ProM 4.0: Comprehensive Support for *Real* Process Analysis

W.M.P. van der Aalst[1], B.F. van Dongen[1], C.W. Günther[1], R.S. Mans[1], A.K. Alves de Medeiros[1], A. Rozinat[1], V. Rubin[2,1], M. Song[1], H.M.W. Verbeek[1], and A.J.M.M. Weijters[1]

[1] Eindhoven University of Technology, Eindhoven, The Netherlands
{w.m.p.v.d.aalst}@tue.nl
[2] University of Paderborn, Paderborn, Germany

Abstract. This tool paper describes the functionality of *ProM*. Version 4.0 of ProM has been released at the end of 2006 and this version reflects recent achievements in *process mining*. Process mining techniques attempt to extract non-trivial and useful information from so-called "event logs". One element of process mining is *control-flow discovery*, i.e., automatically constructing a process model (e.g., a Petri net) describing the causal dependencies between activities. Control-flow discovery is an interesting and practically relevant challenge for Petri-net researchers and ProM provides an excellent platform for this. For example, the theory of regions, genetic algorithms, free-choice-net properties, etc. can be exploited to derive Petri nets based on example behavior. However, as we will show in this paper, the functionality of ProM 4.0 is not limited to control-flow discovery. ProM 4.0 also allows for the discovery of other perspectives (e.g., data and resources) and supports related techniques such as conformance checking, model extension, model transformation, verification, etc. This makes ProM a versatile tool for process analysis which is not restricted to model analysis but also includes log-based analysis.

1 Introduction

The first version of ProM was released in 2004. The initial goal of ProM was to unify process mining efforts at Eindhoven University of Technology and other cooperating groups [4]. Traditionally, most analysis tools focusing on processes are restricted to *model-based analysis*, i.e., a model is used as the starting point of analysis. For example, the alternating-bit protocol can be modeled as a Petri net and verification techniques can then be used to check the correctness of the protocol while simulation can be used to estimate performance aspects. Such analysis is only useful *if the model reflects reality*. Process mining techniques use *event logs* as input, i.e., information recorded by systems ranging from information systems to embedded systems. Hence the starting point is not a model but the observed reality. Therefore, we use the phrase *real process analysis* to position process mining with respect to classical model-based analysis. Note that

ProM also uses models (e.g., Petri nets). However, these models (1) are discovered from event logs, (2) are used to reflect on the observed reality (conformance checking), or (3) are extended based on information extracted from logs.

Process mining is relevant since more and more information about processes is collected in the form of event logs. The widespread use of information systems, e.g., systems constructed using ERP, WFM, CRM, SCM, and PDM software, resulted in the omnipresence of vast amounts of event data. Events may be recorded in the form of audit trails, transactions logs, or databases and may refer to patient treatments, order processing, claims handling, trading, travel booking, etc. Moreover, recently, more and more devices started to collect data using TCP/IP, GSM, Bluetooth, and RFID technology (cf. high-end copiers, wireless sensor networks, medical systems, etc.).

Table 1. Comparing ProM 1.1 presented in [7] with ProM 4.0.

Version	ProM 1.1	ProM 4.0
Mining plug-ins	6	27
Analysis plug-ins	7	35
Import plug-ins	4	16
Export plug-ins	9	28
Conversion plug-ins	3	22
Log filter plug-ins	0	14
Total number of plug-ins	29	142

At the Petri net conference in 2005, Version 1.1 of ProM was presented [7]. In the last two years ProM has been extended dramatically and currently dozens of researchers are developing plug-ins for ProM. ProM is open source and uses a plug-able architecture, e.g., people can add new process mining techniques by adding plug-ins without spending any efforts on the loading and filtering of event logs and the visualization of the resulting models. An example is the plug-in implementing the α-algorithm [5], i.e., a technique to automatically derive Petri nets from event logs. The version of ProM presented at the Petri net conference in 2005 (Version 1.1) contained only 29 plug-ins. Version 4.0 provides 142 plug-ins, i.e., there are almost five times as many plug-ins. Moreover, there have been spectacular improvements in the quality of mining algorithms and the scope of ProM has been extended considerably. This is illustrated by Table 1 which compares the version presented in [7] with the current version. To facilitate the understanding of Table 1, we briefly describe the six types of plug-ins:

