Model Checking of Mixed-Paradigm Process Models in a Discovery Context

Finding the Fit Between Declarative and Procedural

Johannes De Smedt[1], Claudio Di Ciccio[2], Jan Vanthienen[1], and Jan Mendling[2]

[1] Department of Decision Sciences and Information Management, Faculty of Economics and Business, KU Leuven, Leuven, Belgium
{johannes.desmedt;jan.vanthienen}@kuleuven.be
[2] Department of Information Systems and Operations, Vienna University of Economics and Business, Vienna, Austria
{claudio.di.ciccio;jan.mendling}@wu.ac.at

Abstract. The act of retrieving process models from event-based data logs can offer valuable information to business owners. Many approaches have been proposed for this purpose, mining for either a procedural or declarative outcome. A blended approach that combines both process model paradigms exists and offers a great way to deal with process environments which consist of different layers of flexibility. In this paper, it will be shown how to check such models for correctness, and how this checking can contribute to retrieving the models as well. The approach is based on intersecting both parts of the model and provides an effective way to check *(i)* whether the behavior is aligned, and *(ii)* where the model can be improved according to errors that arise along the respective paradigms. To this end, we extend the functionality of Fusion Miner, a mixed-paradigm process miner, in a way to inspect which amount of flexibility is right for the event log. The procedure is demonstrated with an implemented model checker and verified on real-life event logs.

Keywords: Declarative process models, Model checking, Process mining

1 Introduction

The field of process discovery [1] has witnessed the introduction of a vast amount of approaches, both of a procedural [2,3,4] and a declarative [5,6] nature. Both paradigms put a different emphasis on the retrieval of control flow constructs. On the one hand, the former fits activities in paths that are extended with control flow routing objects such as (X)OR- and AND-splits and -joins. Exemplary models include Petri nets and BPMN [7]. The latter, on the other hand, typically encompasses a constraint-based approach that captures behavior in the event log by fitting sequence rules over the activities. Most notably, the Declare language [8] has often been applied in this context [5,6]. Intermediate approaches exist in the form of Hybrid Miner [9] and Fusion Miner [10,11]. The latter mines process

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-319-58457-7_6

models with intertwined state spaces, whereas the former bases upon atomic subprocesses that use either paradigm, as in [9]. This type of process mining goes beyond the traditional techniques and fits different types of behavior in the log with the appropriate paradigm, i.e., the fixed sequences are captured with procedural process discovery techniques, while behavior that is hard to fit within such fixed sequence is mined with declarative techniques. Tackling a log in such a fashion yields more comprehensive though fitting and precise models, especially in the case of environments where multiple levels of flexibility exist. The validation aspect of the retrieved models, however, has still not been investigated yet. In this paper, the checking of mixed-paradigm models is elaborated by finding the common ground of both model types in the form of a global automaton. By using such a single executable model, the behavior of the discovery produce can be checked for inconsistent behavior, which can be pinpointed along the different paradigms.

The paper is organized as follows. Section 2 provides a motivation for the usefulness of the approach. Section 3 introduces the formalisms which are further used to explain the model checking and mining techniques in Section 4. Section 5 evaluates the approach on a real-life data log, which is followed by the conclusion and future work in Section 6.

2 Motivation

Mixed-paradigm models consist of a blend of procedural and declarative process models. More precisely, this entails models which on the one hand contain fixed execution paths, while on the other hand incorporate activity-based rules. Constructing such models is not always straightforward, as one has to be able to grasp the intricacies of both parts, as well as the effect they have on one another. Especially for process discovery, a consistency problem regarding the internal behavior can occur. Because both models are mined separately, though over the same alphabet, many conflicts can occur. For instance, the procedural model might allow for an activity to be enabled, while the declarative model does not, or vice versa. In this case, the activity has to conform to the most restricting model and become disabled. However, this might cause deadlocks for the other model later on, where the activity is not enabled or did not enable another activity that is needed to reach the final state(s) of the model. Consider the model in Figure 1. A procedural Petri net model is combined with multiple Declare constraints. In order to reach a marking containing *p5*, *precedence(g,c)* and *not no-existence(f,g)* cannot reside in the model together, as b requires f to fire first, which means that c can never be executed as g cannot fire anymore and c is in a *precedence* relation with g.

