

# Process Mining: Using CPN Tools to Create Test Logs for Mining Algorithms

A.K. Alves de Medeiros and C.W. Günther

Department of Technology Management, Eindhoven University of Technology  
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.  
{a.k.medeiros,c.w.gunther}@tm.tue.nl

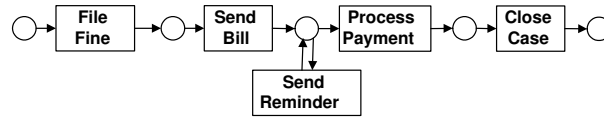
**Abstract.** Process mining aims at automatically generating process models from event logs. The main idea is to use the discovered models as an *objective start point* to deploy systems that support the execution of business processes (for instance, workflow management systems) or as a *feedback mechanism* to check if the prescribed models fit the executed ones. When developing an algorithm to do process mining, one needs some logs to test it. Using real-life logs seems to be the natural choice. However, the real-life event logs usually contain imperfections that can hinder the tuning of the mining algorithm. For instance, real-life logs can be incomplete and/or contain noise. Thus, a more common approach is to first test the accuracy of new process mining algorithms in logs created via simulation. This allows the researcher to have more control about the properties of the event log and to fine tune his/her mining algorithm. Besides, having the original model (the simulated one) may also be a useful aid to assess the quality of the mining algorithm. In our research group, we work with the ProM framework [6] mining tool which receives as input an XML event log. This XML format is also supported by process mining tools of other research groups [4]. This paper shows how to extend CP-nets to generate XML event logs that can be mined by process mining tools supporting this format. This way we benefit from the simulation capabilities of CPN Tools and, therefore, we avoid reinventing the wheel. The extension we made consisted of implementing (i) some ML functions that can be used to annotate the CP-net, and (ii) a ProM<sub>import</sub>-framework [2] plug-in that bundles up the files (generated by the CP-net simulation) into a single XML file that is ready to be mined.

## 1 Introduction

Process mining targets the *automatic* discovery of information from an event log. This discovered information can be used to deploy new systems that support the execution of business processes or as a feedback tool that helps in analyzing and improving enacted business processes. The starting point of any process mining technique is an event log. This log can have lots of data, such as the tasks that are executed, their time of execution, the person/system that performed them, the data fields related to these tasks, and so forth. For instance, consider the event log in Table 1. This log has four executions (cases) of a process that handles fines for drivers in a certain province.

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

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



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

The amount of data in the event log determines which *perspectives* of process mining can be discovered. If the log provides the tasks that are executed in the process and it is possible to infer their order of execution, the *control-flow perspective* can be mined. The log in Table 1 has this data (cf. fields “Case ID”, “Task Name” and “Timestamp”). So, for this log, mining algorithms could discover the process in Figure 1. Basically, the process describes that after a fine is entered in the system, the bill is sent to the driver. If the driver does not pay the bill within one month, a reminder is sent. When the bill is paid, the case is archived. If the log provides information about the persons/systems that executed the tasks, the *organizational perspective* can be discovered. The organizational perspective discovers information like the social network in a process, the transferring of work etc. For instance, the log in Table 1 shows that “Anne” transfers work for both “Mary” (case 2) and “John” (cases 3 and 4), and “John” sometimes transfers work for “Mary” (case 4). Besides, by inspecting the log, the mining algorithm could discover that “Mary” never has to send a reminder more than once, while “John” does not seem to perform as good. The managers could talk to “Mary” and check if she has another approach to send reminders that “John” could benefit from. This can help in making good practices a common knowledge in the organization. When the log contains more details about the tasks, like the values of data fields that the execution of the task modifies, the *case perspective* can be discovered. So, for instance, a forecast for executing cases can be made based on previous already completed cases, exceptional situations can be discovered etc. In our particular example, logging information about the profiles of drivers (like age, gender, car etc) could help in assessing the probability that they would pay their fines on time. Moreover, logging information about the places where the fines were applied could help in improving the traffic measures in these places.

