

PROCESS MINING

Lab Session 3-4

Hands-on Session on Log Filtering and Process Discovery in ProM

Content

Laboratory Session	2
1. Getting Familiar with ProM	4
A. Log Visualizer	4
B. Log Filtering	7
C. Log Variant Analysis	9
Exercises	10
2. Discovering a Model with Inductive Miner	11
Exercise: Alpha Miner vs Inductive Miner	14
3. Constructing Transition Systems and Discovering Models with Region-based Miner	15
Exercise	21
4. Discovering a model with Heuristic Miner	22
5. Final Exercises	27
Solution of Exercise 1	28
Solution of Exercise 2	31

Laboratory Session

ProM is an **extensible** framework that supports various process mining techniques in the form of plug-ins. It is **platform-independent** as it is implemented in Java and can be downloaded **for free**¹. (We used *ProM Lite 1.3* Revision 43952 while preparing this tutorial.) ProM offers hundreds of plug-ins that enable you to apply the latest developments in process mining research.

Please let ProM Lite have enough time to install all packages on the opening screen.

Commercial tools, such as Disco, which we experienced during the first week of the course, are certainly more robust and more user-friendly. Unfortunately, they provide limited functionalities compared with ProM.


The aim of this first hands-on session with ProM is hence twofold:

- To let acquire familiarity with ProM, which will be used in two further ProM laboratory sessions,
- To experience different discovery algorithms to appreciate the pros and cons of the alternatives. In this laboratory session, we will focus on Alpha Miner, Region-based Miner, and Inductive Miner, which are nowadays among the best miners. In a future session, we will also experience the Heuristic Miner.

This document refers to two laboratory sessions. **The expectation is that the first laboratory session will cover the work described in the first three chapters, while the second the last two.** Of course, it is possible to go on after the first three chapters during the first laboratory session, when the previous ones are carried out quickly.


2

ProM checks for updates in each opening. If you are sure about all packages are already updated, you can skip this checking step by making the following configurations

- Open the ProM.ini file  under the ProM folders (It is mainly under this path: "C:\Users\onurd\ProM Lite 1.3\ProM.ini", "onurd" will be your username)
- Make the following changes: *CHECK_PACKAGES = false* and *AUTO_UPDATE = no*

¹ Download link: <https://www.promtools.org/>

ProM uses a part of your memory. It is 4GB as default. If you have enough memory, you can change it by the following configuration:

- Open the ProMLite13.14j.ini file  under the ProM folders (It is mainly under this path: "C:\Users\onurd\ProM Lite 1.3\ProMLite13.14j.ini", "onurd" will be your username)
- Change "*Xmx10G*" (assuming you have at least 16GB memory). "10G" indicates how many gigabytes you allow ProM to use while you run any action.

1. Getting Familiar with ProM

We start from the event log referring to executions of a process at a Dutch financial institute in 2011. The event log represents an application process for a personal loan or overdraft within a global financing organization. The event log is a merger of three intertwined sub-processes: i) the events starting with A refer to the execution of the activities for the loan application that the applicant initiates, ii) the events starting with O refer to those referring to the offers made by the institute and iii) the events starting with W are related to the internal activities of the institute's employees. We will explore this event log with the help of different log visualizations in ProM, followed by answering some questions.

A. Log Visualizer

- i. Start by importing the *2011.xes.gz* file in ProM and import *ProM log files (Naive)* after opening log file as shown in [Figure 1](#).

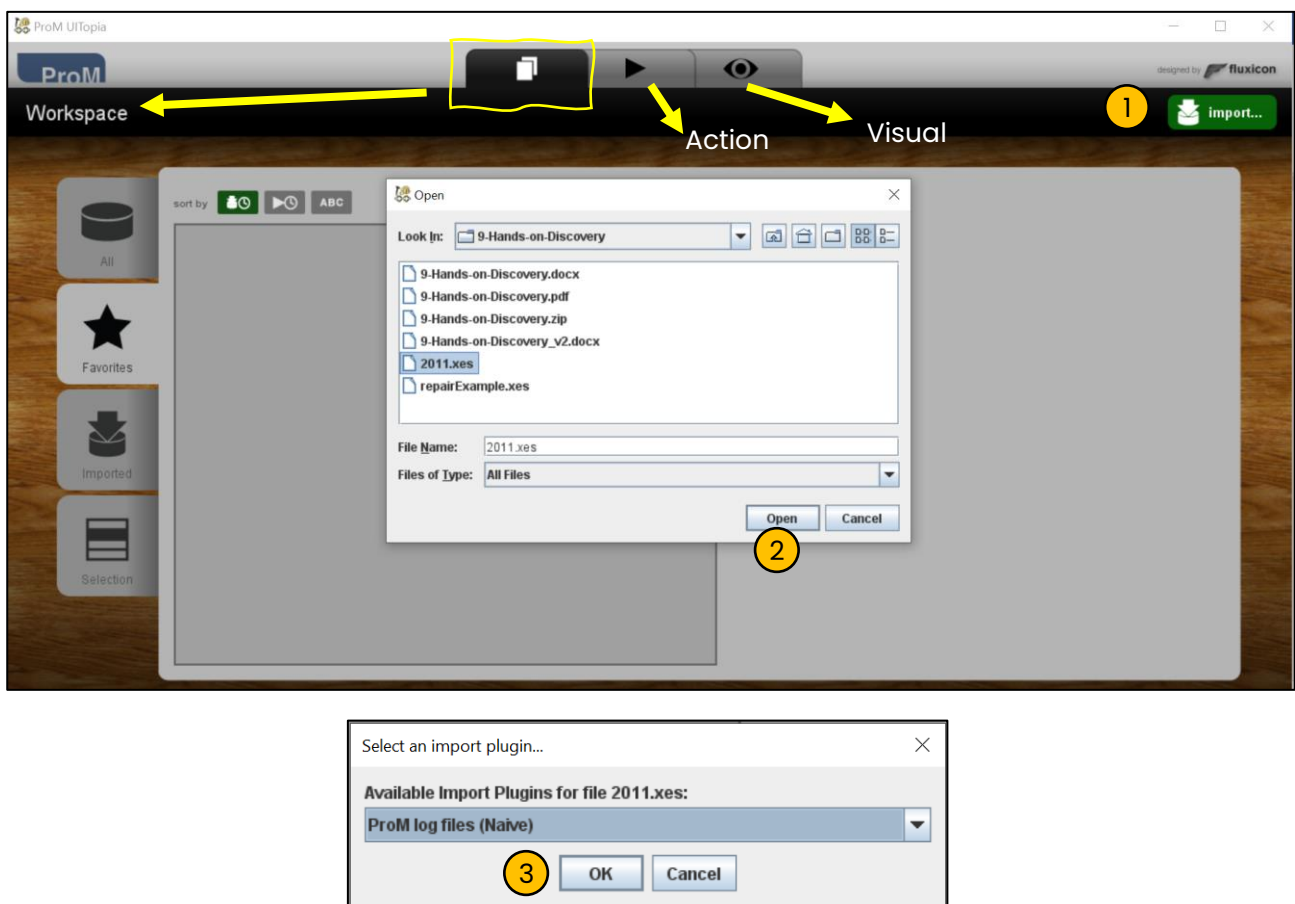


Figure 1. Importing data to ProM

- ii. Select the event log and click the eye icon shown in [Figure 2](#) to explore the event log using the *View Resource*.

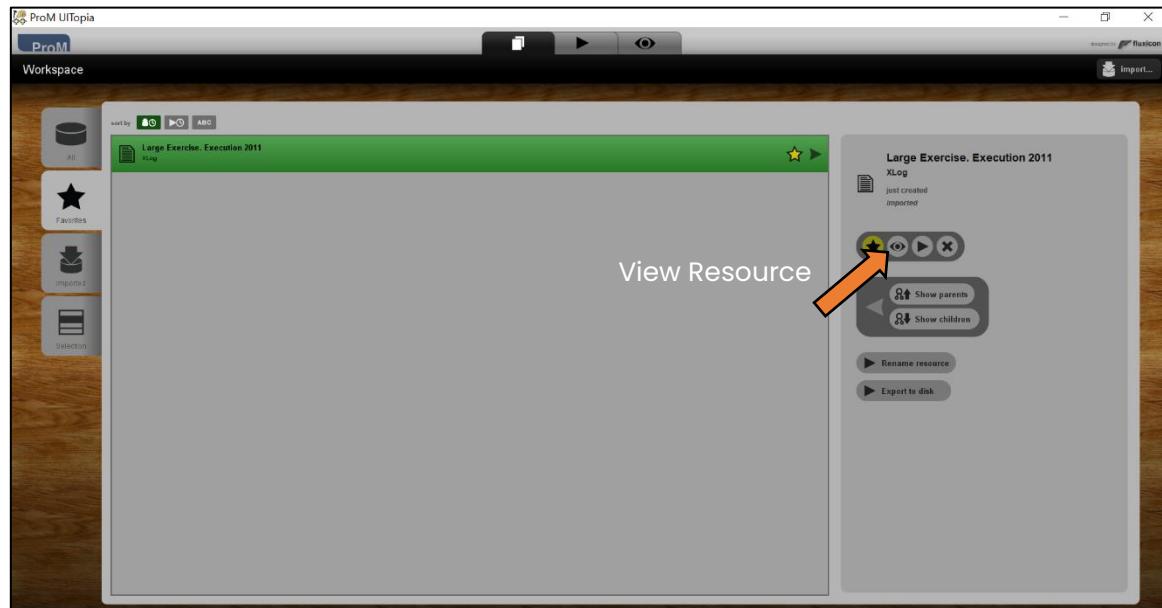


Figure 2. View Resources

- iii. The *Dashboard panel* (① in Figure 3) is opened by default. Check it to get information about the summary of the process, number of events, number of cases, etc. The dashboard panel shows key data and case-based graphs. Key data (②) include the number of processes, cases, events, etc. Case-based graphs (③) depict event per case and event classes per case. You can see the minimum, average and maximum number of events/event classes per case in these visuals.

5

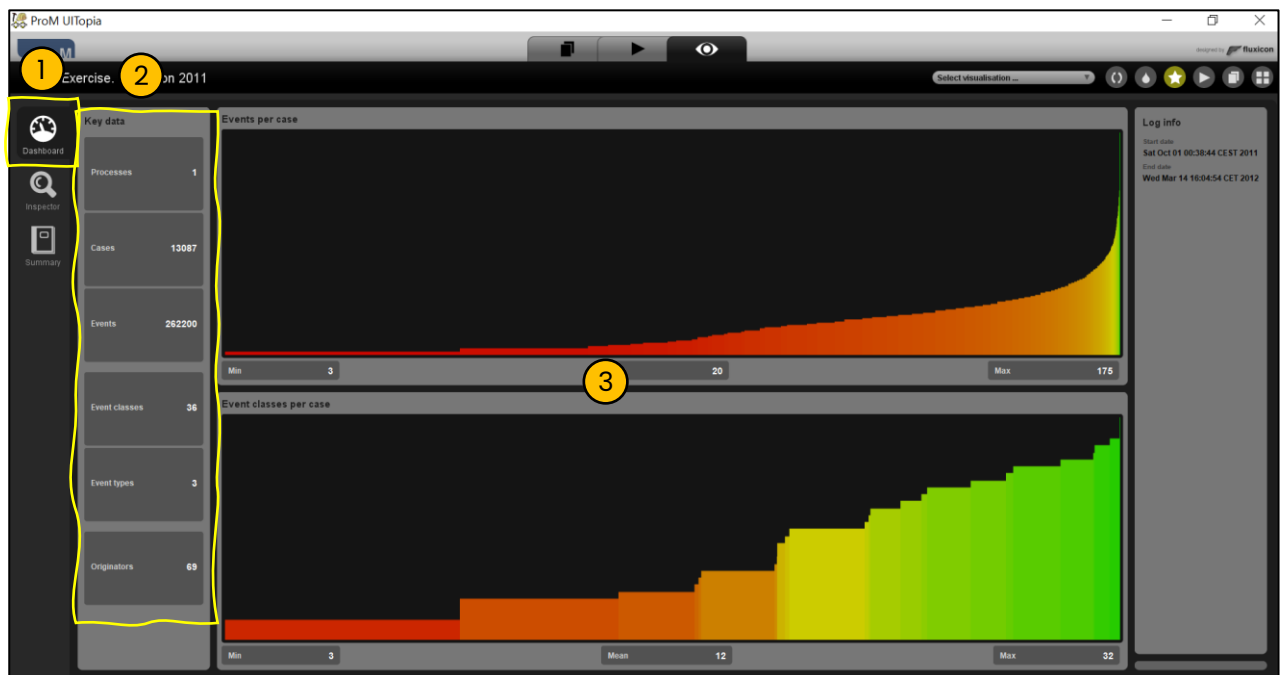


Figure 3. Dashboard

- iv. Click the second resource view, *Inspector* (① in Figure 4), to get information about the event log's traces, and log attributes. The *Browser tab* (②) contains trace IDs (1) and activity details (2). The *Explorer tab* (③) allows looking at traces from a different perspective by coloring cases considering frequency. *Log Attributes tab* (④) refers to the structural format of the event log.



Figure 4. Inspector

- v. Click the third and last resource view, *Summary tab* (Figure 5). You can answer the questions: "How many cases/tasks/resources are in the log?" by looking at the Summary tab. Moreover, you can learn first and last activities in the log and their number and percentage of occurrence.

