical whitepaper. 2004.

[10] R.S. Kaplan and D.P. Norton. *Translating Strategy into Action: The Balanced Scorecard.* Harvard Business School Press, Boston, 1996.

[11] M.J. Lebas. Performance measurement and performance management. *International Journal of Production Economics*, 41:23–35, 1995.

[12] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77-4:541–580, 1989.

[13] Processmining.org. Process mining website. `http://www.processmining.org`, 5-12-2006.

[14] A. Ramesh. Process mining in peoplesoft. Master's thesis, 2006.

[15] F.J. Reh. Management: You can't manage what you don't measure. `http://management.about.com/cs/generalmanagement/a/keyperfindic.htm`, 22-8-2006.

[16] W. Reizig and G. Rosenberg. Lectures on petri nets 1, basic models. *Lecture notes in Computer Science*, 1491, 1998.

[17] A. Rozinat and W.M.P. van der Aalst. Conformance testing: Measuring the alignment between event-logs and process models. *BETA Working Paper Series, WP 144*, 2005.

[18] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison Wesley, 1999.

[19] IDS Scheer. Aris process performance manager. measure, analyze and optimize your business process performance (whitepaper). *IDS Scheer*, 2002.

[20] W.F.M. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. The MIT Press, Cambridge, 2002.

[21] W.M.P. van der Aalst, A.K. Alves de Medeiros, and A.J.M.M Weijters. Genetic process mining. *26th International Conference on Applications and Theory of Petri Nets (ICATPN 2005), G. Ciardo and P. Darondeau, LNCS 3536*, 2005.

[22] W.M.P. van der Aalst and M. Song. Mining social networks: Uncovering interaction patterns in business processes. *International Conference on Business Process Management (BPM 2004)*, 2004.

[23] W.M.P. van der Aalst and B.F. van Dongen. Discovering workflow performance models from timed logs. *Lecture Notes in Computer Science*, 2002.

[24] W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow mining: A survey of issues and approaches. *Data and Knowledge Engineering*, 2003.

[25] W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow mining: Which processes can be rediscovered? *BETA Working Paper Series*, 2002.

[26] B.F. van Dongen, A.K.A. 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. *6th International Conference on Applications and Theory of Petri Nets, G. Ciardo and P. Darondeau*, LNCS 3536:444–454, 2005.

[27] B.F. van Dongen and W.M.P van der Aalst. Multi-phase process mining: Building instance graphs. *In Atzeni, P., Chu, W., Lu, H., Zhou, S., Ling, T., eds.: Conceptual Modeling - ER 2004. LNCS 3288*, pages 362–376, 2004.

[28] B.F. van Dongen and W.M.P van der Aalst. Multi-phase process mining: Aggregating instance graphs into epcs and petri nets. *2nd International Workshop on Applications of Petri Nets to Coordination, Worklflow and Business Process Management (PNCWB) at the ICATPN 2005*, 2005.

[29] Wikipedia.org. Business performance management. `http://en.wikipedia.org/wiki/Business_Performance_Management`, 18-8-2006.

Appendix A

Mining XML format

This appendix describes the Mining XML (MXML) format as introduced in [26], to which logs that serve as input to the ProM framework have to comply. In figure A.1(a) a visual description of the MXML format is given. The root element in this format is the *WorkflowLog* element, which indicates that the log is a workflow log. Each *WorkflowLog* element contains an optional *Data* element, which can be used to store arbitrary textual data, an optional *Source* element, which can be used to store information about the source of this log (the information

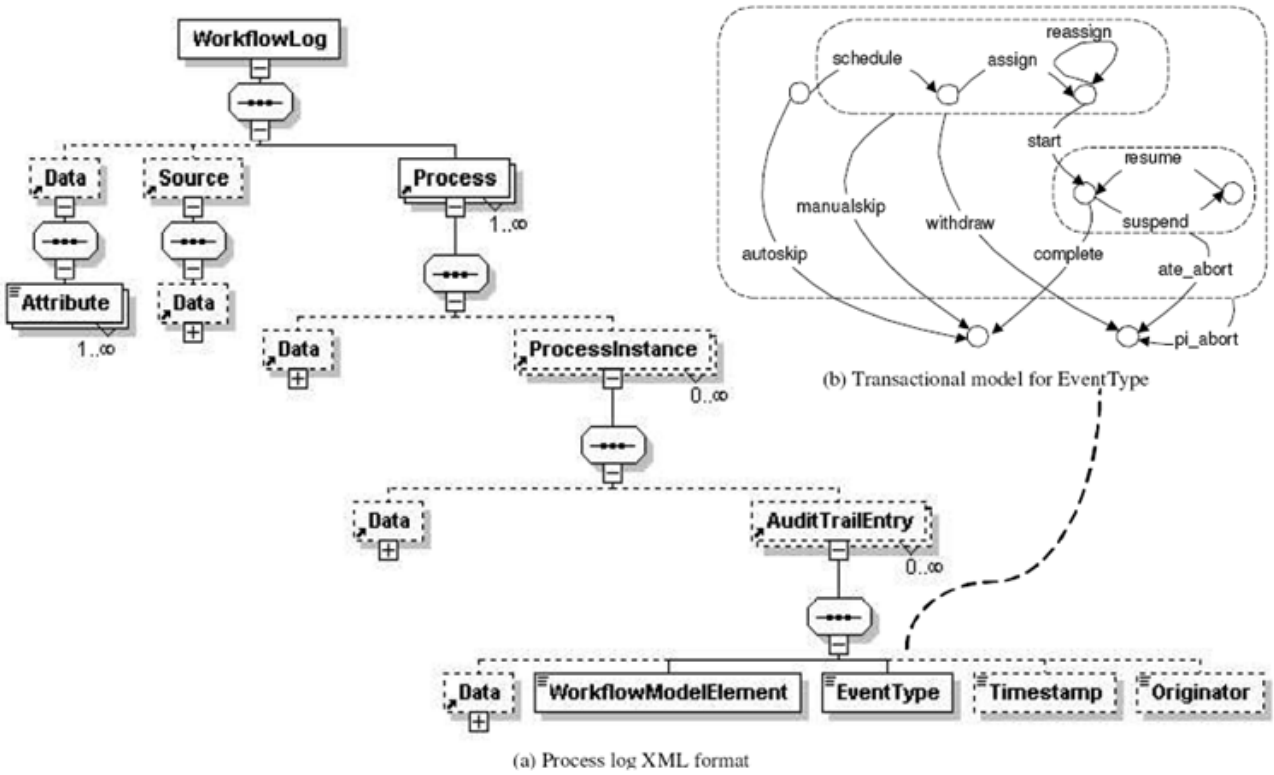

Figure A.1: The Mining XML (MXML) format (a) and transactional model (b)

system that created it), and a number of *Process* elements (at least one), in which information about a specific process is stored. A *Process* element contains a number of *Process Instance* elements and an optional *Data* element, which again can be used to store arbitrary textual data. One *Process Instance* contains one case, i.e. one execution of the *Process*. In a *Process Instance* a number of events is described in *AuditTrailEntry* elements. An *AuditTrailEntry* consists of a *WorkflowModelElement* (an activity or task ID), an *EventType*, a (optional) *Timestamp* and an (optional) *Originator* (a person performing the event). Furthermore, additional information about the event may be stored in *Data* elements.

There can exist various possible *EventTypes* in an event log. In figure A.1(b) the different event-types that can occur for an activity are shown, as well as the possible orderings in which they can appear for one activity. An activity starts by being *scheduled* or by being skipped automatically (*autoskip*). When an activity has been scheduled, the control over that activity is put into the information system, which can then *assign* the activity to someone or to a group of persons. It is possible to *reassign* an assigned activity, which can be done by the information system, or by a user. A user can then *start* working on an activity that was assigned to him, or can decide to *withdraw* the activity or skip it (*manualskip*). This can be done before the activity was assigned even. A person that is working on an activity can *suspend* and *resume* it several times, but in the end he either has to *complete* or abort (*ate_abort*) it. An activity can get aborted (*pi_abort*) during its entire life cycle.
Some information systems do not keep track of the type of event that occurred though. In such cases, each event is said to be of type *normal*.

Appendix B

Sequence diagrams

A sequence diagram is a Unified Modeling Language (UML) diagram. UML is the standard notation for modeling software-intensive systems [18]. UML sequence diagrams are used to model the dynamic behavior of systems, i.e. the behavior of a system over time. A sequence diagram shows the sequence of messages, which are exchanged between the roles that implement the behavior of a system, arranged in time. It visualizes the flow of control across the objects that collaborate in the context of a scenario.

A sequence diagram is a chart which has two dimensions. The vertical dimension is the time dimension (time increases from top to bottom) and the horizontal dimension shows the classifier roles that represent individual objects in the collaboration. Classifier roles include classes, components and nodes. Each classifier role is shown as a lifeline - i.e. a vertical line that represents the role over time. During the time an object exists, the lifeline is depicted as a dashed line. During the time an object is active, the lifeline is drawn as a double line. A message between two classifier roles is shown as an arrow from the lifeline of the source object to that of the destination object.

In figure B.1, an example of a sequence diagram is shown. This sequence diagram displays a scenario of possible behavior of a bank system after the request of a customer to get the balance of his bank account. Classifier roles in this example are *Customer*, *Bank*, *Account Ledger* and *Account*. After the request of the customer to the bank, the bank becomes active. The bank requests the account of the customer from the account ledger, which becomes active and sends the account to the bank. The bank then gets the balance from the account, which it then sends to the customer, who can examine it. After having handled the request, the bank stops being active.

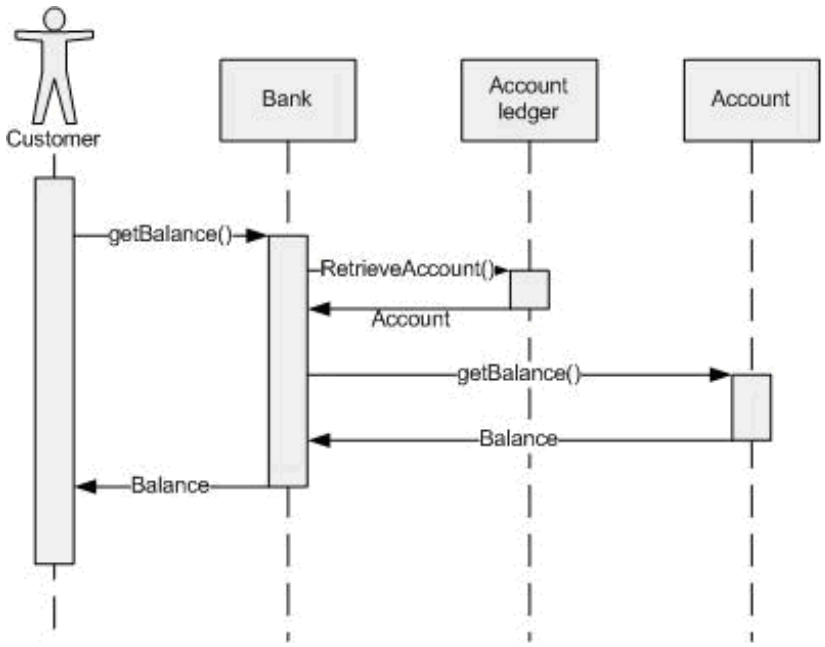

Figure B.1: Sequence diagram of a process at a bank

Appendix C

Motivations

It is not possible for us to implement all the functionality as described in chapter 3 of this document. That is why decisions had to be made about which functionality to implement and which not. These decisions were made, based on a contemplation between the Pros and the Cons of each type of functionality. In the table below the functionality has been listed, guided with a motivation why it should or should not be implemented.

Functionality:	**Motivation:**
Key Performance Indicators	The most relevant KPIs, which can be distilled from most event logs will be implemented. Adding more KPIs will make the functionality more useful, but will also result in an increase in the amount of effort needed. Furthermore, extra restrictions on the log may be required to be able to derive the values of these additional KPIs.
Using Petri net as UI	A process model gives a clear overview of the process. The user can easily relate this model to the process, and the placing of performance information herein will thus make sense to the user. Another advantage is that bottlenecks can be detected at a glance. Next to this, ProM its greatest strength is deriving process models from event logs. Making use of the process model as UI thus fits within the concept of ProM. Petri nets are chosen because all other process models can be converted to Petri nets in ProM.
Time in between two transitions	This functionality increases the flexibility of the performance analysis, because it can be used for the user to derive the values of additional time-related performance indicators.

Functionality:	**Motivation:**
Graphs and Dimensions	Graphs and tables increase insight into the performance of processes, and the causes of extreme values in KPIs. This functionality can become very complex however, and would probably require a huge effort. Other functionality that can assist in cause detection will be implemented instead.
Statistical distributions of KPIs	Increases insight into how many and which process instances cause extreme behavior. We will only state the average, minimum, maximum and standard deviation of time-related KPIs, and the averages of the fastest, slowest, and normal cases, where the user can determine which percentage of cases is fast, slow or normal. If a statistical distribution is required, then the user can make use of exporting to comma-separated lists. These lists can be used by other tools, for instance Statgraphics, to derive the statistical distribution. The functionality that will be implemented is therefore considered to be sufficient.
Routings	Probabilities will be shown at XOR-splits within the Petri net, so the routings that are followed by cases can easily be determined.
Views	When there are many KPIs to monitor, then the use of views can be very useful, but in our situation it is not worth the burden.
Displaying case properties responsible for choice at XOR-splits	This can be very useful for detection of the root causes of problems, but it requires a large effort and goes beyond performance analysis. It may be added to the ProM framework in the future though.
Reporting and exporting functionality	PDF and HTML reports can be very useful, but this functionality is not essential and costs much time to implement. Exporting performance information to comma-separated lists is deemed to be more useful and will be implemented. It can be used to export performance information which can then be used in other tools (e.g. to get a statistical distribution).

Functionality:	**Motivation:**
Sequence, and Pattern Diagrams	Sequence and pattern diagrams provide insight in how work is normally transferred within business processes, but also about which behavior is rare. What is more, these diagrams, and the additional information (e.g. average throughput time) can assist in detecting the causes of rare or unwanted behavior. This information may be used to prevent such behavior from occurring in the future.
Dotted charts	Dotted charts can give the user an idea on how data-elements are related, and can therefore help to discover the causes of certain extreme behavior. Furthermore, this functionality is unique, it is not present in any other tool. However, these charts can be quite useless at times as well and this functionality, of course, also requires some effort to implement. That is why this functionality will not be implemented.