Having these three perspectives in mind, and the different mining tools<sup>1</sup> to tackle one or more of these perspectives, an effort was made in [4] to define a single XML format, called the Mining XML (MXML) format, that could be used as input to the different tools. By converting simulated or real-life logs to the MXML format, one could use the mining techniques in multiple contexts. Therefore, we test our mining algorithms with logs that have the MXML format.

Real-life logs typically contain some subset of data fields that allow for mining. However, real-life logs are not always the best data to first test a mining algorithm that is being developed because real-life logs may be incomplete and/or contain noise. So, a better approach to test mining algorithms is to use simulated data. Note that the use of simulated data gives the researcher more control over event log properties. This more controlled environment may help with tuning the

---

<sup>1</sup> Examples of mining tools are InWolvE [8,10], Process Miner [11], EMiT [7], Little Thumb [12], MiSoN [3] and ProM framework [6]. The InWolvE, Process Miner, EMiT and Little Thumb mine the control-flow perspective. The MiSoN mines the organizational perspective. The ProM framework has plug-ins that to mine all three perspectives. Actually, the mining tools EMiT, MiSoN and Little Thumb were respectively implemented as the ProM mining plug-ins “Alpha algorithm”, “Social network miner” and “Heuristics miner”.

algorithm under development. After being able to correctly mine simulated data, the algorithm can be tested with real-life logs.

CPN Tools supports the modelling, execution and analysis of *Coloured Petri nets* (CP-nets) [9]. Additionally, there is a fair amount of CPN models that can be used as input to test mining algorithms. Thus, we decided to extend CPN Tools to support the creation of MXML logs. The main idea is to create random MXML logs by simulating CP-nets in CPN Tools. The extension is fairly simple and took us no more than twelve man-hours to finish it. The first part of the extension consisted of implementing the ML functions to support the logging from a CP-net. The second part consisted of implementing the “CPN Tools” plug-in in the the ProM<sub>import</sub> framework [2] to bundle the logged files into a single MXML file.

In short, two steps are necessary to create MXML logs using CPN Tools:

1. Modify a CP-net to invoke the set of ML functions that will create logs for every case executed by the CP-net. This step involves modifying the *declarations* of the CP-net and the *input/output/action* transition inscriptions.
2. Use the CPN Tools plug-in, in the ProM<sub>import</sub> framework, to group the logs for the individual cases into a single MXML log.

The rest of this paper is organized as follows. Section 2 explains the MXML format. Understanding how this format supports the different perspectives of mining helps in understanding how CPN Tools was extended. Section 3 describes how to modify a CP-net to create partial MXML logs during its simulation (Step 1 above). Section 4 shows how to use the ProM<sub>import</sub> framework to bundle these partial MXML logs into a single log that can be mined (Step 2 above). Section 5 presents some conclusions and future work.