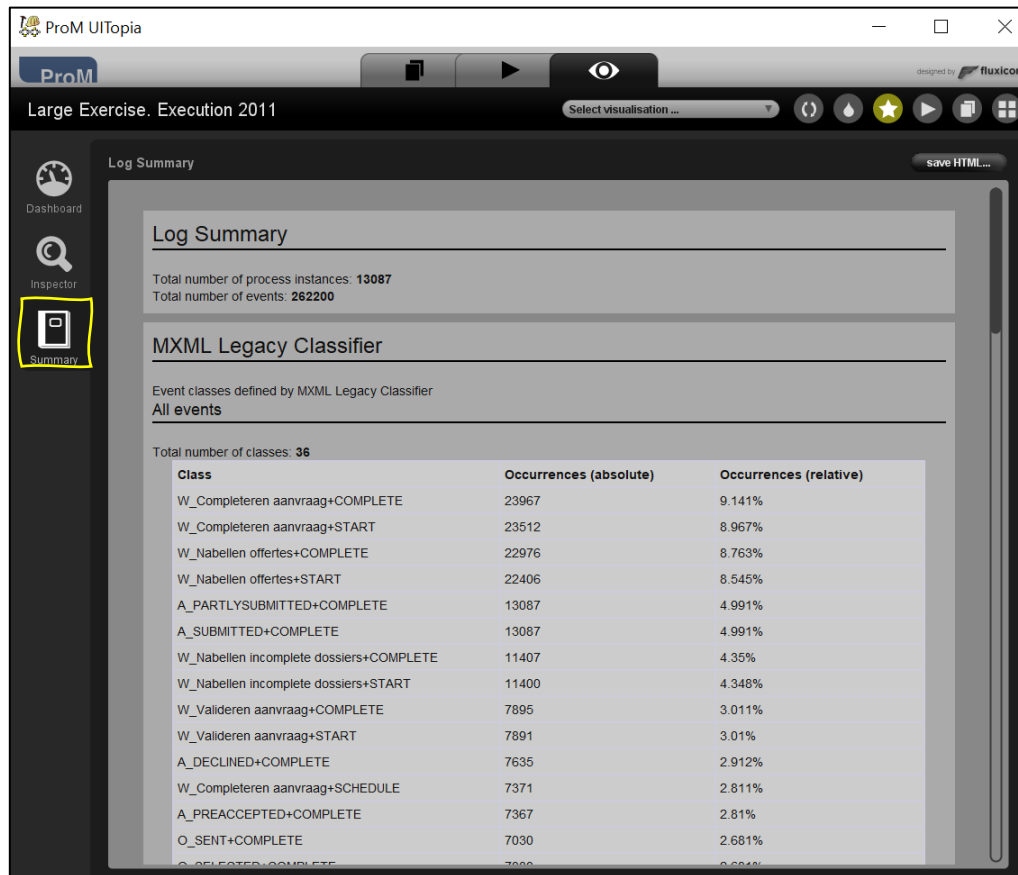


Figure 5. Summary

B. Log Filtering

We mentioned that the event log contains events belonging to three sub-processes, depending on whether the activity name starts with A, O, or W. We need to focus on the events related to the A activities to apply process discovery to the Application sub-process. Hence, we need to filter out the other events (see Figure 3):

- i. You can come back to the *Workspace* window by clicking (① in Figure 6). Select the event log and press the *Play* button (②).

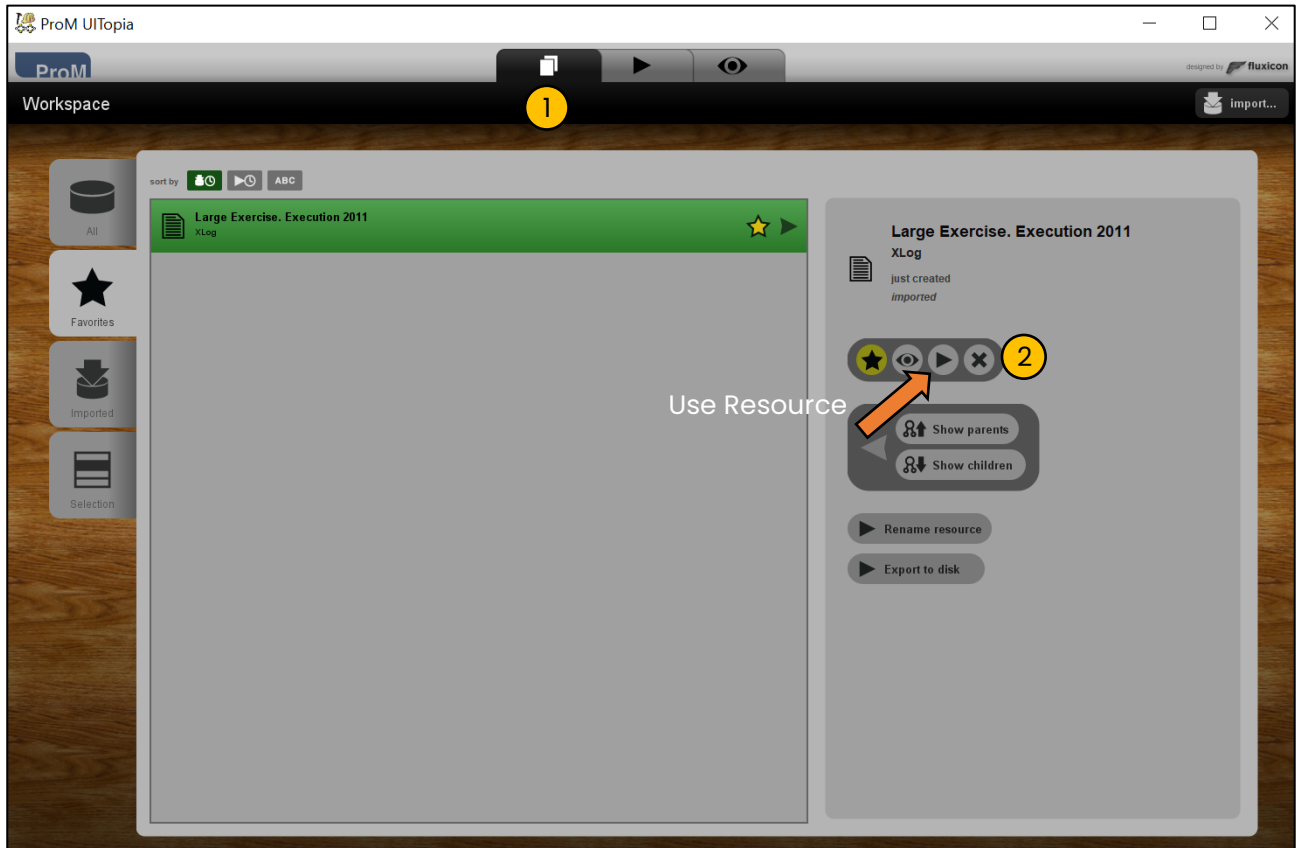


Figure 6. Use resource

8

- ii. In Figure 7, search for the plug-in “*Filter Log using Simple Heuristics*” in the search field. Select this plug-in and click *Start*.



Figure 7. Selecting action to filter the log

- iii. In [Figure 8](#), select every activity as potential start activity (②) and end activity (③) of the process. Finally, select all the activities starting with A (⑤), as shown below. It would lead to only activities belonging to one sub-process. Name the log as A-Subprocess (⑥) and click finish (⑦).

Please note that reduce only the number of events, not the number of log traces. If some activities were not selected as possible start activity (②) and end activity (③), the corresponding traces would be filtered out in a way you do not want.

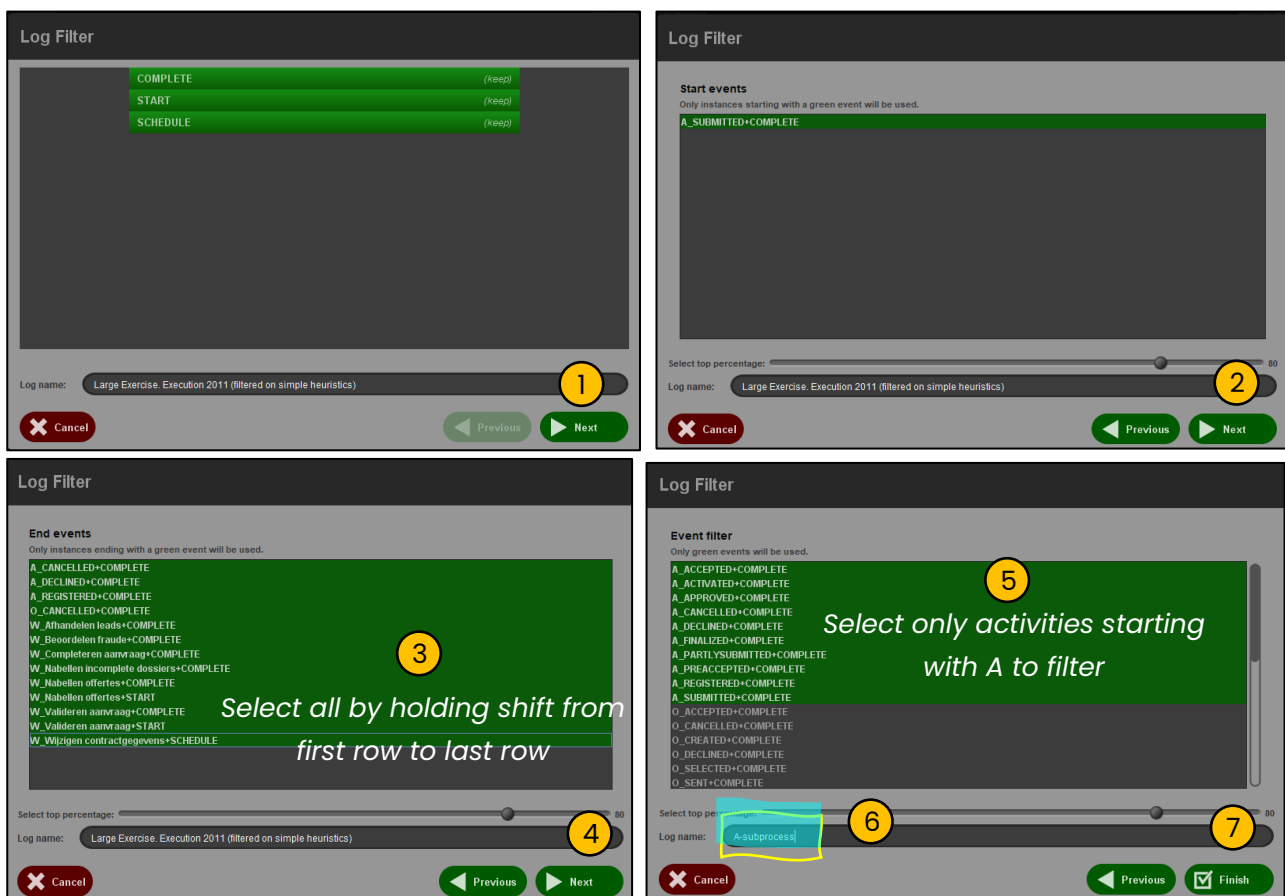


Figure 8. Adjusting actions for filtering

C. Log Variant Analysis

Next, we focus on analyzing different sequence variants of the process. Each variant is a sequence of events such that every case within the variant has the same sequence of events. This analysis is used to identify the most common variants. This can be achieved as follows:

- Visualize the [A-Subprocess](#) event log if it is not on the screen open the visual area.
- Select visualization item (① in [Figure 9](#)) "Explore event log (trace variants/...)" from the dropdown menu. The traces are sorted from top to down from most frequent to least frequent in [Figure 10](#). Note that this dropdown menu contains different visualization types according to the objects.

- iii. Traces of the different variants can be analyzed by clicking on the variant. Similarly, individual traces can be queried, and variants can be selected and exported as sub-logs. For instance, to export a sub-log containing only the first two variants, press *Ctrl* and select the first and second variants. Then right click "*Export 2 selected trace variants as log*". The new log should now be present in the *Workspace*.