Appendix D

User Requirements

In this appendix the original requirements placed upon the Process Performance Analysis functionality are listed along with their priority. Priorities have been assigned to requirements based on the relevance of the functionality. The following priority levels exist:
1 : the highest priority level, requirements of this priority level must be met.
2 : the middle priority level, requirements of this priority level should be met.
3 : the lowest priority level, requirements of this priority level may be met.

Functional requirements (URCAR):
Performance Analysis with process model functionality[1]:

Number	**Requirement**	**Priority**
URCAR 1	The input Petri net must be used as a user interface.	1
URCAR 2	The user can select a place in the Petri net.	1
URCAR 3	The user can select up to two transitions in the Petri net.	1
URCAR 4	The user can view the throughput time of the process.	1
URCAR 5	The user can view the arrival rate of the process.	1
URCAR 6	The user can view the total number of cases in the event log.	1
URCAR 7	The user can view the waiting time of each place in the Petri net	1
URCAR 8	The user can view the synchronization time of each place in the Petri net.	1
URCAR 9	The user can view the sojourn time of each place in the Petri net.	1
URCAR 10	The user can view the total number of measurements taken for each place.	1
URCAR 11	The user can view the arrival rate of each place.	1
URCAR 12	The user can see in the Petri net which places are considered to be bottlenecks.	1

[1]This functionality is described in more detail in section 4.2

Number	Requirement	Priority
URCAR 13	The user can view the the time it takes cases to get from one selected transition to another.	1
URCAR 14	The user can view the waiting time of each activity that appears in the log. If no exact values are recorded, then bound values will be displayed instead (if possible).	1
URCAR 15	The user can view the execution time of each activity that appears in the log. If no exact values are recorded, then bound values will be displayed instead (if possible).	1
URCAR 16	The user can view the sojourn time of each activity that appears in the log. If no exact values are recorded, then bound values will be displayed instead (if possible).	1
URCAR 17	The user can view for each time-related KPIs (e.g. throughput time of the process, waiting time of a place): the average, minimum, maximum, and standard deviation, as well as the average of the $x\%$ of measurements that are counted as fast (i.e. have a low value), the average of the $y\%$ of measurements that are counted as being slow (i.e. have a high value), and the average of the $100 - x - y\%$ of measurements that are counted as being normal.	1
URCAR 18	The user can adjust the percentage of measurements that are counted as fast and the percentage of measurements that are counted as slow, for each time-related KPI.	2
URCAR 19	The user can change the classifications which define at which average waiting time a place is considered to be a bottleneck.	1
URCAR 20	The user can view the routing probabilities per branch at each XOR-split in the Petri net.	1
URCAR 21	The user can select process instances.	1
URCAR 22	The user can view the values of KPIs, that are based on only the selected process instances.	1
URCAR 23	The user can export the values of time-related KPIs to comma-delimited text-files.	3

Performance Sequence diagram functionality [2]:

Number	Requirement	Priority
URCAR 24	The user can view the communication behavior between instances of data-elements in a sequence diagram.	1
URCAR 25	The user can set the sequence diagrams to real time or to logic steps.	3
URCAR 26	The user can view the throughput time of each sequence.	1
URCAR 27	The user can view the time that is spend in blocks, i.e. in periods of activity of a data-element instance, for each sequence.	2

Number	Requirement	Priority
URCAR 28	The user can view the time that is spend in arrows, i.e. in periods in which there is a transfer of work, for each sequence.	2
URCAR 29	The user can view all patterns existing in a sequence diagram in a separate diagram.	1
URCAR 30	The user can view the number of times that each pattern occurs within a sequence diagram.	1
URCAR 31	The user can view the average, minimum, and maximum throughput time for each pattern and the standard deviation.	1
URCAR 32	The user can view the average time that is spend in blocks, i.e. periods of activity of a data-element instance, for each pattern.	2
URCAR 33	The user can view the average time that is spend in arrows, i.e. periods in which there is a transfer of work, for each pattern.	2

Restriction requirements (URCOR)

Interface requirements:

Number	Requirement	Priority
URCOR 1	The Process Performance Analysis functionality must be added to the ProM framework in form of plug-ins.	1
URCOR 2	A timed event log in MXML-format is required as input for both performance analysis plug-ins.	1
URCOR 3	A Petri net is required as input for 'performance analysis with Petri net'.	1
URCOR 4	Values of time-related KPIs can be exported to comma-separated	3

Quality requirements:

Number	Requirement	Priority
URCOR 5	The source-code should conform to the coding standards in use at the ProM-group	1
URCOR 6	The plug-ins should be accompanied by suitable user guides, integrated in the Plug-in Help System of ProM	1

Appendix E

Use cases

In this appendix the use cases that were developed for the two plug-ins will be described. These use cases have been used to set up tests for the plug-ins.

E.1 Performance Analysis with Petri net

In figure E.1 a diagram containing the use cases that were defined for the *Performance Analysis with Petri net* plug-in is depicted. Only one very general actor is defined: *user*. This can be anyone that uses the plug-in, e.g. managers, process analysts, students. In the remainder of this section the defined use cases will be described in detail.

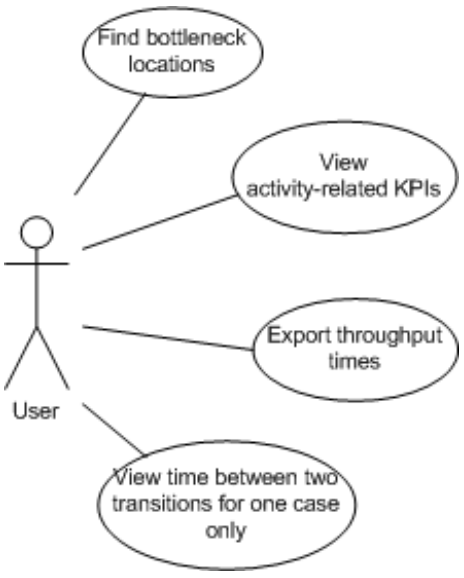

Figure E.1: Use cases *Performance Analysis with Petri net*

E.1.1 Find bottleneck locations

Brief Description:
The plug-in can assist in finding bottleneck locations within the process. A bottleneck location in a Petri net is a place with a relatively high average waiting time.

Preconditions:
A Petri net, a log, and a relation between event classes in the log and transitions in the Petri net can be accessed through the ProM framework.

Main Path:
1. User starts the plug-in.
2. Plug-in requests the user to set bounds and colors for the low, medium, and high waiting time level of places.
3. User sets bounds and colors for low, medium, and high waiting time level and in this manner defines when a place is considered to be a bottleneck (i.e. has a high waiting time level).
4. Plug-in requests the user to set the non-conformance option that defines how the plug-in should deal with failing transitions (see also section 4.2.6).
5. User sets the non-conformance option.
6. Plug-in provides a visualization of the input Petri net, but enhanced in such a way that places are colored according to their average waiting time.

Postcondition:
A Petri net visualization is available, in which places are colored according to their waiting time level. Places with a high waiting time level are considered to be bottlenecks.

Alternative Paths:

- At step 3, instead of setting bounds values and colors, the user can let the plug-in choose the bound values and colors. Approximately one third of the places is contained in each waiting time level when the plug-in uses these 'standard settings'. This does not affect the remainder of the path.
- After step 6, the user has the possibility to adjust the bounds and colors of the waiting time levels. After the settings have been adjusted, the visualization is updated.
- In the visualization that results from following the main path, also probabilities at XOR-splits are shown.

E.1.2 View activity-related KPIs

Brief Description:
The values of the activity-related KPIs of each activity can be viewed. The activity related KPIs are defined in section 4.2.5.

Preconditions:
A Petri net, a log, and a relation between event classes in the log and transitions in the Petri net can be accessed through the ProM framework.

Main Path:
1. User starts the plug-in.
2. Plug-in requests the user to set the non-conformance option that defines how the plug-in should deal with failing transitions.
3. User sets the non-conformance option.
4. Plug-in provides a way to select activities.
5. User selects the activity (s)he is interested in.
6. Plug-in displays the activity-related KPIs of the selected activity.

Postcondition:
Activity-related KPIs of the selected activity are displayed.

Alternative Paths:

- Next to the values of activity-related KPIs, also the values of process-related KPIs (defined in 4.2.2), place-related KPIs (defined in 4.2.3), and the time between two transitions (defined in 4.2.4) can be viewed.

  The values of process-related KPIs are provided after step 3 in the main path, at the same time as the plug-in provides a way to select activities. Values of place-related KPIs and the time between two transitions can be viewed in a similar manner as the values of activity-related KPIs. At step 4 of the main path, the plug-in provides a way to select places and transitions. When the user selects a place, the values of the place-related KPIs of the selected place are displayed. When the user selects two transitions, the time between these two transitions is displayed.

E.1.3 View time between two transitions for one case only

Brief Description:
Instead of the statistical values for the time between two transitions, measured over all replayed cases in the log, also the time between two transitions for one case only can be viewed.

Preconditions:
A Petri net, a log, and a relation between event classes in the log and transitions in the Petri net can be accessed through the ProM framework.

Main Path:
1. User starts the plug-in.
2. Plug-in requests the user to set the non-conformance option that defines how the plug-in should deal with failing transitions.
3. User sets the non-conformance option.

4. Plug-in provides a way to select two transitions.
5. User selects the transitions (s)he is interested in.
6. Plug-in provides a way to select cases.
7. User selects a case.
8. The plug-in shows the time between the two selected transitions for the selected case.

Postcondition:
The time between the selected transitions for the selected case is displayed, provided that both transitions fire at least once during replay of the case.

Alternative Paths:

- At step 7, instead of one process instance, the user can also select multiple process instances. All KPIs are based only on the selected process instance(s).
- Activity-related KPIs, process-related KPIs, and place-related KPIs can also be based on one or multiple of the registered cases in the log.

E.1.4 Export throughput times

Brief Description:
The throughput times of all cases can be exported to a comma-separated text-file.

Preconditions:
A Petri net, a log, and a relation between event classes in the log and transitions in the Petri net can be accessed through the ProM framework.

Main Path:
1. User starts the plug-in.
2. Plug-in requests the user to set the non-conformance option that defines how the plug-in should deal with failing transitions.
3. User sets the non-conformance option.
4. Plug-in shows process-related KPIs, including the statistical values for the throughput time.
5. User requests to export the throughput times.
6. Plug-in asks for a filename.
7. The user provides filename.
8. Plug-in creates a comma-separated file with the specified name, which contains the throughput times of all cases.

Postcondition:
A comma-separated file exists, which contains the throughput times of all cases.

Alternative Paths:

- The values of all time-related KPIs can be exported to comma-separated text-files in a similar manner.

E.2 Performance Sequence Diagrams Analysis

In figure E.2 a diagram containing the use cases that were defined for the *Performance Sequence Diagram Analysis* plug-in is depicted. Again only one very general actor is defined: *user*. In the remainder of this section the defined use cases will be described in detail.

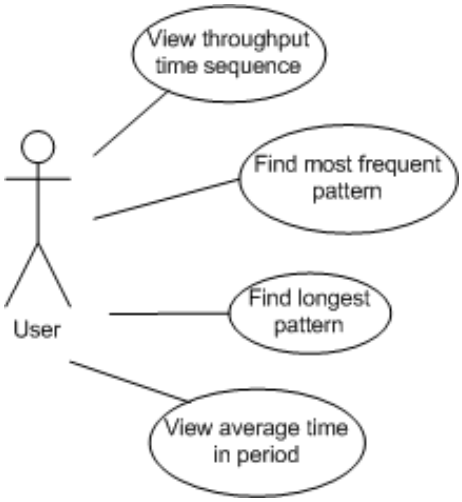

Figure E.2: Use cases *Performance Sequence Diagram Analysis*

E.2.1 View throughput time sequence

Brief Description:
The throughput time of all derived sequences can be viewed.

Preconditions:
A log in which events are registered is accessable through the ProM framework.

Main Path:
1. User starts the plug-in.
2. Plug-in provides a list of the available data-elements that appear on the audit trail level in the log and requests the user to choose one.
3. User selects a data-element.
4. Plug-in shows the sequence diagram.
5. User selects the sequence (s)he is interested in.
6. Plug-in displays the throughput time of the selected sequence.

Postcondition:
The throughput time of the selected sequence is displayed.

E.2.2 Find most frequent pattern

Brief Description:
The most frequent pattern is the pattern that represents the most sequences, i.e. the pattern that represents the most common behavior.

Preconditions:
A log in which events are registered is accessable through the ProM framework.

Main Path:
1. User starts the plug-in.
2. Plug-in provides a list of the available data-elements that appear on the audit trail level in the log and requests the user to choose one.
3. User selects a data-element.
4. Plug-in asks which equivalence type to use: flexible-equivalence or strict-equivalence.
5. User selects an equivalence type.
6. Plug-in shows the pattern diagram and the frequency of each pattern. The first pattern is the most frequent one.

Postcondition:
Plug-in shows the pattern diagram and the frequency of each pattern. The pattern at the top of the diagram is the most frequent one. When there are a number, say x, most frequent patterns, then of course the first x patterns in the diagram are the most frequent ones.

E.2.3 Find longest pattern

Brief Description:
The longest pattern is the pattern with the highest average throughput time.

Preconditions:
A log in which events are registered is accessable through the ProM framework.

Main Path:
1. User starts the plug-in.
2. Plug-in provides a list of the available data-elements that appear on the audit trail level in the log and requests the user to choose one.
3. User selects a data-element.
4. Plug-in asks which equivalence type to use: flexible-equivalence or strict-equivalence.
5. User selects an equivalence type.
6. Plug-in shows the pattern diagram and a list of the statistical values of the throughput time of each pattern.

Postcondition:
The pattern with the highest average throughput time is the longest pattern. This can be more than one pattern.

E.2.4 View average time in period

Brief Description:
View the average time-length of a period of activity of a data-element instance in a pattern.

Preconditions:
A log in which events are registered is accessable through the ProM framework.

Main Path:
1. User starts the plug-in.
2. Plug-in provides a list of the available data-elements that appear on the audit trail level in the log and requests the user to choose one.
3. User selects a data-element.
4. Plug-in asks which equivalence type to use: flexible-equivalence or strict-equivalence.
5. User selects an equivalence type.
6. Plug-in shows the pattern diagram.
7. User selects a block, i.e. a period in the pattern diagram.
8. Plug-in shows the average duration of the selected period.

Postcondition:
The average duration of the selected period of activity is shown.

Alternative Paths:

- At step 7, instead of selecting a block, an arrow could have been selected, after which the source data-element instance, the destination data-element instance, and the average time between beginning and end of the arrow is shown.

Appendix F

Prototypes

In this appendix, prototypes for the Performance Analysis plug-ins are described. In section F.1, prototypes for the *Performance Analysis with Petri net* plug-in are discussed. In section F.2, prototypes for the *Performance Sequence Diagram Analysis* plug-in are discussed.

