The Automated Discovery of Hybrid Processes

Fabrizio Maria Maggi[1], Tijs Slaats[2,3], and Hajo A. Reijers[4,5]

[1] University of Tartu, Estonia
[2] IT University of Copenhagen, Denmark
[3] Exformatics A/S, Lautrupsgade 13, 2100 Copenhagen, Denmark
[4] Eindhoven University of Technology, The Netherlands
[5] Perceptive Software, The Netherlands
f.m.maggi@ut.ee, tslaats@itu.dk, h.a.reijers@tue.nl

Abstract. The declarative-procedural dichotomy is highly relevant when choosing the most suitable process modeling language to represent a discovered process. Less-structured processes with a high level of variability can be described in a more compact way using a declarative language. By contrast, procedural process modeling languages seem more suitable to describe structured and stable processes. However, in various cases, a process may incorporate parts that are better captured in a declarative fashion, while other parts are more suitable to be described procedurally. In this paper, we present a technique for discovering from an event log a so-called *hybrid* process model. A hybrid process model is hierarchical, where each of its sub-processes may be specified in a declarative or procedural fashion. We have implemented the proposed approach as a plug-in of the ProM platform. To evaluate the approach, we used our plug-in to mine a real-life log from a financial context.

1 Introduction

Process models are an important aid to capture how business operations are organized. One direction to simplify the tasks of creating, maintaining, and reading such models involves the use of declarative techniques for process modeling. In contrast to the procedural approach, which is dominant for modeling business processes, a declarative approach leaves implicit in what exact sequences activities must be carried out. Instead, the emphasis is on the constraints that must be respected in carrying out the process – any behavior that respects these goes. In contexts where activities can be executed in highly different combinations, a declarative approach arguably produces simpler representations of the involved process logic. Examples of concrete declarative modeling techniques are Declare, DCR Graphs [3], and SCIFF.

In [10], we reported that a *hybrid* process modeling technique was considered by practitioners as more attractive than a completely declarative or procedural one. Hybrid, in this context, refers to the potential use of both procedural and declarative model elements in the same model. The rationale is that the two types of modeling paradigms allow for a natural fit with different types of process behavior. In places where the process is highly flexible, a declarative modeling approach leads to a compact and simple description of such a "pocket of flexibility" [11]. Instead of describing all the different types of feasible behavior, the focus is then on ruling out what is not allowed (if

anything). By contrast, for parts of the process that are highly structured, a procedural description may be the way to go: It is then simpler to describe what is allowed than what is to be ruled out. For processes that both incorporate structured and unstructured pockets, a hybrid model delivers a compact and simple description.

This paper should be seen as a direct follow-up to our earlier work. Specifically, we developed a technique *to automatically generate a hybrid model from an event log*. This is a novel contribution, since existing techniques can only generate a process model that is either procedural or declarative. By contrast, our technique flexibly alternates between employing a procedural or declarative mining approach in accordance with the nature of the traces it processes. By doing so we are able to avoid the "spaghetti"-like process models that are commonly generated by traditional process mining techniques.

Against this background, the paper is structured as follows. In Section 2, we will outline the notion of a hybrid model and pinpoint its semantics. Section 3 describes our core contribution, the discovery approach. We will will evaluate this approach in Section 4 on a real-life log. After a discussion of related work, we conclude this paper with a reflection on the presented work and future steps in Section 6.

2 Semantics of a Hybrid Model

Our interest in this paper is with hybrid models where the procedural and declarative parts are contained in separate sub-processes. In this sense, there is a resemblance with the *pockets of flexibility* concept [11]. A hybrid process consists of a procedural or declarative top-level process, which may contain a number of atomic activities as well as sub-processes. Each sub-process can be either procedural or declarative and may contain sub-processes of its own. Our approach is applicable to any combination of procedural and declarative languages, but in this paper we will apply Petri nets for our procedural models and Declare [9] for our declarative models. Sub-processes are considered atomic, meaning that once a sub-processes is started the control is passed from the parent process to that sub-process. No other activities can be executed until the sub-process has completed. A sub-process can only complete while it is *accepting*. When exactly it is accepting depends on the language used. In the case of a Petri net this means reaching a final marking, while for a Declare model it means having no violated constraints. For the language of a hybrid model we consider the start and completion of sub-processes as silent transitions, which means that there will be no start- and complete-events for the sub-processes in the log. This underlines the fact that the sub-processes are really just a tool for improving the understanding of the process and not a part of the actual enactment of the process.