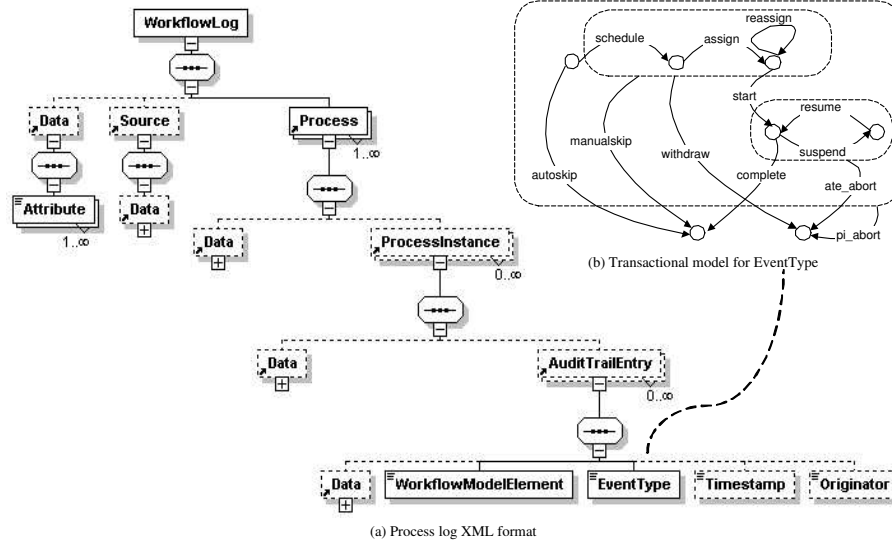
## 2 The MXML format

The Mining XML<sup>2</sup> format (MXML) started as an initiative to make different mining tools have a common input format [4]. This way, event logs could be shared among different mining tools. Actually, defining a common input format like MXML is the first step towards the creation of a repository on which process mining researchers can test their algorithms. In this section we explain the MXML format because this helps to understand the ML functions defined for the extension of CP-nets. The schema for this MXML format (depicted in Figure 2) is available at <http://www.processmining.org/WorkflowLog.xsd>.

As can be seen in Figure 2, an event log (field *WorkflowLog*) has the execution of one or more processes (field *Process*), and optional information about the source program that generated the log (field *Source*) and additional data elements (field *Data*). Every process (field *Process*) has zero or more cases or process instances (field *ProcessInstance*)<sup>3</sup>. Similarly, every process instance has zero or more tasks (field *AuditTrailEntry*). Every task or audit trail entry (ATE) should at least

<sup>2</sup> More information about the Extensible Markup Language (XML) can be found in [1].

<sup>3</sup> In the rest of this document, the words “execution”, “case” and “process instance” are interchangeable.



**Fig. 2.** The visual description of the schema for the Mining XML (MXML) format.

have a name (field *WorkflowModelElement*) and an event type (field *EventType*). The event type determines the state of the tasks. There are 13 supported event types: schedule, assign, reassign, start, resume, suspend, autoskip, manualskip, withdraw, complete, ate\_abort, pi\_abort and unknown. The other task fields are optional. The *Timestamp* field supports the logging of time for the task. The *Originator* field records the person/system that performed the task. The *Data* field allows for more logging of additional information. More details about the MXML format can be found in [6].

Mapping the MXML format to the three mining perspectives, we see that the *control-flow* perspective mainly focuses on the *WorkflowModelElement*, the *EventType* and the *Timestamp*<sup>4</sup> fields. The *organizational* perspective chiefly depends on the *Originator* field. The *case* perspective especially relies on the extra *Data* fields.

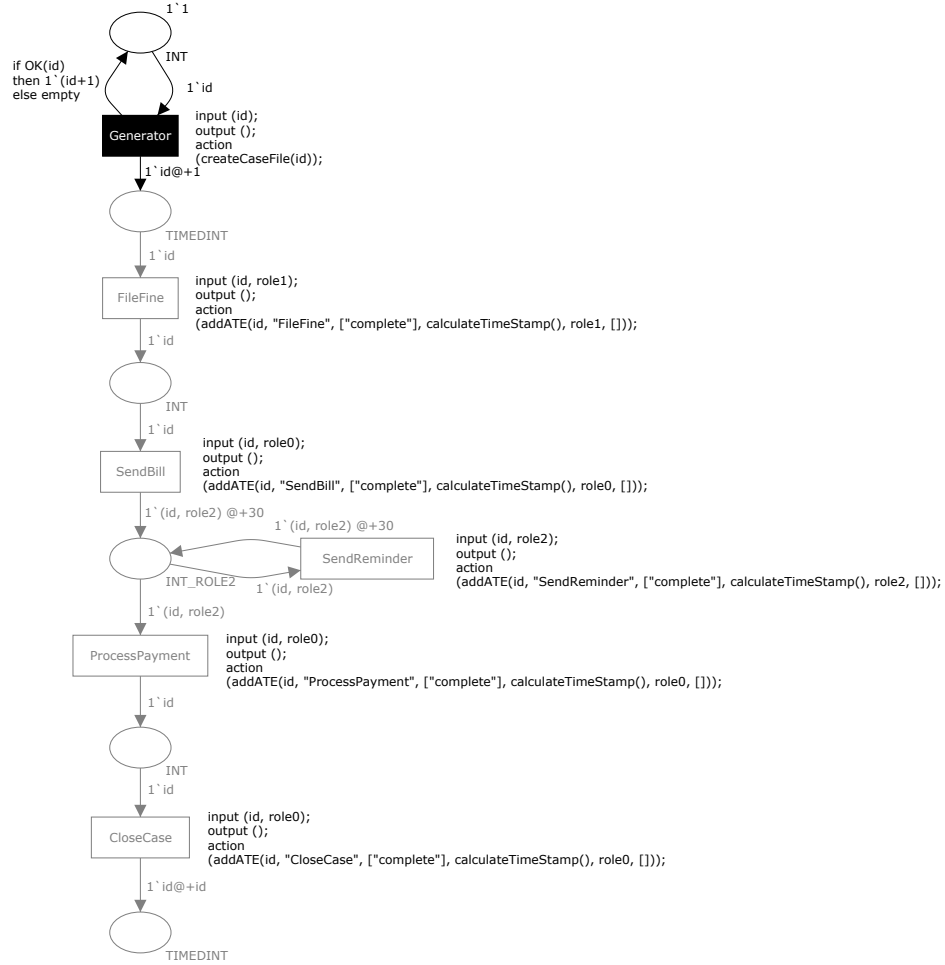
Note that in CPN Tools the process corresponds to the CP-net, the tasks (or ATEs) are the transitions in the CP-net, and each simulation of the CP-net corresponds to the creation of a process instance.

