Mining Hybrid Business Process Models: A Quest for Better Precision

Dennis M. M. Schunselaar[1], Tijs Slaats[2⋆],
Fabrizio M. Maggi[3], Hajo A. Reijers[1], and Wil M.P. van der Aalst[4]

[1] Vrije Universiteit Amsterdam, The Netherlands
[2] University of Copenhagen, Denmark
[3] University of Tartu, Estonia
[4] RWTH Aachen University, Germany
d.m.m.schunselaar@vu.nl, slaats@di.ku.dk, f.m.maggi@ut.ee,
h.a.reijers@vu.nl, wvdaalst@pads.rwth-aachen.de

Abstract. In this paper, we present a technique for the discovery of hybrid process models that combine imperative and declarative constructs. In particular, we first employ the popular Inductive Miner to generate a fully imperative model from a log. Like most imperative miners, the Inductive Miner tends to return so-called flower models for the less structured parts of the process. These parts are often imprecise. To counter these imprecise parts, we replace them with declarative models to increase the precision since declarative models are good at specifying which behavior is disallowed. The approach has been implemented in ProM and tested on several synthetic and real-life event logs. Our experiments show that hybrid models can be found to be more precise without overfitting the data.

Keywords: Hybrid Process Model, Process Mining, Process Discovery, Process Tree, Declare

1 Introduction

In recent years, different comparative investigations have been conducted to better understand the distinctive characteristics of imperative and declarative process modeling languages and to support the choice of the most suitable paradigm to be used in different scenarios [25,26]. While advantages and limitations of the two paradigms are still a matter of investigation, both in academic research and in industry [12,21], a trend has emerged to consider hybrid approaches combining a mixture of imperative and declarative specifications. The motivations behind this trend rely on the surmise that many real-life processes are characterized by a mixture of *(i)* less structured processes with a high level of variability, which can usually be described in a compact way using declarative languages such as Declare [24], SCIFF [23], or DCR Graphs [10], and *(ii)* more stable processes

⋆ This work is supported in part by the Hybrid Business Process Management Technologies project funded by the Danish Council for Independent Research.

with well-structured control flows, which are more appropriate for traditional imperative languages such as Petri nets and BPMN.

As a result, there have been several recent efforts to fully develop and formalize such hybrid modeling notations and methodologies. These hybrid approaches can be categorized based on how tightly the imperative and declarative paradigms are integrated, with each approach falling in one of three mayor categories: *(i)* a *complementary* approach, where imperative and declarative notations are used in separate models, without any semantic interaction, but with the aim of representing different aspects of the same process (see, e.g., [11]). *(ii)* A *hierarchical* approach, where both imperative and declarative notations are used in the same overall model, but the model is separated into several sub-processes and each sub-process uses a single uniform notation. To our knowledge most existing proposals for hybrid approaches fall within this category (see, e.g., [1,27,30]). *(iii)* A fully *mixed* approach, where imperative and declarative constructs fully overlap in the same model. While this approach can be considered the most flexible, it is also the most complex since the modeler needs to consider and understand in minute detail how the elements of the different notations interact. An example of such an approach is the one proposed in [36].

One field where the hybrid approach may lead to significant advances is that of automatic process discovery [5], which aims at generating useful process models from real-life event logs. Many imperative and declarative process discovery algorithms (usually referred to as miners) exist, and it remains an open challenge to determine a suitable discovery approach given the characteristics of the event logs [7]. When faced with very variable logs, imperative miners have a tendency to create "spaghetti models", or "flower models". The former are generally unreadable, the latter allow any behavior and therefore contain very little useful information about the process. Declarative miners, on the other hand, are not well-suited to dealing with very structured logs: when faced with such a log they will often generate an abundance of constraints, which makes the model unreadable for human users. Hybrid process discovery approaches aim at providing a middle way between these two extremes, generating an imperative model for the structured parts of the log and declarative constraints for the unstructured ones. Two primary approaches to hybrid discovery have been considered: *(i)* a *model-driven* approach, where first an imperative or a declarative model is created, from which "bad" parts are identified (e.g., flower sub-models in imperative models, or over-constrained activities in declarative models) and secondly these "bad" parts are re-mined with a more suitable mining algorithm. *(ii)* A *log-driven* approach where an up-front analysis of the log is used to identify structured and unstructured parts, which are then separated into sub-logs. Each sub-log is mined with the most suitable discovery algorithm.

We can categorize the full spectrum of hybrid miners by orthogonally considering their overall mining technique and the type of models that they output. The current paper falls within the Hierarchical/Model-driven section of this categorization. In particular, we propose a hybrid miner that builds on the Inductive Miner [16] by Leemans et al., which, for a given input log, generates an impera-

tive process model in the form of a process tree. The Inductive miner guarantees a fitness of 1. This guarantee sometimes comes at the cost of precision, e.g., when no structure can be found in the log, the Inductive miner defaults to a flower loop, which is able to capture all behavior and much more. Being hierarchical in nature, process trees can be easily adopted into a hierarchical hybrid approach. To do so, we take the process tree returned from the Inductive miner and identify nodes from the process tree to be replaced with Declare models. We have used Declare since there are several well accepted mining algorithms for this notation, whereas support to process discovery for other declarative notations is either non-existent or in a very early stage.

