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
{j.c.b.rantham.prabhakara,h.m.w.verbeek,w.m.p.v.d.aalst}@tue.nl
[2] Philips Healthcare, Veenpluis 5–6, Best, The Netherlands

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.[1] The hierarchy is enabled through the automated discovery of abstractions (of activities) with domain significance.

Key words: process discovery, process maps, hierarchical models, abstractions, common execution patterns

1 Introduction

Process discovery is one of the three main types of process mining [1]. A discovery technique takes an event log and produces a model without using any *apriori* information e.g., the α-algorithm discovers a Petri net based on sequences of events [2]. 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. As a result, the discovered models are often incomprehensible (spaghetti-like) and add little value. This can be attributed to the fact that the majority of techniques pertains to the discovery of control-flow models that are "flat" [2, 3, 4, 5]. A notable exception is the Fuzzy miner [6]. Flat models have inherent limitations and are one of the primary sources of incomprehensibility. For a log with $|\mathcal{A}|$ event classes (activities), a flat model can be viewed as a graph containing $|\mathcal{A}|$ nodes with edges corresponding to the dependency between activities defined by the execution behavior in the log. Graphs become quickly

[1] 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.

overwhelming and unsuitable for human perception and cognitive systems even if there are a few dozens of nodes [7].

In [8], 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* (the abstractions uncovered have strong domain significance from a functionality point of view). These abstractions are used in our *two-phase approach to process discovery* [9]. 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. The two-phase approach to process discovery has been implemented as a set of interrelated plugins in the ProM framework. In this paper, we demonstrate the discovery of hierarchical process models using a chain of these plugins.

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 job, 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. Fig. 15 in Appendix A depicts the high-level workflow of the digital photo copier represented as an YAWL [10] model. This high-level workflow contains the composite tasks (sub-processes) *capture image*, *rasterize image*, *image processing* and *print image*. Fig. 16 in Appendix A depicts the workflow of the image processing sub-process. This sub-process contains another composite task, viz., *half toning*, which is depicted in Fig. 17 in Appendix A. Fig. 18 in Appendix A depicts the *fusing* sub-process within *print image*.

We have modeled this workflow of the copier in CPN tools [11] and generated event logs by simulation [12]. CPN tools lets the users model processes and have more control about the properties of the event logs. This is achieved by varying the parameters of the model and the parameters for simulation. The advantage of using synthetic event logs is that we can conduct various controlled experiments. We also applied the approach presented in this paper to several real-life event logs. These experiments confirm our findings based on the simulated model used in this paper. To illustrate our approach we use one event log generated for the copier. The event log consists of 100 process instances, 76 event classes and $40,995$ events[2]. 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] The event log is available at: `http://www.win.tue.nl/~jcbose/DigitalCopier.xes.gz`.

The remainder of this paper is organized as follows. Section 2 presents an overview of the two phase approach to process discovery. Section 3 describes how the two-phase approach can be used to discover hierarchical processes. Section 4 explains the pattern abstractions plugin in ProM that assists in the abstraction of events and the transformation of an event log to a desired level of granularity. Section 5 explains the enhanced fuzzy miner plugin in ProM that enables the discovery of process maps. We present the experimental results in Section 6. Section 7 concludes the paper.

2 Two-phase Approach to Process Discovery

The first phase in the two-phase approach to process discovery involves the simplification of the log to a desired level of granularity. The second phase involves the discovery of models from this simplified log.

