Discovering Hierarchical Process Models Using ProM

R.P. Jagadeesh Chandra Bose[1,2], Eric H.M.W. Verbeek[1] and Wil M.P. van der Aalst[1]

[1] Department of Mathematics and Computer Science, University of Technology, Eindhoven, The Netherlands
[2] Philips Healthcare, Veenpluis 5–6, Best, The Netherlands
{j.c.b.rantham.prabhakara,h.m.w.verbeek,w.m.p.v.d.aalst}@tue.nl

Abstract. Process models can be seen as "maps" describing the operational processes of organizations. Traditional process discovery algorithms have problems dealing with fine-grained event logs and less-structured processes. The discovered models (i.e., "maps") are spaghetti-like and are difficult to comprehend or even misleading. One of the reasons for this can be attributed to the fact that the discovered models are flat (without any hierarchy). In this paper, we demonstrate the discovery of hierarchical process models using a set of interrelated plugins implemented in ProM.[3] The hierarchy is enabled through the automated discovery of abstractions (of activities) with domain significance.

Keywords: process discovery, process maps, hierarchical models, abstractions, common execution patterns

1 Introduction

We have applied process mining techniques in over 100 organizations. These practical experiences revealed two problems: (a) processes tend to be less structured than what stakeholders expect, and (b) events logs contain fine-grained events whereas stakeholders would like to view processes at a more coarse-grained level. In [1], we showed that common execution patterns (e.g., tandem arrays, maximal repeats etc.) manifested in an event log can be used to create powerful *abstractions*. These abstractions are used in our *two-phase approach to process discovery* [2]. The first phase comprises of pre-processing the event log based on abstractions (bringing the log to the desired level of granularity) and the second phase deals with discovering the process maps while providing a seamless zoom-in/out facility. Figure 1 highlights the difference between the traditional approach to process discovery and our two-phase approach. Note that the process model (map) discovered using the two-phase approach is much simpler.

The *two-phase approach to process discovery* [2] enables the discovery of hierarchical process models. In this paper, we demonstrate the discovery of hierarchical process models using a chain of plugins implemented in ProM. The chain of plugins and their order of application is illustrated in Figure 2.

[3] ProM is an extensible framework that provides a comprehensive set of tools/plugins for the discovery and analysis of process models from event logs. See http://www.processmining.org for more information and to download ProM.