Figure 9. Selecting visualization type (Event log exploration)

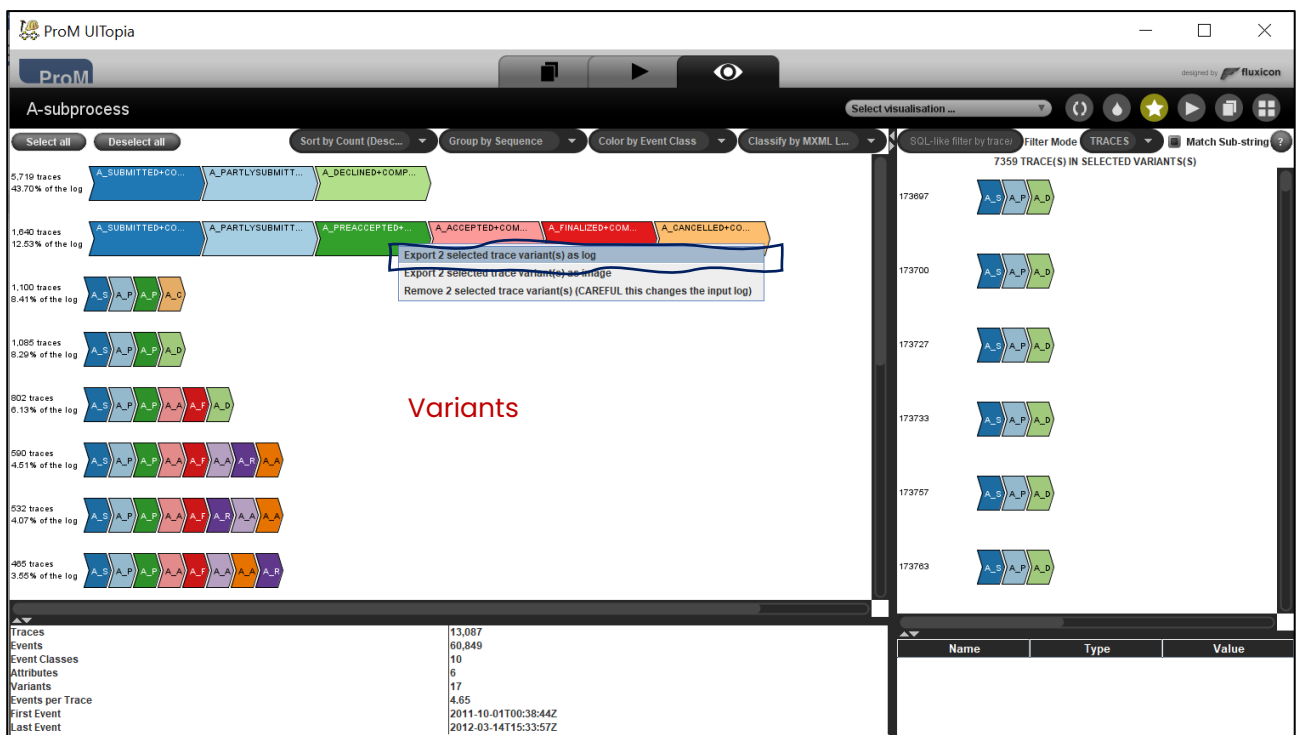


Figure 10. Variant analysis in ProM

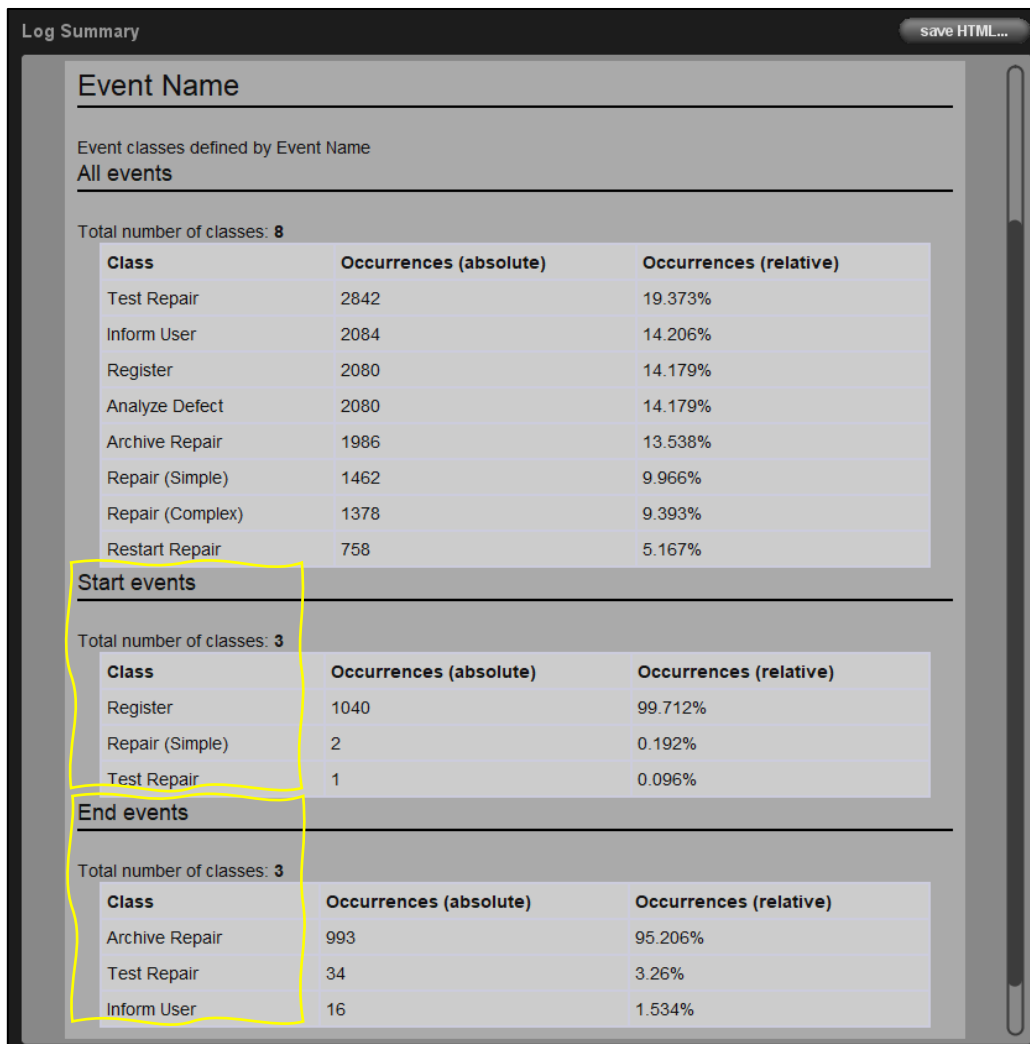
Exercises

1. What is the least frequent variant, and which cases belong to it?
2. Select the top five variants and create a new sub-log. Use it as input for the Alpha Miner to discover a process model.

2. Discovering a Model with Inductive Miner

The second part focuses on introducing the *Inductive Miner*. This miner is one of the most enhanced miners currently available. We have not studied the algorithm during the theoretical classes because it is rather complex, but using it is still worthwhile. In the remainder, we will employ the event log *repairExample.xes*, which is artificial, and record the executions of a hypothetical process to repair, e.g., mobile phones. After loading this event log in ProM and inspecting the *Summary* (as seen in *Page 4, Section 1*), we observe that the log contains events of both “*start*” and “*complete*” types (cf. *Figure 11*).

This section primarily focuses on *Visual Inductive Miner*, which can consider the events of type *complete* and *start* as separate events referring to the life cycle of activity instance. In fact, this information is used to improve the model.



Event Name		
Event classes defined by Event Name		
All events		
Total number of classes: 8		
Class	Occurrences (absolute)	Occurrences (relative)
Test Repair	2842	19.373%
Inform User	2084	14.206%
Register	2080	14.179%
Analyze Defect	2080	14.179%
Archive Repair	1986	13.538%
Repair (Simple)	1462	9.966%
Repair (Complex)	1378	9.393%
Restart Repair	758	5.167%
Start events		
Total number of classes: 3		
Class	Occurrences (absolute)	Occurrences (relative)
Register	1040	99.712%
Repair (Simple)	2	0.192%
Test Repair	1	0.096%
End events		
Total number of classes: 3		
Class	Occurrences (absolute)	Occurrences (relative)
Archive Repair	993	95.206%
Test Repair	34	3.26%
Inform User	16	1.534%

Figure 11. Log summary of the event log

This miner for process discovery is available in plug-in *Mine with the Inductive Visual Miner* (write to *Action search* area), which only takes an event log as input. After clicking on Start, the plug-in is started, and the screen should be similar to [Figure 12](#).

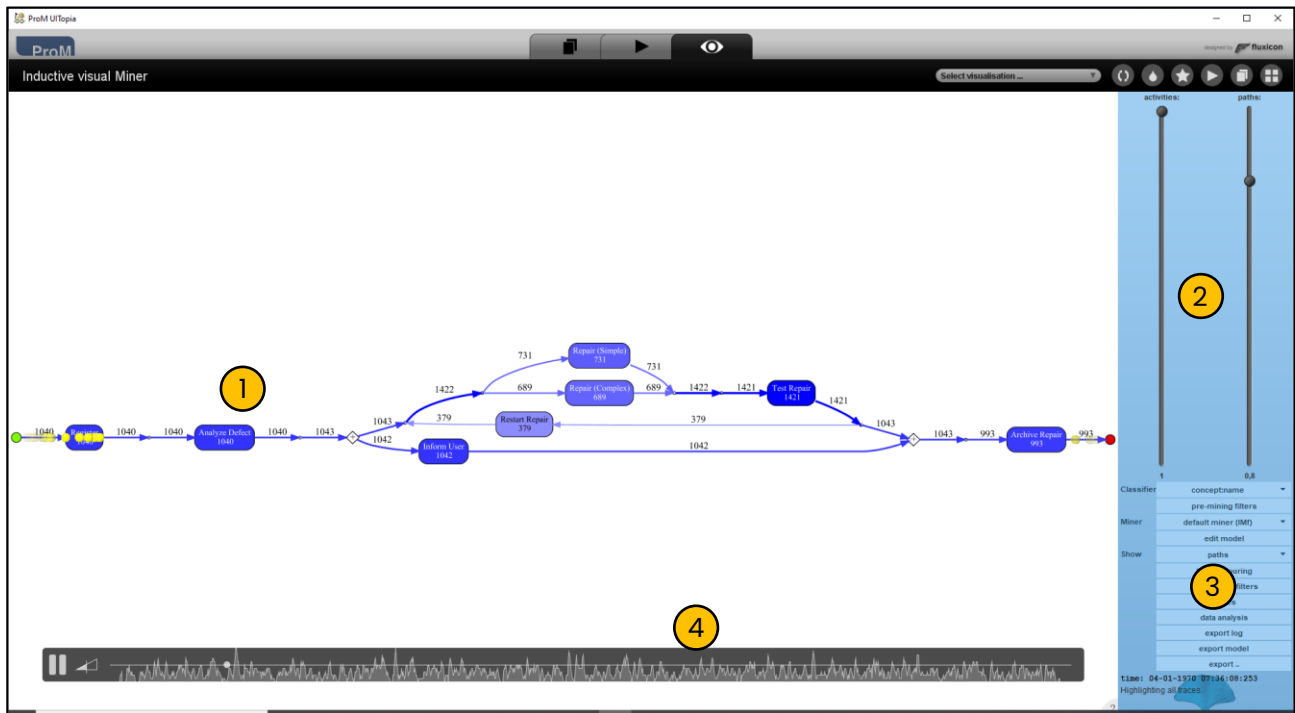


Figure 12. Discovered process model with Inductive Visual Miner

Area ① shows the mined process model. Each activity is filled with a color whose darkness indicates the frequency of occurrence of the associated activity. The darker the color, the more frequent the activity. The numbers on the arcs indicate the number of times those transitions are occurred. The numbers inside the activities refer to the number of their execution. A portion of the event log is replayed on the model. Each dot indicates that a running case is flowing through a branch of the model.

In **Area ②**, it is possible to discover a model that allows for a smaller or larger fraction of the behavior observed in the event log. If a model allows for a larger fraction of the observed behavior, it is generally less understandable than a model that allows for less behavior. Therefore, the model must balance simplicity and other quality (as discussed during the lecture). With this plug-in, this trade-off can be made visually. Through two slides, we can determine the fraction of observed paths which should be allowed by the model and decide the fraction of activities observed in the event log that we want to incorporate into the model.

Area ③ is used to provide further customizations and to export models and videos for other usages inside or outside ProM. This area is further broken down into three elements: *Classifier*, *Miner*, *Show*.

The *classifier area* contains two elements:

- ❖ The first allows us to decide which classifier to use for discovery. A classifier is a log attribute used as an activity name. The default is "*concept:name*", the standard activity-name attribute in XES format.
- ❖ The second is about *pre-mining filter settings*. It makes it possible to filter certain (portions) of the log traces before applying the Inductive Miner.

The *miner area* allows us to decide the miner type employed, such as considering the entire life cycle when mining the process model or the "complete" events. By default, the "complete" events are only used; for instance, selecting the option of *life cycle miner*, the plug-in will use the "start" and "complete" life cycle events. The second part of this area with the button *edit model* allows one to edit the model after being mined.

The *show area* contains some customization options, which are all very interesting. For the sake of time, we mainly focus on the main components.

- ❖ The dropdown list is used to customize what is visualized. It is selected "*paths*" by default. The "*paths and deviations*" option shows the paths observed in the event log but disallowed by the model. Also, time-related information can be visualized by selecting options "*path and queue lengths*", "*path and sojourn time*", "*path and waiting times*" and "*path and service information*". (These options will be discussed in detail when the course focuses on the additional perspectives on December 1, 2022.)
- ❖ The *traces* option is used to visualize the traces and the deviations quickly. The deviations are shown as moves in the log or model.
- ❖ The *highlighting filters* option hides the model parts that do not respect specific criteria. By clicking on the button, we can define many criteria. The plug-in will hide the activities and paths that are never executed or followed when those criteria are not met. Also, the frequency information is adapted accordingly.
- ❖ The *data analysis* option allows us to know the data-based results of the model, such as log attributes, trace attributes, event attributes, and cohort analysis.
- ❖ The *export log* option creates a sub-log according to your filters and other settings. The *export model* is used to generate a Petri Net object, which is made available in ProM for further use (e.g., for conformance checking). Models can be exported as PNML files that can be loaded into Woped.

The slider in *Area* ④ controls the replay of a portion of the event log onto the model. The slider becomes active when one passes over it with the mouse. The graph shows the activities being executed (i.e. not yet completed) at each moment. The time interval is divided into buckets. There are as many buckets as the number of pixels. If the event log contains "start" and "complete" events, *point* (x, y) in the graph indicates that, in the time interval for the x th bucket, y activities are being executed. If the event log only contains the "complete" events, the semantics is different: *point* (x, y) in the graph indicates y activities have been completed in the time interval for the x th bucket.