Phase-1: Preprocessing Log

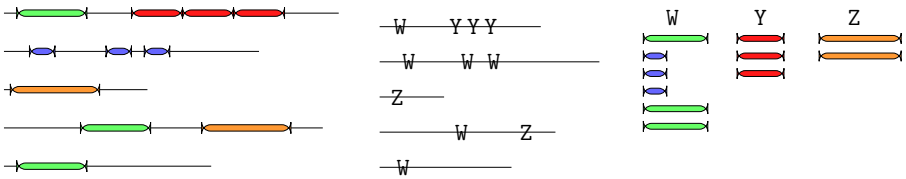
In this phase, the log is simplified based on the desired traits of the context of analysis. Some notion of abstraction of the low-level events to a desired level of granularity needs to be defined. In some applications and contexts, this can be provided by analysts based on domain knowledge e.g., there can be certain activities that demarcate the execution of a particular medical procedure on an X-ray machine. A sequence of events between these activities pertains to an execution sequence of this procedure e.g., the sequence of activities between `Start Fluoroscopy` and `Stop Fluoroscopy` define an instance of a `Fluoroscopy` procedure applied to a patient on an X-ray machine. Alternatively, abstractions can be defined by uncovering common execution patterns in the log and establishing relationships between them as discussed in [8]. These common execution patterns typically capture a sub-process/functionality. Such subprocess behavior in its totality can be captured as an *abstract activity*. Abstractions can be considered as a mapping, $\mathcal{M} \subseteq 2^{\Sigma} \times \mathcal{A}$, between the *original alphabet*, Σ, of the event log, and an *abstract alphabet* $\mathcal{A}$. An example mapping is $\mathcal{M} = \{(\{\mathtt{a},\mathtt{b}\},\mathtt{x}), (\{\mathtt{b},\mathtt{c},\mathtt{d}\},\mathtt{y}), (\{\mathtt{e}\},\mathtt{z}), (\{\mathtt{d}\},\mathtt{z})\}$. This mapping is analogous to the grouping and tagging of streets as a town/city in cartography and to the selection of a desired perspective of viewing maps (restaurant maps vs. fuel station maps)[3]. Each $(B,\mathtt{a}) \in \mathcal{M}$ may reflect a set of sequences $\mathbf{s} \in B^{+}$ capturing the set of patterns defined by the alphabet B for the abstraction $\mathtt{a}$.

Using this mapping, the original event log $\mathcal{L}$, is transformed into an *abstract log* $\mathcal{L}'$. Each trace $\mathbf{t} \in \mathcal{L}$ is transformed into a corresponding trace $\mathbf{t}' \in \mathcal{L}'$. At

[3] One can also take the analogy of *atoms* and *molecules* from chemistry. The individual activities in the original event log are atoms while the mapping associates groups of atoms (activities) to molecules (abstract activities).

the same time, we create one sub-log for each abstract activity $\mathtt{a} \in \mathcal{A}$. The basic idea of the transformation algorithm is to scan each trace from left to right and in the process determine whether there exists a pattern in the trace for which an abstraction is defined. If such a pattern exists, the manifestation of the pattern in the trace is replaced by its abstract activity and simultaneously the manifestation of the pattern that is replaced is added as a trace (process instance) in the sub-log corresponding to that abstract activity. It could be the case that the manifestation of patterns is disturbed by the presence of concurrent activities and/or noise in a trace. Therefore, we consider not just exact but also inexact (non-continuous and approximate) manifestations of patterns to deal with such scenarios. Fig. 1 depicts the general idea of the event log transformation.

$\mathcal{D} = \bigcup_{(B,\mathtt{a}) \in \mathcal{M}} B$ denotes the set of activities in Σ for which a mapping is defined. The activities in $\Sigma \setminus \mathcal{D}$ being not involved in the definition of mapping indicate activities that are insignificant from the context of analysis and are filtered from $\mathbf{t}$ during this transformation. The transformation of logs can be associated to the concept of *artifacts* [13] or *proclets* [14] in business processes. Artifact-centric process models partition a monolithic process model into smaller loosely-coupled process fragments, each describing the life-cycle of a concrete and identifiable artifact.



(a) original log with common execution patterns

(b) transformed log and the sub-logs for each abstraction

Fig. 1: Transformation of the original log into an abstracted log. Also, one sub-log is created for each abstract activity. `W`, `Y`, and `Z` are abstract activities.

Phase-2: Mining Maps

The second phase is to mine a process model on the abstracted (transformed) log. The mapping defined in Phase-1 induces a hierarchy over the abstract activities. Upon zooming into an abstract activity, a process model depicting the subprocess captured by this abstract activity can be shown. The sub-log for the abstract activity is used to create this sub-process model. Multiple levels of hierarchy can be obtained by a repetitive application of Phase-1 i.e., abstractions are defined over the transformed log (pre-processed log with abstractions) obtained in iteration i in iteration $i + 1$. This results in new abstractions to be defined over existing abstractions, thus inducing a hierarchy.

Fig. 2 illustrates 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.