- *Mining plug-ins* implement some mining algorithm, e.g., the α-miner to discover a Petri net [5] or the social network miner to discover a social network [1].
- *Export plug-ins* implement some "save as" functionality for specific objects in ProM. For example, there are plug-ins to save Petri nets, EPCs, social networks, YAWL, spreadsheets, etc. often also in different formats (PNML, CPN Tools, EPML, AML, etc.).
- *Import plug-ins* implement an "open" functionality for specific objects, e.g., load instance-EPCs from ARIS PPM or BPEL models from WebSphere.

- *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 to compare a log and a model (i.e., conformance checking) or a log and an LTL formula. Moreover, there are analysis plug-ins related to performance measurement (e.g., projecting waiting times onto a Petri net).
- *Conversion plug-ins* implement conversions between different data formats, e.g., from EPCs to Petri nets or from Petri nets to BPEL.
- *Log filter plug-ins* implement different ways of "massaging" the log before applying process mining techniques. For example, there are plug-ins to select different parts of the log, to abstract from infrequent behavior, clean the log by removing incomplete cases, etc.

In this paper we do not elaborate on the architecture and implementation framework for plug-ins (for this we refer to [7]). Instead we focus on the functionality provided by the many new plug-ins in ProM 4.0.

The remainder of this paper is organized as follows. Section 2 provides an overview of process mining and briefly introduces the basic concepts. Section 3 describes the "teleclaims" process of an Australian insurance company. A log of this process is used as a running example and is used to explain the different types of process mining: Discovery (Section 4), Conformance (Section 5), and Extension (Section 6). Section 7 briefly mentions additional functionality such as verification and model transformation. Section 8 concludes the paper.

2 Overview

The idea of process mining is to discover, monitor and improve *real* processes (i.e., not assumed processes) by extracting knowledge from event logs. Today many of the activities occurring in processes are either supported or monitored by information systems. Consider for example ERP, WFM, CRM, SCM, and PDM systems to support a wide variety of business processes while recording well-structured and detailed event logs. However, process mining is not limited to information systems and can also be used to monitor other operational processes or systems. For example, we have applied process mining to complex X-ray machines, high-end copiers, web services, wafer steppers, careflows in hospitals, etc. All of these applications have in common that *there is a notion of a process* and that *the occurrence of activities are recorded in so-called event logs*.

Assuming that we are able to log events, a wide range of *process mining techniques* comes into reach. The basic idea of process mining is to learn from observed executions of a process and can be used to (1) *discover* new models (e.g., constructing a Petri net that is able to reproduce the observed behavior), (2) check the *conformance* of a model by checking whether the modeled behavior matches the observed behavior, and (3) *extend* an existing model by projecting information extracted from the logs onto some initial model (e.g., show bottlenecks in a process model by analyzing the event log). All three types of analysis

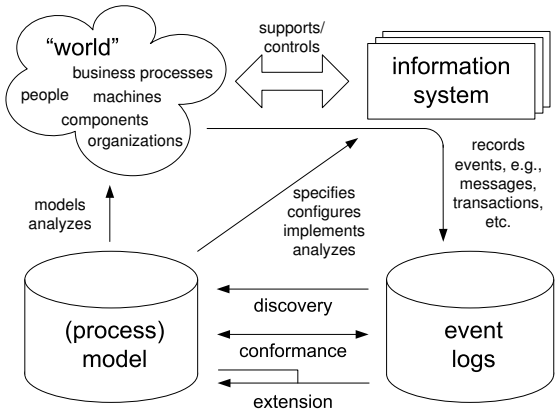

Fig. 1. Overview showing three types of process mining supported by ProM: (1) Discovery, (2) Conformance, and (3) Extension.

have in common that they assume the existence of some *event log*. Figure 1 shows the three types of process mining. Each of these is supported by ProM through various plug-ins as will be shown in the remainder using a running example.