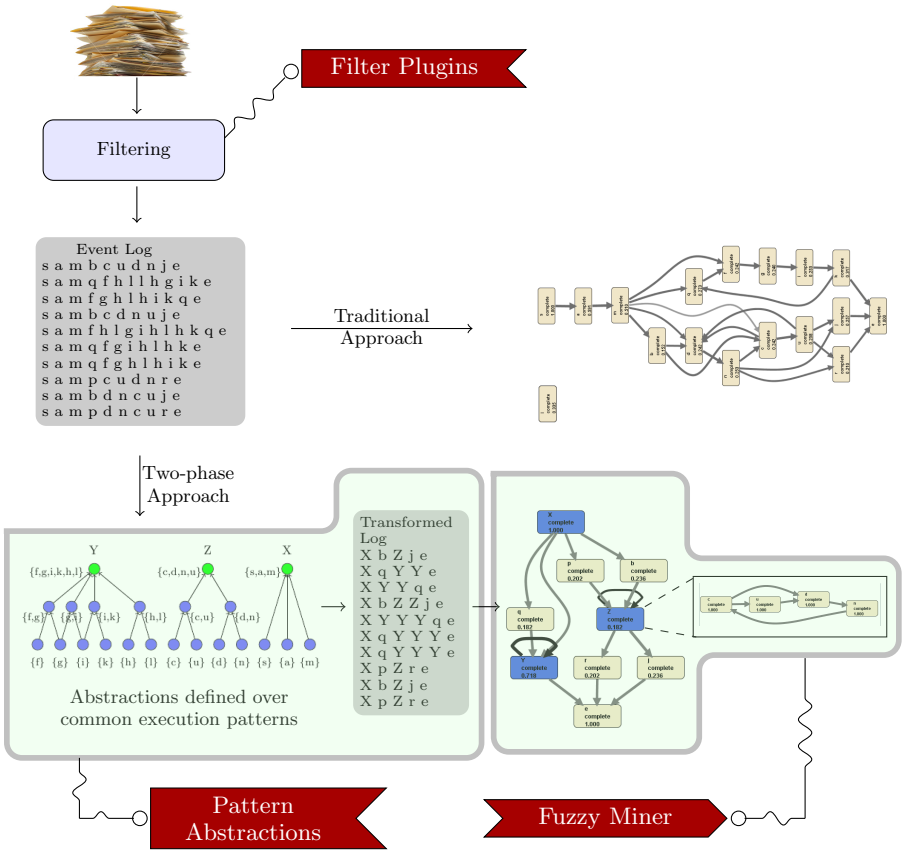

Fig. 1: Traditional approach versus our two-phase approach to process discovery. ProM plugins are used to filter the event log. ProM's *Pattern Abstractions* plugin and the *Fuzzy Miner* plugin are used to realize simple and intuitive models.

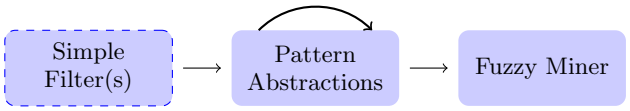

Fig. 2: The chaining of plugins that enables the discovery of hierarchical process models.

The event log may first be cleansed using some simple filters (e.g., adding artificial start/end events, filtering events of a particular transaction type such as considering only 'complete' events etc.). The *Pattern Abstractions* plugin is then applied on this filtered log one or several times. The Pattern Abstractions plugin has been implemented as a *log visualizer* in ProM and caters to *the discovery of common execution patterns, the definition of abstractions over them, and the pre-processing of the event log with these abstractions*. The transformed log (pre-processed log with abstractions) obtained in iteration i is used as the input for the Pattern Abstractions plugin in iteration $i+1$. It is this repetitive application

of the Pattern Abstractions plugin that enables the definition of multiple levels of hierarchy (new abstractions can be defined over existing abstractions). During the pre-processing phase, for each defined abstraction, the Pattern Abstractions plugin generates a sub-log that captures the manifestation of execution patterns defined by that abstraction as its process instances. The Fuzzy Miner plugin [3] is then applied on the transformed log obtained after the last iteration. The Fuzzy Miner plugin in ProM has been enhanced to utilize the availability of sub-logs for the defined abstractions. Process models are discovered for each of the sub-logs and are displayed upon zooming in on its corresponding abstract activity.

Running Example. We use the workflow of a simple digital photo copier as our running example. The copier supports photocopying, scanning and printing of documents in both color and gray modes. The scanned documents can be sent to the user via email or FTP. Upon receipt of a job, the copier first generates an image of the document and subsequently processes the image to enhance its quality. Depending on whether the job request is for a copy/scan or print, separate procedures are followed to generate an image. For print requests, the document is first interpreted and then a rasterization procedure is followed to form an image. The image is then written on the drum, developed, and fused on to the paper.

We have modeled this workflow of the copier in CPN tools [4] and generated event logs by simulation. We use one such event log in this paper. The event log consists of 100 process instances, 76 event classes and 40,995 events. The event log contains fine-grained events pertaining to different procedures (e.g., image processing, image generation etc.) mentioned above. An analyst may not be interested in such low level details. We demonstrate the discovery of the workflow at various levels of abstractions for this event log.

2 Pattern Abstractions Plugin

The basic building blocks of the Pattern Abstractions plugin are shown in Figure 3. Figures 4 and 5 illustrate these building blocks.

Fig. 3: Building blocks of the Pattern Abstractions plugin

– *Discover Common Execution Patterns:* The Pattern Abstractions plugin supports the discovery of tandem arrays (loop patterns) and maximal repeats (common subsequence of activities within a process instance or across process instances) [1]. These can be uncovered in *linear* time and space with respect to the length of the traces.