### 3 Extending a CP-net to Produce MXML Event Logs

This section shows how to annotate a CP-net with the ML functions that we created to log MXML files. The ML functions can be downloaded from the section

<sup>4</sup> When the *Timestamp* field is not logged, the sequence in which the tasks appear in the log is used to infer their order of execution.

“Tools”<sup>5</sup> in [2]. To illustrate the extension process, we use the CP-net in Figure 3. The extension of this CP-net involves editing its declaration and transition inscriptions.



**Fig. 3.** Example of an extended CP-net for the process described in Figure 1. The highlighted transition “Generator” and the input/output/action inscriptions were added during extension of the CP-net.

**CPN Declarations** The declarations of a CP-net need to be modified to import the ML functions to log transitions. These functions are in the file *logging-FunctionsMultipleFiles.sml*. The ML functions in this file use two constants:

<sup>5</sup> Search for the link to the file “CPNToolsConverter.zip”.

*FILE* and *FILE\_EXTENSION*. The constant *FILE* sets the location and the name prefix of the MXML files that the CP-net will create for every case it executes. The constant *FILE\_EXTENSION* sets the extension that these created files have. For instance, to (1) create the XML log files for every case at the subdirectory *logs* from the directory where the CP-net is located and name every log with the prefix *logsCPN*; and (2) assign the extension *.cpnxml* to every created log, the following should be declared:

1. *val FILE = “./logs/logsCPN”*
2. *val FILE\_EXTENSION = “.cpnxml”*
3. *use “loggingFunctionsMultipleFiles.sml”;*

Note that the use of the file *loggingFunctionsMultipleFiles.sml* **must be declared after** declaring the constants *FILE* and *FILE\_EXTENSION*. Table 3 shows what these declarations look like in the CP-net of Figure 3. Additionally, be aware that ML is case sensitive and the subdirectories provided in the constant *FILE* should already exist.

```
(* Standard declarations *)
colset E = with e;
colset INT = int;
colset BOOL = bool;
colset STRING = string;
(* Net declarations *)
colset TIMEDINT = int timed;
colset ROLE0 = subset STRING with [“system”];
colset ROLE1 = subset STRING with [“Anne”];
colset ROLE2 = subset STRING with [“Mary”, “John”];
colset INT_ROLE2 = product INT * ROLE2 timed;
var id: INT;
var role0: ROLE0;
var role1: ROLE1;
var role2: ROLE2;
fun OK(id) =
  if id < 1000 then true
  else false;
(* Log declarations *)
val FILE = “./logs/logsCPN”
val FILE_EXTENSION = “.cpnxml”
use “loggingFunctionsMultipleFiles.sml”;
```

**Table 2.** Declarations for the CP-net in Figure 3. The declarations in **bold** were used to extend the model to log MXML files.