3 Running Example

As a working example, we consider the "teleclaims" process of an Australian insurance company described in [2]. This process deals with the handling of inbound phone calls, whereby different types of insurance claims (household, car, etc.) are lodged over the phone. The process is supported by two separate call centres operating for two different organizational entities (Brisbane and Sydney). Both centres are similar in terms of incoming call volume (approx. 9,000 per week) and average total call handling time (550 seconds), but different in the way call centre agents are deployed, underlying IT systems, etc. The teleclaims process model is shown in Figure 2. The two highlighted boxes at the top show the subprocesses in both call centres. The lower part describes the process in the back-office.

This process model is expressed in terms of an Event-Driven Process Chain (EPC) (see [8] for a discussion on the semantics of EPCs). For the purpose of the paper it is not necessary to understand the process and EPC notation in any detail. However, for a basic understanding, consider the subprocess corresponding to the call centre in Brisbane. The process starts with *event* "Phone call received". This event triggers *function* "Check if sufficient information is available". This function is executed by a "Call Center Agent". Then a choice is made. The circle represents a so-called *connector*. The "x" inside the connector and the two outgoing arcs indicate that it is an exclusive OR-split (XOR). The XOR connector results in event "Sufficient information is available" or event "Sufficient information is not available". In the latter case the process ends. If the information is available, the claim is registered (cf. function "Register claim"

also executed by a "Call Center Agent") resulting in event "Claim is registered". The call centre in Sydney has a similar subprocess and the back-office process should be self-explaining after this short introduction to EPCs. Note that there are three types of split and join connectors: AND, XOR, and OR, e.g., in the back-office process there is one AND-split ($\wedge$) indicating that the last part is executed in parallel.

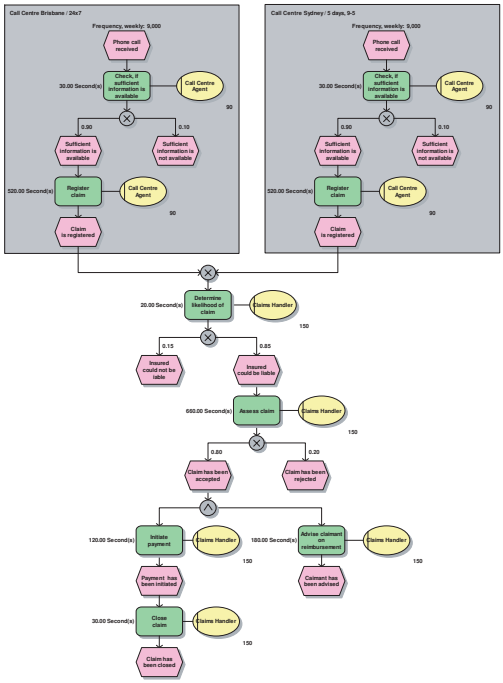

Fig. 2. Insurance claim handling EPC [2].

```
...
<ProcessInstance id="3055" description="Claim being handled">
  <AuditTrailEntry>
    <Data><Attribute name = "call centre">Sydney </Attribute>
    </Data><WorkflowModelElement>incoming claim
      </WorkflowModelElement>
    <EventType >complete</EventType>
    <Timestamp>2006-12-01T07:51:05.000+01:00</Timestamp>
    <Originator>customer</Originator>
  </AuditTrailEntry>
  <AuditTrailEntry>
    <Data><Attribute name = "location">Sydney </Attribute>
    </Data><WorkflowModelElement>check if sufficient
      information is available</WorkflowModelElement>
    <EventType >start</EventType>
    <Timestamp>2006-12-01T07:51:05.000+01:00</Timestamp>
    <Originator>Call Centre Agent Sydney</Originator>
  </AuditTrailEntry>
  <AuditTrailEntry>
    <Data><Attribute name = "location">Sydney </Attribute>
    </Data><WorkflowModelElement>check if sufficient
      information is available</WorkflowModelElement>
    <EventType >complete</EventType>
    <Timestamp>2006-12-01T07:51:25.000+01:00</Timestamp>
    <Originator>Call Centre Agent Sydney</Originator>
  </AuditTrailEntry>
  ...
  <AuditTrailEntry>
    <Data><Attribute name = "outcome">processed </Attribute>
    <Attribute name = "duration">1732 </Attribute>
    </Data><WorkflowModelElement>end</WorkflowModelElement>
    <EventType >complete</EventType>
    <Timestamp>2006-12-01T08:19:57.000+01:00</Timestamp>
    <Originator>Claims handler</Originator>
  </AuditTrailEntry>
</ProcessInstance>
...
```

Fig. 3. Fragment of the MXML log containing 3512 cases (process instances) and 46138 events (audit trail entries).