Exercise: Alpha Miner vs Inductive Miner

Use event log *repairExample.xes* to mine the process model using the *Alpha Miner*. Before applying the plug-in, it is necessary to filter out the events related to starting activities.

1. Is the model discovered through *Alpha Miner* satisfactory? Could you explain the reason why the *Alpha Miner* discovers a specific model?
2. If the model is unsatisfactory, try to filter out the traces affecting the results. Remember that complete traces start with activity "Register" and conclude with activity "Archive Repair".
3. Repeat the experience with the Inductive Miner on the original event log (i.e. before filtering out): is the model already satisfactory?

3. Constructing Transition Systems and Discovering Models with Region-based Miner

In this section, we use the again *repairExample.xes* event log. The first plug-in to employ is *Mine Transition System* focusing on building a transition system from the event log (see Figure 13). The discovery of a model using the region theory requires a *transition system* (i.e., a final-state automaton). The only input is an event log.

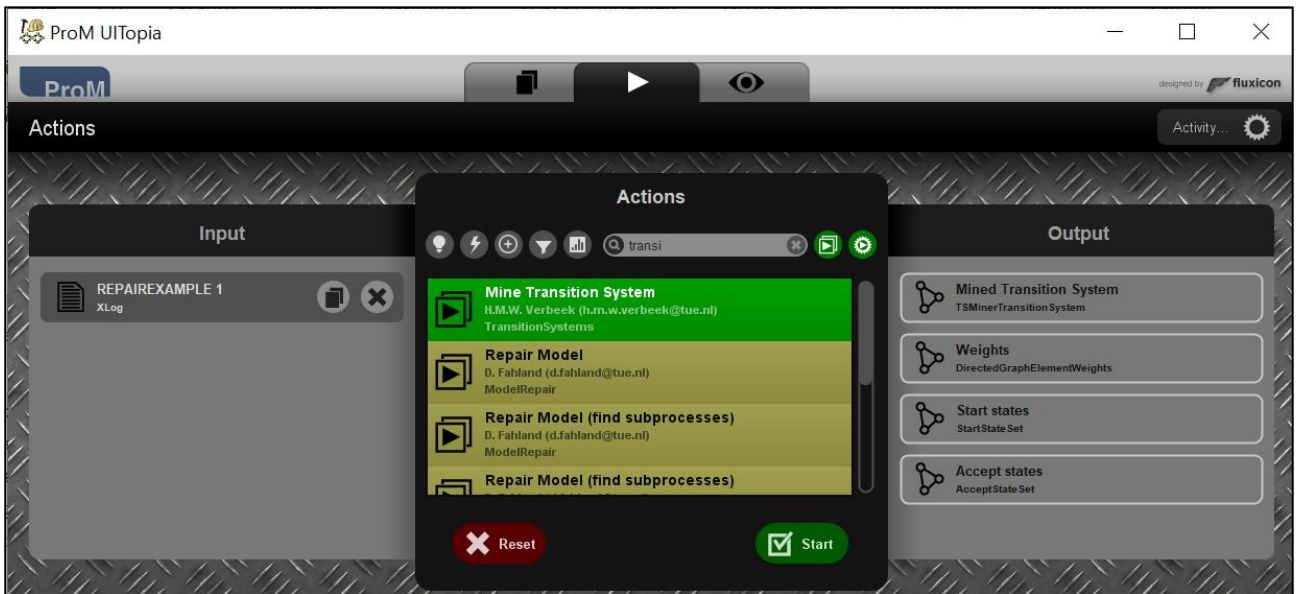


Figure 13. The action of Mine Transition System

After pressing Start, the wizard will guide you through the process of configuring this miner. The configuration options for this miner can be divided into three categories:

- options for configuring state keys (states differ if and only if their keys differ)
- options for configuring transition labels
- options for configuring post-mining conversions

The wizard will allow you to configure these three categories in the given order.

When the plug-in is started, options for configuring state keys window is appeared like in Figure 14, which can be left unchanged. This enables us to decide whether to consider past or future events when constructing the transition system. It also allows one to choose which attribute to use as labels. The current configuration is such that a past abstraction will be used, using the event name.

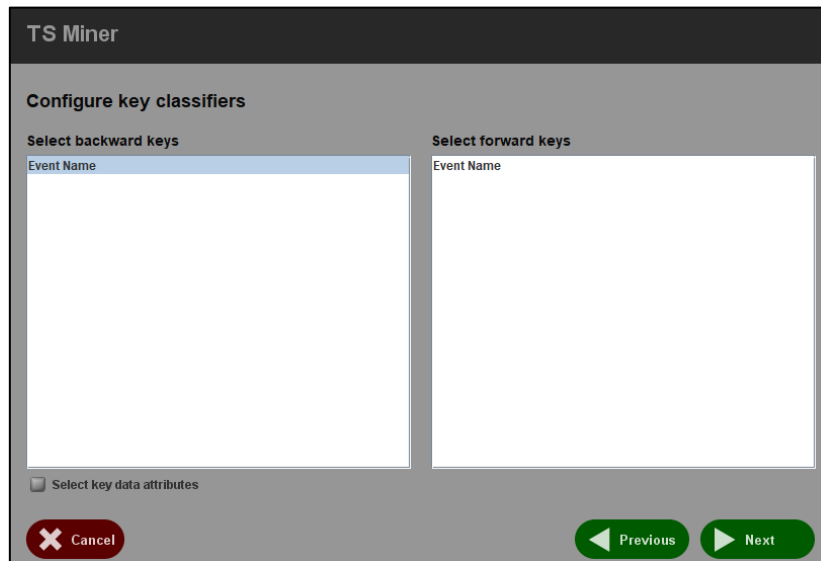


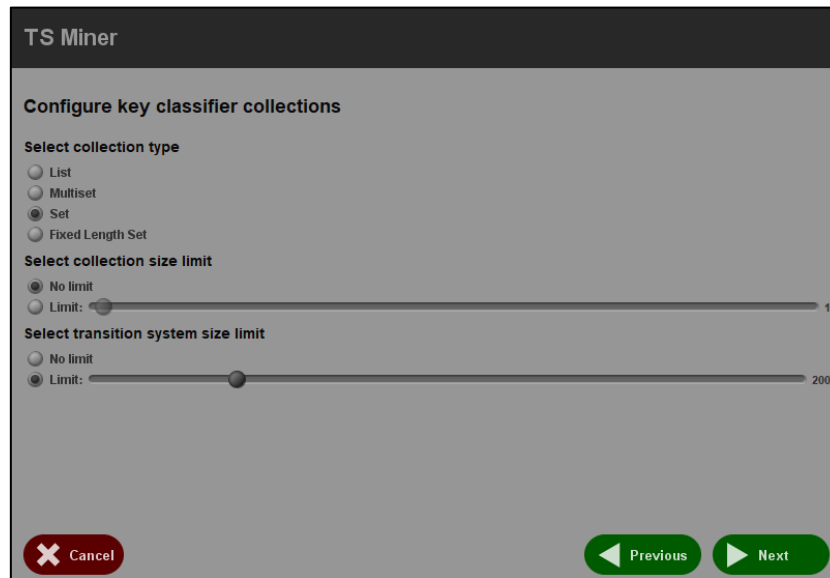
Figure 14. Options for configuring state keys

After clicking on Next, the window for *options for configuring transition labels* is appeared (see Figure 15).

- ❖ *Select collection type* allows us to decide whether to use the sequence abstraction (here named list), the multiset or set collection type (please ignore the fourth).
- ❖ *Select collection size limit* allows us to decide whether to use the previous event, a user-defined number of prior events, or the entire history. The option *No limit* corresponds to using the whole record. Alternatively, you can choose the option limit to customize how many events will be used through the slider.
- ❖ *Select transition system size limit* allows one to limit the number of states of the transition system: the default opinion is *limit to 200 states*, which is far larger than what we can obtain with the log we are considering.

We opt for using a set abstraction that contains the entire history, which is the configuration observed in Figure 15.

Note that, compared with the default, the option “Select collection size limit” has been changed to No Limit in the figure!



TS Miner

Configure key classifier collections

Select collection type

- ☐ List
- ☐ Multiset
- ☒ Set
- ☐ Fixed Length Set

Select collection size limit

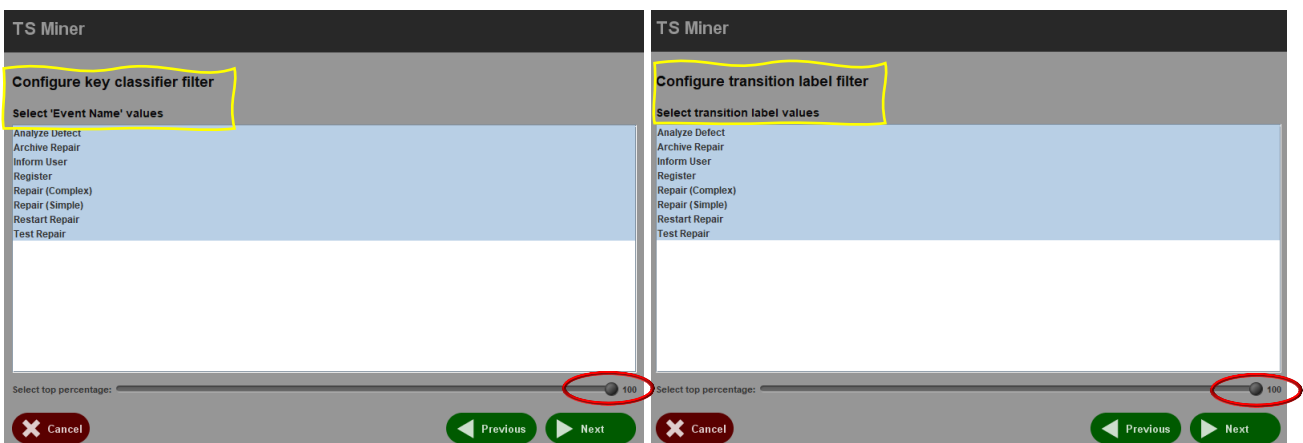
- ☒ No limit
- ☐ Limit:

Select transition system size limit

- ☐ No limit
- ☒ Limit:

Figure 15. Options for configuring transition labels-1

The following two wizard screens filter specific event labels (see Figure 16). For instance, you want to avoid certain transition labels (i.e., activities) in the Petri net to keep the model more straightforward. Since the Petri net would already contain a limited number of transition labels, we aim to consider all event and transition labels. It is possible by selecting every element in the lists or moving the sliders to 100 or CTRL+A after choosing one of the items from the list.



TS Miner

Configure key classifier filter

Select 'Event Name' values

- Analyze Defect
- Archive Repair
- Inform User
- Register
- Repair (Complex)
- Repair (Simple)
- Restart Repair
- Test Repair

Select top percentage:

TS Miner

Configure transition label filter

Select transition label values

- Analyze Defect
- Archive Repair
- Inform User
- Register
- Repair (Complex)
- Repair (Simple)
- Restart Repair
- Test Repair

Select top percentage:

Figure 16. Options for configuring transition labels-2

In the wizard windows of *Configure post-mining conversion*, we can click on the *Next* button (see Figure 17). The last wizard window is simply a summary of the parameters chosen in the previous wizard windows. This concludes the configuration, yielding the transition system in Figure 18.

The thickness of an arc (i.e., transition) is proportional to the number of event-log traces that traverse that arc. The arcs are also annotated with the transition labels (i.e., the activity whose execution triggered the state transition). Each state corresponds to a given set of activities. You can inspect the set associated with the state by passing with the mouse over any state.