**CPN Transitions** Once the declarations of the CP-net have been updated, the *input/output/action* inscriptions of transitions can be modified to invoke the

logging functions. The CP-net will create a partial MXML log for *every* case that it executes. In the example in Figure 3, the transition *Generator* creates the unique case identifiers. After a partial MXML log has been created, the transitions (or tasks) executed for a case are written to its partial MXML log. Thus, two ML functions are provided: `createCaseFile` and `addATE`.

The function `createCaseFile(int caseld)` opens the log file for a case. This function should be invoked only *once per case*, and before the function `addATE` is invoked for this same case. The transition *Generator* in Figure 3 illustrates how to use the function `createCaseFile`. Note that this function receives an *integer* (the case identifier!) as input.

The function `addATE(int caseld, String transitionName, ListOfStrings eventType, StringTimestamp timestamp, String originator, ListOfStrings data)` logs the execution of a transition to the log of a case. For instance, the transition *FileFine* in Figure 3 has an invocation of this function. The parameters of the function `addATE` are:

1. **caseld**: *integer* that uniquely identifies a case. In Figure 3, the case id is given by the variable *id*.
2. **transitionName**: *string* that has the name of the transition to log. Note that all strings in ML should be in quotes (""). In Figure 3, the task name for transition *FileFine* is "*FileFine*".
3. **eventType**: *list of strings*. If the event type is supported, the list should contain a *single* element and have the format [**name**], where *name* in {"assign", "withdraw", "reassign", "start", "suspend", "resume", "complete", "autoskip", "manualskip", "pi\_abort", "ate\_abort"}. If the event type is *unknown*, this list should have *two* elements and the format ["**unknown**", "**name**"], where *name* is the unknown event type name. In Figure 3, the event type for transition *FileFine* is ["complete"].
4. **timestamp**: *string* that represents the date and time in which the task was executed. The **timestamp** has the XML pre-defined format *dateTime* [5]. For instance, a valid timestamp string is "2005-06-30T14:55:00.000+01:00". The function `calculateTimeStamp()` is provided to automatically calculate the timestamp field based on the current time (in minutes) of a CP-net. The starting date of the function `calculateTimeStamp()` is the starting date of Unix (1-1-1970). The function `calculateTimeStamp()` is included in the file *loggingFunctionsMultipleFiles.sml*. In Figure 3, the function `calculateTimeStamp()` was used to provide the timestamp.
5. **originator**: *string* that has the name of the originator (person or system) that executed the transition. In Figure 3, the user with *role1* executed the task *FileFine*.
6. **data**: *list of strings* containing the additional data fields that may be associated to a task. This list must have the format [attributeName1, attributeValue1, attributeName2, attributeValue2, ..., attributeNameX, attributeValueX]. In Figure 3, no extra data is associated to the task *FileFine*.



The parameters `timestamp`, `originator` and `data` can be empty (see optional fields in Figure 2). The first two are empty if the string “” is given as input. The `data` parameter is empty when [] is given as input.

The simulation of the extended CP-net will create the partial MXML log files whose aggregation is described in the next section.