In this paper, we primarily focus on the precision of the model. There are two main reasons for this: (1) precision, contrary to other quality dimensions like simplicity, has widely accepted metrics for measuring it, and (2) flower loops tend to be very imprecise since they allow for any possible behavior, thus giving an opportunity to have a more accurate understanding of the process captured in the event log by replacing them with declarative models.

The approach has been implemented as a plug-in of the process mining tool ProM and tested on both synthetic and real-life logs. The evaluation shows positive results for both synthetic and real-life logs, with up to 52.18% improvement in the precision of the mined models for synthetic logs and up to 3471% increase in precision for the real-life logs.

The rest of the paper is structured as follows: in Section 2, we discuss the related work. In Section 3, we introduce the hybrid models we use. In Section 4, we describe the proposed approach. In Section 5, we report on our evaluation. Section 6 concludes the paper and spells out directions for future work.

2 Related Work

In the literature, there are different families of approaches that have been proposed for the discovery of imperative process models. The first family of approaches extracts some footprint from the event log and uses this footprint to directly construct a process model (see, e.g., [5,34]). The second family of process discovery approaches first constructs an intermediate model and then converts it into a more refined model that can express concurrency and other advanced control-flow patterns (e.g., [4]). The third family of approaches tries to break the discovery problem into smaller problems. For example, the Inductive miner [16] aims at splitting the event log recursively into sub-logs, which are eventually mined. Techniques originating from the field of computational intelligence form the basis for the fourth family of process discovery approaches. An example of this type of approaches is the genetic mining approach described in [22].

In the area of declarative process discovery, Maggi et al. [18,19] first proposed an unsupervised algorithm for mining Declare models. They base the discovery of constraints on the replay of the log on specific automata, each accepting only those traces that are compliant with one constraint. In [8,14,15], the authors use inductive logic programming techniques for the supervised learning of con-

straints expressed using SCIFF [6]. The approach for the discovery of Declare constraints presented in [13] is divided in two steps. The first step computes statistic data describing the occurrences of activities and their interplay in the log. The second step checks the validity of Declare constraints by querying the statistic data structure built in the previous step. In [28], the authors present a mining approach that works with RelationalXES, a relational database architecture for storing event log data. The relational event data is queried with conventional SQL.

Recent research has put into evidence synergies between imperative and declarative approaches [25,26]. Accordingly, hybrid process modeling notations have been proposed. In particular, in [9], the authors provide a conservative extension of BPMN for declarative process modeling, namely BPMN-D, and show that Declare models can be transformed into readable BPMN-D models. In [36], the authors propose to extend Colored Petri nets with the possibility of linking transitions to Declare constraints directly. The notion of *transition enablement* is extended to handle declarative links between transitions. A recent implementation of this technique has been made available in CPN Tools 4.0 [35]. In [31], the authors extend the work in [36] by defining a semantics based on mapping Declare constraints to R/I-nets and by proposing modeling guidelines for the mixed paradigm. In [32], a discovery approach based on this paradigm is proposed. Differently from these approaches devoted to obtain a fully mixed language, in [30], a hybrid process model is hierarchical, where each of its sub-processes may be specified in either a procedural or declarative fashion. In [20], an approach for the discovery of hybrid process models based on this semantics is presented. In contrast to the approach of the current paper, which is model-driven, the approach in [20] is log-driven. In addition, our approach guarantees that the resulting hybrid model has 100% fitness.

3 Hybrid Models

As discussed in the introduction, we use the hierarchical approach to hybrid models. In our previous work [30], we have formalized the execution semantics of any combination of different formalisms used in a hierarchical hybrid model, e.g., Petri nets, BPMN, Declare, DCR Graphs, and process trees. In the mining approach presented here, we limit ourselves to hybrid models consisting of process trees and Declare models, and, in particular, to hybrid models where the top-level model is a process tree and some sub-processes can be Declare models.

In Fig. 1, we show an example of a hybrid model. The top-level model is a process tree, and there are two declarative sub-processes. The top-level process tree has the regular activities a, b, c, d, e and o and two abstract activities $X1$ and $X2$, each mapping to a declarative sub-process. The declarative sub-process $X1$ contains activity s and the sub-process $X2$ contains activities p q, r and t.

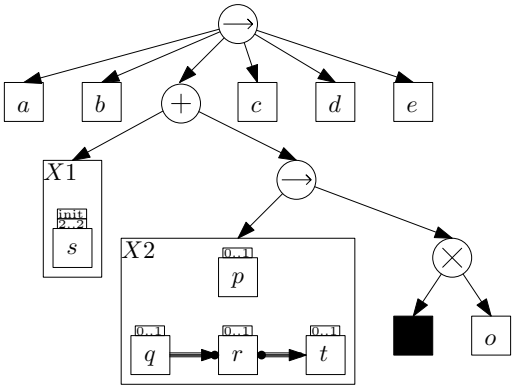


Fig. 1: Example of a Hybrid Model

3.1 Process trees

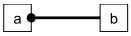
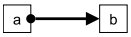
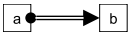
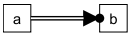
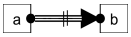
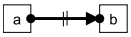
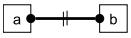
Process trees are a tree-structured modeling formalism. The branches of the trees represent control-flow constructs and the leaves of the tree represent atomic activities (e.g., a), including silent activities (τ). Various dialects exist for process trees (see, e.g., the ones used in [16,29]). We adopt the one from [16], which has four types of control-flow nodes: *(i)* the sequential composition ($\rightarrow$) denotes that the child nodes should occur in sequence, *(ii)* the exclusive choice ($\times$) denotes that (exactly) one of the child nodes should occur, *(iii)* the parallel composition ($+$) denotes that the child nodes should occur in parallel, and *(iv)* the redo loop ($\circlearrowleft$), which contains a *do* part (the first child) and a *redo* part (all subsequent children), and denotes a kind of looping behavior where the *do* is the main body of the loop and one can repeat the loop after doing one of the *redo* children.