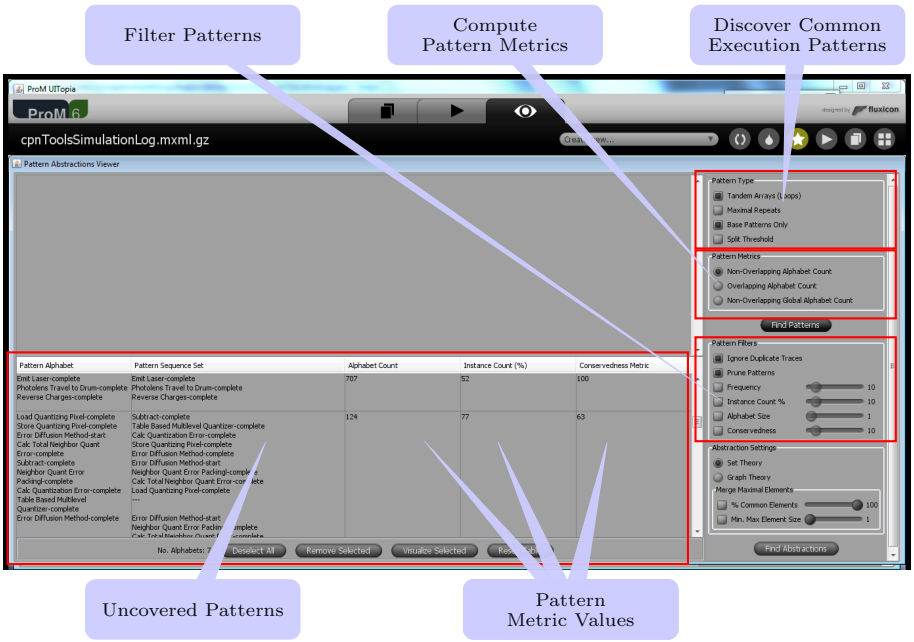

Fig. 4: The discovery of common execution patterns, computation of pattern metrics, filtering and inspection of patterns in the Pattern Abstractions plugin.

- *Compute Pattern Metrics:* Various metrics (e.g, overlapping and non-overlapping frequency counts, instance count etc.) to assess the significance of the uncovered patterns are supported.
- *Filter Patterns:* It could be the case that too many patterns are uncovered from the event log. To manage this, features to filter patterns that are less significant are supported.
- *Form and Select Abstractions:* Abstractions are defined over the filtered patterns. Patterns that are closely related are grouped together to form abstractions. The approach for forming abstractions is presented in [1]. Furthermore, various features to edit/select abstractions such as merging two or more abstractions and deleting activities related to a particular abstraction are supported. Figure 5 depicts a few abstractions defined over loop patterns for the copier event log e.g., *half-toning*, a procedure for enhancing the image quality, is uncovered as an abstraction.
- *Transform Log:* The event log is pre-processed by replacing activity subsequences corresponding to abstractions. A replaced activity subsequence is captured as a process instance in the sub-log for the corresponding abstract activity.

At any iteration, if n abstractions are selected, the Pattern Abstractions plugin generates a transformed log, and n sub-logs (one for each of the n chosen abstractions). We recommend to process for loop patterns in the initial iterations and maximal repeats in the subsequent iterations. For the example event log, we have performed three iterations. The transformed log after the third iteration

has 19 event classes and 1601 events. In the process, we have defined various abstractions such as *half-toning, image processing, capture image*, etc.