F.1 Performance Analysis with Petri net

Before the values of the various KPIs can be obtained by means of the log replay method, some options need to be set. Therefore, at start-up of the plug-in, a window should appear in which these options can be set. In figure F.1 a prototype of such a window is shown. At the top of the window, one of the *non-conformance options* must be selected. The *non-conformance options* define what the plug-in should do with measurements taken during replay of cases that do not fit in the Petri net. Furthermore, waiting time level settings can

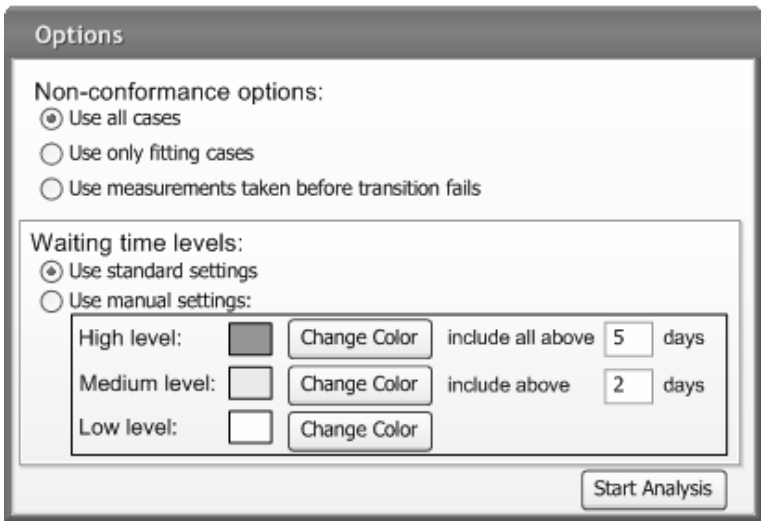

Figure F.1: Prototype of options screen

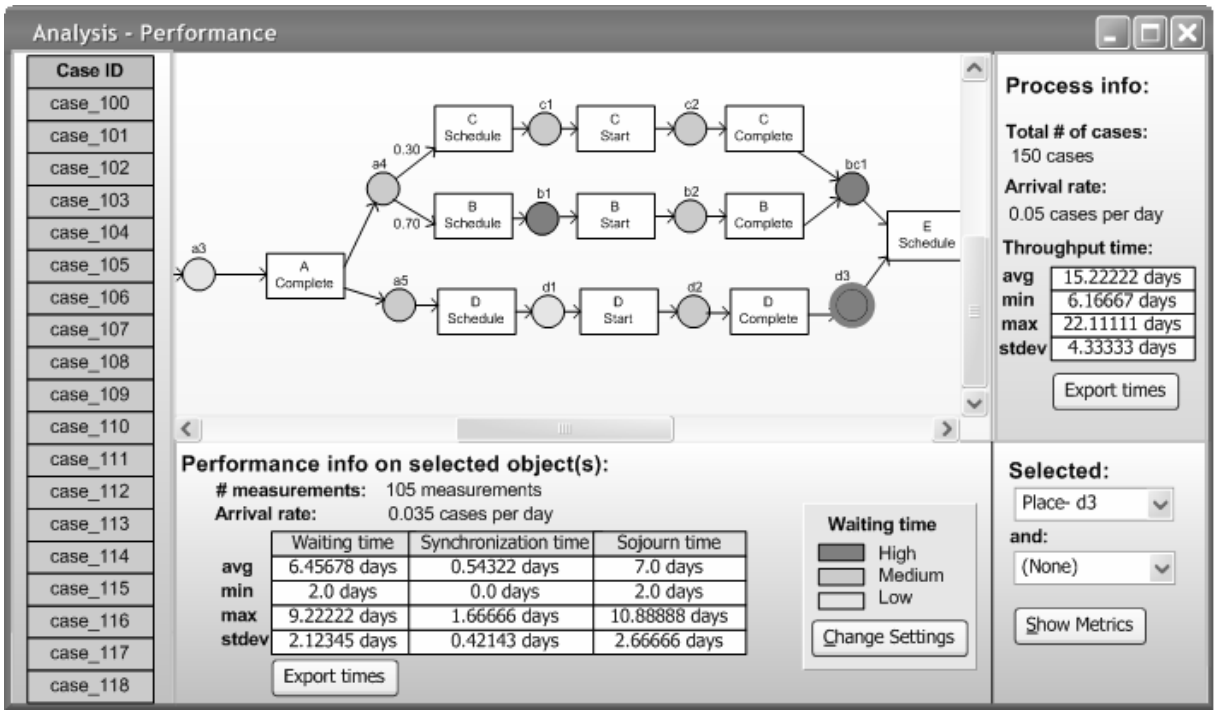

Figure F.2: Prototype of main GUI

be set in this window. The user can choose between using standard waiting time level settings or to manually specify these settings. When the latter option is chosen, the user can set the bound values and the colors of each waiting time level.

In figure F.2 a prototype is shown of the GUI that may appear after log replay is completed. In this GUI, the input Petri net is displayed, but with the places colored according to their average waiting time level and probabilities shown at XOR-splits. Below the Petri net a legend is depicted which describes which color belongs to which level. The settings for these waiting time levels can be changed by clicking on the *Change Settings* button below the legend, after which a window (similar to the initial options window) pops-up in which the current settings can be adjusted.

Left of the Petri net, the identifier names of the process instances that are contained in the input log are shown in a table. These process instances can be selected or deselected. The displayed performance information should be based only on the selected process instances. On the right side of the Petri net, information concerning the whole process is displayed, such as the arrival rate of cases and the average throughput time.

Places and transitions can be selected in the selection boxes on the lower right side of the window, but can also be selected within the Petri net itself. In the example in figure F.2 the place $d3$ is selected, and below the Petri net, performance information (e.g. waiting time,

synchronization time, sojourn time) of the selected place is shown. When two transitions are selected instead, the time cases spend in between these two transitions is shown. When one transition is selected, the performance information of the activity that is (referred to by the event class that is) related to the transition is shown.

F.2 Performance Sequence Diagram Analysis

A prototype of the GUI of the *Performance Sequence Diagram Analysis* plug-in is displayed in figure F.3. On the left side of this window, the (initial) options can be set and/or changed. The options available here are the component type (i.e. data-element) on which the plug-in should focus and the type of equivalence that should be used when deriving patterns. After the options have been set, the user can press the *Show diagram* button. The sequence and pattern diagram are derived then and can be viewed by selecting the corresponding tab (at the top of the window). In figure F.3 the sequence diagram is displayed. It is depicted on the right side of the window. Information about each sequence (throughput time of the sequence, name of the process instance that is represented by the sequence) can be viewed by moving the mouse cursor over the sequence. The information will then be displayed in a tooltip. Specific information about the arrows (duration of arrow) and the periods/blocks (duration of block) of a sequence can be summoned by moving the mouse cursor over the object (arrow or block) of interest.

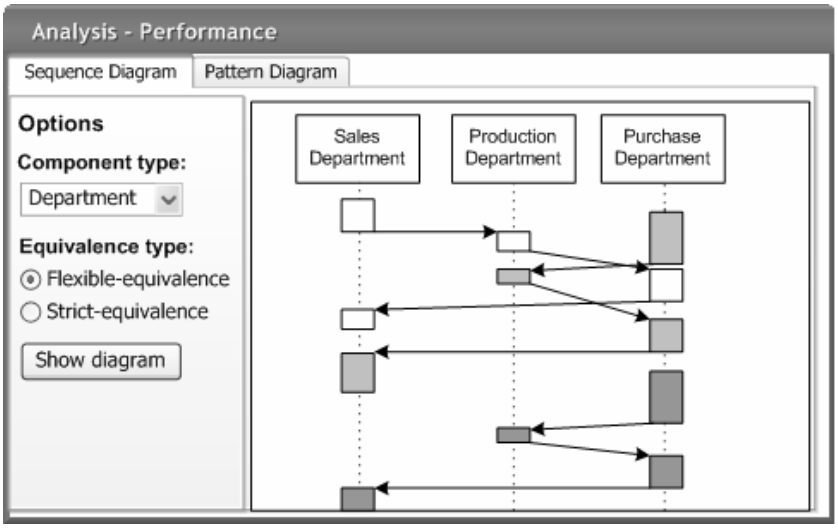

Figure F.3: Prototype: Performance Analysis Sequence Diagram

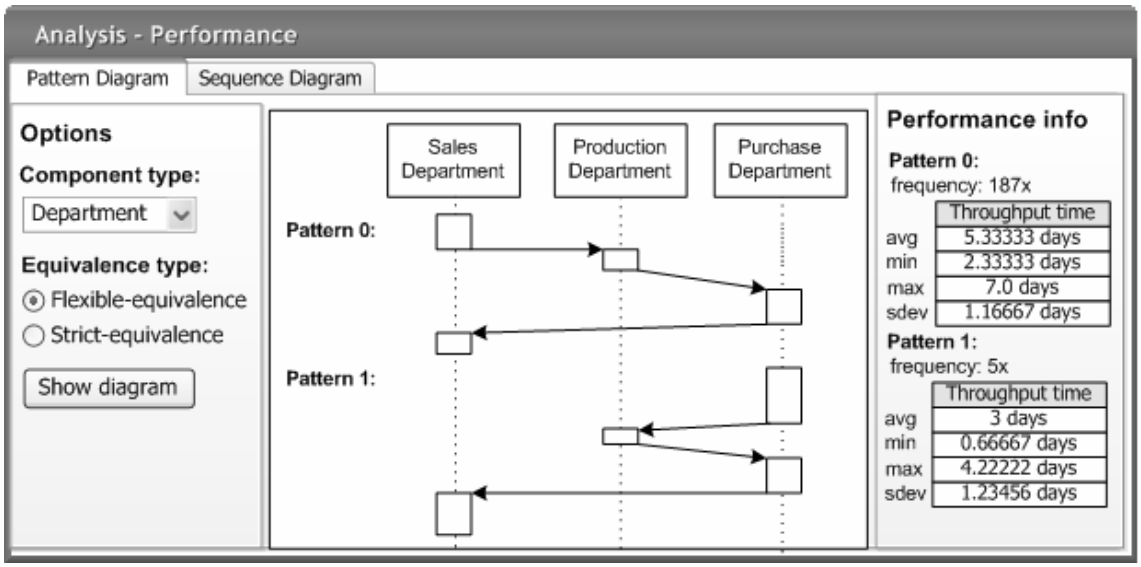

Figure F.4: Prototype: Performance Analysis Pattern Diagram

In figure F.4 an example of the GUI is shown where the pattern diagram has been selected. The pattern diagram is displayed in the center of the window. On the right side of the pattern diagram, the statistical values of the throughput time of each pattern and the frequency of sequences that follow this pattern are shown. Specific information of the arrows (average duration of arrow) and the periods/blocks (average duration of block) of a pattern can be summoned by moving the mouse cursor over the object of interest.

Appendix G

Test cases

In this appendix test cases will be described. In these test cases we will verify whether the use cases, which were defined in appendix E, can be performed with the implemented plug-ins.

G.1 Performance Analysis with Petri net

All use cases of the *Performance Analysis with Petri net* plug-in have the same precondition: A Petri net, a log, and a relation between event classes in the log and transitions in the Petri net can be accessed through the ProM framework. We assume the precondition is met at the start of each test case. In the remainder of this section we will address the use cases of appendix E.1 one by one, to verify whether they can be performed with the *Performance Analysis with Petri net* plug-in.

G.1.1 Find bottleneck locations

1. *User starts the plug-in.*
   Start the plug-in by first selecting *Analysis* from the menu bar of the ProM framework and then selecting the *Performance Analysis with Petri net* item from the menu that appears.

2. *Plug-in asks the user to set bounds and colors for the low, medium, and high waiting time level of places.*
   The window as depicted in figure G.1a appears. In this window you can choose between *Use auto bottleneck settings* and *Use manualbottleneck settings*. When the latter option is chosen, the window will show options to define waiting time levels (see figure G.1b).

3. *User sets bounds and colors for low, medium, and high waiting time level and in this manner defines when a place is considered to be a bottleneck.*

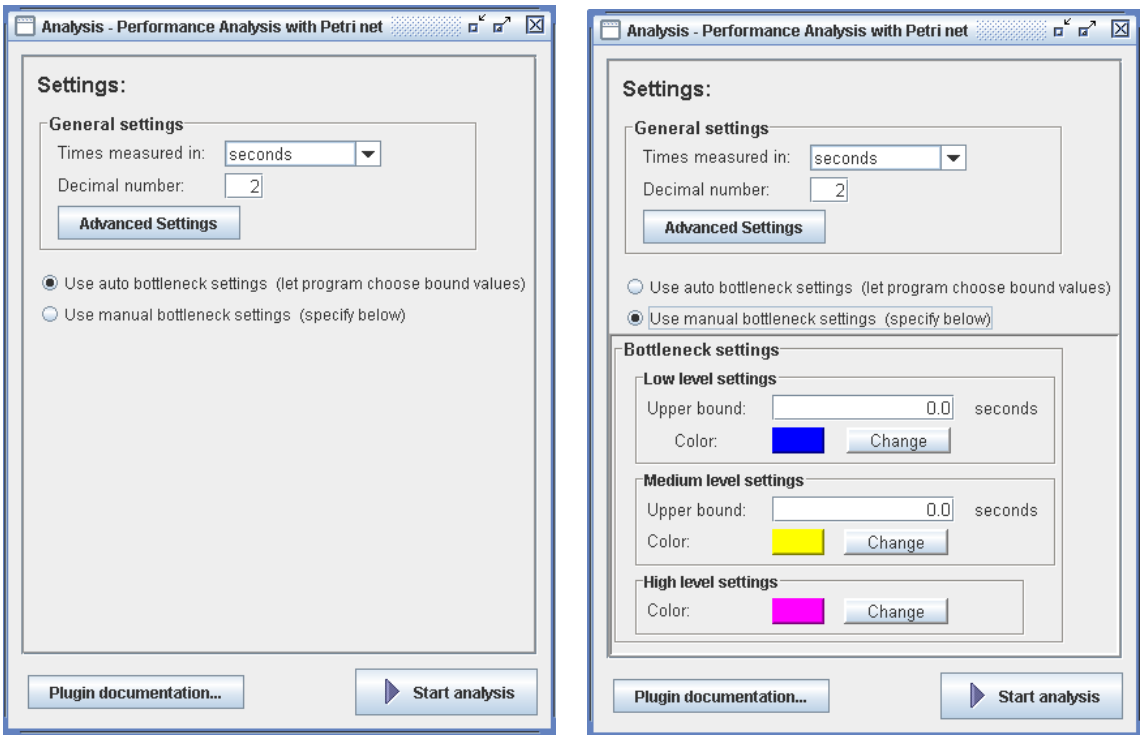

Figure G.1: **(a)** Settings window (*standard*) **(b)** Settings window (*manual bottleneck settings*)

Below the *General settings*, select *Use manual settings*. At *Low level settings*, set the *upper bound* to the wanted value. In our example, we will set this value to 300 seconds. All places with an average waiting time equal to or below this value will then be given the color of the low level after log replay. At *Medium level settings*, set the *upper bound* to the wanted value. In our example we will use 1200 seconds. All places with an average waiting time equal to or below 1200 seconds, and above 300 seconds, will then be given the color of the medium level. All places with an average waiting time above 1200 seconds, will be given the color of the high level. For this test case, we will keep the standard colors (these can be changed by pressing the *Change* button behind the color that you wish to change).