3 Discovering Hybrid Process Models

Fig. 1 gives an overview of our approach. In the following paragraphs we describe our approach step by step.

Distinguishing Structured and Unstructured Events. We start by separating the events of the log into two distinct sets: one containing those events that occur in a structured

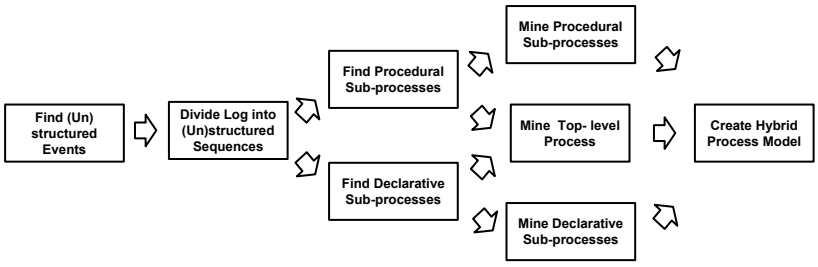

Fig. 1. Overview of our approach

context and one containing those events that occur in an unstructured context. To distinguish structured from unstructured events we use a novel technique, which we refer to as *context analysis*. Our first step is to determine for each event the number of unique predecessors and successors to that event. We then consider an event with a large number of both predecessors and successors to be unstructured (according to a user-defined threshold, in our experimentation we used 4), while an event with a small number of predecessors or a small number of successors is considered to be structured. The reasoning behind these cases is as follows: if an event has a high number of predecessors and a high number of successors, then there are few rules constraining when exactly the event can occur. We then consider it likely to fit well into a declarative model. Similarly, if an event has only a small number of predecessors and a small number of successors, then it is probably more easily modeled procedurally, for example, as a sequence or a choice from a low number of options. In the case that an event has a small number of predecessors, but a large number of successors, we consider it likely that the event is either the last element in a structured sequence, which is followed by an unstructured sequence, or that the event is followed by a choice from a large number of options. In both cases it makes sense to consider this as a structured event and model it procedurally. Similarly, in the case that an event has a small number of successors, but a large number of predecessors we consider it likely that the event is either the first element in an structured sequence, which was preceded by an unstructured sequence, or that the event joins a choice from a large number of options. In both cases it seems fitting to consider this as a structured event and model it procedurally.

Dividing the Log into Structured and Unstructured Sequences. The context analysis gives us two sets: one that contains structured events and one that contains unstructured events. In the following step, we use these events to identify structured and unstructured sequences by parsing the log and starting a new sequence whenever an event does not belong to the same set as its preceding event. After this step, our approach splits into two branches, one handling the structured sub-logs and the other handling the unstructured sub-logs.

Finding and Mining procedural Sub-processes. By grouping together each structured event with all the direct successors and all the direct predecessors, we obtain a set of disjoint clusters of the structured sequences. We then mine procedural sub-processes for

each of these clusters. Finally, we abstract the main log by replacing each sequence with the identifier of the sub-process that it belongs to. It should be noted that the clusters of procedural sequences that are discovered could be further split up using existing clustering techniques. This did not seem necessary on basis of the examples we used in our experimentation with the technique.

Finding and Mining Declarative Sub-processes. For finding declarative sub-processes, we first use an indirect association rule mining algorithm on the set of unstructured sequences to find declarative patterns. Recall that an indirect association rule can be used to find events that rarely occur together, yet there are other "mediator events" with which they appear relatively frequently. We use this type of algorithms since it gives us the opportunity to not only discover Declare constraints that express positive relations, but also constraints like, for example, not coexistence constraints, which are more likely to be satisfied when the events involved do not occur together in the same trace. In a second stage, we use a mining algorithm for standard association rules on the remaining sequences. These rules reflect relationships that exist between events that often co-occur in common transactions. For this reason, these rules allow us to group together events that are very likely connected with each other through positive relations in Declare. The patterns are abstracted in the main log, and any remaining event that is at this point not identified as belonging to a declarative pattern is left as an atomic event.