We can define the semantics of process trees in terms of the formal languages (sets of sequences) that they represent. We reuse the definition from [2], but extend it to handle sub-process nodes. In the definition, we use two special operators on languages: *(i)* the concatenation operator ($\cdot$) concatenates the languages, e.g., $\{\langle a\rangle\} \cdot \{\langle b\rangle\} \cdot \{\langle c\rangle, \langle d\rangle\} = \{\langle a, b, c\rangle, \langle a, b, d\rangle\}$, and *(ii)* the shuffle operator ($\diamond$) returns all possible interleavings of the languages, e.g., $\{\langle a\rangle\} \diamond \{\langle b, c\rangle, \langle d, e\rangle\} = \{\langle a, b, c\rangle, \langle b, a, c\rangle, \langle b, c, a\rangle, \langle a, d, e\rangle, \langle d, a, e\rangle, \langle d, e, a\rangle\}$.

Definition 1 (Process Tree Semantics). *For a process tree Q with alphabet A, the language $\mathcal{L}(Q)$ of the process tree is defined recursively over its nodes as:*

1. $\mathcal{L}(\tau) = \{\langle\rangle\}$;
2. $\mathcal{L}(a) = \{\langle a\rangle\}$, *if* $a \in A$;
3. $\mathcal{L}(\overline{X}) = \mathcal{L}(\mathcal{M}_X)$, *where $\mathcal{M}_X$ is the model of the sub-process X;*
4. $\mathcal{L}(\rightarrow (Q_1, Q_2, \ldots, Q_n)) = \mathcal{L}(Q_1) \cdot \mathcal{L}(Q_2) \cdot \ldots \cdot \mathcal{L}(Q_n)$;
5. $\mathcal{L}(\times(Q_1, Q_2, \ldots, Q_n)) = \mathcal{L}(Q_1) \cup \mathcal{L}(Q_2) \cup \ldots \cup \mathcal{L}(Q_n)$;
6. $\mathcal{L}(+(Q_1, Q_2, \ldots, Q_n)) = \mathcal{L}(Q_1) \diamond \mathcal{L}(Q_2) \diamond \ldots \diamond \mathcal{L}(Q_n)$;
7. $\mathcal{L}(\circlearrowleft (Q_1, Q_2, \ldots, Q_n)) = \{\sigma_1 \cdot \sigma'_1 \cdot \sigma_2 \cdot \sigma'_2 \ldots \sigma_m \in A^* \mid m \geq 1 \wedge \forall_{1 \leq j \leq m} \sigma_j \in \mathcal{L}(Q_1) \wedge \forall_{1 \leq j < m} \sigma'_j \in \cup_{2 \leq i \leq n} \mathcal{L}(Q_i)\}$.

Table 1: Semantics of Declare templates

Template	Semantics	Notation
existence(a)	a must occur at least once	
absence2(a)	a can occur at most once	
init(a)	a must occur in the first position	
responded existence(a, b)	If a occurs, b must occur as well	
response(a, b)	If a occurs, b must eventually follow	
alternate response(a, b)	If a occurs, b must eventually follow, without any other a in between	
chain response(a, b)	If a occurs, b must occur next	
precedence(a, b)	b can occur only if a has occurred before	
alternate precedence(a, b)	b can occur only if a has occurred before, without any other b in between	
chain precedence(a, b)	b can occur only immediately after a	
not chain succession(a, b)	If a occurs, b cannot occur next	
not succession(a, b)	If a occurs, b cannot eventually follow	
not co-existence(a, b)	If a occurs, b cannot occur	

Given this semantics, the language of the process tree in Fig. 1 is: $\{\langle a, b, X1, X2, o, c, d, e\rangle, \langle a, b, X2, X1, o, c, d, e\rangle, \langle a, b, X2, o, X1, c, d, e\rangle, \langle a, b, X1, X2, c, d, e\rangle, \langle a, b, X2, X1, c, d, e\rangle\}$.

3.2 Declare

Declare [24] is a declarative process modeling language that encodes a set of declarative templates into a formal logic. Encodings into several different logics have been proposed, such as a linear temporal logic (LTL) and regular expressions. When using Declare in the context of process mining, it is most sensible to use one of their finite variants, as event logs contain only finite traces. Therefore, we employ the encoding into the finite variant of LTL (LTL_f) for the current paper. Table 1 shows an overview of the core relations used in Declare models. For the *existence* and *absence* templates, the language supports all countable variations, e.g., 0..2, 0..3, 4..$*$, etc.