The challenge is to find whether the behavior of both models is compatible and can be used as a whole, and if not, where the discrepancies reside. This might lead to insights into how the model types interact, e.g., the procedural model might be too restrictive to allow for the more flexible behavior of the declarative model. By pinpointing which constraints are not working out with the procedural

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-319-58457-7_6

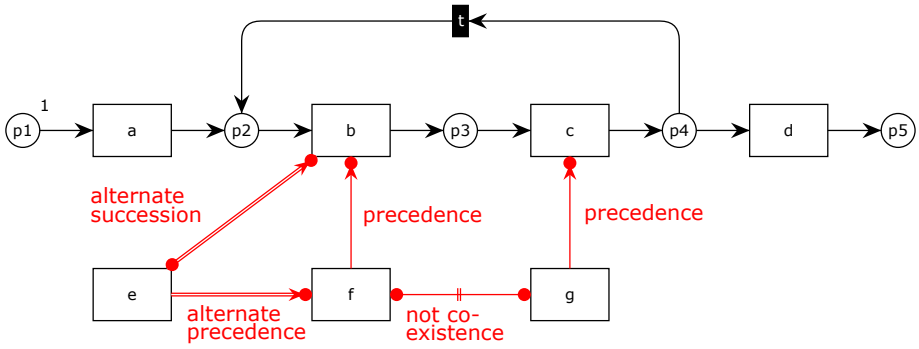

Fig. 1: An example of a mixed-paradigm model containing inconsistencies.

model, the modeler or miner might react accordingly. This is the principle that is used by the model checking approach proposed in this paper. By incrementally matching both models, the maximal conjoined behavior is sought after, ideally yielding a full match of behavior. Furthermore, Fusion Miner is adopted to also recognize procedural models with a vast state space. This reflects the presence of either a vast model, or of a model with many routing constructs for achieving a wide array of execution paths. The latter indicates that the model is either overfitting, or tries to capture a vast deal of the flexibility still, which conflicts with the aim of the approach to capture flexibility with the declarative process model. Hence, the algorithm adapts itself automatically to revise its outcome to shift the balance of the model towards the declarative part.

3 Preliminaries

In this section, we outline the notions upon which our approach is devised. We explain the concept of a mixed-paradigm process, along with the description of the modeling languages that are utilized respectively for the declarative and the procedural part, i.e., Declare and Petri nets. Previous work of which the implementation in CPN Tools is a well-known example [12], proposes building the different execution automata and joining them on-the-fly for simulation purposes. The aim of this work, however, is not to ensure execution ex-ante, but rather to check whether a discovered mixed-paradigm process model is executable ex-post.

3.1 Mixed-Paradigm Models

Mixed-paradigm models consist of a combination of both procedural and declarative model constructs. In this paper we use Petri nets [7] and Declare [13] to represent either paradigm, because they are commonly used languages for such endeavors [14,15].

More formally, we define the activities of the model as a finite set A, for which any type of connection $F \subseteq A \times A$ can exist. There exist four types of activities:

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-319-58457-7_6

$D \subseteq A$**:** the activities appearing in the declarative model,
$S \subseteq A$**:** the activities appearing in the procedural model,
$DD \subseteq D \setminus S$**:** the activities only appearing in the declarative model, and
$SS \subseteq S \setminus D$**:** the activities only appearing in the procedural model.

Hence, there are four corresponding connection types:

$F_D \subseteq D \times D$**:** the connections in the declarative model,
$F_{DS} \subseteq D \times S$**:** the connections from the declarative to the procedural model,
$F_{SD} \subseteq S \times D$**:** the connections from the procedural to the declarative model, and
$F_S \subseteq S \times S$**:** the connections in the procedural model.

The declarative model, being a constraint-based Declare model, is defined as a tuple $DM = (D, F_D)$ with F_D the set of constraints defined over D. They can be of any kind as listed in Table 1. For simplicity, we only consider unary and binary constraints, but the proposed technique would be able to incorporate any type of constraints that are expressed in the same (finite) execution semantics. All constraints can be expressed as regular expressions, which yield finite state automata (FSA) [16,17]. The automaton of a single constraint $f \in F_D$ is denoted as $\alpha(f)$. The conjunction of two or more constraints corresponds to the product $\otimes$ of the related automata. Therefore, to get the full behavior of the model, the separate automata are conjoined to build the global automaton by means of the product operation: $\Phi = \prod_{f \in F_D} \alpha(f)$.