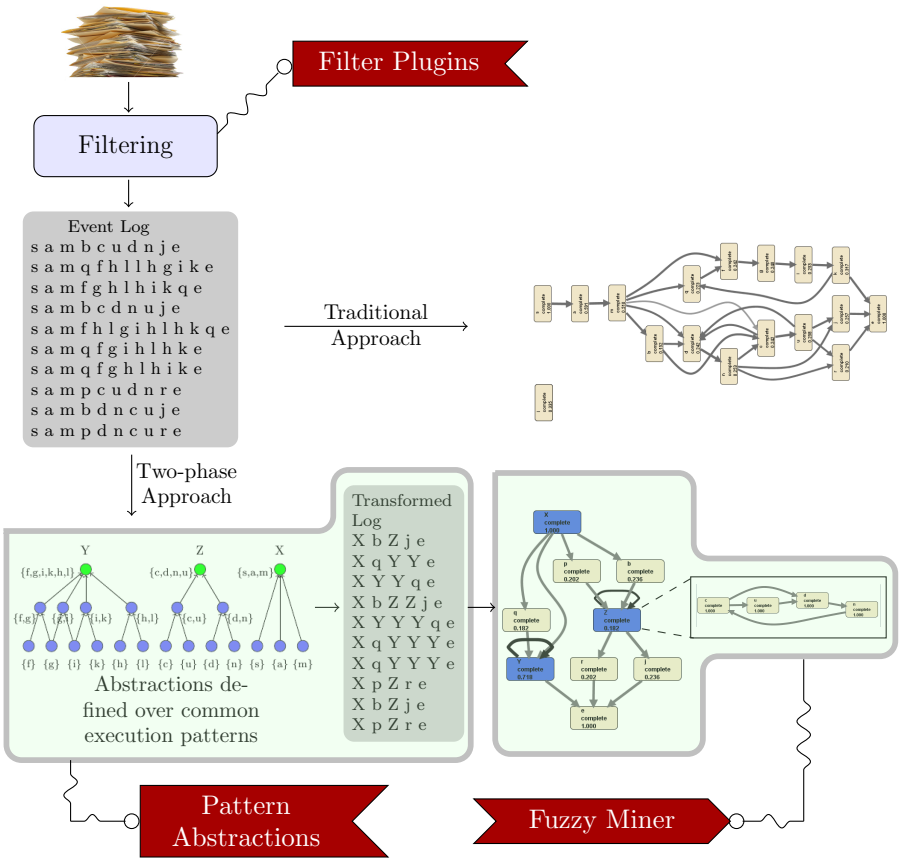


Fig. 2: Traditional approach versus our two-phase approach to process discovery. ProM plugins are used to filter the event log. ProM's *Pattern Abstractions* plugin and the *Fuzzy Miner* plugin are used to realize simple and intuitive models.

3 Discovering Hierarchical Processes

The *two-phase approach to process discovery* described in Section 2 enables the discovery of hierarchical process models [9]. In this paper, we demonstrate this using a chain of plugins implemented in ProM. The chain of plugins and their order of application is illustrated in Fig. 3. The event log may first be cleansed using some simple filters (e.g., adding artificial start/end events, filtering events

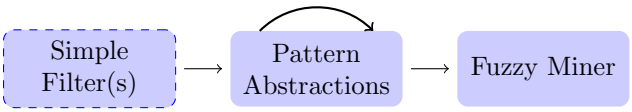


Fig. 3: The chaining of plugins that enables the discovery of hierarchical process models.

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 discovery 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 [6] 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.

In the next two sections, we explain the functionality and features of the Pattern Abstractions and the (Enhanced) Fuzzy Miner plugins.

4 Pattern Abstractions Plugin

The basic building blocks of the Pattern Abstractions plugin are shown in Fig. 4. Figures 5 and 6 illustrate these building blocks.