Definition 2 (Declare Model). *A Declare model is a pair $M = (A, C)$, where A is a set of activities and C is a set of constraints over the activities in A.*

The language of a Declare model $M = (A, C)$ is the set of all traces that satisfy every constraint in C. Formally:

Definition 3 (Declare Semantics). *The language $\mathcal{L}(M)$ of a Declare model $M = (A, C)$ is defined as: $\mathcal{L}(M) = \{\sigma \in A^* \mid \forall_{c \in C}\, \sigma \models c\}$.*

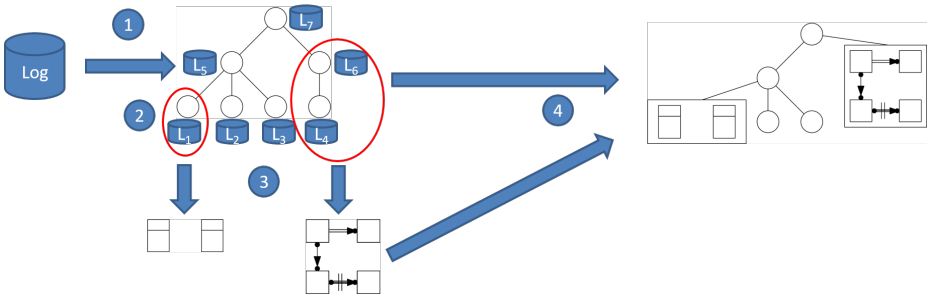


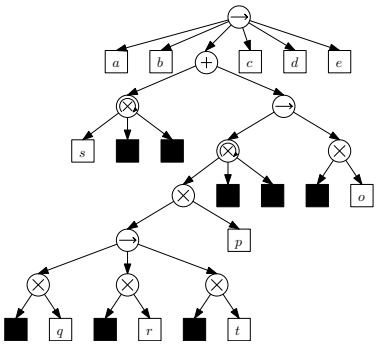
Fig. 2: Overview of the Approach

Sub-processes $X1$ and $X2$ in Fig. 1 are examples of Declare models. $X1$ is very simple and contains three constraints: $absence3(s)$, $existence2(s)$ and $init(s)$, indicated by the two boxes on activity s, one containing the text 2..2, and the other the text *init*. These constraints together indicate that s is expected to occur exactly 2 times, and is always the first activity in that subprocess to be executed. Therefore, the language of the Declare model $X1$ is $\mathcal{L}(X1) = \{\langle s, s\rangle\}$. Model $X2$ is a bit more complex. It contains an $absence2(x)$ constraint for each activity $x \in \{p, q, r, t\}$, but no existence constraints, meaning that all activities can occur either once or not at all. In addition, there is a *chain precedence*(q, r) constraint indicating that r can only occur immediately after q and a *chain response*(r, t) constraint indicating that after r, t must happen immediately. Note that "immediately" is interpreted in the context of the Declare model, so while neither p or q may happen between them, other activities in the hybrid model that can happen in parallel with $X2$ may still occur between them. This means that the language of the Declare model $X2$ is: $\mathcal{L}(X2) = \{\langle\rangle, \langle t\rangle, \langle q\rangle, \langle q, t\rangle, \langle t, q\rangle, \langle q, r, t\rangle, \langle p\rangle, \langle p, t\rangle, \langle t, p\rangle, \langle q, p\rangle, \langle p, q\rangle, \langle p, q, t\rangle, \langle p, t, q\rangle, \langle q, p, t\rangle, \langle q, t, p\rangle, \langle t, p, q\rangle, \langle t, q, p\rangle, \langle p, q, r, t\rangle, \langle q, r, t, p\rangle\}$.

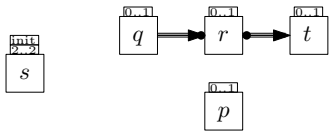
Now that we have introduced its component notations, we can give the language of the entire hybrid model in Fig. 1. Because of the many possible interleavings, giving the full set of all possible traces would be fairly cumbersome, so instead we provide it by using the operators used in Definition 1: $\{\langle a, b\rangle\} \cdot (\mathcal{L}(X1) \diamond (\mathcal{L}(X2) \cdot \{\langle\rangle, \langle o\rangle\})) \cdot \{\langle c, d, e\rangle\}$.

4 Hybrid Model Discovery

Our approach is inspired by two observations: (1) imperative miners in general, and the Inductive miner in particular, have difficulties in dealing with unstructured behavior without sacrificing precision (while maintaining a high fitness value), and (2) declarative models are aimed at describing unstructured behavior in a concise way. This led us to an approach where we first mine an imperative model (a process tree). Afterwards, we try to substitute various sub-processes with declarative models to improve precision without sacrificing fitness. Fig. 2 gives a broad overview of the approach, consisting of 4 steps. To exemplify our approach, we use the example log L_1.



(a) Process tree for log L_1



(b) Declare models discovered from the sub-logs corresponding to the loop sub-processes

Fig. 3: Intermediate outcomes of the proposed approach

Definition 4 (L_1: Running example). *Recall that the multiset* $[a^4, b^6]$ *is a set containing four times element* a *and six times element* b. $L_1 = [\langle a, b, s, s, c, d, e\rangle^5, \langle a, b, o, s, s, c, d, e\rangle^5, \langle a, b, s, o, s, c, d, e\rangle^5, \langle a, b, q, s, r, s, t, c, d, e\rangle^5, \langle a, b, s, s, q, p, c, d, e\rangle^5, \langle a, b, s, q, s, r, t, c, d, e\rangle^5, \langle a, b, s, t, p, s, c, d, e\rangle^5, \langle a, b, s, s, t, q, c, d, e\rangle^5, \langle a, b, p, t, s, q, s, c, d, e\rangle^5, \langle a, b, s, s, p, t, c, d, e\rangle^5, \langle a, b, p, s, q, s, r, t, c, d, e\rangle^5, \langle a, b, s, s, q, t, o, c, d, e\rangle^5]$.