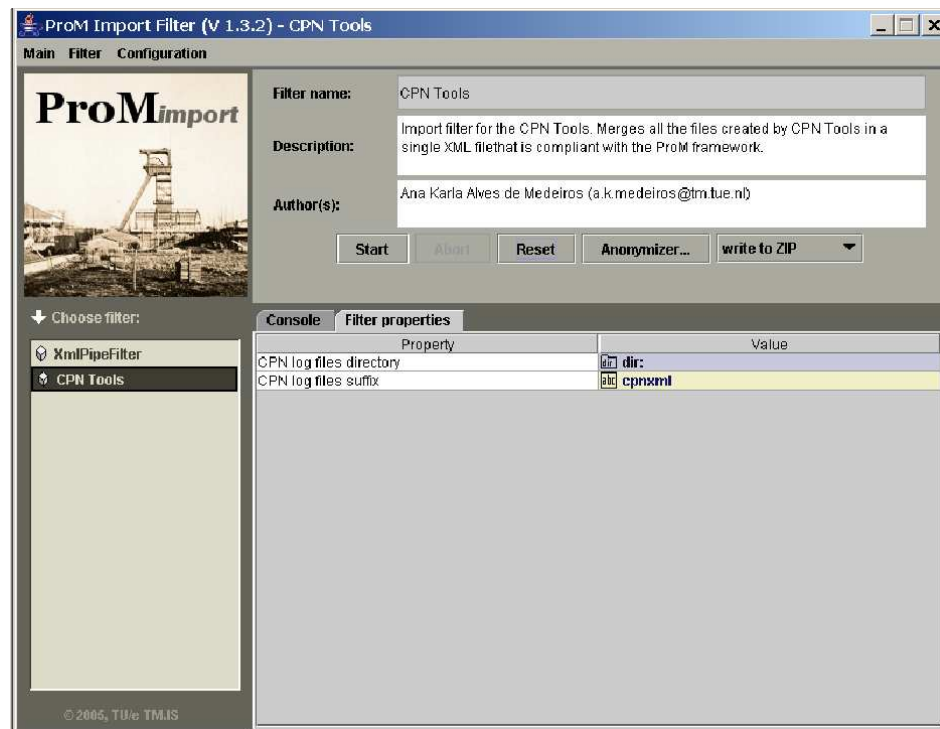
## 4 Final Log Aggregation

Once the CP-net has been simulated with logging extensions enabled, the generated log output has to be correctly aggregated and converted, such that it can be read and interpreted by mining tools like the ProM framework. As explained in Section 3, simulating a CP-net with logging extensions included and enabled will yield one log file per invocation of `createCaseFile`, including all logged events. The task of log aggregation is now to combine these files as process instances within one single MXML log file, representing all logs for the respective process model (i.e., the simulated CP-net).

This aggregation pass has been implemented as the “CPN Tools” import plug-in for the ProM<sub>import</sub> framework [2]. This framework has been developed to serve as a common environment for converting and importing logs from all kinds of information systems, and subsequently creating MXML compliant log files from them. The actual procedures for importing logs from a specific source system can be implemented as plug-ins, which can be dynamically loaded and removed from the running framework. The framework has a common graphical user interface for configuring and controlling import plug-ins, keeps all configuration data persistent, and provides a set of useful classes which can be used by all import plug-ins in order to ease development.

As the implemented ML functions `createCaseFile` and `addATE` (cf. Section 3) write MXML compliant log fractions during simulation runs, all that is left to do is to generate a common enclosing log file. For each file created by the simulation runs, a process instance is created within that common log file. In this process instance, all log events from one input file are added in their given order. In the end, the output log file thus includes all data from the simulation run logs in an aggregated manner, ready to be analyzed by mining tools like the ProM framework.