Note that instead of the *Use manual settings* option, also the *Use auto settings* option could have been chosen. In that case, the plug-in would set the bound values for the waiting time levels in such a manner that after log replay approximately one third of the places would belong to each of the waiting time levels.

In the windows of figure G.1 also general options can be set. The following general options are available:

- *Time measured in*: the time unit in which the values of time-related KPIs are

given. The standard value of this option is *seconds*, other possible values are: *milliseconds, minutes, hours, days, weeks, months (30 days)* and *years (365 days)*.

- *Decimal number*: the number of decimal places that the values of KPIs can have at most. The standard value of this option is 2.

In this test case, we will use the standard general settings.

4. *Plug-in requests the user to set the non-conformance option that defines how the plug-in should deal with failing transitions.*
   The plug-in uses standard non-conformance options, but these can be viewed and changed through the window that appears after pressing the *Advanced Settings* button.

5. *User sets non-conformance option.*
   Press the *Advanced Settings* button. The window as depicted in figure G.2 will appear.

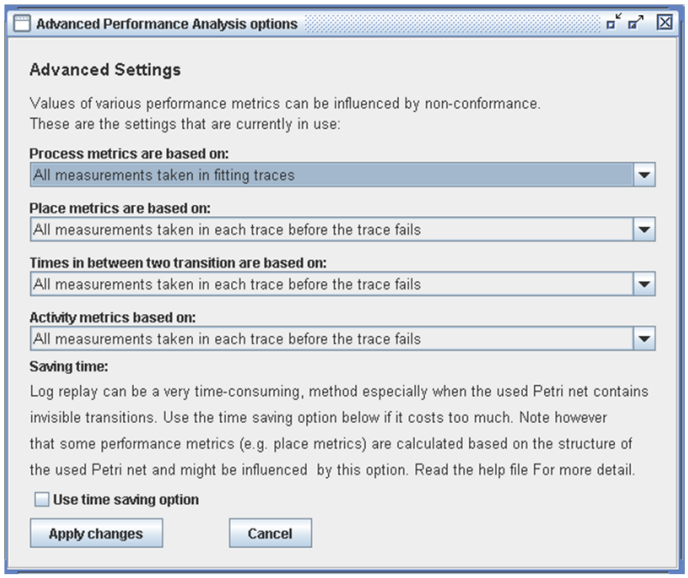

Figure G.2: Screenshot: Advanced settings window

Here you can specify how the plug-in should deal with measurements that are taken in those cases that do not 'fit' in the Petri net (i.e. where at least one transition fails during log replay). Instead of one general option, the plug-in has a different option for each type of KPI (where the types of KPIs are: process-related, place-related, activity-related, and time between two transitions). Since bottlenecks are places, we focus on the settings for place-related KPIs. For this test case, we select the following option: *All measurements taken in each trace before the trace fails.*

The idea behind this option is that, when a case does not fit in the Petri net, all measurements that are taken during replay of the case before a deviation (i.e. a failing transition) occurs are not influenced by this deviation. Those measurements can then be used normally in calculations. Take for instance the average waiting time of a place. This time is calculated based on the waiting times of the tokens that 'visit' the place during log replay. When during replay of a case a token is added to and removed from the place before a transition fails execution, then the waiting time of that token can be included in the calculation of the average waiting time. If a token is (added to or) removed from the place after a transition fails, then the waiting time of that token should not be used for the calculation of the average waiting time, because it may have been influenced by the deviation that occurred before.

Note that selecting another option here does not influence the remainder of this test case, but only the results. Proceed by pressing the *Apply changes* button. The previous window (figure G.1a) is then visible again. In this window, press the *Start analysis* button to continue.

6. *Plug-in provides a visualization of the input Petri net, but enhanced in such a way that places are colored according to their average waiting time.*
   In the center of the window that appears (similar to the one depicted in G.3), a visualization of the Petri net is shown. In this Petri net visualization the places are colored according to the waiting time levels. Below the Petri net a legend is shown, which shows which color corresponds to which level.

   Bounds and colors of the levels can be adjusted by pressing the *Settings* button below the legend. A window very similar to the settings window (figure G.1b) will then appear.

G.1.2 View activity-related KPIs

1. *User starts the plug-in.*
   Start the plug-in by first selecting *Analysis* from the menu bar of the ProM framework and then selecting the *Performance Analysis with Petri net* item from the menu that appears.

2. *Plug-in requests the user to set the non-conformance option that defines how the plug-in should deal with failing transitions.*
   At start-up of the plug-in the *Settings* window as depicted in figure G.1a is created. To view the non-conformance settings, press the *Advanced settings* button. The window in figure G.2 appears.

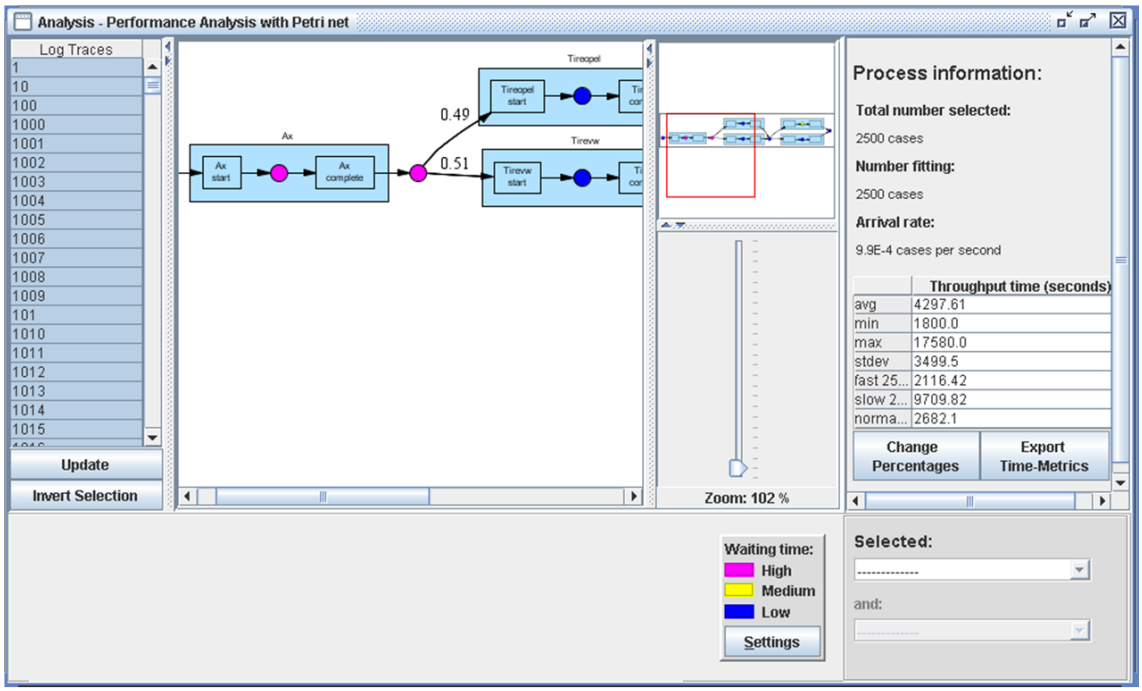

Figure G.3: Screenshot: Main window

3. *User sets non-conformance option.*
   At *Activity metrics*, select a non-conformance option. In this test case we will use the *All measurements taken in fitting traces* option. Only the measurements for activity-related KPIs that are taken during replay of cases that fit (i.e. no transition fails during log replay) are then used for calculation of the values of these KPIs. Proceed by pressing the *Apply changes* button. The previous window (figure G.1a) is then visible again.

   For this example we will use the standard general settings and auto bottleneck settings. Press the *Start analysis* button to continue.

4. *Plug-in provides a way to select activities.*
   When the analysis (i.e. the log replay method) has completed, a window similar to the one depicted in figure G.3 will appear. An activity can be selected here by selecting one of the transitions that is related to (an event class that refers to) the activity. A transition can be selected by selecting its name in the upper selectionbox on the lower right side of the window. Another way to select a transition is by 'clicking' on it in the Petri net visualization in the center of the window.

5. *User selects the activity (s)he is interested in.*
   Select one of the transitions that is related to the activity. In our example, we select

the *Ax-complete* transition.

6. *Plug-in displays the activity-related KPIs of the selected activity.*
   Below the Petri net, the activity-related KPIs of the selected activity will be displayed. In the example in figure G.4, the KPIs related to activity *Ax* are displayed. In this example bound values are given for the statistical values of the sojourn and the waiting time of this activity (bound values are colored red). The statistical values of the execution time of activity *Ax* are exact values.

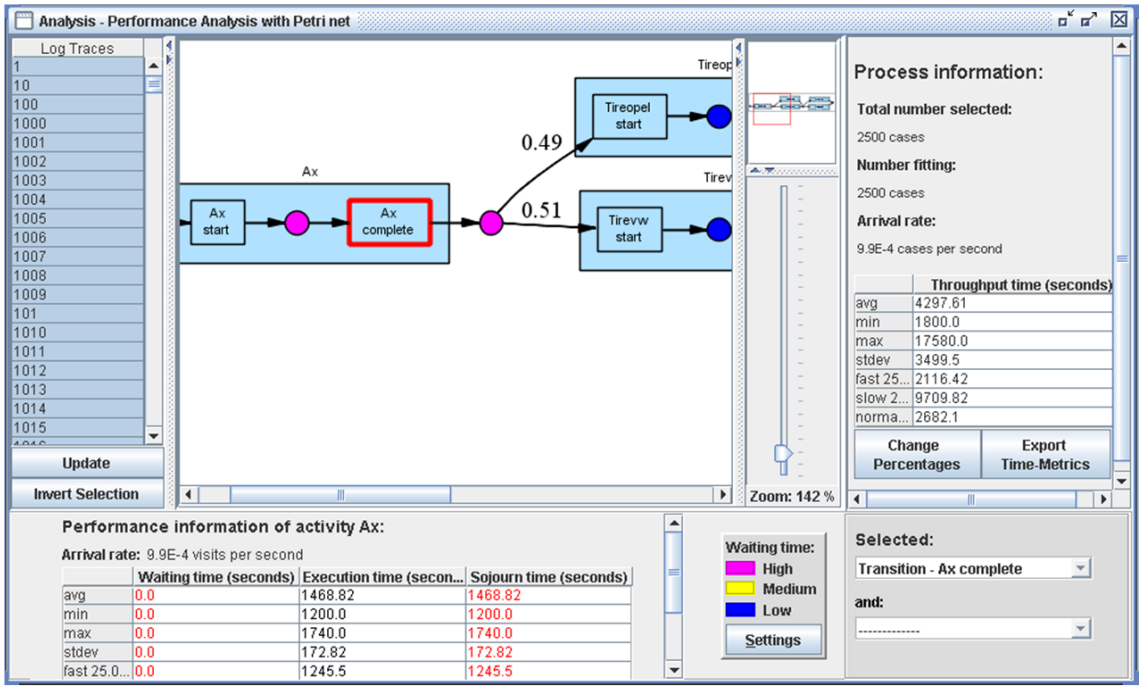

Figure G.4: Screenshot: Activity selected in main window

In a similar manner the values of place-related KPIs can be viewed (by selecting the place in the selectionbox or in the Petri net) and the time between two transitions (by selecting the transitions in the two selectionboxes or in the Petri net). The values of these KPIs are displayed below the Petri net as well.

G.1.3 View time between two transitions for one case only

The first two steps of this use case are the same as the first two steps of the previous use case. At that point the advanced settings are shown (as displayed in figure G.2).

3. *User sets non-conformance option.*
   Select the non-conformance option for *Time between two transitions*. In this test case we will set the option to *All measurements taken*. The calculations of this time will then

be based on the time between the two transitions of all cases (fitting or not) during replay of which the transitions both fire at least once. Press the *Apply changes* button to proceed. When the plug-in returns to the general options window (similar to figure G.1a), press the *Start analysis* button.

4. *Plug-in provides a way to select two transitions.*
   A window similar to the one displayed in figure G.3 appears. Transitions can be selected in the Petri net or in the selectionboxes on the lower right side of the window.

5. *User selects the transitions (s)he is interested in.*
   Select the two transitions within the Petri net or in the selectionboxes. In our example, we have chosen transition *Ax-complete* and transiton *Tireopel-start*.

6. *Plug-in provides a way to select cases.*
   On the left side of the window, the identifier names of the cases can be (de)selected in a table.

7. *User selects a case.*
   Select the name of the case in which you are interested (in our example case *1001*) and press the *Update* button below the table.

8. *The plug-in shows the time between the two selected transitions for the selected case.*
   The time between the two selected transitions for the selected case will be shown below the Petri net (see figure G.5), if such a time is recorded for the selected case.

G.1.4 Export throughput times

The first two steps of this use case are again the same as the first two steps of the previous two use cases. At that point the advanced settings are shown (as displayed in figure G.2).

3. *User sets non-conformance option.*
   Since this use case is about the throughput time, a process-related KPI, set the non-conformance option for process metrics to the required value. In our example we will use *All measurements taken*. The throughput times of all cases, fitting or not will then be used for calculation of the statistical values of the throughput time of the process.

   Press the *Apply changes* button to proceed. When the plug-in returns to the general options window (figure G.1a), press the *Start analysis* button.