Mining the Top-level Process. When we are done finding (but not necessarily mining) procedural and declarative sub-processes and have all sub-processes abstracted in the main log, we can then either choose to apply the approach iteratively on the abstracted log, starting from the first step where we distinguish structured and unstructured activities based on their context, or we can choose to finalize the approach by mining the main log. In the latter case, we compute the average string edit distance for all traces in the abstract main log and in case of a high similarity among the traces (>50%) we mine the log procedurally using a procedural miner. In case of a lower similarity, we mine it using a declarative miner. The use of the string edit distance is a simple way to distinguish between structured and unstructured logs. We validated this approach based on experiments on synthetic logs. The results of these experiments have shown that traces in structured logs are more similar to each other with respect to traces in an unstructured log. Of course, more sophisticated techniques can be used for discriminating between them.

Creating a Hybrid Process Model. When all mining tasks have finished, we can combine the resulting process models into a single hybrid model, based on which abstract activities in the top-level model correspond to which sub-process. The exact method will depend on the miners used and the languages that they use to generate models. In our implementation, we simply generated separate Petri nets and Declare models. However, to improve usability, a tool that supports the visualization and management of such hybrid models would be needed (for example, to graphically represent a Declare sub-process within a top-level Petri net model). At this point, this is left for future work.

Fig. 2. Procedural Models

4 Evaluation

To evaluate our approach, we have implemented it as a plug-in of the process mining tool ProM.[1] For the evaluation, we turned to the real-life event log, which was made available as part of the BPI Challenge 2012.[2] The process represented in the event log is an application process for a personal loan within a global financing organization. The log itself contains some 262.200 events in 13.087 cases.

Our evaluation took on the following form. We set out to compare a model that would result from a traditional mining approach on the selected log with a hybrid model that is generated as proposed in this paper. We will refer to these as the *procedural* and the *hybrid* models. The aim then is to compare these models specifically with respect to the understandability of the generated models. We decided to create three procedural models of the event log by using the Alpha, Heuristic, and ILP miner, respectively. The resulting procedural models are shown in Fig. 2.

We created the hybrid model by using the Declare miner on the clusters of unstructured sequences, while using the Heuristic miner on the clusters of structured sequences. Since the root model in this case also could be classified as structured, it was mined with the Heuristic miner as well. The hybrid model can be seen in Fig. 3. In this figure, the procedural root net is shown, as well as links to its sub-nets. Note that two sub-nets are of a declarative nature (D1.1 and D2.1); the other sub-nets are procedural.

To make sure that a comparison with respect to the simplicity of the various models is fair, we first reflect on their *fitness* [13]. This expresses how well the model is able to "replay" the observed behavior in the log. The values are provided in Table 1. As can be seen, the fitness values for the procedural models range from 0.01 for the ILP

[1] `http://www.promtools.org/prom6/HybridMiner`
[2] `http://dx.doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f`