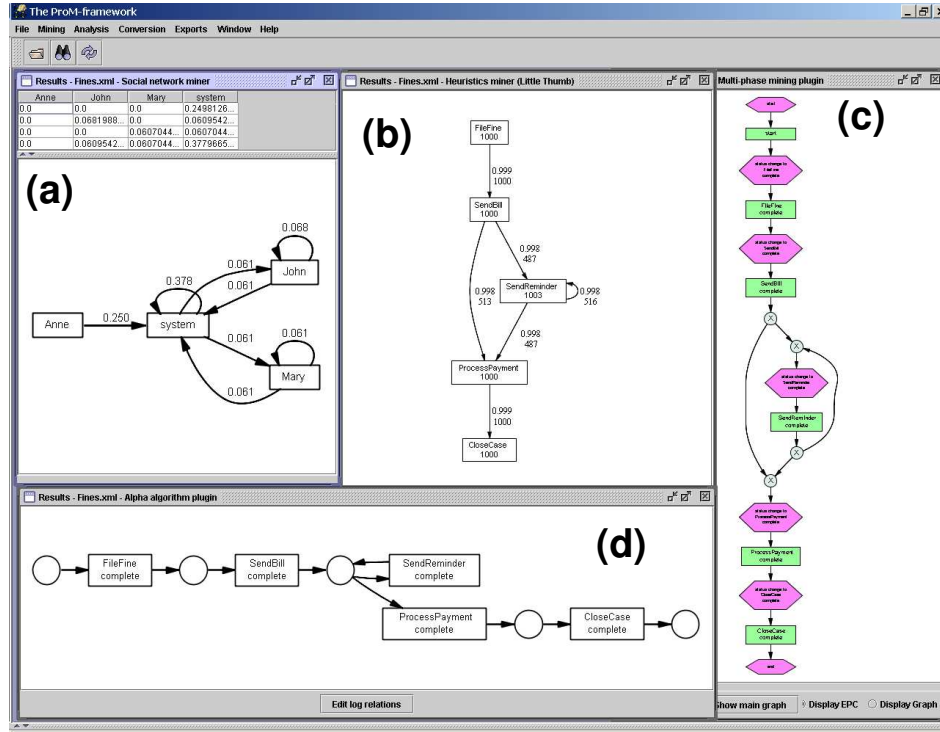
Using the “CPN Tools” import plug-in is fairly straightforward: The configuration pane (see Figure 4) has two filter properties that allow the user to select the input directory for the partial MXML files and the suffix of these files. After running the simulation passes in CPN Tools, the files created for each trace of one process model can be selected here. As this plug-in is geared towards aggregating logs that have resulted from executing one process model, it will accordingly write one single output log file. In this, the import framework provides the choice between writing the log into a compressed ZIP file, or as plain XML file. In either case, the resulting file can subsequently be loaded from within, for instance, the ProM framework, where a multitude of Process Mining plug-ins are available for analysis.