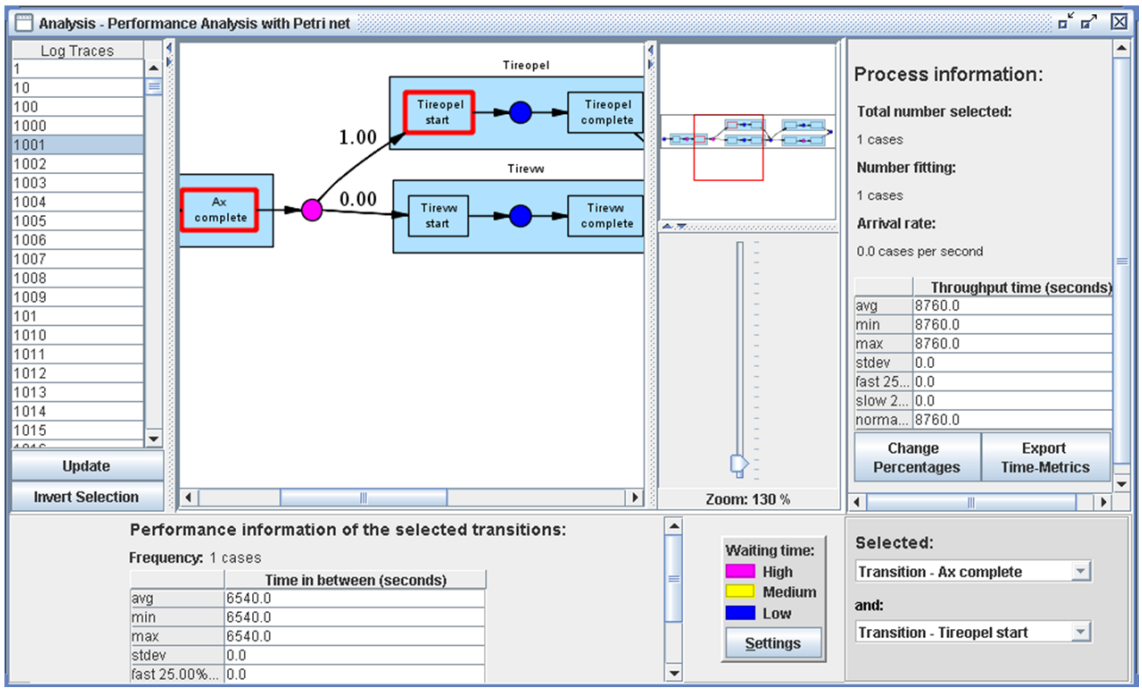

Figure G.5: Screenshot: Two transitions selected in main window

4. *Plug-in shows process-related KPIs, including the statistical values for the throughput time.*
   On the right side of the window that appears (similar to figure G.3), the process-related KPIs are depicted.

5. *User requests to export the throughput times.*
   Press the *Export time-metrics* button that is located below the table in which the statistical values of the throughput time of the process are shown.

6. *Plug-in asks for a filename.*
   The window as depicted in figure G.6 appears.

7. *The user provides filename.*
   At *File Name* fill in `file` (or another name of your liking). In this window, below the filename, also the file extension (.txt or .csv) can be set. In this example we keep the standard setting for the file extension: `.csv`. Press the *Export* button.

8. *Plug-in creates a comma-separated file, which contains the throughput times of all cases in the log.*

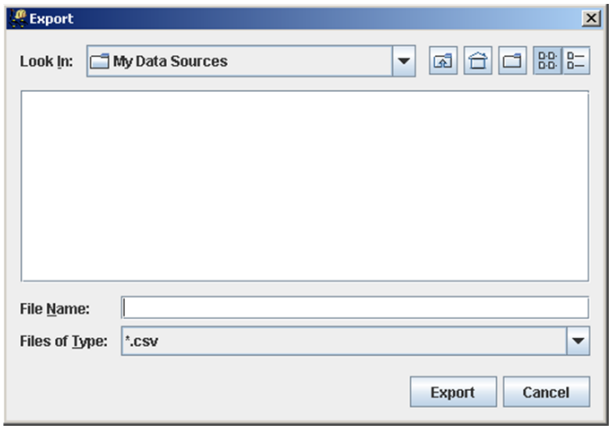

Figure G.6: Screenshot: Export window

The plug-in creates a comma-separated file with name `file`, which contains the throughput times of all cases in the log. Comma-separated text-files can be loaded into other programs, such as Microsoft Excel or Statgraphics, and can be used for further analysis.

G.2 Performance Sequence Diagram Analysis

All use cases of the *Performance Sequence Diagram Analysis* plug-in have the same precondition: A log can be accessed through the ProM framework. We assume the precondition is met at the start of each test case. In the remainder of this section we will address the use cases of appendix E.2 one by one, to verify whether they can be performed with the *Performance Sequence Diagram Analysis* plug-in.

G.2.1 View throughput time sequence

1. *User starts the plug-in.*
   Start the plug-in by first selecting *Analysis* from the menu bar of the ProM framework and then selecting the *Performance Sequence Diagram Analysis* item from the menu that appears.

2. *Plug-in provides a list of the available data-elements that appear on the audit trail level in the log and requests the user to choose one.*
   At start-up of the plug-in, the window as depicted in figure G.7 is displayed. On the left side of this window, options can be set.

3. *User selects a data-element.*
   At *Component type*, select the data-element that your are interested in. In our example

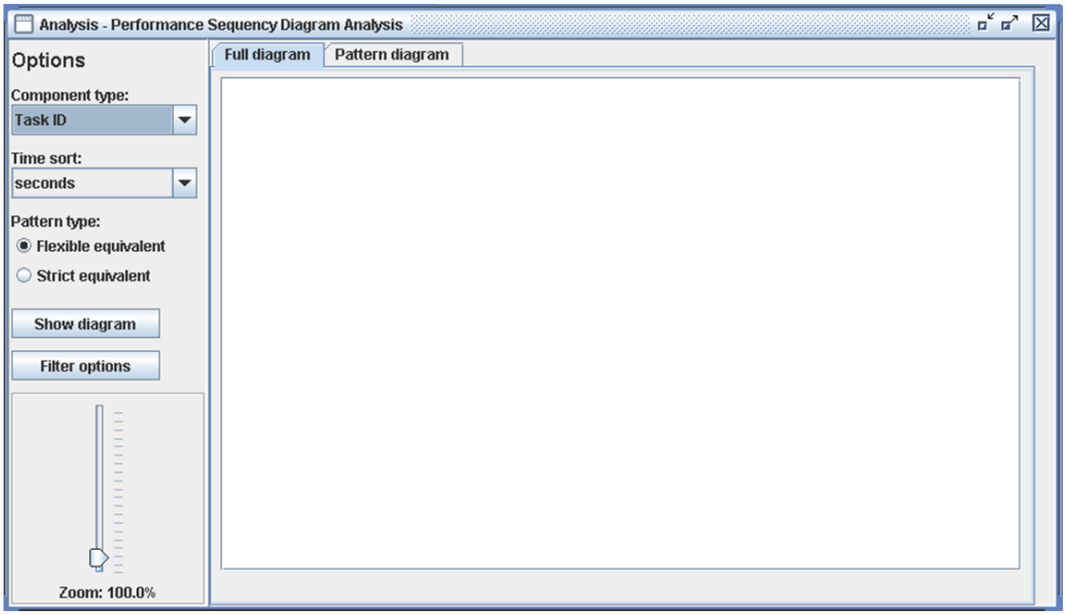

Figure G.7: Screenshot: Main window *Performance Sequence Diagram Analysis*

we will use the *Originator* data-element, which is present in most logs. After selecting the data-element, press the *Show Diagrams* button.

4. *Plug-in shows the sequence diagram.*
   On the right side of the window, the derived sequence diagram appears. See for instance figure G.8.

5. *User selects the sequence (s)he is interested in.*
   Move the mouse cursor over the sequence in which you are interested.

6. *Plug-in displays the throughput time of the selected sequence.*
   As can be seen in figure G.8, in a tooltip information about the sequence will appear. This information includes the throughput time of the sequence.

G.2.2 Find most frequent pattern

The first two steps of this use case are equal to the first two steps of the previous use case.

3. *User selects a data-element.*
   At *Component type*, select the data-element that your are interested in.

4. *Plug-in asks which equivalence type to use: flexible-equivalence or strict-equivalence.*
   On the left side of the window (as depicted in figure G.7) a *Pattern type* can be selected.

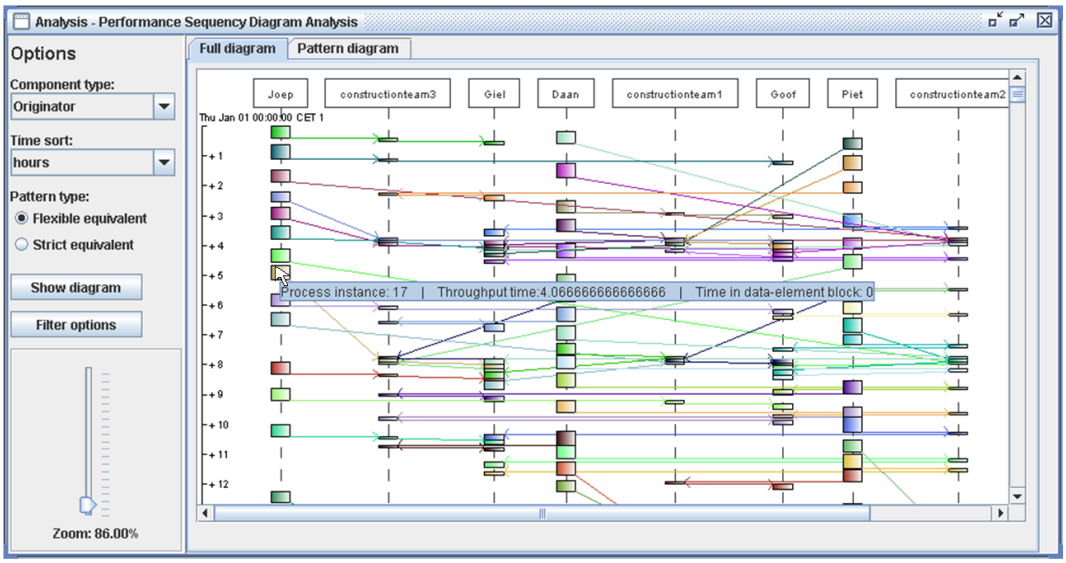

Figure G.8: Screenshot: Sequence diagram

5. *User selects an equivalence type.*
   At *Pattern type* choose between *flexible-equivalent* and *strict-equivalent*. In this example we will use *flexible-equivalent*. Press the *Show diagrams* button and select the *Pattern diagram* tab at the top of the screen.

6. *Plug-in shows the pattern diagram and the frequency of each pattern. The first pattern is the most frequent one.*
   The window will now look similar to the one depicted in figure G.9. In the centre of the window, the pattern diagram is shown. The top pattern is the most frequent one. In the example in figure G.9, the top pattern has a frequency of 153, i.e. there exist 153 sequences in the sequence diagram that belong to this pattern.

G.2.3 Find longest pattern

The first five steps of this use case are equal to the first five steps of the previous use case. At that point, the window will look similar to the one depicted in figure G.9.

6. *Plug-in shows the pattern diagram and a list of the statistical values of the throughput time of each pattern.*
   On the right side of the window, KPIs are displayed including the statistical values of the throughput time. The pattern with the highest average throughput time is the longest pattern.

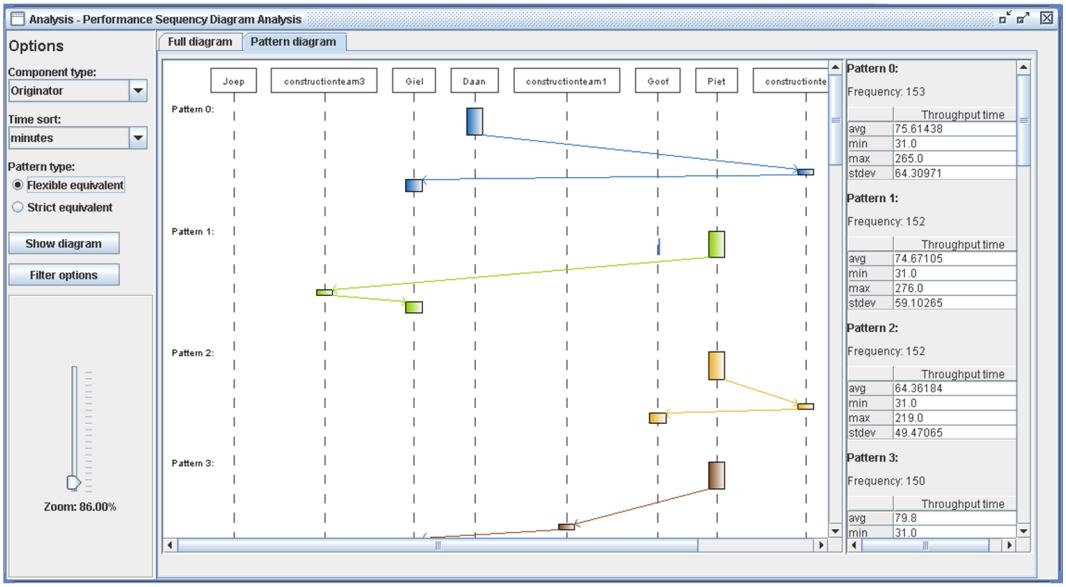

Figure G.9: Screenshot pattern diagram

G.2.4 View average time in period

The first five steps of this use case are equal to the first five steps of the previous use case.

6. *Plug-in shows the pattern diagram and a list of the statistical values of the throughput time of each pattern.*
   The window will look similar to the one depicted in figure G.9.

7. *User selects a block, i.e. a period in the pattern diagram.*
   Move the mouse cursor over the block that you are interested in.

8. *Plug-in shows the average duration of the selected period.*
   In the tooltip, information about the 'selected' block will appear, including the average time spend in the block, i.e. the average duration of the selected period.

Appendix H

User manuals

This appendix contains the user manuals of the *Performance Analysis with Petri net* plug-in and the *Performance Sequence Diagram Analysis* plug-in.

H.1 User manual: Performance Analysis with Petri net

1. Introduction

The purpose of the *Performance Analysis with Petri net* plug-in is to provide you with a means to assess the performance of your processes. Main focus here is to provide you with Key Performance Indicators (KPIs), which can be summoned in an intuitive way. This is done with help of an event log and a process model (in form of a Petri net) of the process under consideration. The process model that is used for this can be user-defined (and imported into the ProM-framework) or obtained by process mining. By replaying the log traces of the event log in the process model, the plug-in retrieves the values of various KPIs. When the log traces have been replayed, the values of these can be accessed through the process model.