4.1 Step 1: Mine process tree

For the first step of our approach, we use the Inductive miner [16] to generate a process tree from the input log. The Inductive miner builds a directly-follows-graph for the unique activities of the log and uses it to find structures and construct a process tree. If the Inductive miner cannot find any structure, it will return a flower model where any behavior is allowed. This makes the Inductive miner ideal for our approach: not only is it one of the most popular mining algorithms, but also it avoids finding structures when none exists and instead simply returns a pocket-of-flexibility, which can then be mined declaratively. We use the standard Inductive miner with a noise threshold of 0. In this way, the generated model is guaranteed to have 100% fitness and we can make a fair comparison between a pure imperative and a hybrid approach, where we only consider improvements on the precision of the model. For our example log, the Inductive miner creates the process tree shown in Fig. 3a.

4.2 Step 2: Select Unstructured Nodes

Once we have mined the process tree, we determine which parts should be replaced with a declarative sub-process. For every permutation of nodes, we replace those nodes with Declare sub-processes, determine the precision of the model, and return the most precise permutation. If a node is selected to be replaced with a declarative sub-process and also a higher node in the tree is selected,

then we only substitute the higher node in the tree, i.e., the node and all its descendants are replaced.

Note that, when we apply our approach to the example log and process tree from Fig. 3a by replacing the loop sub-processes with declarative sub-processes, this results in an increase of precision. This is because the loops allow for any number of occurrences of s and p, q, r, t, while, in reality, the allowed number of occurrences is bound. This bound can be found by the Declare miner.

4.3 Step 3: Mine Declarative Models

For each permutation of nodes to be replaced, we extract the sub-logs corresponding to these nodes. Afterwards, we run a declarative miner on each sub-log to generate a declarative model corresponding to the selected node and its descendants. In our particular case, we use the Declare miner [17] for this task, but we have also made a wrapper for MINERful [13]. The sub-log corresponding to the first loop sub-process is $[\langle s, s\rangle^{60}]$. Passing this log to the Declare miner generates the Declare model in Fig. 3b on the left. The sub-log corresponding to the second loop sub-process is $[\langle\rangle^5, \langle o\rangle^{10}, \langle q, r, t\rangle^{10}, \langle q, p\rangle^5, \langle t, p, \rangle^5, \langle t, q\rangle^5, \langle p, t, q\rangle^5, \langle p, t\rangle^5, \langle p, q, r, t\rangle^5, \langle q, t, o\rangle^5]$. Passing this log to the Declare miner generates the Declare model in Fig. 3b on the right.

4.4 Step 4: Construct Hybrid Model

Once we have a Declare model corresponding to each selected node, we replace these nodes and their descendants with an abstract activity representing a sub-process. The sub-process is then defined by the declarative model mined from the sub-log for that node. Applying this technique to the process tree from Fig. 3a and the Declare models from Fig. 3b, we can derive the hybrid model in Fig. 1.

5 Evaluation

Our approach has been implemented in the HYBRID package in ProM. The main plug-in for mining is called "Mine Hybrid Model". Fig. 4 shows the visualization of a hybrid model in the plug-in.

We evaluated our approach on both synthetic and real-life logs. We use the synthetic examples to show that the algorithm is able to detect and construct hybrid models when these are clearly present in the log. We use the real-life logs to demonstrate how well the approach works on actual logs coming from industrial settings, and to discover if any of these have a clear hybrid structure. We have evaluated the output of the Hybrid miner (HM) in terms of precision. This in order to show that a hybrid model indeed can help in making the unstructured parts of the process tree resulting from the Inductive miner (IM) more precise. To show this increase in precision, we introduce the *relative increase in precision* (RIiP) as the percentage of increase in precision.

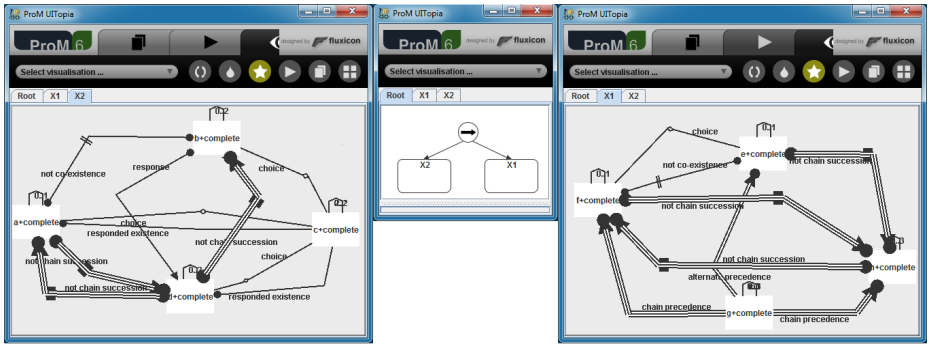


Fig. 4: Hybrid model for log L_3

5.1 Precision

We use a metric based on *escaping edges* [3] for the computation of precision. Our main consideration for choosing this metric is that the precision is defined based on a transition system, a formalism that both a process tree and a Declare model can be transformed into. For every visited state in the transition system, the outgoing transitions used by the log are compared to the total number of possible outgoing transitions. Transitions that are never visited by the log, the *escaping edges*, are counted against the precision of the model. The metric takes into account the frequencies of states, i.e., a frequently visited state has a higher impact on the precision than an infrequently visited state, and states that are never visited by the log are not taken into consideration.