Figure 3 shows a fragment of the log in MXML format, the format used by ProM. In this case, the event log was obtained from a simulation using CPN Tools. Using ProMimport one can extract logs from a wide variety of systems, e.g., workflow management systems like Staffware, case handling systems like FLOWer, ERP components like PeopleSoft Financials, simulation tools like ARIS and CPN Tools, middleware systems like WebSphere, BI tools like ARIS PPM, etc., and it has also been used to develop many organization/system-specific conversions (e.g., hospitals, banks, governments, etc.). Figure 3 illustrates the typical data present in most event logs, i.e., a log is composed of process instances (i.e., cases) and within each instance there are audit trail entries (i.e., events) with various attributes. Note that it is not required that systems log all of this information, e.g., some systems do not record transactional information (e.g., just the completion of activities is recorded), related data, or timestamps. In the MXML format only the ProcessInstance (i.e., case) field and the WorkflowModelElement (i.e., activity) field are obligatory, i.e., *any event*

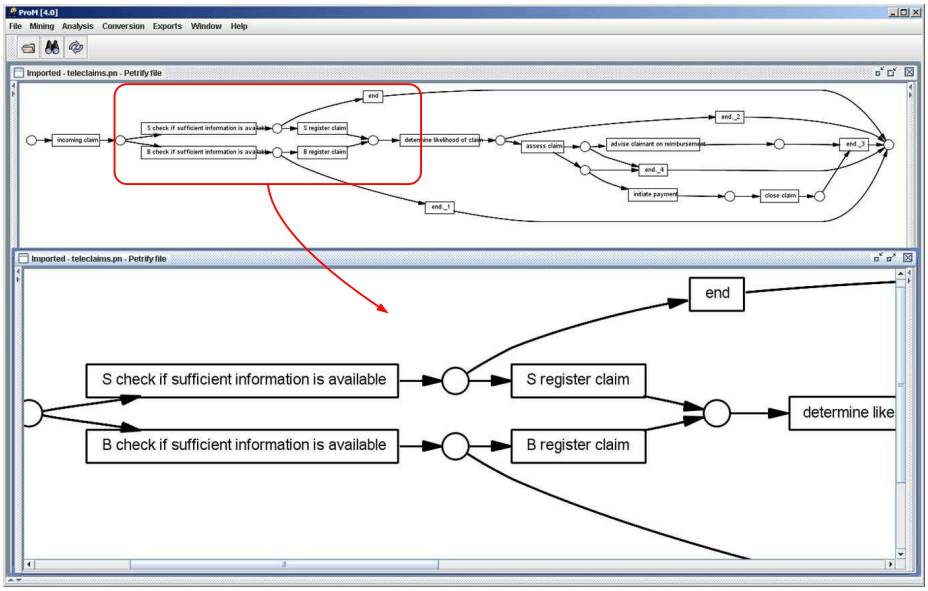

Fig. 4. A Petri net discovered using ProM based on an analysis of the 3512 cases.

needs to be linked to a case (process instance) and an activity. All other fields (data, timestamps, resources, etc.) are optional.

For control-flow discovery, e.g., deriving a Petri net model from an MXML file, we often focus on the ordering of activities within individual cases. In this context, a single case σ can be described by a sequence of activities, i.e., a trace $\sigma \in A^*$ where A is the set of activities. Consequently, such an abstraction of the log can be described by a multiset of traces.