2. Performance Information

The log replay method (also used in the conformance checker plug-in) is used to derive performance information from a log with help of a process model. The log replay method simulates the process instances in the input log in the Petri net. To be able to do this, events in the log have to be associated with transitions in the Petri net. During log replay, every time an event occurs in a log trace, (one of) its corresponding transition(s) in the Petri net is fired and measurements are taken. In this way, performance information can be obtained, like for instance routing probabilities at XOR-splits and the time that cases, represented by tokens, spend at certain places within the process. We distinguish four groups of performance information:

- Process metrics
- Place metrics

- Two-transition metrics
- Activity metrics

Process metrics
The values of the following process metrics (i.e. process-related KPIs) are derived by this plug-in:

- **Total number selected**: the total number of process instances.
- **Number fitting**: the number of process instances that complete properly and successfully, i.e. the number of instances that can be replayed in the Petri net without any problems. For details about fitness, see the *Conformance Checker* help file and documentation.
- **Arrival rate**: the number of arrivals of cases to the process per time unit.
- **Throughput time**: the statistical values (average, minimum, maximum, etc.) of the time cases spend in the process.

Place metrics
The values of the following place-related KPIs are derived:

- **Frequency**: the number of visits of tokens to the place during replay of the process instances in the Petri net.
- **Arrival rate**: the rate at which tokens arrive to the place per time unit.
- **Place time metrics**:
  - **Sojourn time**: the statistical values of the total time tokens spend in the place (Waiting time + Synchronization time).
  - **Waiting time**: the statistical values of the time that passes from the (full) enabling of a transition until its firing, i.e. time that tokens spend in the place waiting for a transition (to which the place is an input place) to fire and consume them.
  - **Synchronization time**: the statistical values of the time that passes from the partial enabling of a transition (i.e. at least one input place marked) until full enabling (i.e. all input places are marked). Time that tokens spend in a place, waiting for the transition (to which this place is an input place) to be fully enabled.
  - **Probability at each outgoing arc of XOR-splits**: The probability that tokens that are in the XOR-split (i.e. place with multiple outgoing arcs) are consumed by the transition that is connected to the place by the arc.

Two-transition metrics
For each two (visible) transitions in the Petri net, the following metrics are available:

- **Frequency**: the number of process instances in which both transitions fire at least once.
- **Time in between**: the statistical values of the (absolute) time between the first firing of the one transition and the first firing of the other transition during replay of cases.

Activity metrics
Often transitions are part of an activity, i.e. a task. For instance, an activity *clean* can be represented by the transitions *clean-schedule*, *clean-start* and *clean-complete*. In such case, activity metrics can be derived. These are:

- **Arrival rate**: rate at which work-items arrive to the activity.
- **Activity time metrics**:
  - **Waiting time**: the statistical values of the time between the moment at which the activity is scheduled and the moment at which execution of the activity is started. (Statistical values of the time between schedule and start events of the activity).
  - **Execution time**: the statistical values of the time in which the activity is actually executed. (Statistical values of the time between start and complete events of the activity, without the time the activity was suspended in between the start and complete).
  - **Sojourn time**: the statistical values of the time between the moment at which the activity is scheduled and the moment at which it finishes execution (Statistical values of the time between schedule and complete events of the activity).

The exact values of activity time metrics (waiting, execution and sojourn time) can only be calculated when in the used log and Petri net there are schedule, start and complete events present for all activities. However, when you are using a log and a Petri net which contain less information, upper bounds of the values of activity time metrics are calculated and displayed instead. This can be done because the process model that we have at our disposal, defines how activities are interrelated. For instance, suppose the log and the Petri net only contain start and complete events/transitions. In such case, execution times can be calculated in a regular fashion. Since we do not know when the activity was scheduled, we can at best calculate an upper bound for the waiting time and the sojourn time of the activity. This can be done by pretending that a schedule did take place at the moment that the last activity that is a direct precessor of this activity was completed. The current activity may actually

have been scheduled at every moment after this moment and before the start event. The precise moment is unknown.

It is not always possible to calculate upper bounds for all activity metrics however. When for instance, in a log only complete events exist, only an upper bound for the sojourn time (which then naturally is also an upper bound for the waiting time and execution time) can be calculated. On the other hand, if only start events occur, it is possible to calculate upper bounds for all activity metrics, but in this case the upper bounds of execution time and sojourn time of each activity do overlap with the upper bounds of waiting time and sojourn time of its direct successor(s). To show the difference between normal (i.e. exact) values and bound values for activity metrics, bound values are displayed in red.

3. Getting Started

This plug-in requires a Petri net and an associated log as input. Associating a log to a Petri net can be done in the following way:
First, open the log file (in MXML format). Then import the process model via choosing:
File → Open [process format] file → With: [log] from the menu.
This process model must either be available in some Petri net format (a .tpn or .pnml file) or in another modeling paradigm that can be read by ProM (such as an EPC or a YAWL model), which can then be converted into a Petri net by dedicated conversion plug-ins within the framework. Another possibility is of course to mine a process model from the event log using one of the mining plug-ins.

When a Petri net and an associated log have been provided, and the plug-in has been started, you will first see a screen where you can specify the initial settings. In the example below, the user has chosen to have all values of the time-related KPIs to be given in days and to have the KPIs given with an accuracy of up to five decimal places. Instead of days, the user could have selected: milliseconds, seconds, minutes, weeks, months (30 days) and years (365 days)

The user has also specified to use 'manual bottleneck settings'. A bottleneck in a process is a place with a high average waiting time. Three bottleneck levels are distinguished: low, medium and high and each of these levels has its own settings. After calculation, the used Petri net will be visualized with places colored according to their waiting time level.
At low level and medium level, you can specify the upper bound of the levels. In the example, the user has specified the upper bound at low level to be 2 days, which means that every place that has an average waiting time associated to it of 2 days or less is considered to be of low level, and will be colored blue (you can change this color, by clicking on the 'Change' button and choosing the color of your liking). At medium level the user has set the upper bound to 5 days, so every place with an average waiting time between 2 and 5 days (including 5, excluding 2) is considered to be of medium level and will be colored yellow (or the color

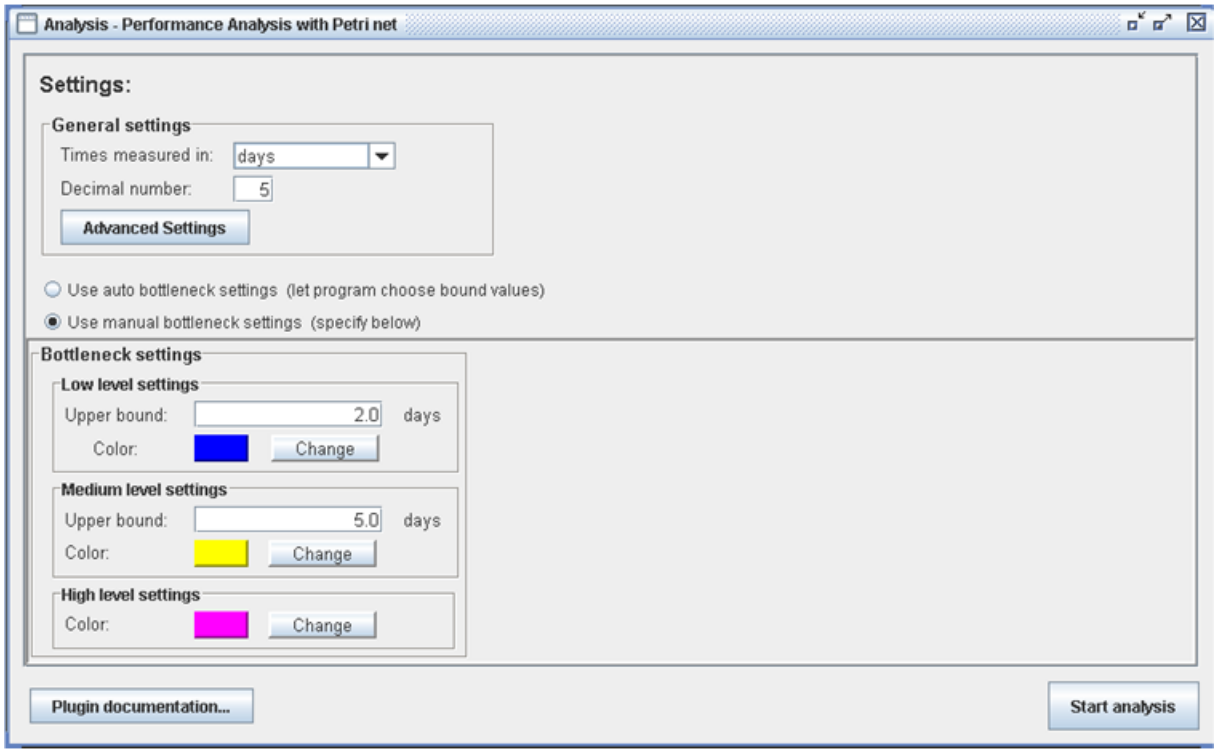

Figure H.1: Screenshot of the *options* screen of the plug-in

you selected) and all places with an average waiting time above 5 days will be colored purple (or the color you selected).

If you have no clue on which upper bound values you want to use, you are advised to select the 'use auto bottleneck settings' option instead. Values for upper bounds will then be estimated by the plug-in and standard coloring will be used. Each level will then contain approximately one third of the places.

4. Advanced Settings

From the options screen, you have the possibility to access the 'advanced settings' by pressing the corresponding button. In the window (as depicted in figure H.2) that pops up you can specify which conformance settings you wish to use and if you wish to use the 'time saving' option here.

Dealing with non-conformance

As mentioned, when calculating the values of the KPIs, all process instances in the input log are replayed in the input Petri net. When doing this, it can happen that a process instance does not fit in the Petri net, i.e. it can happen that when a process instance is replayed, an event occurs in the log at a moment at which the corresponding transition in the Petri net is not yet enabled. Since process instances are replayed in the Petri net using the non-blocking log replay method, in such case the transition is artificially enabled by creating the missing

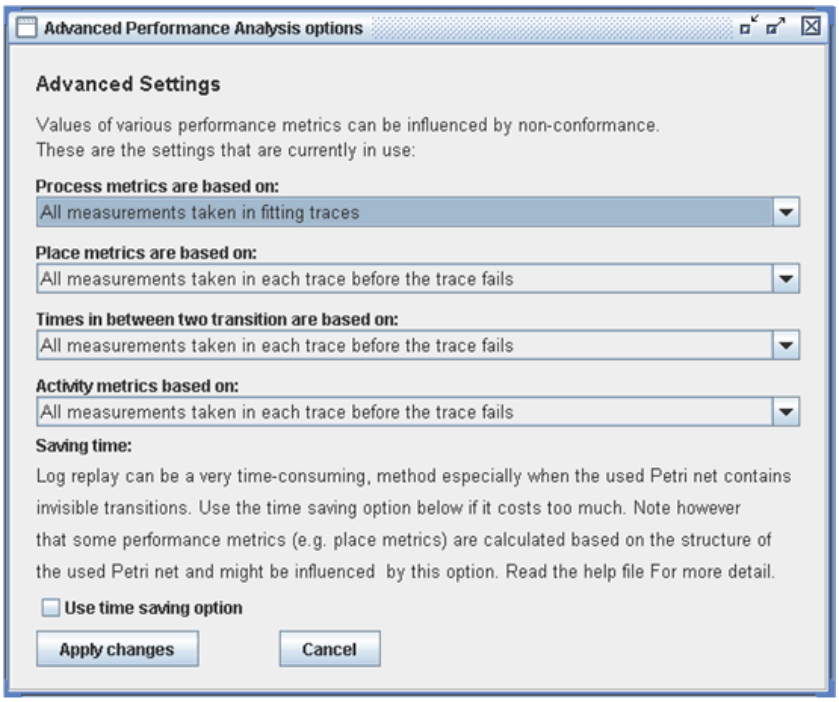

Figure H.2: Screenshot of the *advanced settings* screen of the plug-in

tokens, and then fired anyway. This has its consequences for the measurements taken. There are many ways to deal with this non-conformance, which may be appropriate in different situations. It is therefore possible to set up how you want the plug-in to deal with this behavior. These advanced settings are related to the different metric-types.

Process metrics can be based on:

- All measurements taken: the process instances that fit in the Petri net, as well as the ones that do not are used when calculating the values of process metrics. Beware however, that using non-fitting traces may lead to strange results. For example, in case the log contains incomplete traces, then the calculated average throughput time may be much lower than expected.
- All measurements taken in fitting traces: only those process instances that fit are used in calculation. This is the standard setting for process metrics.

Place metrics can be based on:

- All measurements taken: all measurements taken in the process instances that fit in the Petri net, as well as the ones that do not are used when calculation the values of place metrics.
- All measurements taken in each trace before the trace fails: all measurements which are taken when replaying process instances that fit in the Petri net are used, as well as

the measurements taken in the traces that do not completely fit in the Petri net, but which are taken before the moment at which an event occurs in the log that does not fit in the Petri net, i.e. when the transition corresponding to the event is not enabled. One can expect these measurements, taken before a failure occurs, to not be influenced by the non-conformance of the trace in which they were taken. That is why this is the standard setting.

- All measurements taken in each trace where no related transition has failed execution: In case a related transition (a transition of which this place is an input or output place is) of the place fails in a trace, all measurements which are taken in this trace are not used in calculations.

- All measurements taken in fitting traces: only those process instances that fit are used in calculation.

Time-in-between metrics can be based on:

- All measurements taken: all measurements taken in the process instances that fit in the Petri net, as well as the ones that do not are used when calculating the time between two transitions. Note that in all traces at most one measurement is taken for the time-in-between is taken.

- All measurements taken in each trace before the trace fails: all measurements belonging to the two transitions that are taken in fitting traces are used when calculating time-between metrics and next to these, of each non-fitting trace: the measurements that are taken during replay of the trace, before the moment at which an event occurs in the log that does fit with the Petri net. This means that when a transition fails before the last of the two transitions fires during log replay, the measurement is not used. One can expect that these measurements, taken before a failure occurs, are not influenced by the non-conformance of the trace in which they were taken. That is why this is the standard setting.

- All measurements taken in each trace where both transitions do not fail: all time in between measurements taken in traces where both the selected transitions fire regularly (so do not fail). Note that this does not exclude the measurements taken when a transition fails which is located in between these two transitions, which can be of influence to the results of calculations.

- All measurements taken in fitting traces: only those process instances that fit are used in calculation.

Activity metrics can be based on:

- All measurements taken: measurements taken in fitting as well as in non-fitting traces, are used when calculating activity metrics.
- All measurements taken in each trace before the trace fails: all measurements belonging to the activity that are taken in fitting traces are used when calculating activity metrics and next to these, of each non-fitting trace: the measurements that are taken during replay of the trace, before the moment at which an event occurs in the log that does not fit in the Petri net. One can expect that these measurements, taken before a failure occurs, are not influenced by the non-conformance of the trace in which they were taken. That is why this is the standard setting.
- All measurements taken where no transition corresponding to the activity fails: all measurements taken in traces where no transition, which corresponds to the activity, has to be artificially enabled. For example, when a 'complete'-transition of an activity fails, not only the sojourn time and execution time measurements are discarded, but also the waiting time measurement even though the 'schedule' and the 'start' transition do not fail.
- All measurements taken where no transition corresponding to a metric of the activity fails: the same as the last option, though now the measurements of the metrics that are not influenced by failing transitions are taken into account. In the example of the previous option, the waiting time measurement is used.
- All measurements taken in fitting traces: only those process instances that fit are used in calculation.