Fig. 4: 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

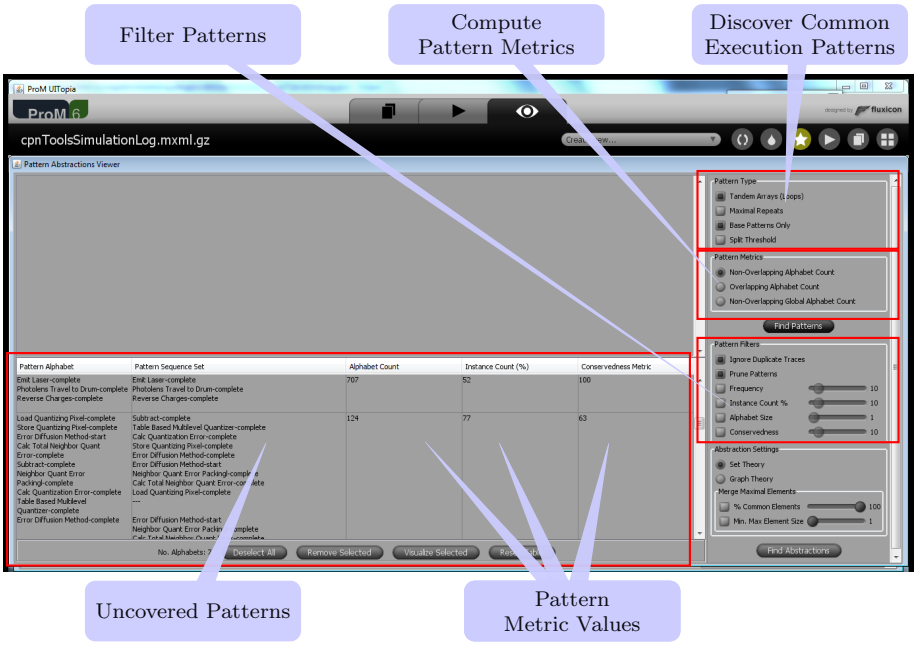


Fig. 5: The discovery of common execution patterns, computation of pattern metrics, filtering and inspection of patterns in the Pattern Abstractions plugin.

(common subsequence of activities within a process instance or across process instances) [8]. These can be uncovered in *linear* time and space with respect to the length of the traces.

- *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 [8]. Furthermore, various features to edit/select abstractions such as merging two or more abstractions and deleting activities related to a particular abstraction are supported. Fig. 6 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.

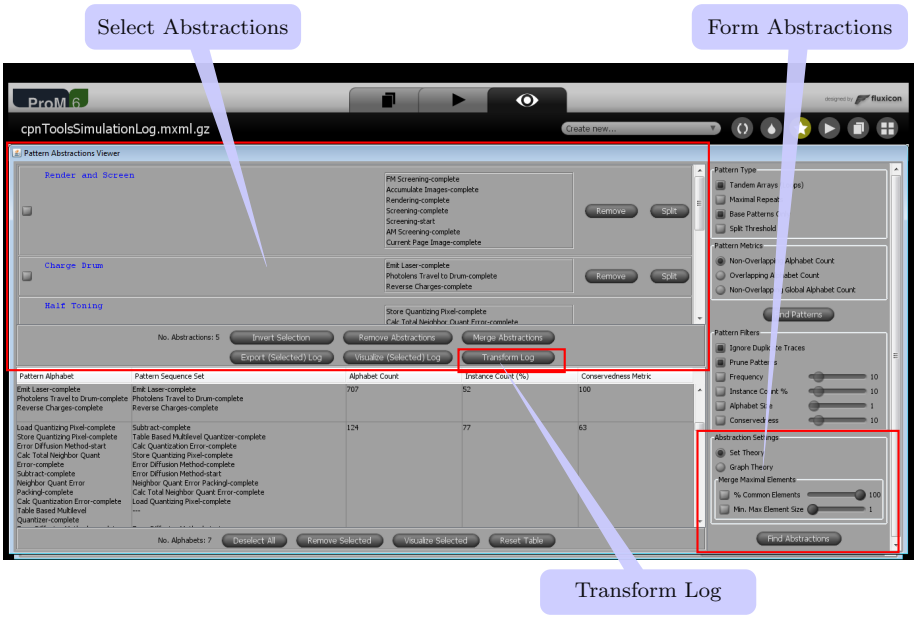


Fig. 6: The generation and selection of abstractions in the Pattern Abstractions plugin.

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

5 (Enhanced) Fuzzy Miner Plugin

The Fuzzy Miner [6, 15] is a process discovery technique 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 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.

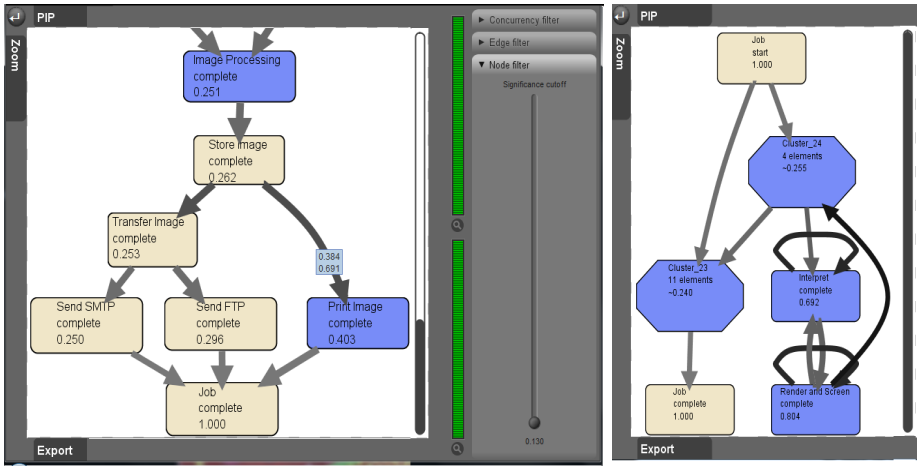


Fig. 7: Fuzzy model and instance.