4 Discovery

Process mining techniques supporting *discovery* do not assume an a-priori model, i.e., based on an event log, some model is constructed (cf. Figure 1). ProM 4.0 offers 27 mining plug-ins able to construct a wide variety of models. One of the first plug-ins was the α-miner [5] which constructs a Petri net model from an MXML log, i.e., based on an analysis of the log which does not contain any explicit process information (e.g., AND/XOR-splits/joins), a process model is derived. However, the α-miner is unable to discover complex process models. For example, it is unable to *correctly* discover the teleclaims process illustrated in Figure 2. However, ProM 4.0 has several new mining plug-ins that are able to correctly discover this process using various approaches (regions, heuristics, genetic algorithms, etc.) and representations (Petri nets, EPCs, transitions systems, heuristic nets).

Figure 4 shows a Petri net discovered by ProM. The top window shows the overall process while the second window zooms in on the first part of the discovered model. This model is behaviorally equivalent to the EPC model in

Figure 2 and has been obtained using an approach which first builds a transition system (see Figure 5) and then uses extensions of the classical theory of regions [6] to construct a Petri net. ProM provides various ways to extract transition systems from logs, a plug-in to construct regions on-the-fly, and an import and export plug-in for Petrify [6] (see [3] for details).

Process mining is not limited to process models (i.e., control flow). ProM also allows for the discovery of models related to data, time, transactions, and resources. As an example, Figure 6 shows the plug-in to extract social networks from event logs using the technique presented in [1]. The social network shown in Figure 6 is constructed based on frequencies of work being transferred from one resource class to another. The diagram adequately shows that work is generated by customers and then flows via the call centre agents to the claims handlers in the back office.

It is impossible to provide an overview of all the discovery algorithms supported. However, of the 27 mining plug-ins we would like to mention the heuristics miner (Figure 7) able to discover processes in the presence of noise and the multi-phase miner using an EPC representation. Both approaches are more robust than the region-based approach and the classical α-algorithm. It is also possible to convert models of one type to another. For example, Figure 8 shows the EPC representation of the Petri net in Figure 4.

5 Conformance

Conformance checking requires, in addition to an event log, some a-priori model. This model may be handcrafted or obtained through process discovery. Whatever its source, ProM provides various ways of checking whether reality conforms to such a model. For example, there may be a process model indicating that purchase orders of more than one million Euro require two checks. Another example is the checking of the so-called "four-eyes principle". Conformance checking may be used to detect deviations, to locate and explain these deviations, and to measure the severity of these deviations. ProM 4.0 also supports conformance checking, i.e., comparing an a-priori model with the observed reality stored in some MXML log. For example, we could take the discovered model shown in Figure 4 and compare it with the log shown in Figure 3 using the conformance checking plug-in in ProM. Figure 9 shows the result. This analysis shows that the fitness of the model is 1.0, i.e., the model is able to "parse" all cases. The conformance checker also calculates metrics such as behavioral appropriateness (i.e., precision) and structural appropriateness [9] all indicating that the discovered model is indeed a good reflection of reality. Note that, typically, conformance checking is done not with respect to a discovered model, but with respect to some normative/descriptive hand-crafted model. For example, given an event log obtained from the real teleclaims process it would be interesting to detect potential deviations from the process model in Figure 2. In case that there is not a complete a-priori process model but just a set of requirements (e.g., business rules), ProM's LTL checker can be used.

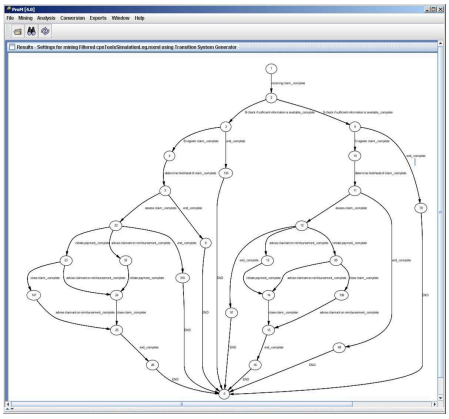

Fig. 5. Transition system system used to construct the Petri net in Figure 4.