The procedural model is defined as a Petri net, namely a tuple $PN = (P, S^{PN}, F^{PN}, L)$ with P a set of finite places, S^{PN} the set of transitions, which coincide with the aforementioned procedural activities, and $F^{PN} \subseteq (P \times S^{PN}) \cup (S^{PN} \times P)$. We consider Petri nets without weighed arcs, reset and inhibitor arcs, coloring, or stochastic extension, but with an injective labeling function $L : S^{PN} \to S \cup \tau$ with τ silent transitions used for routing purposes. For every $s \in S^{PN}$, the preset of places is defined as $\bullet s = \{p \,|\, (p, s) \in F^{PN}\}$ and the postset of places as $s\bullet = \{p \,|\, (s, p) \in F^{PN}\}$. A marking is a function $M : P \to \mathbb{N}$ which assigns a number of tokens to a place. M_0 is the initial marking of the net. A transition $s \in S^{PN}$ is said to be enabled iff $\forall p \in \bullet s$, $M(p) > 0$. When a transition s fires, all output places in $s\bullet$ receive an extra token, and from all input places in $\bullet s$ a token is subtracted. The new marking M' is thus such that: $\forall p \in \bullet s \quad M'(p) = M(p) - 1$, $\forall p \in s \bullet \quad M'(p) = M(p) + 1$, and $\forall p \in P \setminus \{s \bullet \cup \bullet s\} \quad M'(p) = M(p)$. Every marking that is generated by a sequence of firings from M is said to be reachable from M. A net is bounded when the number of reachable markings from M_0 is finite.

For a bounded Petri net, a reachability graph can be calculated [7,20]. The reachability graph of a bounded Petri net is a transition system constructed as follows. The initial marking is the initial state. Every reachable marking from M_0 is a state. Transitions between pairs of states represent the transitions that lead from a marking to another by means of a firing. A state in which no transitions are enabled anymore is called a final state. In the following, we assume that the Petri nets used to represent procedural models are bounded.

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-319-58457-7_6

Template	Regular Expression [18,19]	Description
Existence(A,n)	`.*(A.*){n}`	Activity A happens at least n times.
Absence(A,n)	`[^A]*(A?[^A]*){n}`	Activity A happens at most n times.
Exactly(A,n)	`[^A]*(A[^A]*){n}`	Activity A happens exactly n times.
Init(A)	`(A.*)?`	Each instance has to start with activity A.
Last(A)	`.*A`	Each instance has to end with activity A.
Responded existence(A,B)	`[^A]*((A.*B.*)|(B.*A.*))?`	If A happens at least once then B has to happen or happened before A.
Co-existence(A,B)	`[^AB]*((A.*B.*)|(B.*A.*))?`	If A happens then B has to happen or happened after after A, and vice versa.
Response(A,B)	`[^A]*(A.*B)*[^A]*`	Whenever activity A happens, activity B has to happen eventually afterward.
Precedence(A,B)	`[^B]*(A.*B)*[^B]*`	Whenever activity B happens, activity A has to have happened before it.
Alternate response(A,B)	`[^A]*(A[^A]*B[^A]*)*`	After each activity A, at least one activity B is executed. A following activity A can be executed again only after the first occurrence of activity B.
Alternate precedence(A,B)	`[^B]*(A[^B]*B[^B]*)*`	Before each activity B, at least one activity A is executed. A following activity B can be executed again only after the first next occurrence of activity A.
Chain response(A,B)	`[^A]*(AB[^A]*)*`	Every time activity A happens, it must be directly followed by activity B (activity B can also follow other activities).
Chain precedence(A,B)	`[^B]*(AB[^B]*)*`	Every time activity B happens, it must be directly preceded by activity A (activity A can also precede other activities).
Not co-existence(A,B)	`[^AB]*((A[^B]*)|(B[^A]*))?`	Either activity A or B can happen, but not both.
Not succession(A,B)	`[^A]*(A[^B]*)*`	Activity A cannot be followed by activity B, and activity B cannot be preceded by activity A.
Not chain succession(A,B)	`[^A]*(A+[^AB][^A]*)*A*`	Activities A and B can never directly follow each other.
Choice(A,B)	`.*[AB].*`	Activity A or activity B has to happen at least once, possibly both.
Exclusive choice(A,B)	`([^B]*A[^B]*)|.*[AB].*([^A]*B[^A]*)`	Activity A or activity B has to happen at least once, but not both.