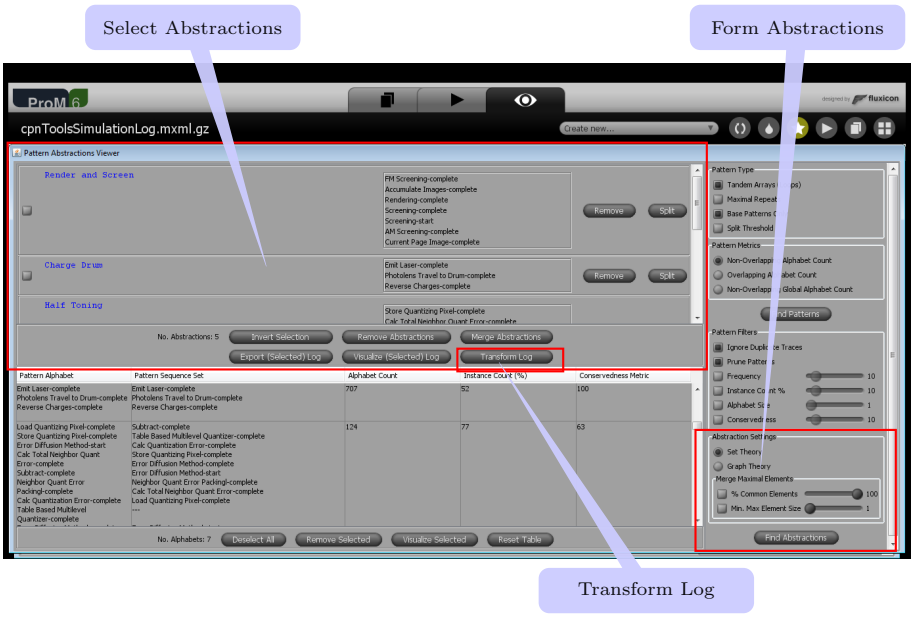

Fig. 5: The generation and selection of abstractions in the Pattern Abstractions plugin.

The Pattern Abstractions plugin supports additional features such as visualizing patterns and exporting the traces that contain the patterns.

3 (Enhanced) Fuzzy Miner Plugin

The Fuzzy Miner [3, 5] is a process miner that mines an event log for a family of process models using a "map" metaphor. As many maps exist that show the city of Amsterdam at different levels of abstraction, also different maps exist for a process model mined from an event log. In this map metaphor, an object of interest in Amsterdam (like the Rijksmuseum or the Anne Frank House) corresponds to a node in the process model, where streets (like the Kalverstraat or the PC Hooftstraat) correspond to edges in the model. For sake of convenience, we call a single map a *fuzzy instance* whereas we call a family of maps (like all Amsterdam maps) a *fuzzy model*.

Like high-level maps only show major objects of interest and major streets, high-level fuzzy instances also show only major elements (nodes and edges). For this purpose, the Fuzzy Miner computes from the log a significance weight for every element and an additional correlation weight for every edge. The higher these weights are, the more major the element is considered to be. Furthermore, the Fuzzy Miner uses a number of thresholds: Only elements that meet these thresholds are shown. As such, these thresholds correspond to the required level

of abstraction: The higher these thresholds are, the higher the level of abstraction is. For sake of completeness we mention here that a fuzzy instance may contain clusters of minor nodes: If some objects of interest on the Amsterdam map are too minor to be shown by themselves on some map, they may be shown as a single (and major) object provided that they are close enough. For this reason, the Fuzzy Miner first attempts to cluster minor nodes into major cluster nodes, and only if that does not work it will remove the minor node from the map.