5.2 Synthetic Logs

In order to show the possible gains of using a hybrid approach over solely using the Inductive miner, we have created four different synthetic logs, each serving as a representative for an interesting class of hybrid behavior.

L_1*: Running example.* The first log we examined was that of the running example, which is an example of a log where part of the behavior is very structured (events a, b, c, d and e), and part is unstructured (events o, p, q, r, s and t) that always occurs in a specific location (between events b and c). The most precise model for L_1 is more complex than the simplified Hybrid model used for illustrative purposes in Fig. 1. By mining the log with the Hybrid miner, we achieve a precision improvement of +0.194 with respect to the Inductive miner. The relative improvement (how much the precision of the hybrid model improved when compared to the precision of the imperative model) RIiP is of 29.59%. The resulting model is clearly hybrid, consisting of a process tree with 7 nodes and a declarative sub-process of 30 constraints.

L_2*: Running example unbalanced.* In the running example, the traces of the log are balanced (they occur equally frequently). In the second example, we

experimented with making the traces exhibiting unstructured behavior more frequently, with the aim of discovering if this increased the relative precision of the hybrid model. Definition 5 shows the log used.

Definition 5 (L_2: Running example unbalanced). $L_2 = [\langle a, b, s, s, c, d, e\rangle^1, \langle a, b, o, s, s, c, d, e\rangle^1, \langle a, b, s, o, s, c, d, e\rangle^1, \langle a, b, q, s, r, s, t, c, d, e\rangle^{100}, \langle a, b, s, s, q, p, c, d, e\rangle^1, \langle a, b, s, q, s, r, t, c, d, e\rangle^1, \langle a, b, s, t, p, s, c, d, e\rangle^1, \langle a, b, s, s, t, q, c, d, e\rangle^1, \langle a, b, p, t, s, q, s, c, d, e\rangle^{100}, \langle a, b, s, s, p, t, c, d, e\rangle^5, \langle a, b, p, s, q, s, r, t, c, d, e\rangle^{100}, \langle a, b, s, s, q, t, o, c, d, e\rangle^5]$.

For this log, the Hybrid miner provides an improvement from 0.543 to 0.827 with respect to the Inductive miner. Compared to L_1, the hybrid model accounts for the unstructured parts of the log being more pronounced by adding 2 additional constraints to the declarative sub-process. Note that, while there is an absolute improvement that is slightly better (+0.284) than the one obtained for L_1, the relative improvement RIiP is much higher (52.18%).

L_3: Abstract structure. For the third experiment, we used log L_3.

Definition 6 (L_3: Abstract structure). $L_3 = [\langle a, c, c, d, g, h, e, g, g\rangle^5, \langle c, b, c, d, g, g, f, g, h\rangle^5, \langle c, a, c, d, d, g, h, g, h, g, h, e\rangle^5, \langle d, c, c, a, g, h, g, f, g\rangle^5, \langle c, a, c, g, g, f, g, h\rangle^5, \langle c, c, d, g, h, g, e, g, h\rangle^5, \langle c, b, c, d, g, g, g, f\rangle^5, \langle d, c, b, c, d, g, g, g, e\rangle^5, \langle b, c, c, b, d, g, f, g, g\rangle^5, \langle c, c, g, g, g, e\rangle^5]$.

The log exhibits an abstract structure, i.e., while all the activities of the log occur in an unstructured manner, they can be separated in two sets $\{a, b, c, d\}$ and $\{e, f, g, h\}$, where events of the second set always occur after the events of the first set. As expected the Hybrid miner finds a hybrid model consisting of a sequence flow and two declarative sub-processes, one containing 14 constraints, the other 15. The hybrid model results in a precision improvement from 0.459 to 0.616.

L_4: Unstructured sequences. For the fourth experiment, we used log L_4.

Definition 7 (L_4: Unstructured sequences). $L_4 = [\langle a, b, c, f, g\rangle^5, \langle f, g, a, b, c, d, e\rangle^5, \langle a, b, c, h, i, d, e\rangle^5, \langle a, b, c, f, g, d, e\rangle^5, \langle h, i, a, b, c, h, i, h, i\rangle^5, \langle h, i, a, b, c, d, e, a, b, c\rangle^5, \langle a, b, c, a, b, c, f, g, d, e\rangle^5, \langle f, g, a, b, c, a, b, c, d, e\rangle^5, \langle h, i, a, b, c\rangle^5, \langle h, i, h, i, a, b, c, d, e\rangle^5]$.