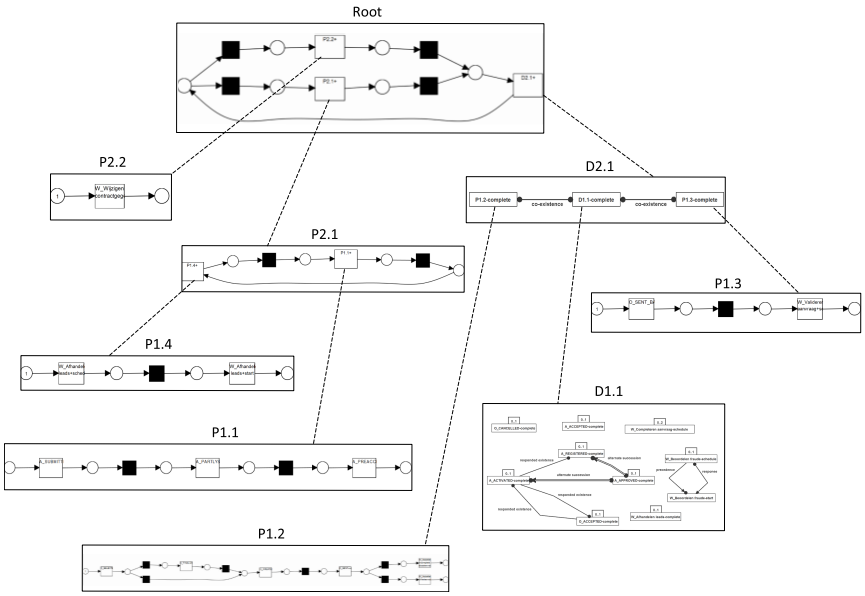

Fig. 3. Hybrid Model

miner to 0.58 for the Alpha miner. For the hybrid model, the fitness values are provided for each of the sub-nets. These values vary from 0.69 to 1.00 (perfect fitness). Without an integrated fitness measure available for hierarchical nets, we propose to take the minimum value as a conservative approximation for the fitness of the hybrid net. On this basis, the replay fitness of the hybrid net can be seen to be at least as good as that of the procedural models. Also, it is not particularly "flower-like", which can be a drawback of aiming at a well-fitting model [13].

A visual inspection of the models seems to indicate that the hybrid model is vastly simpler than the procedural models. All procedural models can be characterized as "spaghetti-like". The hybrid model, by contrast, is composed of 9 different sub-nets, each of which having a fairly simple structure. Arguably the most difficult of these sub-nets are P1.2 and D1.1, which are the largest procedural and declarative sub-nets, respectively. While the modularity of the hybrid model to some extent seems to help the understanding of the process, the overall lack of visual clutter is apparent. The proposed

Procedural			**Hybrid**								
Alpha	**Heuristic**	**ILP**									
			Root	P1.1	P1.2	P1.3	P1.4	P2.1	P2.2	D1.1	D1.2
0.58	0.40	0.01	1.00	0.84	0.73	0.69	0.81	1.00	0.86	1.00	1.00

Table 1. Fitness values for the generated models

approach, therefore, seems to have the potential to automatically generate behaviorally accurate process models that are simple to read.

5 Related Work

Several approaches in the literature focus on the discovery of declarative process models [1,4,2,5,6,7,8]. The algorithms proposed in [4,2,6,8] are tailored to discover Declare specifications. In particular, the technique proposed in [4,2] is based on a two-step approach. First, the input event log is parsed to generate a knowledge base containing information useful to discover a Declare model. Then, in a second phase, the knowledge base is queried to find the set of constraints that hold on the input log. The work proposed in [6] is based on an Apriori algorithm for association rule mining and has been used in this paper for the discovery of the declarative sub-processes of a hybrid model. The approaches proposed in [1,5] are more general and allow for the specification of rules that go beyond the traditional Declare templates. However, these approaches can be hardly used in real-life settings since they are based on supervised learning techniques thus requiring negative examples that are difficult to be derived from real data. In the work proposed in [7], a first-order variant of LTL is used to specify a set of data-aware patterns. Such extended patterns are used as the target language for a process discovery algorithm to produce data-aware Declare constraints from raw event logs.

A recent implementation of a hybrid process modeling technique is made available in CPN Tools 4.0 [15]. Different than what is proposed in the paper at hand, a hybrid CPN net allows for the use of procedural and declarative modeling elements within the same sub-process. Already at an earlier stage, modeling approaches have been proposed that embrace "pockets of flexibility". Specifically, in [11] it is proposed to define at build-time in a workflow process pockets in a way that is highly similar to a declarative style to match their highly flexible behavior; at runtime one has to pick a specific procedural instantiation of the workflow that fits the definition. Two other approaches that combine procedural and declarative elements worth noting are Flexibility-as-a-Service (FAAS) [14] and the Guard-Stage-Milestone model [12]. It should be noted that for none of these approaches automated discovery techniques exist.