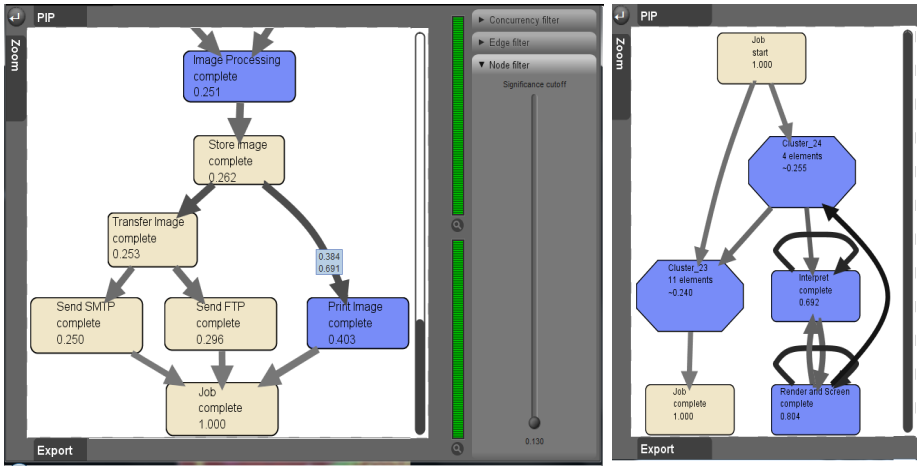

Fig. 6: Fuzzy model and instance

Figure 6 shows an example fuzzy model (left-hand side) and fuzzy instance (right-hand side). Note that both views show a fuzzy instance, but the fuzzy model view allows the user to change the thresholds (by changing the sliders) whereas the fuzzy instance view does not. The significance of a node is displayed as part of its label (for example, the node "Transfer Image" has a significance of 0.253), the significance of an edge is visualized using its wideness (the wider the edge, the more significant it is), and the correlation of an edge is visualized using its color contrast (the darker the edge is, the more correlated its input node and its output node are). The octagonal shaped nodes in the right-hand side view correspond to the cluster nodes (one of the cluster nodes contain 4 activities and the other contains 11 activities). All activities on the left hand side except "Job Complete" are contained in a cluster node on the right. Apparently, the significance weights for these nodes (0.262, 0.253, 0.250, 0.296 and 0.403) were too low to be shown, which indicates that the corresponding threshold was set to at least 0.403. Furthermore, the node "Interpret" (on the right) is highly self-correlated, whereas the nodes "Transfer Image" and "Send SMTP" (on the left) are moderately correlated.

The Fuzzy Miner has been enhanced to utilize the availability of sub-logs obtained from the Pattern Abstractions plugin for the chosen abstractions. Fuzzy models are discovered for each of the sub-logs and are displayed upon zooming in on its corresponding abstract activity. Abstract activities are differentiated from

other activities by means of a distinct color (a darker shade of blue, see also Figure 6).

Figure 7 depicts the top-level process model of the copier example. This model is generated from the transformed log obtained after the third iteration of Pattern Abstractions plugin. The upper branch of the process model corresponds to the creation of the document image for print requests while the lower branch corresponds to image creation for copy/scan requests. The two branches meet after the image is formed and the image is subjected to some image processing functionality. The document is then printed or sent to the user via email or FTP. The lower level details of image creation, image processing, print image have been abstracted in this model. *The Pattern Abstractions plugin enables the discovery of such abstractions with strong domain (functional) significance.* Upon zooming in on the *Image Processing* abstraction, the process model depicted in Figure 8 is shown. This sub-process in turn contains another abstract activity viz., *Half Toning* (the level of hierarchy is two). Zooming in on this abstract activity displays the sub-process defining it as depicted in Figure 8. Figure 9 depicts two other abstractions.

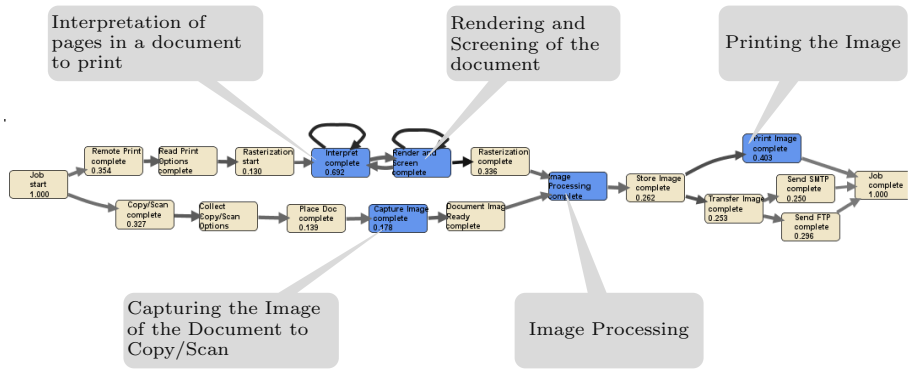

Fig. 7: The top level process model of the copier event log. Blue (dark colored) nodes are abstract activities that can be zoomed in. Upon zooming in, the sub-process defining the abstraction is shown.

In this fashion, using the chain of plugins presented in this paper, one can discover hierarchical process models.