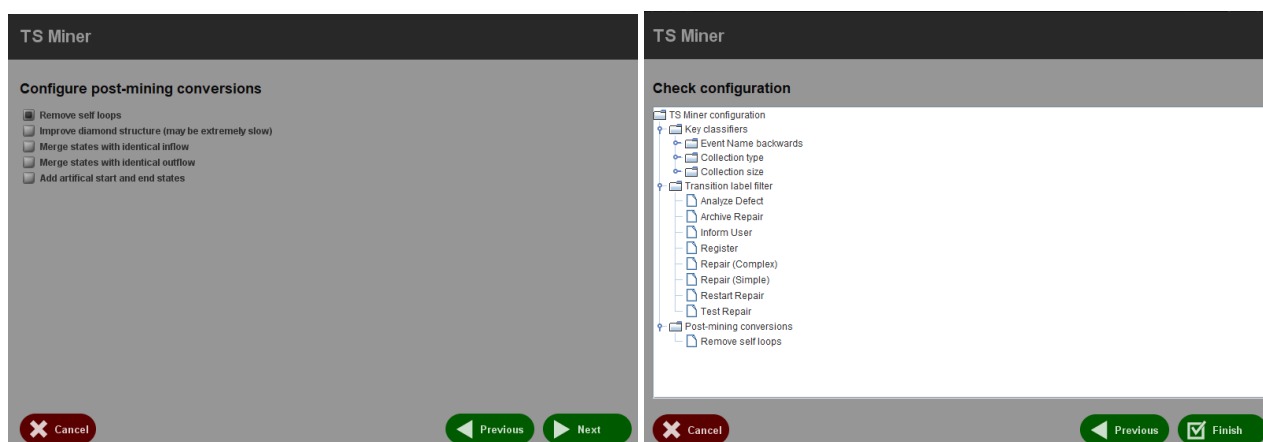


Figure 17. Options for configuring post-mining conversions

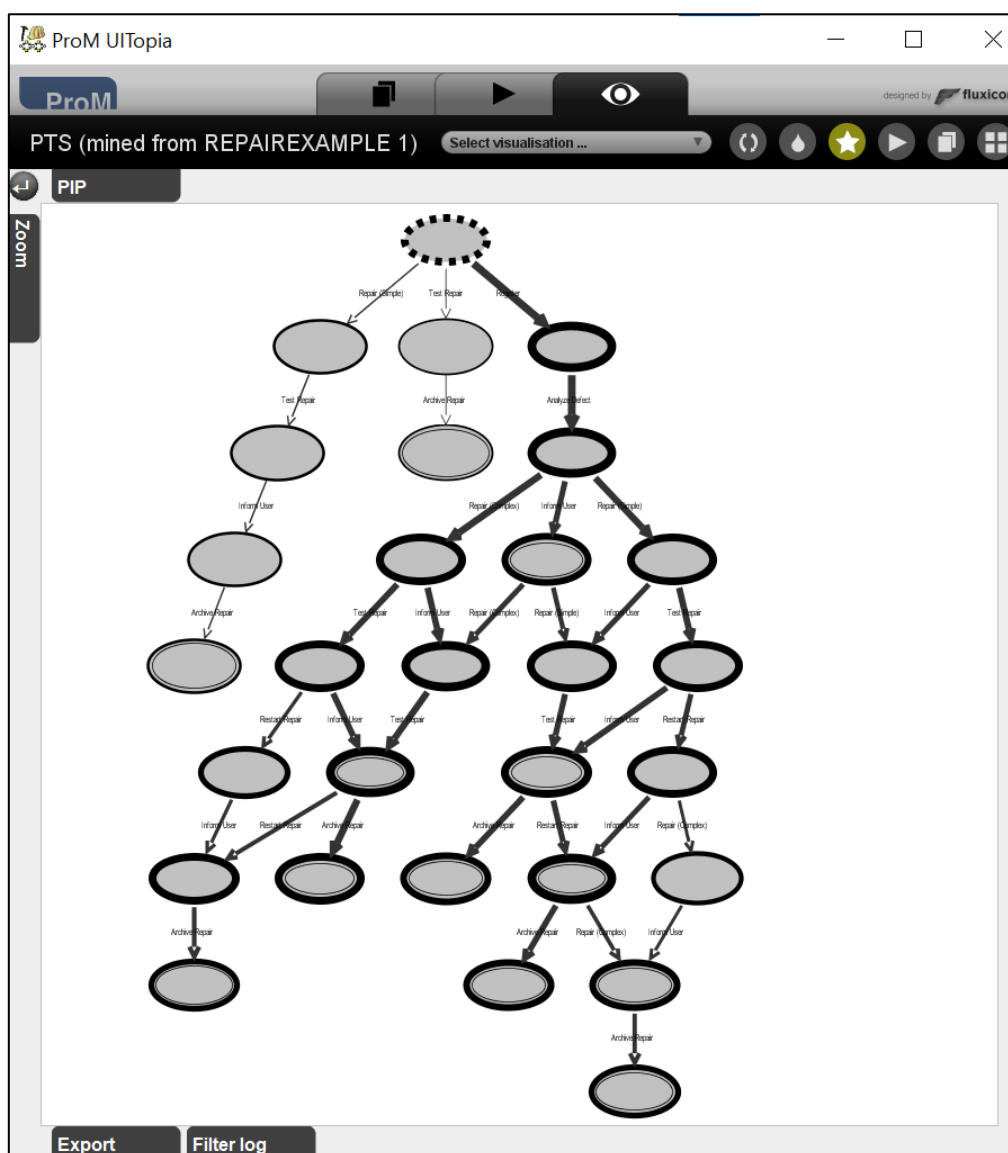


Figure 18. Constructed transition system

In line with the ProM framework, this transition system is visualized as an object in the ProM workspace (see Figure 19). With a transition system at hand, it is possible to invoke the region-based miner, a plug-in named *Convert Petri Net using Regions* (see Figure 20).

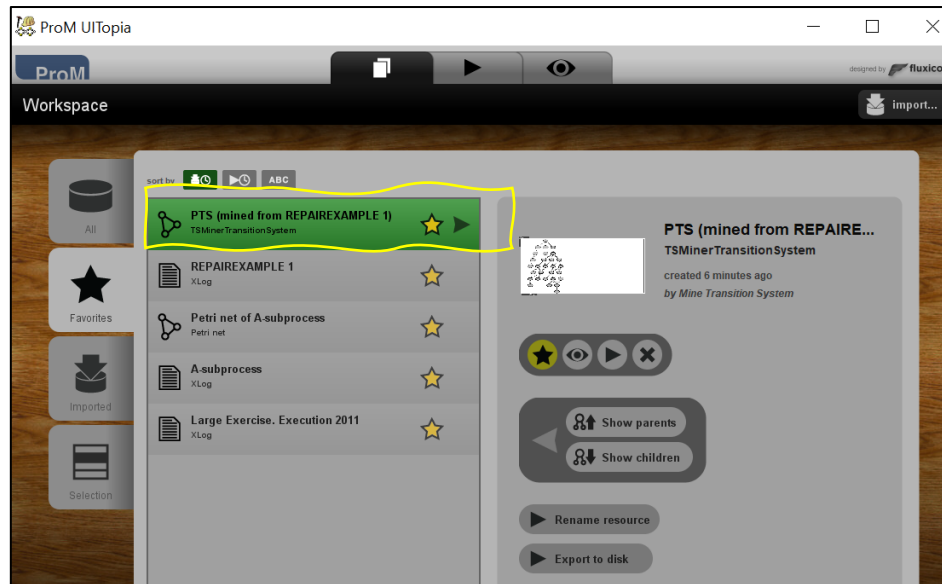


Figure 19. Transition system as an object on the workspace

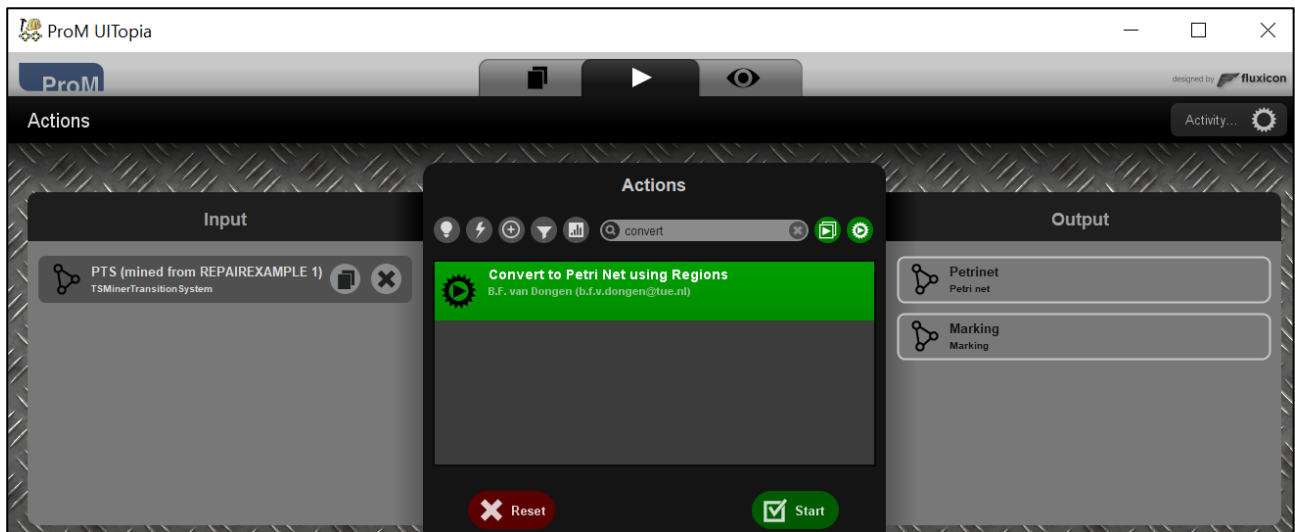


Figure 20. The action of Convert Petri Net using Regions

After clicking on *Start*, the plug-in computes the regions and generates an accordant Petri Nets, as shown in Figure 21. Two places are associated with the number 1 to indicate the presence of one token in each of the two places at the initial marking.

The first observation is that the algorithm implemented in ProM is more advanced than the basic algorithm discussed in the lecture: the basic algorithm does not allow for transition label splitting

and, hence, it is not possible to have multiple transitions that refer to the same activity (i.e. with the same label).

The algorithm implemented in ProM can have multiple transitions of the same activity.

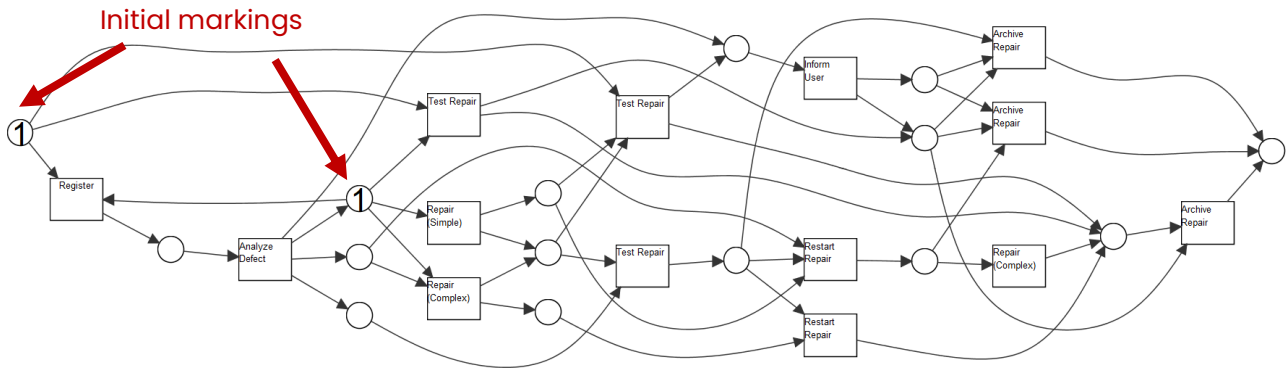


Figure 21. Petri net constructed via the Region Miner starting from the transition system

Why is the model so complex compared with that discovered via Inductive Miner (see Section 2 on page 11)?

Inductive Miner filters the infrequent behavior, whereas the Region-based Miner does not feature this functionality. Therefore, the entire behavior observed in an event log is allowed, including incomplete traces in the Region-based Miner. The inspection of the event log shows that some traces do not start with *Register*, and some end with activities other than *Archive Repair*. Recall that an event log is a dump of the system at a certain time and generally includes traces that are not completed, breaking the assumptions of discovery algorithms that traces refer to complete process executions.

After filtering the event only to retain traces that start with *Register* and end with *Archive Repair*, the resulting model is as in Figure 22. The model is now reasonable but still not as simple as that discovered by the Inductive Miner, which is among the most enhanced algorithms. On the other hand, note that the Region-based Miner does not discover the loop that the Inductive Miner can find. Last but not least, one can observe that there are multiple transitions with the same label (i.e. referring to the same process' activity). The standard discovery algorithm based on region, which we studied in the course, does not duplicate transitions with the same label. ProM features an extended version that allows discovering multiple transitions with the same label.

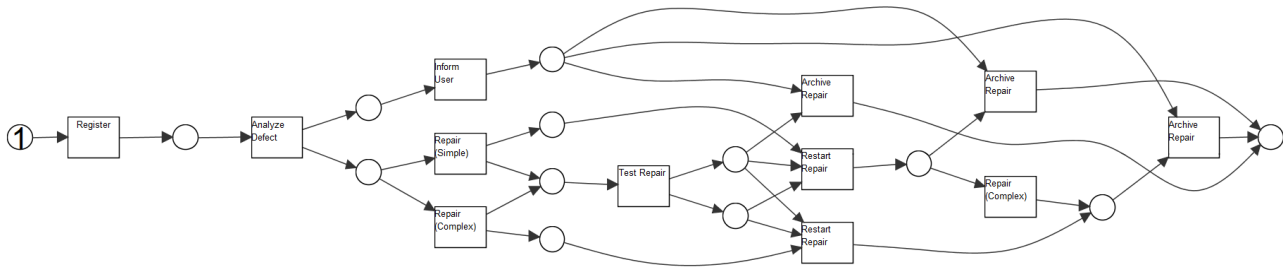


Figure 22. Petri net constructed after filtering traces starting with Register and ending with Archive Repair

Exercise

1. Verify whether or not the Petri net in Figure 22 is sound. If not, try to determine the reason why it is not sound.
2. One of the problems of the models in Figure 22 is that it does not contain the loop of restarting repairing activity, hence not generalizing sufficiently. Try to change the transition-system abstraction and see whether anything changes. Use the sequence abstraction with limit 2². This means that a state is identified by the latest two activities observed in the trace. Is the loop now present? If so or not, why?

² The reason why we add the limitation of two events is to prevent an explosion of the size of the transition system. Try to discover the transition system without limitation. The region-based discovery algorithm will run out of memory when trying to construct the resulting Petri net.