Time saving option

When the Petri net that is used as input to this plug-in is large and contains many invisible transitions, the replaying of the log may take much time due to the method used. This method builds a coverability graph every time that a transition is not immediately enabled, i.e. not all input places contain a token. This is done to discover if there is a sequence of invisible transitions that does enable the transition. When there are many invisible transitions in the Petri net or the Petri net is very complex, then this may take much time. If time is not on your side, you can select the 'use time saving option'. This will result in no coverability graphs being build during log replay. Note however that this may have its affect on the values of the varying performance metrics, since process instances that would normally conform to the Petri net can now become 'non-fitting'.

5. Starting the analysis

When all options have been set, you can press the 'Start Analysis' button, the program will then start the performance analysis using the specified settings. As you can see (in figure

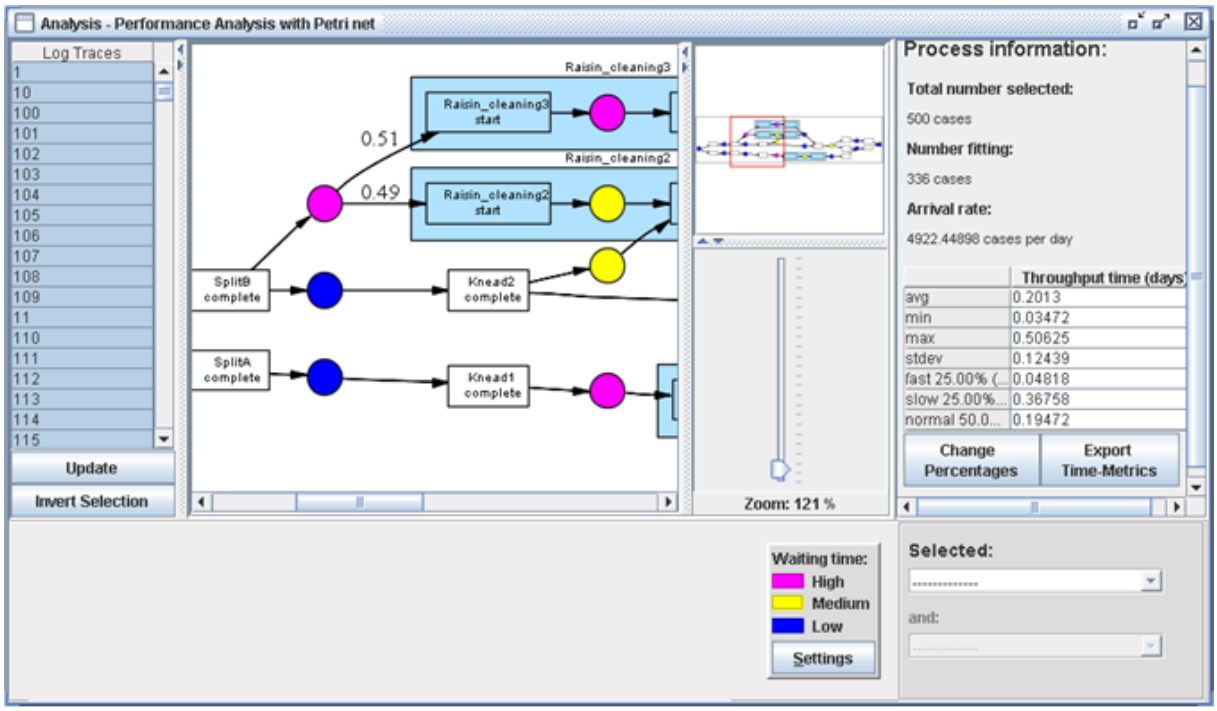

Figure H.3: Screenshot of the *Performance Analysis with Petri net* plug-in

H.3, a visualization of the used Petri net is displayed at the center of the screen. Within the shown Petri net, at XOR-splits, (i.e. places with more than one outgoing arc), the probability that a token is consumed by the transition (to which the arc leads) is shown. At the XOR-split in the example above, you can see that the probability that a token is consumed by transition 'Raising_Cleaning3 - Start' is 0.51 and the probability that a token is consumed by 'Raising_Cleaning2 - Start' is 0.49. Furthermore, places in the Petri net are colored according to their average waiting time. This should make it easier for you to spot the location of possible bottlenecks within your process. Below the Petri net, a legend is shown in which you can see which color belongs to which waiting time level. In the example, the purple-colored places are said to have a high average waiting time and may thus be bottlenecks.

Left of the Petri net visualization, a table is displayed that contains the identifier names of all process instances in the input log. You can select or deselect these (identifier names of) process instances. Initially all process instances are selected. If you press the 'Update' button after selecting process instances, the displayed performance information will be based on the selected instances only. If you press the 'Invert Selection' button all the process instances that are currently not selected get selected and all process instances that are currently selected, get deselected.

On the right hand side of the window, the values process-related metrics are displayed. Note that not only the average throughput time is displayed, but also the minimum, the maximum,

the standard deviation and the average time of the fastest process instances (initially 25% of the total number of selected process instances) which have the lowest throughput time, the average time of the slowest process instances (also 25% initially) and the average time of all other process instances (obviously 50% initially). You can adjust the percentages, by pressing the 'Change Percentages' button below the throughput time table and filling in the new percentages in the dialogs that will appear. Next to this, you also have the possibility to export the throughput times of all selected process instances to a comma-separated text file (.csv or .txt) by pressing the 'Export Time-Metrics' button and filling in or selecting a filename. Such files can be loaded into Microsoft Excel or StatGraphics for further analysis.

The values of metrics belonging to places can be viewed by selecting the place that you want to study more closely. You can select it, by selecting its name in the (upper) selection box on the lower right side of your screen or by clicking on the place within the Petri net. In the example below (figure H.4), the user has selected the (possible bottleneck) place with name p2.

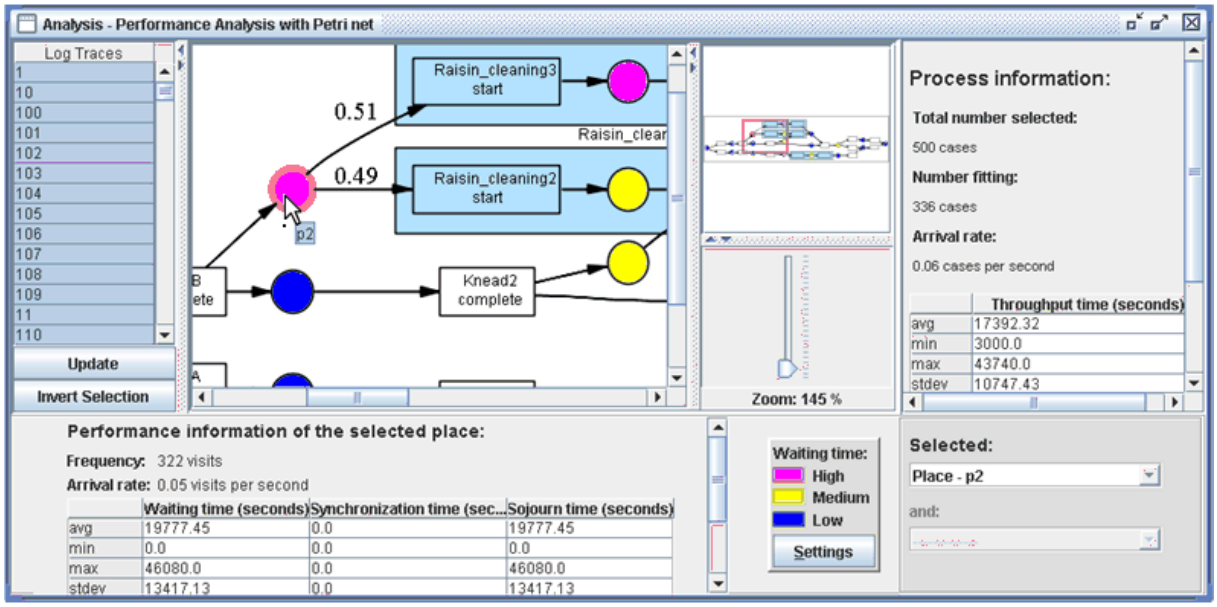

Figure H.4: Screenshot of the *Performance Analysis with Petri net* plug-in, where a place is selected

In a similar manner two transitions can be selected, after which the statistical values of the absolute time tokens spend in between these two transitions is displayed. Note that per process instance only one measurement is taken, namely the time between the first firing of the one transition during log replay and the first firing of the other transition. The values of metrics belonging to an activity can be viewed by selecting a transition that is related to the activity. Each normal (visible) transition is associated with an activity.

Just like with the throughput time, of each time metric (time-between, activity and place time metrics) the average, minimum, maximum, standard deviation and the average time of the fastest measurements (initially 25% of the total number of visits to the place), i.e. measurements with the lowest times, the average time of the slowest measurements (also 25% initially), i.e. measurements with the highest times, and the average time of all other visits (obviously 50% initially) is calculated. The time metrics are displayed in the table on the panel below the Petri net. You can adjust the percentages of slow, fast and normal measurements, by pressing the 'Change Percentages' button below the table, and filling in the percentages that you want to use in the dialogs that will appear. Next to this, you also have the possibility to export the times of all measurements of a time metric to a comma-separated text file (.csv or .txt) by pressing the 'Export Time Metrics' button and filling in or selecting a filename. Comma-separated files can be loaded into other tools, such as for example Microsoft Excel or StatGraphics, for further analysis.

6. Adjusting Settings

When you press the 'Change Settings' button below the waiting time legend, a window will appear which looks quite similar to the 'Initial Settings' window. Here you can set (adjust) the following options:

- The timesort, i.e. time unit, in which metrics are displayed (milliseconds, seconds, minutes, hours, days, weeks, months, years).
- The accuracy of the displayed metrics, i.e. the number of decimal places that a metric can have at most.
- The values of the bounds between the waiting time levels.
- The colors corresponding to the waiting time levels.

If a place was selected when you pressed the 'Change Settings' button, then you can also choose to apply the filled in bound-values and selected level colors to all places in the Petri net, or to apply them only to the selected place.
Furthermore, you can press the 'Advanced Settings' button after which you can specify in the window that pop-ups how the plug-in should deal with non-fitting traces.

H.2 User manual: Performance Sequence Diagram Analysis

Introduction

The purpose of the performance sequence diagram plug-in is to provide you with a means to assess the performance of your processes. This plug-in can especially be of help if you want to know what behavior in your processes is common, what behavior is rare and what behavior may result in extreme situations (e.g. instances with extremely high throughput times). What is more, it can assist you in finding the causes of such behavior.

In the logs of today's information systems, a lot of information is recorded. This plug-in will focus on information that is stored on the audit trail entry level of the logs. In an audit trail entry, information about one event is recorded. Common information stored at the audit trail entry level includes:

- TaskID: Name of the task involved in the event.
- Event type: Type of the event, e.g. schedule, start, complete.
- Timestamp: Timestamp at which the event took place.
- Originator: Name of the originator involved in the event.

Though one could also think of other information that may be stored here, such as:

- Department: Name of the department at which the event was handled.
- Part: Name of the part of the case that is currently being handled. For instance, suppose the process under consideration is the production line of a car. A case is then the production of one car. Parts currently handled may be: doors, wheels, windows etc.
- Country: Name of the country in which the event took place.

This plug-in allows you to focus on a certain data-element (such as taskID, originator, department) and see how transfer of work between instances of the selected data-element (for instance in case of originator this can be Nick or John or another person) takes place for each case. The transfer of work is displayed in a sequence diagram, in which you can easily see which data-elements cooperate. What is more, this plug-in takes the sequences that visualize the transfer of work, and compares them against each other to see if they follow the same pattern. The found patterns are then displayed in a separate diagram: the pattern diagram. Here the patterns are displayed, sorted based on the frequency of sequences that follow the pattern (the pattern with the highest frequency if displayed at the top). Furthermore, information such as the mean throughput time of the patterns is available as well here. Using this information, you can easily determine which patterns appear often (common behavior),

which appear rarely (rare, maybe even unwanted behavior), and which patterns result in a high average throughput time, and thus may indicate unwanted behavior. With help of the patterns, you may be able to understand the causes of this unwanted behavior (much time spend at a certain data-element instance perhaps). This can help you with preventing the behavior from appearing in the future.

How to use

For this plug-in only an event log is required as input. When you start the plug-in, first log relations will be derived from this log. When this is done, you will see a screen as the one depicted below (figure H.5).

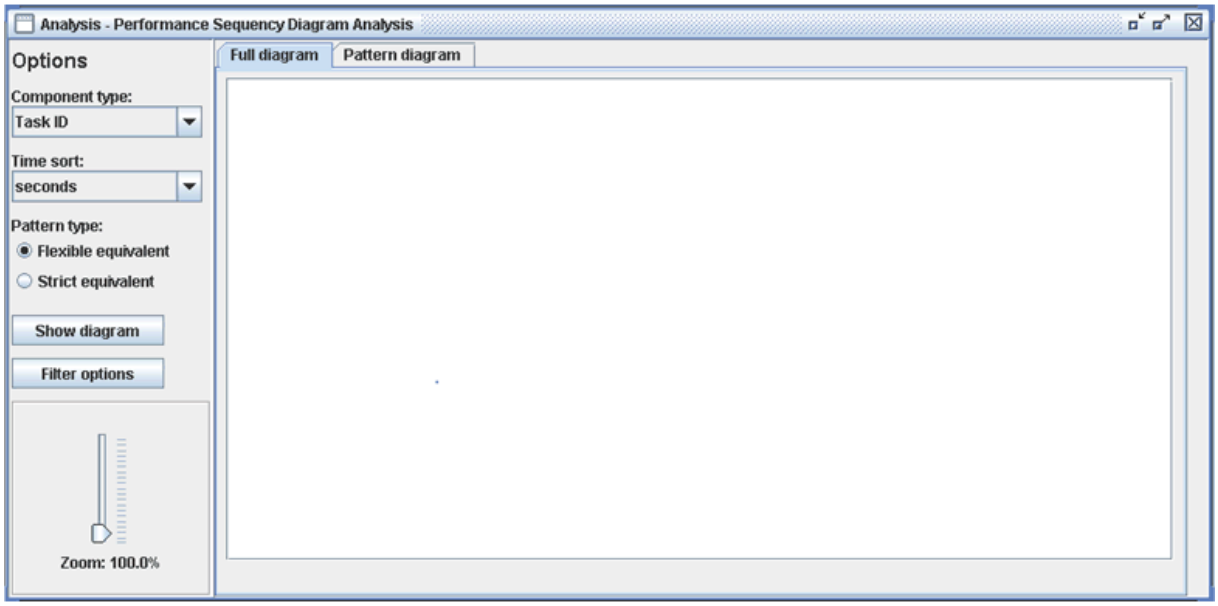

Figure H.5: Screenshot of the *Performance Sequence Diagram Analysis* plug-in at start-up

On the left-hand side of this screen, there are a number of options that you can set. At the top you can choose the data-element that you wish to use. Note that the types available here are dependent on the information contained in the log. Clearly, if departments are not mentioned in the log, then the department data-element type will not be available.
The second option available is the time sort, here you can select the unit in which the available times should be displayed. Time sorts available here are: milliseconds, seconds, minutes, hours, days, months (30 days) and years (365 days).