6 Conclusion

In this paper, we presented an automated discovery technique for hybrid process models. By analyzing the traces that are available in an event log and clustering them together according to their structure (or lack thereof), we are able to mine the structured and less structured pockets within a process with procedural and declarative mining algorithms, respectively. The result is a hierarchical process model with both procedural and declarative sub-processes. Our evaluation on a real-life event log suggests that the proposed technique is indeed capable of producing a much simpler representation of a process than traditional, purely procedural approaches can.

The proposed approach could be improved along theoretical, technical, and empirical angles. On a theoretical side, there is a need to establish proper metrics that tie to the established quality dimensions of fitness, precision, generalization and simplicity [13]

for hybrid, hierarchical process models. At this point, it is not entirely clear how a quality measure for a subprocess propagates to the quality of the overall model. Establishing this will pave the way for a more thorough insight into the strengths and weaknesses of the proposed discovery technique. Technically, a step ahead would be to allow for *duplicate events*, i.e. the same event can be part of a procedural as well as a declarative sub-process. We did not allow for this at this point, but this could be done by identifying "recurrent" predecessors/successors even if these appear only in a certain percentage of cases. From an empirical angle, end users need to be confronted with hybrid models for a thorough evaluation of their usefulness and ease of use.

As to stimulate the uptake of hybrid process models, a number of other developments are called for as well. As we pointed out in our earlier work [10], modeling guidelines and tool support will be essential to allow for the manual creation and maintenance of hybrid process models. We are currently experimenting with such guidelines and our initial insights are that modelers with an intermediate experience with procedural modeling approaches do not find the composition of hybrid models all that difficult. We hope to report on more substantial insights in the near future.

References

1. F. Chesani, E. Lamma, P. Mello, M. Montali, F. Riguzzi, and S. Storari. Exploiting inductive logic programming techniques for declarative process mining. *ToPNoC*, 2009.
2. Claudio Di Ciccio and Massimo Mecella. A two-step fast algorithm for the automated discovery of declarative workflows. In *CIDM*, 2013.
3. Søren Debois, Thomas Hildebrandt, and Tijs Slaats. Hierarchical declarative modelling with refinement and sub-processes. In *BPM*, 2014.
4. C. Di Ciccio and M. Mecella. Mining constraints for artful processes. In *BIS*, 2012.
5. E. Lamma, P. Mello, F. Riguzzi, and S. Storari. Applying inductive logic programming to process mining. In *Inductive Logic Programming*, volume 4894. 2008.
6. F.M. Maggi, R.P.J.C. Bose, and W.M.P. van der Aalst. Efficient discovery of understandable declarative models from event Logs. In *CAiSE*, pages 270–285, 2012.
7. F.M. Maggi, M. Dumas, L. García-Bañuelos, and M. Montali. Discovering data-aware declarative process models from event logs. In *BPM*, pages 81–96, 2013.
8. F.M. Maggi, A.J. Mooij, and W.M.P. van der Aalst. User-guided discovery of declarative process models. In *CIDM*, pages 192–199, 2011.
9. M. Pesic, H. Schonenberg, and W.M.P. van der Aalst. DECLARE: Full Support for Loosely-Structured Processes. In *EDOC 2007*, pages 287–298, 2007.
10. H.A. Reijers, T. Slaats, and C. Stahl. Declarative modeling – An academic dream or the future for BPM? In *BPM*, pages 307–322, 2013.
11. S.W. Sadiq, M.E. Orlowska, and W. Sadiq. Specification and validation of process constraints for flexible workflows. *Information Systems*, 30(5):349–378, 2005.
12. R. Vaculín, R. Hull, T. Heath, C. Cochran, A. Nigam, and P. Sukaviriya. Declarative business artifact centric modeling of decision and knowledge intensive business processes. In *EDOC*, pages 151–160, 2011.
13. W.M.P. van der Aalst. *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.
14. W.M.P. van der Aalst, M. Adams, A.H.M. ter Hofstede, M. Pesic, and H. Schonenberg. Flexibility as a service. In *Database Systems for Advanced Applications*, 2009.
15. M. Westergaard and T. Slaats. Mixing paradigms for more comprehensible models. In *Business Process Management*, pages 283–290. 2013.