**Fig. 4.** Screenshot of the CPN Tools plug-in. This plug-in is implemented in the *ProM<sub>import</sub>* framework.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <WorkflowLog xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://tmitwww.tm.tue.nl/research/processmining/WorkflowLog.xsd" description="Log
  extracted from CPN Tools">
  <Source program="CPN Tools" />
  - <Process id="CpnToolsLog" description="Log file created in CPN Tools">
    - <ProcessInstance id="1" description="">
      - <AuditTrailEntry>
        <WorkflowModelElement>FileFine</WorkflowModelElement>
        <EventType>complete</EventType>
        <Timestamp>0000-01-01T00:01:00.000+01:00</Timestamp>
        <Originator>Anne</Originator>
      </AuditTrailEntry>
      - <AuditTrailEntry>
        <WorkflowModelElement>SendBill</WorkflowModelElement>
        <EventType>complete</EventType>
        <Timestamp>0000-01-01T00:01:00.000+01:00</Timestamp>
        <Originator>system</Originator>
      </AuditTrailEntry>
      - <AuditTrailEntry>
        <WorkflowModelElement>ProcessPayment</WorkflowModelElement>
        <EventType>complete</EventType>
        <Timestamp>0000-01-01T00:31:00.000+01:00</Timestamp>
        <Originator>system</Originator>
      </AuditTrailEntry>
      - <AuditTrailEntry>
        <WorkflowModelElement>CloseCase</WorkflowModelElement>
        <EventType>complete</EventType>
        <Timestamp>0000-01-01T00:31:00.000+01:00</Timestamp>
        <Originator>system</Originator>
      </AuditTrailEntry>
    </ProcessInstance>
    - <ProcessInstance id="10" description="">
      - <AuditTrailEntry>
        <WorkflowModelElement>FileFine</WorkflowModelElement>
        <EventType>complete</EventType>
        <Timestamp>0000-01-01T00:01:00.000+01:00</Timestamp>
        <Originator>Anne</Originator>
      </AuditTrailEntry>
      - <AuditTrailEntry>
        <WorkflowModelElement>SendBill</WorkflowModelElement>
```

**Fig. 5.** Excerpt of the MXML log that was aggregated for a simulation of the CP-net in Figure 3.



**Fig. 6.** Screenshot of the mined Petri net for the in Figure 5. The shown mining plug-ins are: (a) *Social network miner*, (b) *Heuristics miner*, (c) *Multi-phase mining* and (d) *Alpha algorithm*. The *Social network miner* plug-in can mine the *organizational perspective* (cf. Section 1) of an event log. Here we show the *handover of work* setting, considering only direct succession. Note that the users “John” and “Mary” never transfer work to each other. This is compatible with the CP-net in Figure 3. The other plug-ins in this figure can mine the *control-flow perspective* of the event log. As expected, all mined control-flow structures are like the CP-net in Figure 3.

As an illustration, Figure 5 shows an excerpt of the MXML log that was aggregated for the simulation of the CP-net in Figure 3. Additionally, Figure 6 shows a screenshot with the results of applying four different ProM mining plug-ins to this MXML log.

## 5 Conclusions and Future Work

This paper demonstrates how to benefit from the CPN Tools simulation capabilities to build event logs that can be used in the process mining research. The extension to CPN Tools consisted of implementing (i) a set of ML functions to create log files and (ii) a ProM<sub>import</sub> plug-in. The ML functions can be called from the input/output/action transition inscriptions of a CP-net. When the CP-net is simulated, partial logs are created. These partial logs are bundled into a single MXML log by using the ProM<sub>import</sub> CPN Tools plug-in. The resulting log can be mined by mining tools like the ProM framework. All the tools/files necessary for extending a CP-net to create MXML logs can be found at <http://www.processmining.org>.

As future work, we are interested in creating a repository for MXML logs. This repository will contain logs with different properties. In addition, such a repository will allow researchers to test and compare their algorithms in a unified way and with logs that others may have created.

## Acknowledgments

The authors would like to thank Wil van der Aalst for suggesting the use of CPN Tools to create random event logs in the MXML format. They also would like to thank Wil van der Aalst and Boudewijn van Dongen for the discussions about how to log from CPN Tools models. Last but not least, the authors would like to thank the anonymous reviewers for their useful remarks on how to improve this paper readability.

## References

1. Extensible Markup Language (XML). <http://www.w3.org/XML/>.
2. Process mining website. <http://www.processmining.org>.
3. W.M.P. van der Aalst and M. Song. Mining Social Networks: Uncovering interaction patterns in business processes. In J. Desel, B. Pernici, and M. Weske, editors, *International Conference on Business Process Management (BPM 2004)*, volume 3080 of *Lecture Notes in Computer Science*, pages 244–260. Springer-Verlag, Berlin, 2004.
4. 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*, 47(2):237–267, 2003.
5. P. V. Biron and A. Malhotra. XML Schema Part 2: Datatypes (Second Edition). <http://www.w3.org/TR/xmlschema-2/>, 2004.

6. 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. In G. Ciardo and P. Darondeau, editors, *International Conference on Applications and Theory of Petri Nets (ATPN 2005)*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer-Verlag, Berlin, 2005.
7. B.F. van Dongen and W.M.P. van der Aalst. EMiT: A process mining tool. In J. Cortadelle and W. Reisig, editors, *International Conference on Applications and Theory of Petri Nets (ATPN 2004)*, volume 3099 of *Lecture Notes in Computer Science*, pages 454–463. Springer-Verlag, Berlin, 2004.
8. J. Herbst and D. Karagiannis. Workflow mining with inwolve. *Computers in Industry*, 53(3):245–264, 2004.
9. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1*. EATCS monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1997.
10. J. Herbst M. Hammori and N. Kleiner. Interactive workflow mining. In B. Pernici J. Desel and M. Weske, editors, *International Conference on Business Process Management (BPM 2004)*, volume 3080 of *LNCS*, pages 211–226. Springer Verlag, January 2000.
11. G. Schimm. Mining exact models of concurrent workflows. *Computers in Industry*, 53(3):265–281, 2004.
12. A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering Workflow Models from Event-Based Data using Little Thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003.