This log contains four strict sequences, $\langle a, b, c\rangle$, $\langle d, e\rangle$, $\langle f, g\rangle$ and $\langle h, i\rangle$, which themselves occur in an unstructured manner. This example can be seen as being in the inverse class as L_3, i.e., while all activities occur in a structured manner, there is a very clear unstructured abstract behavior in the log. Since the best way of representing this process would be by using a declarative model with imperative sub-processes, mining the log with the Hybrid miner (which is developed to build imperative models with declarative sub-processes) results in a fully declarative model. The declarative model provides a precision improvement from 0.743 to 0.897 with respect to the Inductive miner.

Table 2: Evaluation Results of the Hybrid miner

	Precision			Hybrid Miner		
	IM	HM	RIiP	#nodes	#sub-processes	#constraints
L_1	0.6552	0.8491	29.59%	7	1	30
L_2	0.5432	0.8266	52.18%	7	1	30
L_3	0.4595	0.6165	34.17%	3	2	14,15
L_4	0.7428	0.8966	20.70%	1	1	101
BPIC 2012	0.2401	0.6068	152.74%	1	1	1286
BPIC 2013	0.4044	0.4044	0.00%	44	0	0
BPIC 2017	0.0601	0.0601	0.00%	72	1	1
WABO2	0.0049	0.1673	3347%	1	1	78 100
WABO3	0.0059	0.2101	3471%	1	1	75 893
Sepsis cases	0.1905	0.3762	97.47%	1	1	211
RTFM	0.8181	0.9625	17.65%	1	1	73

5.3 Real-Life Logs

We also experimented on a number of real-life logs, in particular the BPI Challenge (BPIC) 2012, 2013 (incidents) and 2017, WABO 2 and 3, Sepsis cases and Road Traffic Fine Management (RTFM). Each of these logs was retrieved from the 4TU repository.[1] As mentioned, in order to find the most precise model, the Hybrid miner tries all permutations of nodes resulting in models ranging from fully imperative via hybridity to fully declarative. In most cases, a fully declarative model was returned as being the most precise. The only exception are the BPIC 2013 and 2017 logs. The latter returns a hybrid model that contains only one Declare constraint that gives no noticeable precision improvement. For the WABO logs, the models found by the Hybrid miner show a precision improvement of over 30 times even if these models are extremely large, consisting of over 75 000 constraints. The BPIC 2012 and Sepsis cases logs also show a significant increase in precision and return more reasonably sized declarative models.

5.4 Discussion

Table 2 shows the detailed results of all the experiments we ran. Overall the Hybrid miner performed quite well on both synthetic and real-life examples. However, in the case of the real-life logs this was mostly due to the fact that the Hybrid miner was able to determine that a fully declarative model would have the highest precision and provided such a model. Granted, this increase in precision came at the cost of simplicity since more than 75 000 constraints cannot be comprehended by an end-user. This is also one of the limitations of our current approach: getting the right settings on the used miners to give a precise *and* comprehensible result.

In future work, we would like to investigate why the Hybrid miner did not generate any particularly interesting hybrid models from real-life logs. One particularly promising hypothesis is that these logs fit into the category represented

[1] `https://data.4tu.nl/repository/`

by L_4, which would be better represented by a hybrid model where the top-level is declarative, and sub-processes are imperative. Another explanation could be that the structure of the process tree returned by the Inductive miner hinders further improvements. After all, we take the structure of the process tree as-is without considering other structures that are language equivalent to the returned process tree.

In addition, it should be noted that, while we are using one of the most widely accepted precision metric, there is still an ongoing search for better metrics [33]. For example, for a log containing at most 2 occurrences of an activity, the current metric gives only a small benefit to an *absence*3 constraint over a process tree loop, even though, from a language inclusion perspective, the *absence*3 constraint is infinitely more precise. Improvements to the way precision is calculated for hybrid models may also lead to different results.

6 Conclusion

We presented a novel technique for mining hybrid models, which combine the strengths of the imperative and declarative process modeling paradigms. We implemented our technique as a ProM plug-in and evaluated the approach on several synthetic and real-life logs. Compared to the Inductive miner, the miner showed significant improvements in precision for both synthetic and real-life logs. In the case of real-life logs, it mostly found that purely declarative models were the most precise, whereas in the case of synthetic logs, proper hybrid models were found. Precision improvements ranged up to 52.18% for synthetic logs and up to 3471% for real-life logs.

The presented approach is easily extendible and customizable. For example, we can already plug any other miner that provides a process tree as output into our implementation and it is also possible to extend the general approach to other types of imperative models and miners. It is also possible to apply different metrics for precision and to use various declarative miners.

As future work, we would like to implement and experiment with different variants of the approach (e.g., different miners and ways to partition behavior in logs into declarative and imperative parts). Moreover, we also plan to use an approach opposite to the one proposed here: we could first attempt to mine a log declaratively, analyze the resulting model to detect parts which are overly constrained, and replace these with imperative sub-processes.