4. Discovering a model with Heuristic Miner

In this section, we will learn about the implementation of the heuristic miner, which is available through a plug-in named *Interactive Data-aware Heuristic Miner (iDHM)*, using the *repairExample.xes* event log. The first option is to select the classifier (see Figure 23). In ProM, a classifier defines which attributes of an event log are to be considered together to represent activities. *MXML Legacy Classifier* employees the pair with activity name and life cycle transition. Alternatively, you can choose the sole *Event Name* or even the *Resource* name when available.

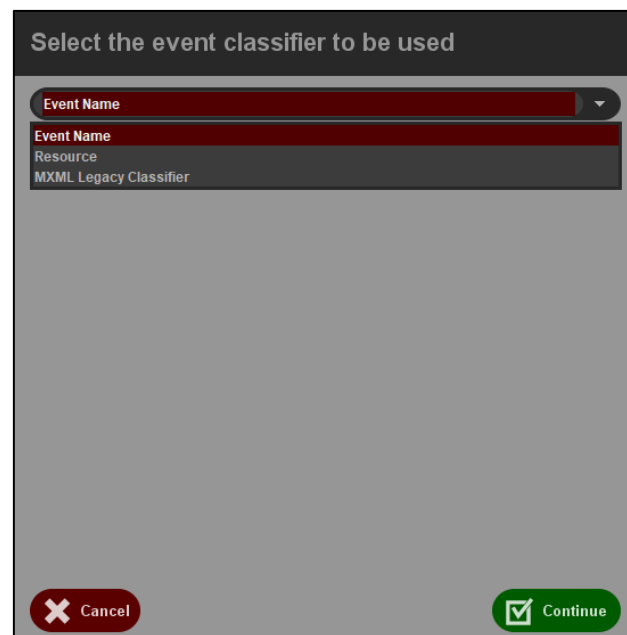


Figure 23. Selecting event classifier for Interactive Data-aware Heuristic Miner

If one chooses classifier *MXML Legacy Classifier*, the resulting model is as in Figure 24. In this case, there is a different activity for each combination of activity name and life cycle transition. Since it is likely not the result that one would desire, the plug-in should be executed again, and Event Name should be chosen as the classifier (see Figure 25).

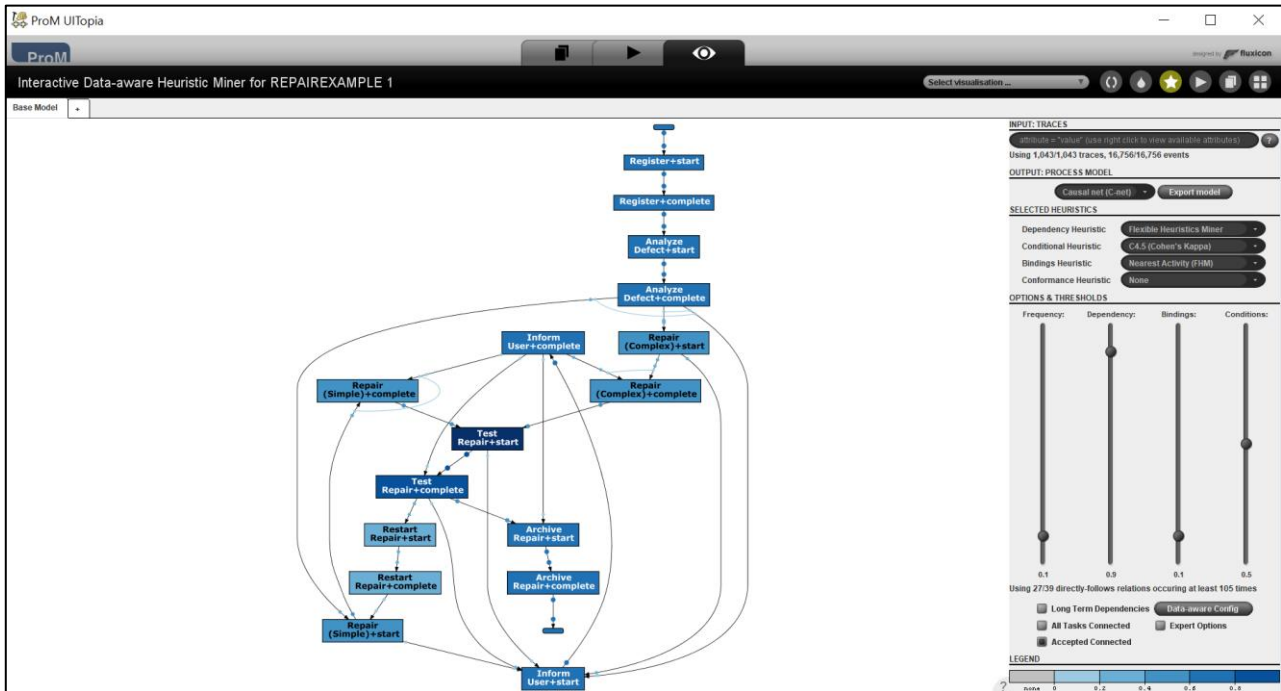


Figure 24. MXML Legacy Classifier for Interactive Data-aware Heuristic Miner

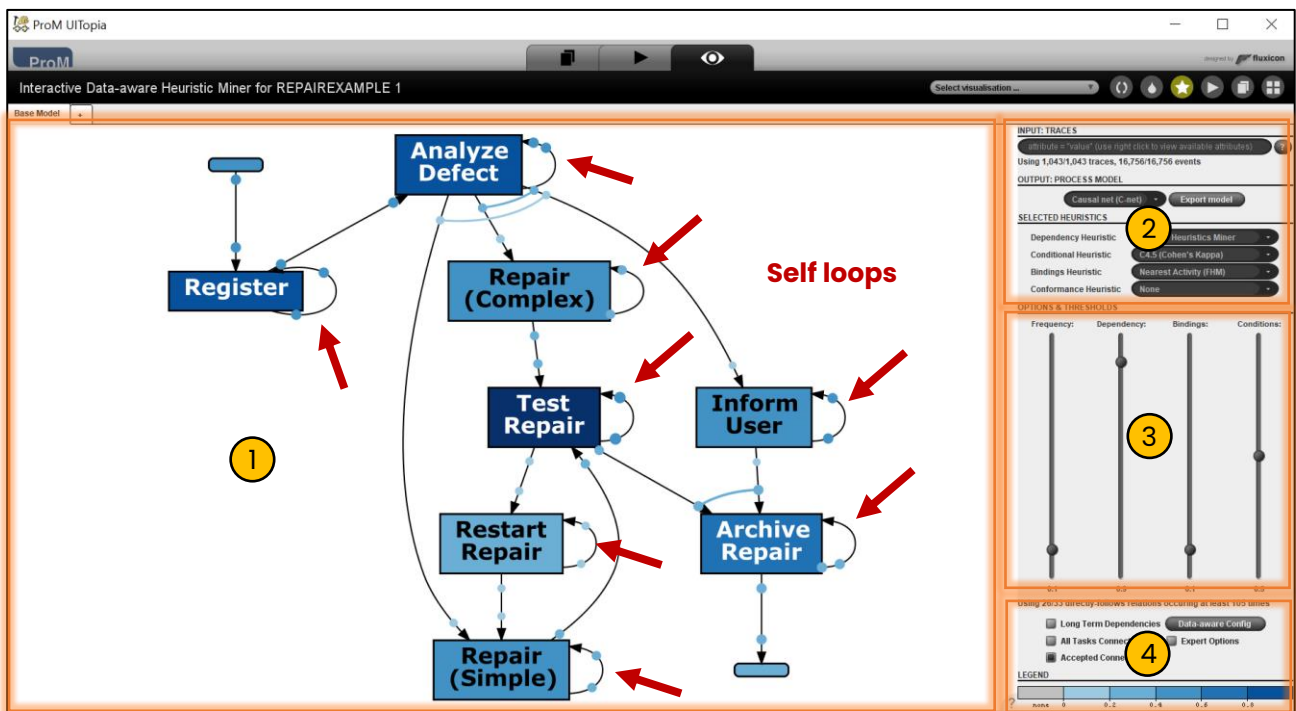


Figure 25. Event Classifier for Interactive Data-aware Heuristic Miner

Four different areas of Interactive Data-aware Heuristic Miner given in Figure 25 can be identified as follows:

Area ① includes the mined process model. Each activity is filled with a colour whose darkness indicates the frequency of occurrence of the associated activity. The darker the colour, the more frequent the activity.

Area ② can be used to decide what type of model will be visualized as an output. There are several possibilities. For this hands-on, the following options are of interest: *Causal net* (default option), *Petri Net*, *Directly-follow graph*.

If the *Casual net* or the *Petri Net* is selected as an option, the model can be exported through the button **Export Model**. In that case, the model is listed among the objects available in ProM for further inspection/visualization and/or as input for other plug-ins.

Area ③ covers the configuration settings of the heuristic miner. The *frequency and the dependency sliders* can be used to configure the heuristic-miner thresholds on the frequency and dependency relation, as discussed during the theoretical classes. Increasing the threshold would reduce the retained activity relation, thus making the model less complex but also less representative of all the observed behavior (as usual in Process Mining, one should make a trade-off between the different qualities of a model). The other two slides are irrelevant for this hands-on (the last slider will be discussed later).

The slider in Area ④ defines some additional configuration options, namely:

- ❖ *Long Term Dependencies*. Activates the standard Long-term-dependency heuristic.
- ❖ *All Tasks Connected*. Ensure that all activities are shown and connected, regardless of the thresholds. If activities were initially disconnected, the plug-in adds the minimum number of arcs to ensure full connectivity. The arcs with higher frequency and higher activity-dependency values are given priority.
- ❖ *Accepted Connected*. Only the minimum number of activities and dependencies that do not fulfill the threshold are added.

The mode contains undesired loops presented with red arrows in Figure 25. This is due to the presence of two events for each activity related to the start and complete events. To obtain a more meaningful result, we need to filter out the start events before employing the plug-in while keeping all activity names as given in Figure 26. After the log filtering for the A-subprocess given in Section 1B on page 7, the Casual Net in Figure 27 is generated.

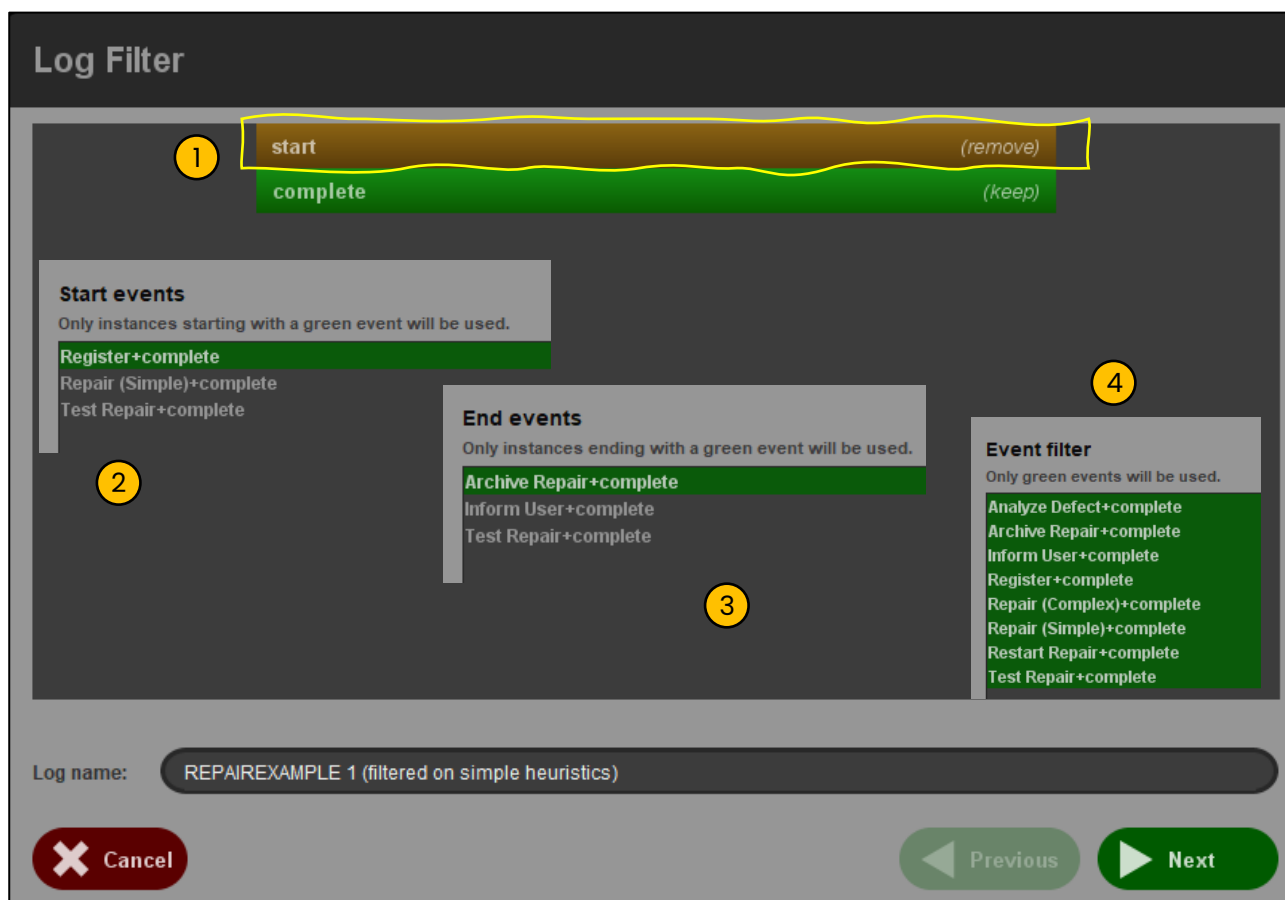


Figure 26. Filtering start event to avoid self loop by Filter log using Simple Heuristic