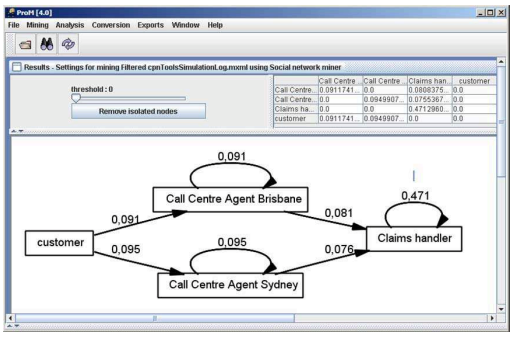

Fig. 6. Social network obtained using the "handover of work" metric.

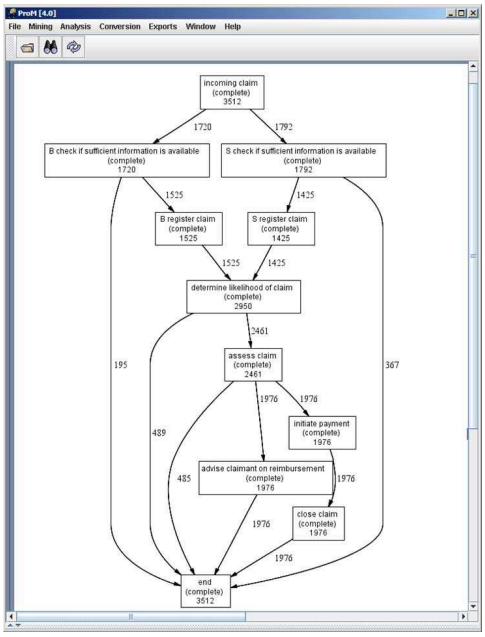

Fig. 7. Heuristics net obtained by applying the heuristics miner to the log of Figure 3.

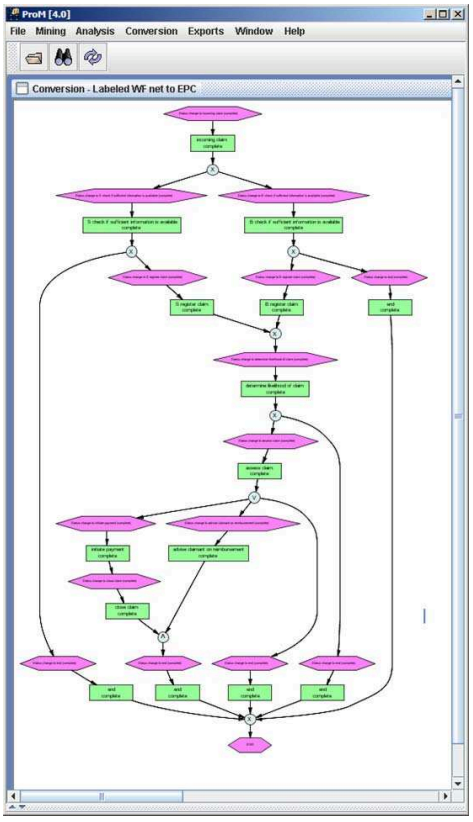

Fig. 8. EPC discovered from the log in Figure 3.

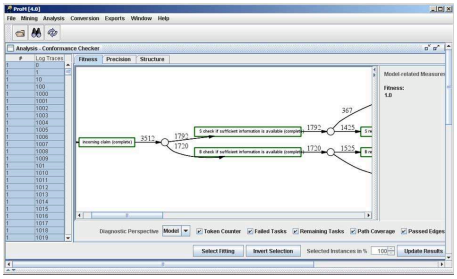

Fig. 9. Conformance checker.

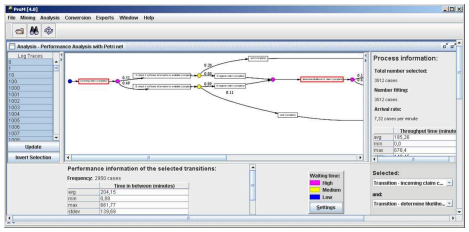

Fig. 10. Performance analyzer.