References

1. van der Aalst, W.M.P., Adams, M., ter Hofstede, A.H.M., Pesic, M., Schonenberg, H.: Flexibility as a Service. In: Database Systems for Advanced Appl. (2009)
2. van der Aalst, W.M.P.: Process Mining - Data Science in Action, Second Edition. Springer (2016)
3. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying history on process models for conformance checking and performance analysis. Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery 2(2), 182–192 (2012)

4. van der Aalst, W.M.P., Rubin, V.A., Verbeek, H.M.W., van Dongen, B.F., Kindler, E., Günther, C.W.: Process mining: a two-step approach to balance between underfitting and overfitting. Software and System Modeling 9(1), 87–111 (2010)
5. 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), 1128–1142 (2004)
6. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Verifiable agent interaction in abductive logic programming: The SCIFF framework. ACM Trans. Comput. Log. 9(4), 29:1–29:43 (2008)
7. Back, C.O., Debois, S., Slaats, T.: Towards an entropy-based analysis of log variability. In: BPM Workshops. pp. 53–70 (2018)
8. Chesani, F., Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Exploiting inductive logic programming techniques for declarative process mining. T. Petri Nets and Other Models of Concurrency 2, 278–295 (2009)
9. De Giacomo, G., Dumas, M., Maggi, F.M., Montali, M.: Declarative process modeling in BPMN. In: CAiSE. pp. 84–100 (2015)
10. Debois, S., Hildebrandt, T.T., Slaats, T.: Hierarchical declarative modelling with refinement and sub-processes. In: BPM. pp. 18–33 (2014)
11. Debois, S., Hildebrandt, T.T., Marquard, M., Slaats, T.: Hybrid process technologies in the financial sector. In: BPM (Industry track). pp. 107–119 (2015)
12. Debois, S., Hildebrandt, T.T., Slaats, T., Marquard, M.: A case for declarative process modelling: Agile development of a grant application system. In: EDOC Workshops. vol. 14, pp. 126–133 (2014)
13. Di Ciccio, C., Mecella, M.: A two-step fast algorithm for the automated discovery of declarative workflows. In: CIDM. pp. 135–142. IEEE (2013)
14. Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Inducing declarative logic-based models from labeled traces. In: BPM. pp. 344–359 (2007)
15. Lamma, E., Mello, P., Riguzzi, F., Storari, S.: Applying inductive logic programming to process mining. In: Inductive Logic Programming. vol. 4894, pp. 132–146 (2007)
16. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - A constructive approach. In: PETRI NETS. pp. 311–329 (2013)
17. Maggi, F.M.: Declarative process mining with the Declare component of ProM. In: BPM (Demos). CEUR Workshop Proceedings, vol. 1021. CEUR-WS.org (2013)
18. Maggi, F.M., Bose, R.P.J.C., van der Aalst, W.M.P.: Efficient discovery of understandable declarative process models from event logs. In: CAiSE. pp. 270–285 (2012)
19. Maggi, F.M., Mooij, A.J., van der Aalst, W.M.P.: User-guided discovery of declarative process models. In: CIDM. pp. 192–199. IEEE (2011)
20. Maggi, F.M., Slaats, T., Reijers, H.A.: The automated discovery of hybrid processes. In: BPM. pp. 392–399 (2014)
21. Marquard, M., Shahzad, M., Slaats, T.: Web-based modelling and collaborative simulation of declarative processes. In: BPM. pp. 209–225 (2015)
22. de Medeiros, A.K.A., Weijters, A.J.M.M., van der Aalst, W.M.P.: Genetic process mining: an experimental evaluation. Data Min. Knowl. Disc. 14(2), 245–304 (2007)
23. Montali, M.: Specification and Verification of Declarative Open Interaction Models - A Logic-Based Approach, vol. 56. Springer (2010)
24. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: Full Support for Loosely-Structured Processes. In: EDOC. pp. 287–300 (2007)

25. Pichler, P., Weber, B., Zugal, S., Pinggera, J., Mendling, J., Reijers, H.A.: Imperative versus declarative process modeling languages: An empirical investigation. In: BPM Workshops. pp. 383–394 (2011)
26. Reijers, H.A., Slaats, T., Stahl, C.: Declarative modeling – An academic dream or the future for BPM? In: BPM. pp. 307–322 (2013)
27. Sadiq, S., Sadiq, W., Orlowska, M.: Pockets of flexibility in workflow specification. In: ER, pp. 513–526 (2001)
28. Schönig, S., Rogge-Solti, A., Cabanillas, C., Jablonski, S., Mendling, J.: Efficient and customisable declarative process mining with SQL. In: CAiSE 2016. pp. 290–305 (2016)
29. Schunselaar, D.M.M.: Configurable Process Trees: Elicitation, Analysis, and Enactment. Ph.D. thesis, Eindhoven University of Technology (2016)
30. Slaats, T., Schunselaar, D.M.M., Maggi, F.M., Reijers, H.A.: The semantics of hybrid process models. In: OTM CoopIS. pp. 531–551 (2016)
31. Smedt, J.D., Weerdt, J.D., Vanthienen, J., Poels, G.: Mixed-paradigm process modeling with intertwined state spaces. Business & IS Eng. 58(1), 19–29 (2016)
32. Smedt, J.D., Weerdt, J.D., Vanthienen, J.: Fusion miner: Process discovery for mixed-paradigm models. Decision Support Systems 77, 123–136 (2015)
33. Tax, N., Lu, X., Sidorova, N., Fahland, D., van der Aalst, W.M.P.: The imprecisions of precision measures in process mining (2017), `https://arxiv.org/abs/1705.03303`
34. 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), 151–162 (2003)
35. Westergaard, M., Slaats, T.: CPN Tools 4: A process modeling tool combining declarative and imperative paradigms. In: BPM (Demos) (2013)
36. Westergaard, M., Slaats, T.: Mixing paradigms for more comprehensible models. In: BPM, pp. 283–290 (2013)