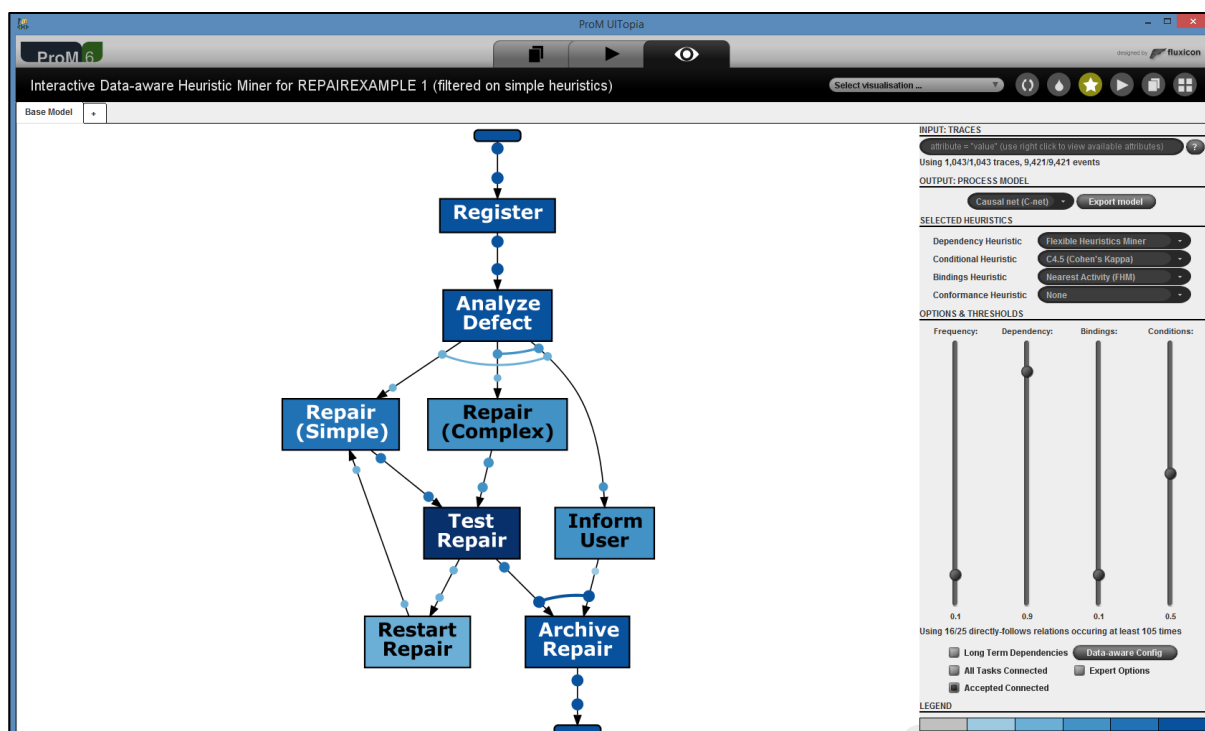


Figure 27. Casual Net mined via Interactive Data-aware Heuristic Miner, after removing self loops

Figure 28 shows the Petri net obtained through the conversion from the casual net in Figure 27 and subsequently exported into the ProM workspace. The conversion and exporting can be achieved through the controls in Area ② in Figure 25: one should select *Petri net* as *Output: Process Model*, and then click on *Export Model*.

The model in Figure 28 contains several *invisible transitions*, shown as black rectangles. Indeed, each causal-net binding is mapped onto a different invisible transition of the Petri net. Many of them are unnecessary. It is possible to have an equivalent Petri net where the unnecessary invisible transitions are removed while the behaviour (and possibly the soundness) is preserved. Reducing a Petri net to the sole necessary transition can be achieved through the plug-in *Reduce Silent Transition*. Figure 29 illustrates the resulting Petri net after removing the unnecessary invisible transitions: it's worth noting that the Petri net allows for behaviour according to which the second repair, i.e. after restarting, is usually classified – and dealt with – as simple, even when the first was complex.

Note that, while Heuristic Miner does not guarantee soundness, the model is sound for this specific example.

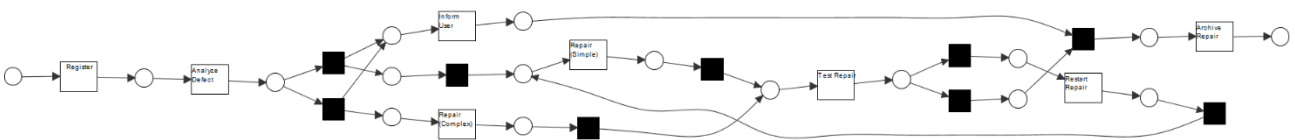


Figure 28. Exported Petri net for the Causal Net in Figure 27

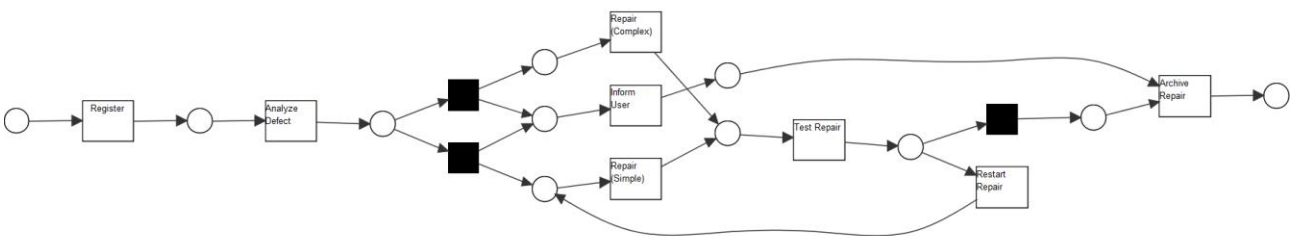


Figure 29. An equivalent Petri Net to that in Figure 28 where unnecessary invisible transitions are removed (via plug-in "Reduce Silent Transition")

5. Final Exercises

In Section 1, you created the *Application sub-process (A-subprocess)* and tried the *Alpha Miner* with little success. Now, try to employ *Inductive and Heuristic Miners* and see whether you can obtain good models. Use the event log of the financial institute (*2011.xes.gz*) to mine the process model.

1. Try to build a model that shows the three processes together and their interaction (i.e., without filtering on the activity type).
2. Try to build a model for each sub-processes, namely filtering on the events that start with A, O, and W, respectively. Have you gained more insight compared with the single model at point 1?

Solution of Exercise 1

Import event log [2011.xes.gz](#). Click Actions and select *Alpha Miner*, *Mine with Inductive Visual Miner* and *Inductive Data-aware Heuristic Miner (iDHM)*, respectively. Export all models as Petri net to be able to compare three process models. Alpha Miner already creates a Petri net. Return to the workspace and rename it "Petri net by Alpha Miner" (see [Figure 31](#)). In the inductive miner, click "export model" and choose Petri net. Return to the workspace and rename it "Petri net by Inductive Miner" (see [Figure 32](#)). In the heuristic miner, choose Petri net from the output of the process model and click the "Export model" button. Return to the workspace and rename it "Petri net by Heuristic Miner". Finally, you should have [Figure 33](#).

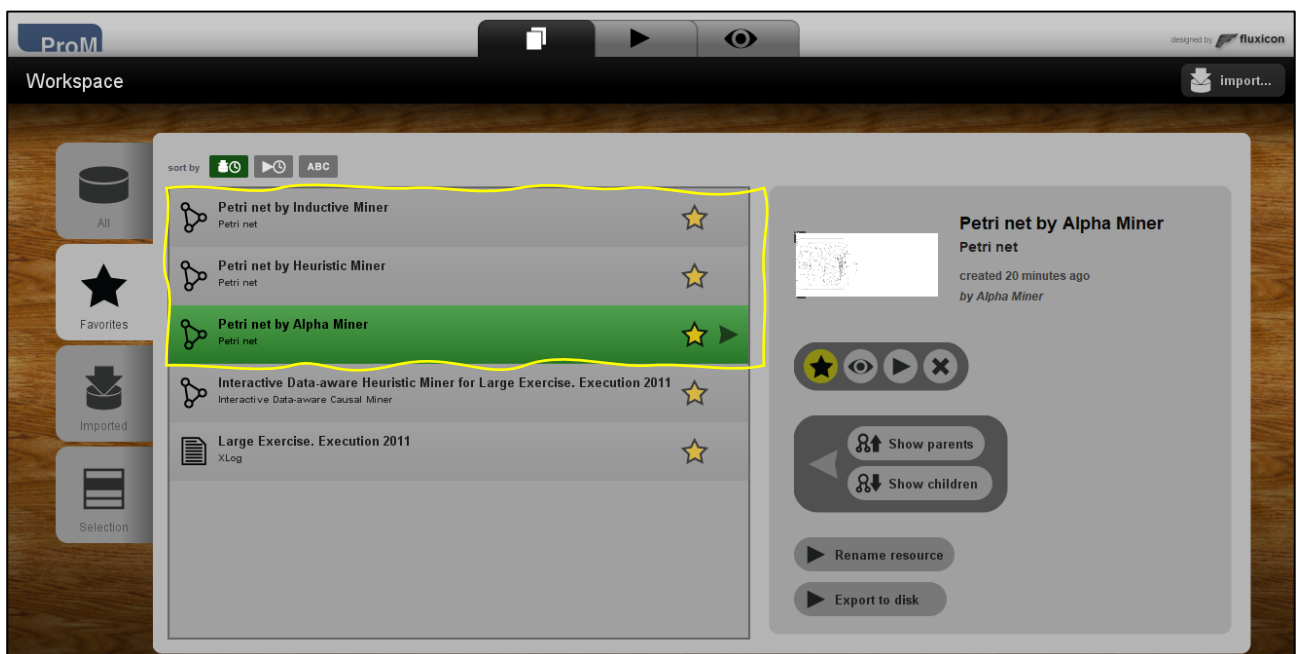



Figure 30. Three process model for 2011.xes.gz

Now you can compare all three models presented by Petri net by clicking [View overview](#)  button at the top-right corner. Alternatively, you can view the models from the workspace.