The last setting that is available is the pattern type. You can choose between using the flexible-equivalent or the strict-equivalent pattern type. The difference between the both is that when using the strict-equivalent pattern type, sequences are defined to be different if data-element blocks do not start and end in exactly the same order. When using the

flexible-equivalent pattern type this is not needed. Instead, for each block in the pattern the requirement is set that all blocks that begin after the block ends, should begin after the block ends in the compared sequence as well. Let us look at an example (figure H.6):

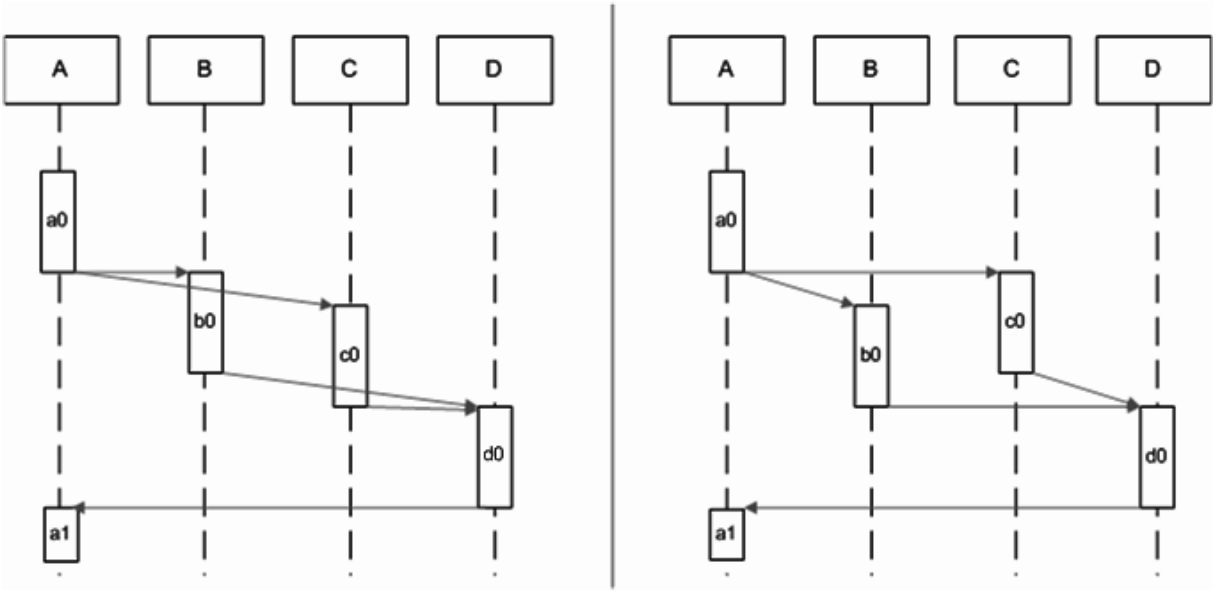

Figure H.6: Example of two sequences

When using the strict-equivalent type, the sequences in the example above are considered to be different, because in the left sequence block $b0$ starts (and ends) before block $c0$ and in the right sequence block $c0$ starts (and ends) before block $b0$. When using the flexible-equivalent type though, they are considered to be equal, because although blocks $b0$ and $c0$ do not start in the same order, the flexible-equivalent requirement is met. For both sequences we have the following:
After the end of block $a0$, blocks $b0$, $c0$, $d0$ and $a1$ start.
After the end of block $b0$, blocks $d0$ and $a1$ start.
After the end of block $c0$, blocks $d0$ and $a1$ start.
After the end of block $d0$, block $a1$ starts.
No block starts after block $a1$ ends.

When you have selected the settings that you wish to use, you can click on the 'Show Diagram' button. The sequence diagram will then be displayed. See for instance the example below (in figure H.7).

When you move your mouse cursor over a sequence, information about the sequence will be displayed in the tooltip. In the example, the mouse is above two overlapping sequences. Information about both these sequences is displayed. This information encompasses:

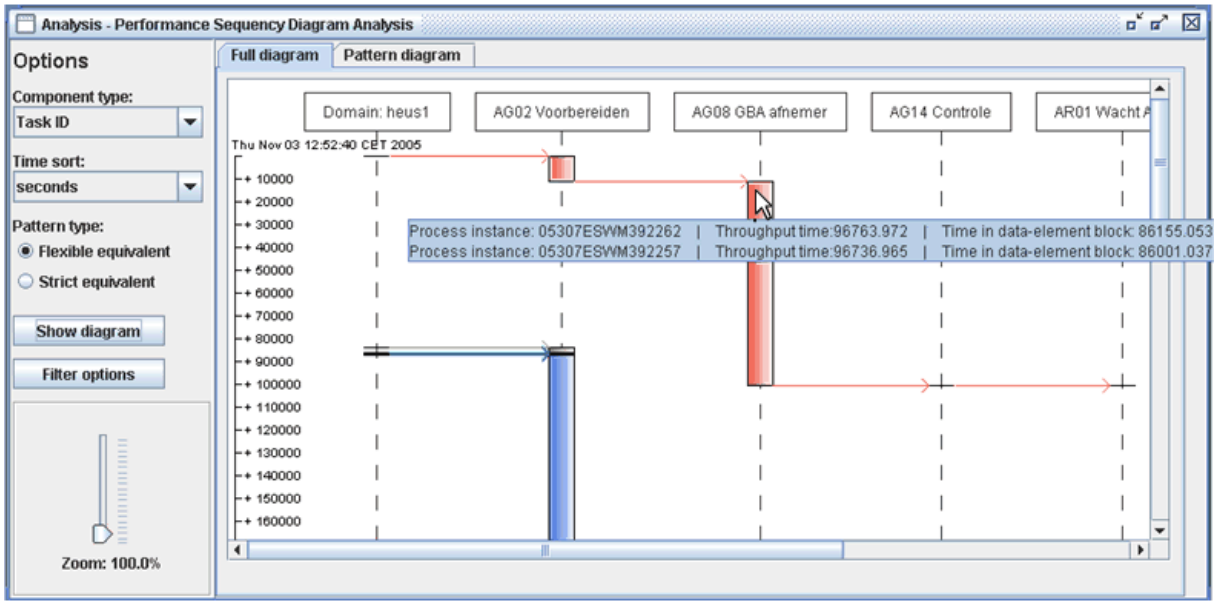

Figure H.7: Example of a sequence diagram in the plug-in

- The name of the process instance of which the sequence is a visualization.
- The throughput time of the sequence.

If the cursor points to an arrow:

- The name of the data-element from which the arrow originates, i.e. the data-element that is the source of the arrow.
- The name of the data-element in which the arrow ends, i.e. the destination of the arrow.
- The time spend between the moment at which the arrow begins and the moment at which the arrow ends.

If the cursor points to a block (as in the example):

- The name of the data-element to which the block belongs.
- The time spend in the block.

You can switch between the sequence diagram and the pattern diagram, by selecting the corresponding tab at the top of the screen. An example of what a pattern diagram can look like is shown below (figure H.8).

As you can see, information about each pattern, such as throughput time and frequency is shown on the right hand side of the screen. You can also see information belonging to the pattern in a tooltip when you move the cursor of your mouse over the pattern. The information in the tooltip encompasses:

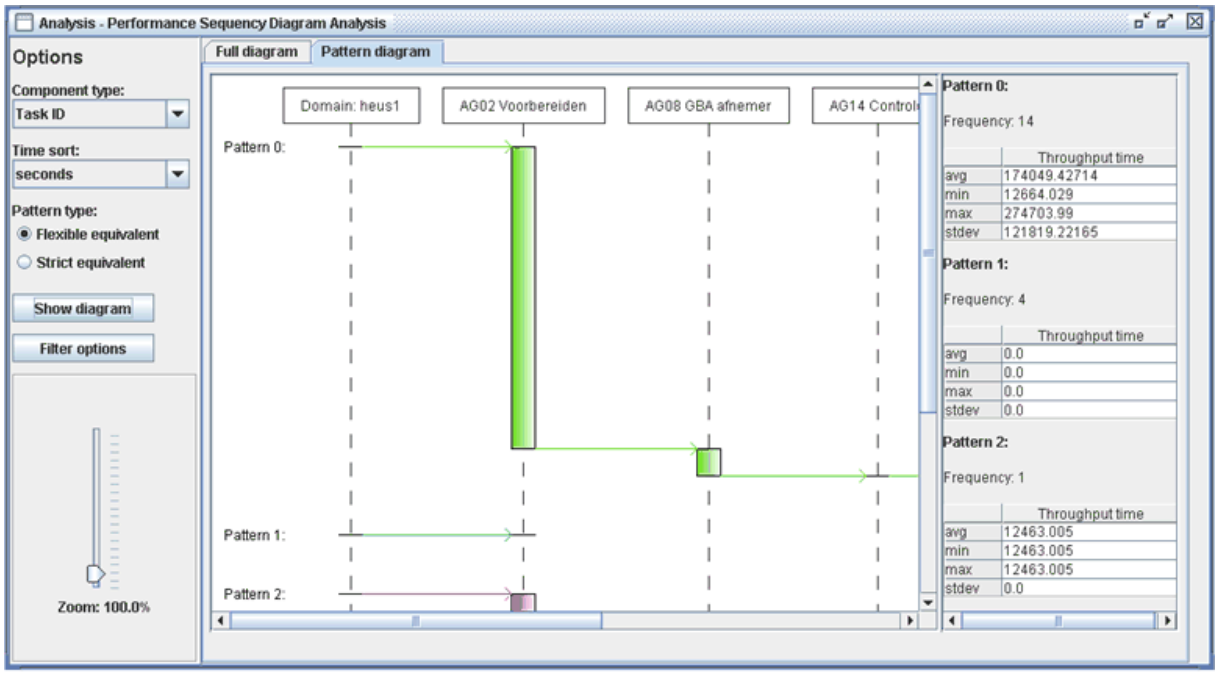

Figure H.8: Example of a pattern diagram in the plug-in

- The pattern number of the pattern.
- The mean throughput time of the pattern.

If the cursor points to an arrow:

- The name of the data-element from which the arrow originates, i.e. the data-element that is the source of the arrow.
- The name of the data-element in which the arrow ends, i.e. the data-element that is the destination of the arrow.
- The mean time spend between the moment at which the arrow begins and the moment at which the arrow ends.

If the cursor points to a block:

- The name of the data-element to which the block belongs.
- The mean time spend in the block.

If the results still contain many patterns, and you only wish to view a limited number, you can click on the 'Filter Options' button. You will then see the screen below (figure H.9).

Here you can select the process-instances that you wish to use by selecting them in the table on the left side of the screen. You can also select instances by selecting one of the following options, and pressing the 'Update' button afterwards. The following option is always available:

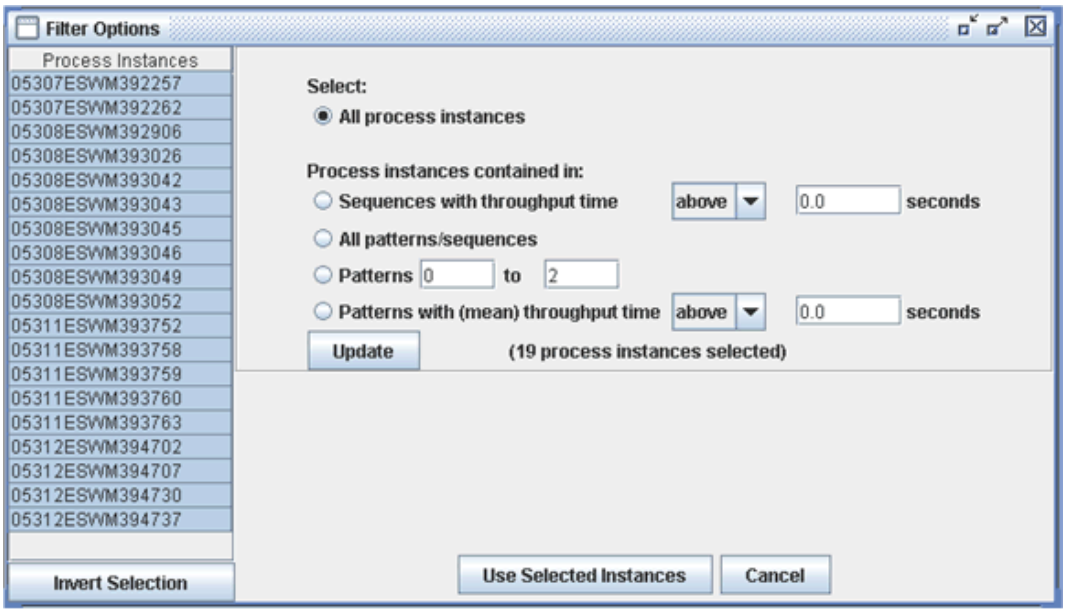

Figure H.9: Screenshot of filter options screen

- All Process instances: Selects all the process instances.

The following option is available after the 'Show Diagram' button has been clicked at least once:

- Sequences with a certain throughput time: Selects all process instances that have a throughput time above/below the filled in bound value.

The following options are available after the pattern diagram has been drawn at least once:

- All Patterns: Selects all process instances that match one of the patterns from the pattern diagram.
- Interval of patterns: Selects all process instances that match one of the patterns in the interval.
- Patterns with a certain mean throughput time: Selects all process instances that match one of the patterns of which the mean throughput time lies above/below the filled in bound value.

After you have selected the process instances that you wish to use, you can click the 'Use selected instances' button, after which the sequence diagram and pattern diagram will be redrawn.