Fig. 7 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

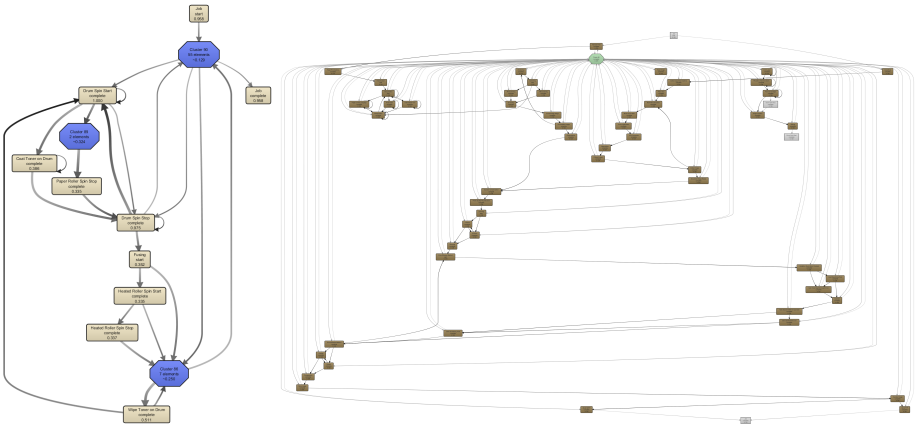
and the other contains 11 activities). All activities on the left hand side except "Job Complete" are contained in a cluster node on the right. Apparently, the significance weights for these nodes (0.262, 0.253, 0.250, 0.296 and 0.403) were too low to be shown, which indicates that the corresponding 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 (called as the Fuzzy Map miner) 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 Fig. 7). Thus, the enhanced fuzzy miner gives the user more control over the discovery process than the classical fuzzy miner [6].

6 Experimental Results and Discussion

In this section, we demonstrate the discovery of hierarchical processes using the concepts presented in this paper firstly on the synthetic log of the digital copier example and later on a real-life case study. We contrast the process models discovered using the proposed approach with the models uncovered using the classical Fuzzy miner. Fig. 8(a) depicts the process model mined on the digital copier event log using the classical Fuzzy miner [6]. As discussed in Section 5, the Fuzzy miner plugin groups activities into cluster nodes (blue colored nodes in Fig. 8(a)) based on their significance. Fig. 8(b) depicts the process model obtained upon zooming one of the cluster nodes containing 55 activities. As we can see, the model is Spaghetti-like and hard to comprehend. The cluster nodes formed by the classical fuzzy miner do not have any semantic significance with respect to the domain/application. In other words, the classical fuzzy miner poses the risk of aggregating unrelated nodes together in a cluster (using the map metaphor, this is similar to streets in Eindhoven being clustered along with streets in Amsterdam).

Fig. 9 depicts the top-level process model of the copier example discovered using the chain of plugins proposed in this paper. 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.* Note the similarity of



(a) top-level model (b) process model upon zooming the cluster node 'cluster 90'

Fig. 8: Process model mined using the classical fuzzy miner.

this discovered model with the original model in Fig. 15. The *Interpret* and *Render and Screen* abstractions in Fig. 9 are within the Rasterize Image composite task of Fig. 15.

Upon zooming in on the *Image Processing* abstraction, the process model depicted in Fig. 10 is shown. Note the similarity of this discovered model with that of the original model in Fig. 16. 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 Fig. 10. Note the similarity of this discovered model for half toning with that of the original model in Fig. 17. Fig. 11 depicts two other abstractions viz., *Interpret* and *Fusing*. Note the similarity of the discovered model for *Fusing* with that of the original model in Fig. 18.

In this fashion, using the chain of plugins presented in this paper, one can discover hierarchical process models. We next present the results of applying this approach in a real-life case study. The case study was performed in close collaboration with *Philips Healthcare* and pertains to the analysis of event logs generated by the *X-ray machines*. More specifically, we considered an event log capturing the activities performed by field service engineers during one of the part replacements on X-ray machines. The event log contains 113 cases and 76, 754 events referring to 92 activities. Fig. 12 depicts the process model uncovered on this log using the classical Fuzzy miner. Again, we see an incomprehensible spaghetti-like model. We uncover common execution patterns, define abstractions, and use our two-phase approach to alleviate this problem. The transformed log (with abstractions) contains 113 cases and 10, 387 events referring to 20 activities. Fig. 13 depicts the top-level process model discovered using the Fuzzy map miner on this