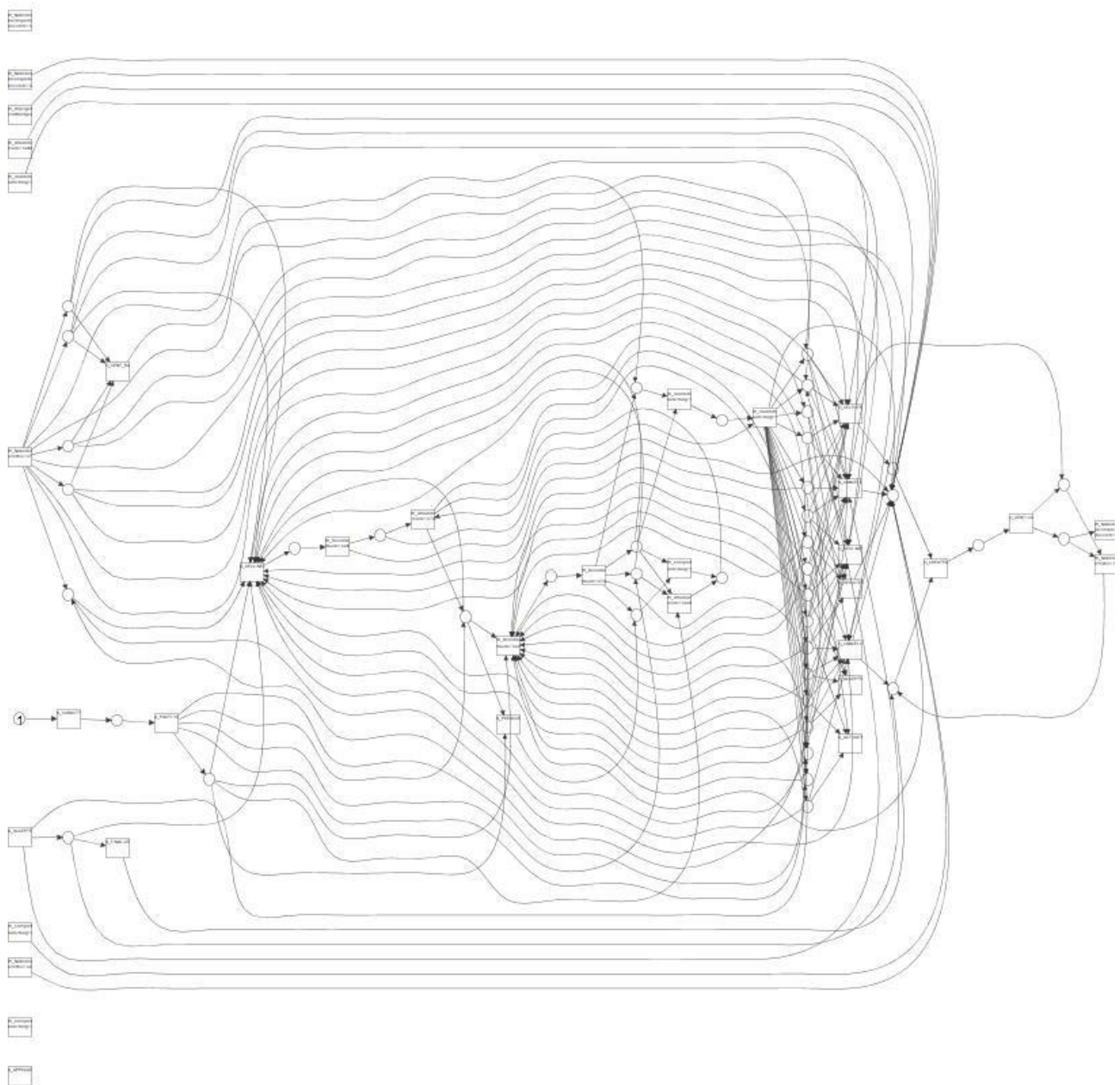


Figure 31. Process model by Alpha miner

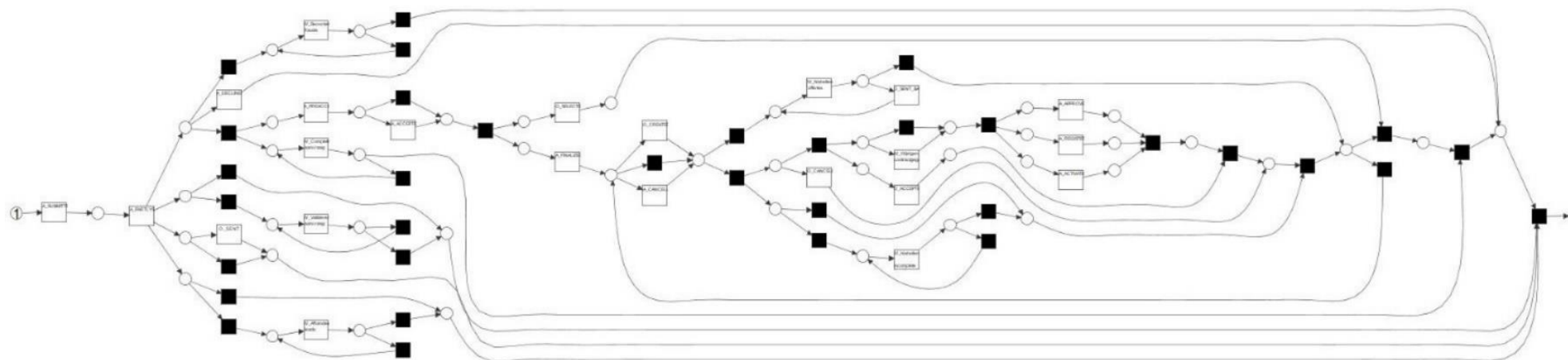


Figure 32. Process model by Inductive miner



Figure 33. Process model by Heuristic miner

Solution of Exercise 2

Apply Filter Log using Simple Heuristic to create sub-processes related to the application (A), offer (O), and internal activities (W). Do not forget to view the event log after creating because it may lead to an error in ProM while discovering process models. Change log names A-subprocess, O-subprocess and W-subprocess, respectively (see [Figure 34](#))

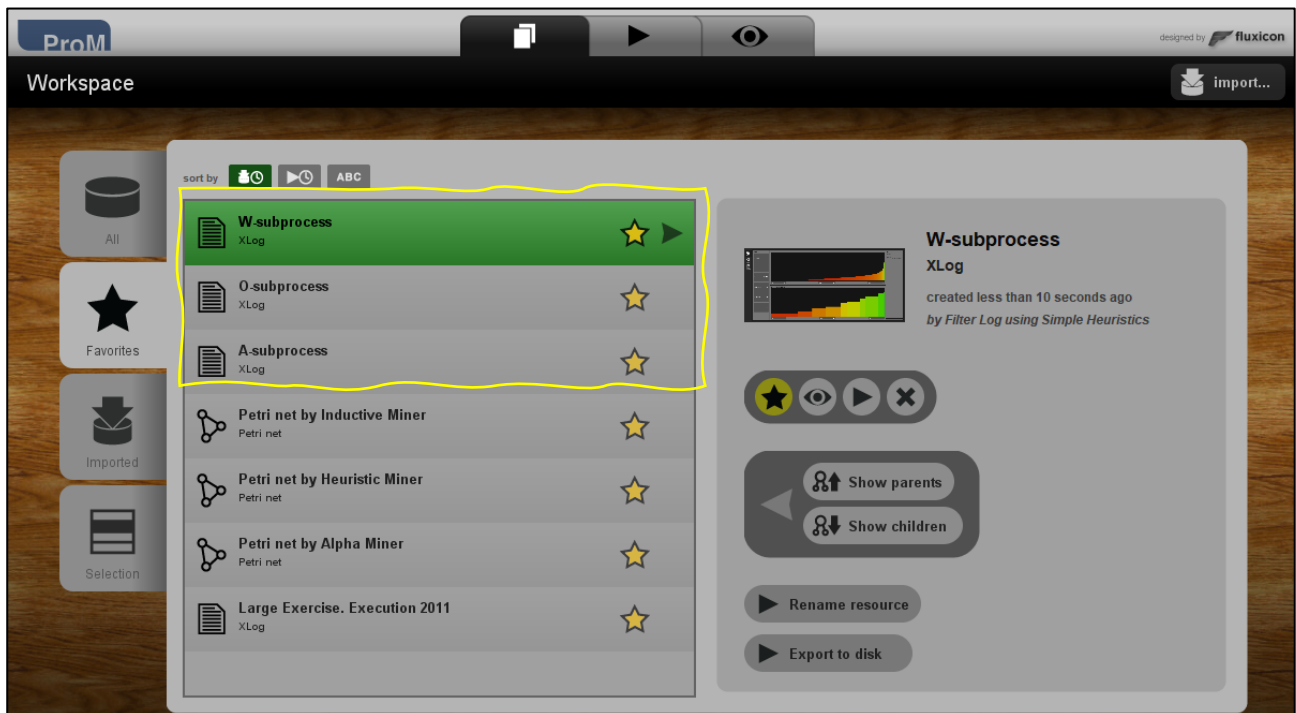


Figure 34. Three subprocess logs for 2011.xes.gz

Then create three process models for the three subprocesses again. This solution presents only the model discovered by the inductive miner for the three subprocesses.

After filtering and renaming the subprocesses, you should have the model in [Figure 35](#).

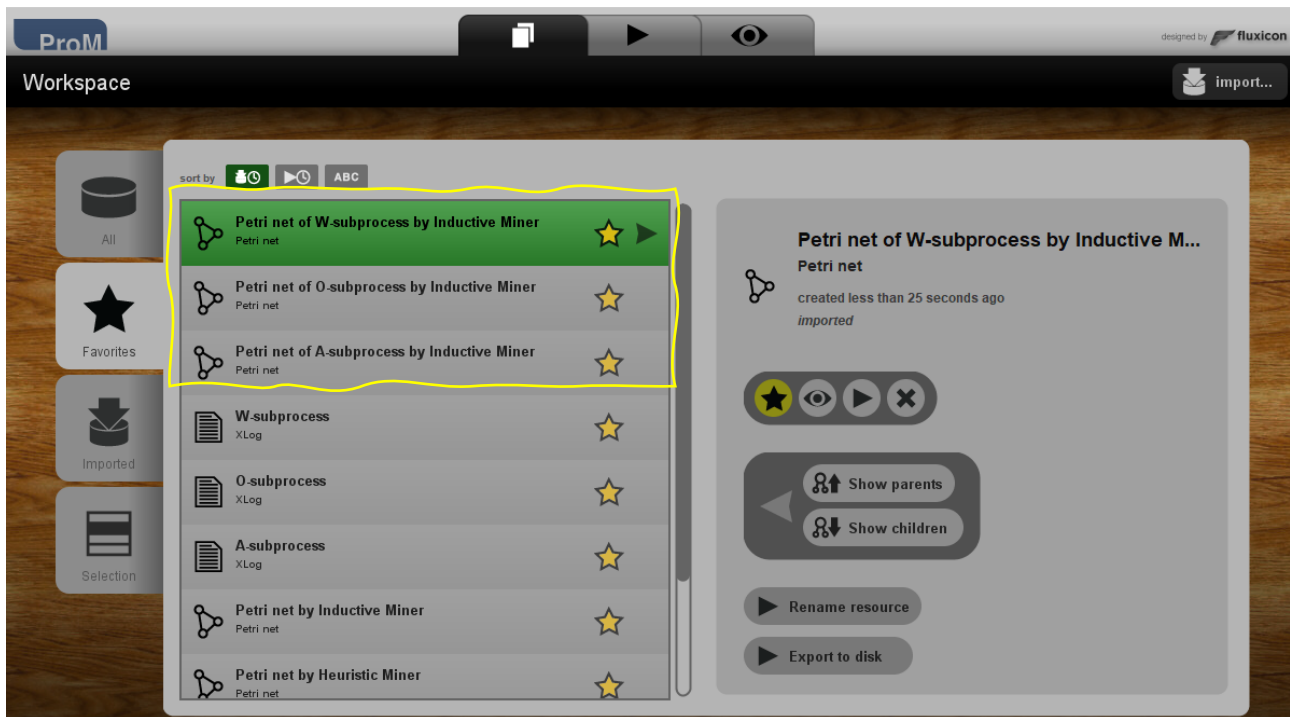


Figure 35. Process model by Inductive miner for three subprocesses

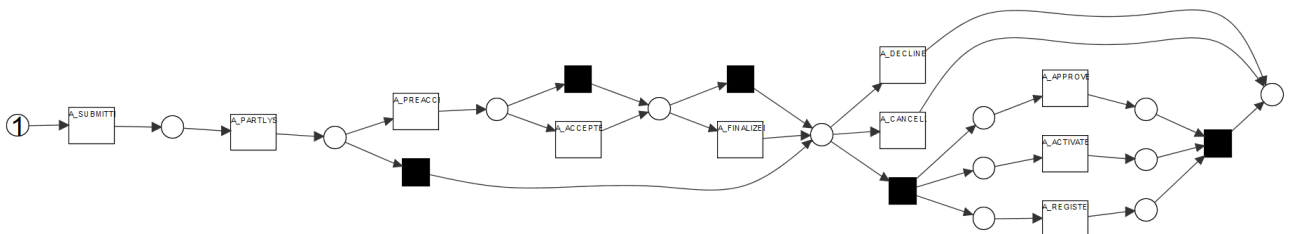


Figure 36. Process model by Inductive miner for A-subprocess

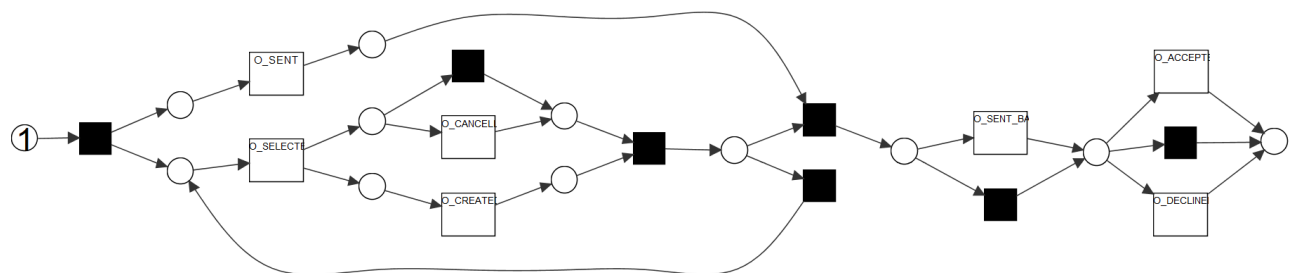


Figure 37. Process model by Inductive miner for O-subprocess

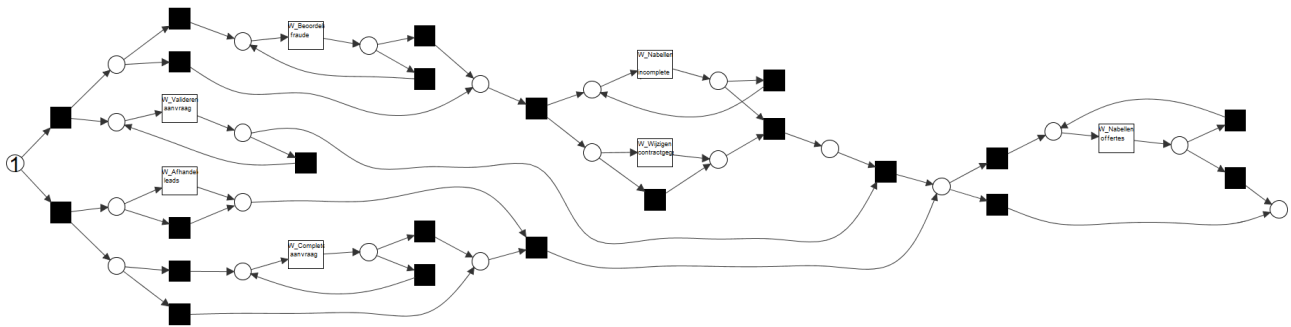


Figure 38. Process model by Inductive miner for W-subprocess

As you can see, the single models, including A, O and W subprocesses, gave us more insight than the whole event log. In fact, the process is the interleaving of three processes, beforehand named “subprocesses”, instead of one single process. The discovery algorithms always assume one single process of which to discover the model. Mining the three processes together and their interleaving break this assumption.