6 Extension

For model *extension* it is also assumed that there is an initial model (cf. Figure 1). This model is extended with a new aspect or perspective, i.e., the goal is not to check conformance but to enrich the model with performance/time aspects, organizational/resource aspects, and data/information aspects. Consider for example a Petri net (either discovered, hand-crafted, or resulting from some model transformation) describing a process which is also logged. It is possible to enrich the Petri net using information in the log. Most logs also contain information about resources, data, and time. ProM 4.0 supports for example decision mining, i.e., by analyzing the data attached to events and using classical decision tree analysis, it is possible to add decision rules to the Petri net (represented as conditions on arcs). Information about resources (Originator field in the MXML log) can be analyzed and used to add allocation rules to a Petri net. Figure 10 shows a performance analysis plug-in which projects timing information on places and transitions. It graphically shows the bottlenecks and all kinds of performance indicators, e.g., average/variance of the total flow time or the time spent between two activities. The information coming from all kinds of sources can be stitched together and exported to CPN Tools, i.e., ProM is able to turn MXML logs into colored Petri nets describing all perspectives (control-flow, data, time, resources, etc.). CPN Tools can then be used to simulate the process without adding any additional information to the generated model.

7 Additional Functionality

It is not possible to give a complete overview of all 142 plug-ins. The figures shown in previous sections reflect only the functionality of 7 plug-ins. However, it is important to note that the functionality of ProM is not limited to process mining. ProM also allows for *model conversion*. For example, a model discovered in terms of a heuristic net can be mapped onto an EPC which can be converted into a Petri net which is saved as a YAWL file that can be uploaded in the workflow system YAWL thereby directly enacting the discovered model. For some of the models, ProM also provides *analysis* plug-ins. For example, the basic Petri net analysis techniques (invariants, reachability graphs, reduction rules, S-components, soundness checks, etc.) are supported. There are also interfaces

to different analysis (e.g., Petrify, Fiona, and Woflan) and visualization (e.g., FSMView and DiaGraphica) tools.

8 Conclusion

ProM 4.0 consolidates the state-of-the-art of process mining. It provides a plug-able environment for process mining offering a wide variety of plug-ins for process discovery, conformance checking, model extension, model transformation, etc. ProM is open source and can be downloaded from www.processmining.org. Many of its plug-ins work on Petri nets, e.g., there are several plug-ins to discover Petri nets using techniques ranging from genetic algorithms and heuristics to regions and partial orders. Moreover, Petri nets can be analyzed in various ways using the various analysis plug-ins.

Acknowledgements The development of ProM is supported by EIT, NWO-EW, the Technology Foundation STW, and the IOP program of the Dutch Ministry of Economic Affairs.

References

1. W.M.P. van der Aalst, H.A. Reijers, and M. Song. Discovering Social Networks from Event Logs. *Computer Supported Cooperative work*, 14(6):549–593, 2005.
2. W.M.P. van der Aalst, M. Rosemann, and M. Dumas. Deadline-based Escalation in Process-Aware Information Systems. *Decision Support Systems*, 43(2):492–511, 2007.
3. W.M.P. van der Aalst, V. Rubin, B.F. van Dongen, E. Kindler, and C.W. Günther. Process Mining: A Two-Step Approach using Transition Systems and Regions. BPM Center Report BPM-06-30, BPMcenter.org, 2006.
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. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.
6. J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Deriving Petri Nets from Finite Transition Systems. *IEEE Transactions on Computers*, 47(8):859–882, August 1998.
7. B.F. van Dongen, A.K. Alves de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM framework: A New Era in Process Mining Tool Support. In G. Ciardo and P. Darondeau, editors, *Application and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer-Verlag, Berlin, 2005.
8. E. Kindler. On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. *Data and Knowledge Engineering*, 56(1):23–40, 2006.
9. A. Rozinat and W.M.P. van der Aalst. Conformance Testing: Measuring the Fit and Appropriateness of Event Logs and Process Models. In C. Bussler et al., editor, *BPM 2005 Workshops (Workshop on Business Process Intelligence)*, volume 3812 of *Lecture Notes in Computer Science*, pages 163–176. Springer-Verlag, Berlin, 2006.