4 Conclusions

We demonstrated the discovery of hierarchical process models using a chain of plugins implemented in ProM. The repetitive application of Pattern Abstractions plugin enables the discovery of multiple levels of hierarchy. We can use this approach to create maps that (i) depict desired traits, (ii) eliminate irrelevant details, (iii) reduce complexity, and (iv) improve comprehensibility.

Acknowledgments R.P.J.C. Bose and W.M.P. van der Aalst are grateful to Philips Healthcare for funding the research in process mining.

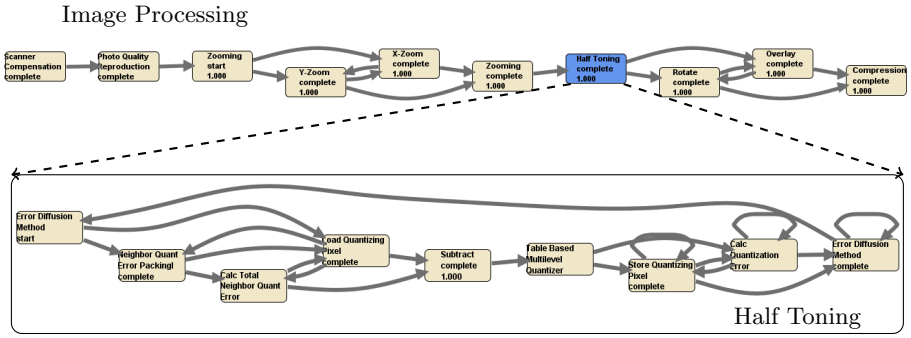

Fig. 8: The sub-process captured for the abstraction 'Image Processing' (in the top-level model). This sub-process in turn contains another abstraction viz., 'Half Toning'. Upon zooming in on 'Half Toning', the sub-process defining that is shown.

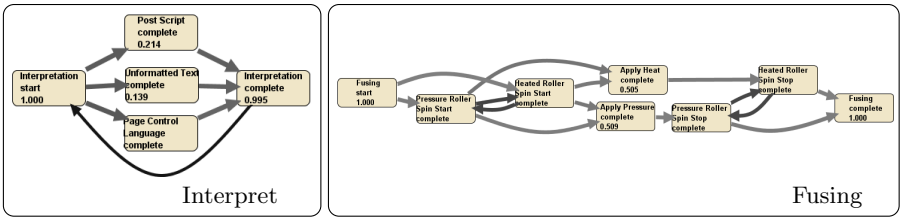

Fig. 9: The sub-processes for the abstractions 'Interpret' and 'Fusing'. 'Interpret' is an abstract activity at the top-level of the process model while 'Fusing' is an abstract activity underneath the 'Print Image' abstraction.

References

1. Bose, R.P.J.C., van der Aalst, W.M.P.: Abstractions in Process Mining: A Taxonomy of Patterns. In Dayal, U., Eder, J., Koehler, J., Reijers, H., eds.: Business Process Management. Volume 5701 of LNCS., Springer-Verlag (2009) 159–175
2. Li, J., Bose, R.P.J.C., van der Aalst, W.M.P.: Mining Context-Dependent and Interactive Business Process Maps using Execution Patterns. In zur Muehlen, M., Su, J., eds.: BPM 2010 Workshops. Volume 66 of LNBIP., Springer-Verlag (2011) 109–121
3. Günther, C., van der Aalst, W.M.P.: Fuzzy Mining: Adaptive Process Simplification Based on Multi-perspective Metrics. In: International Conference on Business Process Management (BPM 2007). Volume 4714 of LNCS., Springer-Verlag (2007) 328–343
4. Vinter Ratzer, A., Wells, L., Lassen, H.M., Laursen, M., Qvortrup, J.F., Stissing, M.S., Westergaard, M., Christensen, S., Jensen, K.: CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets. In: 24th International Conference on Applications and Theory of Petri Nets (ICATPN). Volume 2679 of LNCS., Springer (2003) 450–462
5. Xia, J.: Automatic Determination of Graph Simplification Parameter Values for Fuzzy Miner. Master's thesis, Eindhoven University of Technology (2010)