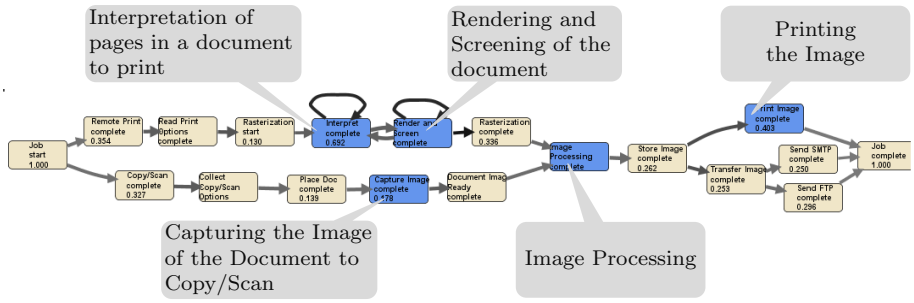


Fig. 9: The top level process model of the copier event log. Blue (dark colored) nodes are abstract activities that can be zoomed in. Upon zooming in, the sub-process defining the abstraction is shown.

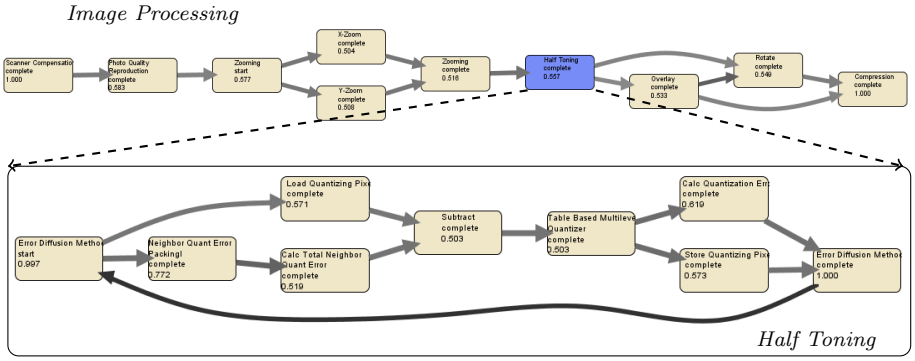


Fig. 10: The sub-process captured for the abstraction 'Image Processing' (in the top-level model). This sub-process in turn contains another abstraction viz., 'Half Toning'. Upon zooming in on 'Half Toning', the sub-process defining that is shown.

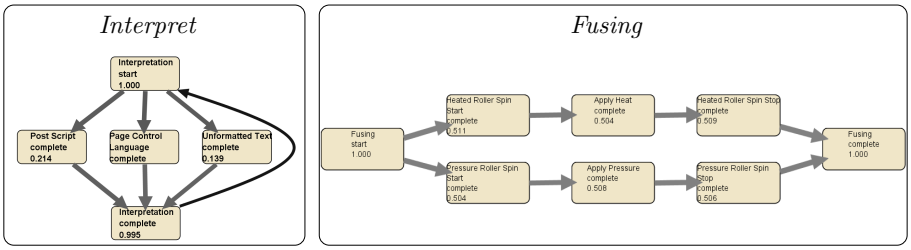


Fig. 11: The sub-processes for the abstractions 'Interpret' and 'Fusing'. 'Interpret' is an abstract activity at the top-level of the process model while 'Fusing' is an abstract activity underneath the 'Print Image' abstraction.

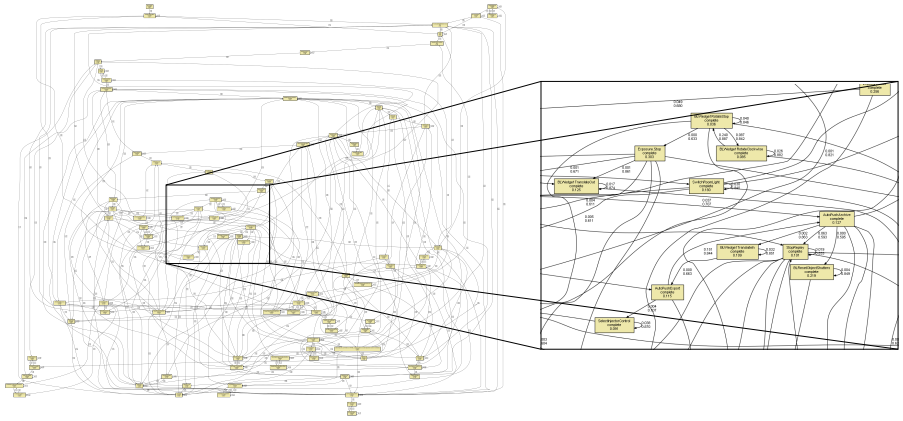


Fig. 12: Process model discovered using the classical Fuzzy miner on the event log capturing the activities of field service engineers during one of the part replacements in X-ray machines.

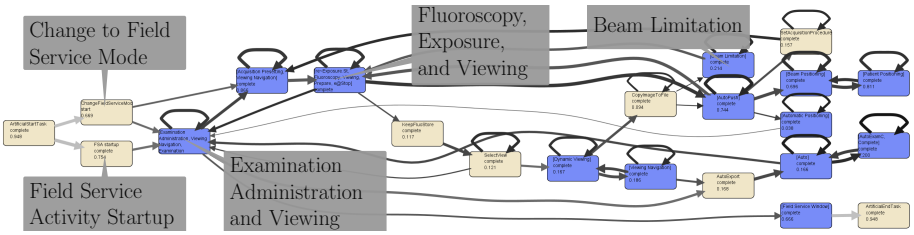
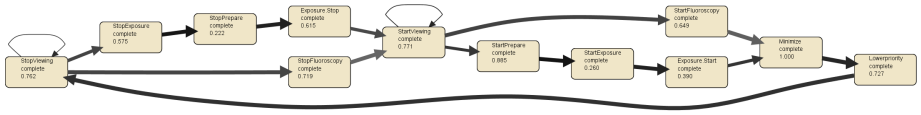


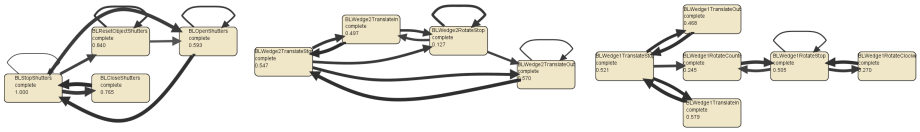
Fig. 13: The top-level of the hierarchical process model using the approach proposed in this paper on the event log capturing the activities of field service engineers during one of the part replacements in X-ray machines.

transformed log. We can see that the model discovered using our two-phase approach is simpler and more comprehensible. Fig. 14(a) depicts the sub-process for the abstract activity Fluoroscopy, Exposure, and Viewing while Fig. 14(b) depicts the sub-process for the abstract activity Beam Limitation.

We have used this approach on several other real-life event logs as well (e.g., see [16]). Our experiences show that the automated abstractions uncovered by the Pattern Abstractions plugin have strong domain (functional) significance. This can be attributed to that fact that these abstractions are obtained by exploiting the common execution patterns in the event log. Common subsequences of activities in an event log that are found to recur within a process instance or across process instances tend to have some domain (functional) significance. One of the limitations of our approach is the semi-automated means of defining abstractions. Currently, the plugin requires the manual merging/pruning of patterns. In the future, we would like to automate some of these functionalities.



(a) Fluoroscopy, Exposure and Viewing



(b) Beam Limitation

Fig. 14: The sub-processes corresponding to the abstract activities Fluoroscopy, Exposure and Viewing and Beam Limitation.

7 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 comprehensible process maps.

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.

References

1. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer-Verlag New York Inc (2011)
2. van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow Mining: Discovering Process Models from Event Logs. IEEE Transactions on Knowledge and Data Engineering **16**(9) (2004) 1128–1142
3. Weijters, A.J.M.M., van der Aalst, W.M.P.: Rediscovering Workflow Models From Event-based Data Using Little Thumb. Integrated Computer-Aided Engineering **10**(2) (2003) 151–162
4. van der Werf, J.M., van Dongen, B.F., Hurkens, C., Serebrenik, A.: Process Discovering Using Integer Linear Programming. Applications and Theory of Petri Nets (2008) 358–387
5. van Dongen, B.F., de Medeiros, A.K.A., Wen, L.: Process Mining: Overview and Outlook of Petri Net Discovery Algorithms. Transactions on Petri Nets and Other Models of Concurrency II (2009) 225–242
6. 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

7. Görg, C., Pohl, M., Qeli, E., Xu, K.: Visual Representations. In Kerren, A., Ebert, A., Meye, J., eds.: Human-Centered Visualization Environments. Volume 4417 of LNCS. Springer (2007) 163–230
8. 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
9. 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
10. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: Yet Another Workflow Language. Information Systems **30**(4) (2005) 245–275
11. Jensen, K., Kristensen, L.M.: Colored Petri Nets: Modeling and Validation of Concurrent Systems. Springer-Verlag New York Inc (2009)
12. Medeiros, A.K.A.D., Günther, C.W.: Process Mining: Using CPN Tools to Create Test Logs for Mining Algorithms. In: Proceedings of the Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools. (2005) 177–190
13. Nigam, A., Caswell, N.S.: Business Artifacts: An Approach to Operational Specification. IBM Systems Journal **42**(3) (2003) 428–445
14. van der Aalst, W.M.P., Barthelmess, P., Ellis, C.A., Wainer, J.: Proclets: A Framework for Lightweight Interacting Workflow Processes. International Journal of Cooperative Information Systems **10**(4) (2001) 443–482
15. Xia, J.: Automatic Determination of Graph Simplification Parameter Values for Fuzzy Miner. Master's thesis, Eindhoven University of Technology (2010)
16. Bose, R.P.J.C., van der Aalst, W.M.P.: Analysis of Patient Treatment Procedures: The BPI Challenge Case Study. Technical Report BPM-11-18, BPMCenter.org (2011) `http://bpmcenter.org/wp-content/uploads/reports/2011/BPM-11-18.pdf`.

A YAWL Models of the Digital Photo Copier

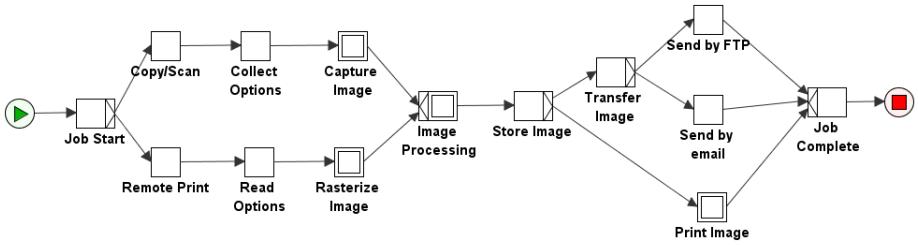


Fig. 15: High-level model of the digital photo copier. The digital copier supports two functionalities viz., copy/scan and print. Documents are interpreted and converted into an image before they are printed. Scanned images of the copy/scan jobs are sent to the user via email or ftp.

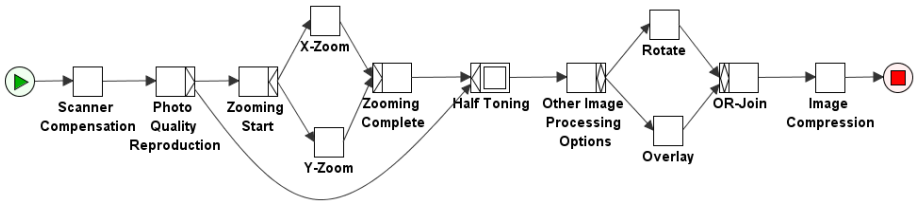


Fig. 16: The image processing sub-process. This subprocess supports operations such as zooming, rotating and overlay and attempts at improving the quality of images.

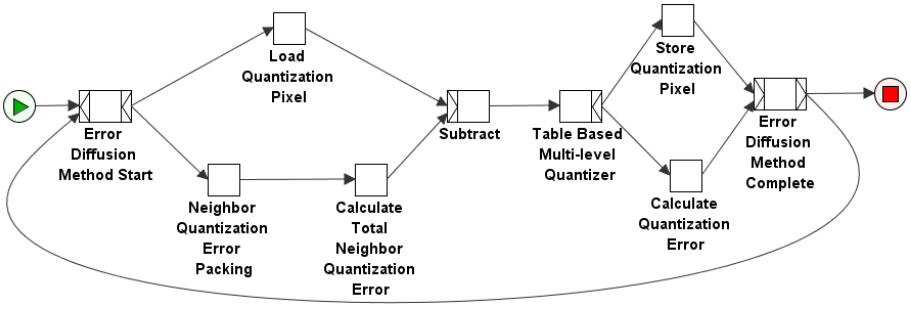


Fig. 17: The half-toning sub-process for image representation. Half toning is a technique that simulates continuous tone imagery through the use of equally spaced dots of varying size.

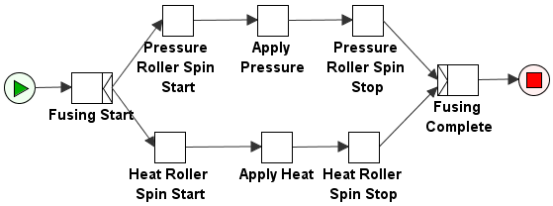


Fig. 18: The fusing sub-process within print image.