Table 1: An overview of Declare constraint templates with their corresponding LTL formulas, regular expressions, and verbose descriptions.

For a mixed-model, we assume to deal with closed-world models, i.e., only the activities in A can be used in the model. Nevertheless, A can be extended to also include activities that are neither constrained in the Declare model, nor included in S. The injective labeling function makes it possible to deal with duplicate activities and silent transitions, as there exists a unique element for each silent or duplicate activity in the Petri net.

4 Model Checking Approach

In the following section, we describe our approach to check the consistency of a mixed-paradigm model. Thereafter, we show that it can be used in the context of mining to reduce the required computational expensiveness.

4.1 Model Checking

The model checking approach is based on automaton multiplication, as inspired by [16]. In order to verify whether a mixed-paradigm model does not have any conflicting states, both models are brought to the same execution model, being a finite state automaton. For the Petri net, the reachability graph is calculated (Algorithm 1, line 2). In the presence of deadlocks, the reachability graph will not reach a state in which there is an end marking without remaining tokens. In this case, the conjoining would not work and the Petri net should be checked for errors

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-319-58457-7_6

on its own. Next, all the constraints in the declarative model are checked for compatibility with the reachability automaton by conjoining its executable form, also an FSA, with the procedural model (line 19-20). In case the full declarative model does not conflict, the result Φ_{MPM} will contain the full behavior of both models without the conflicting states. If not, the declarative model with the most non-conflicting constraints will be returned (line 7-11). Another possibility is to check the declarative model up front with the approach discussed in [16] and available in the MINERful framework[3], which also checks for constraint redundancy.

Iteratively conjoining constraints to get the biggest set of non-conflicting Declare constraints might be computationally inefficient in case of very big constraint sets. This can be resolved by introducing a priority scheme that checks unary and negative constraints (e.g. *exactly2* and *not succession*) last, because they often impose a high degree of interaction with other constraints [21]. Furthermore, this can also help to resolve the issue of choosing one constraint set over the other in the case of equal sizes. Note that the approach can also be tailored towards analyzing process mining results.

Algorithm 1 Model checking procedure for mixed-paradigm models.

Output: Φ_{MPM}

1: **procedure** *calculateModel*(DM, PN)
2: $\quad\Phi_{PN} \leftarrow calculateReachabilityGraph(PN)$
3: $\quad\Phi_{MPM} = \emptyset$, $b = 0$, $V \leftarrow \emptyset$
4: $\quad$**for** $f \in F_D$ **do**
5: $\quad\quad\Phi_T, V \leftarrow checkConstraintForConflicts(\Phi_{PN}, F_D, f, V)$
6: $\quad\quad$**if** $V = F_D$ **then**
7: $\quad\quad\quad\Phi_{MPM} \leftarrow \Phi_T$
8: $\quad\quad\quad$**break**
9: $\quad\quad$**end if**
10: $\quad\quad$**if** $|V| > b$ **then**
11: $\quad\quad\quad b \leftarrow |V|$
12: $\quad\quad\quad\Phi_{MPM} \leftarrow \Phi_T$
13: $\quad\quad$**end if**
14: $\quad$**end for**
15: $\quad$**return** Φ_{MPM}
16: **end procedure**

17: **procedure** *checkConstraintForConflicts*(Φ, F_D, f, V)
18: $\quad V \leftarrow f$
19: $\quad$**if** $\Phi \otimes \alpha(f) \neq \emptyset$ **then**
20: $\quad\quad\Phi_{PN} \leftarrow \Phi \otimes \alpha(f)$
21: $\quad\quad$**for** $g \in F_D \setminus V$ **do**
22: $\quad\quad\quad checkConstraintForConflicts(\Phi_{PN}, g, V)$
23: $\quad\quad$**end for**
24: $\quad$**end if**
25: $\quad$**return** Φ_{PN}, V
26: **end procedure**

Example Consider the model in Figure 1 again. The reachability graph of the procedural model is relatively small and is presented in Figure 2. The initial marking is $M_0(p_1) = 1$. The algorithm iteratively tests whether the constraints in the declarative part of the model can be intersected with this state space.

[3] `https://github.com/cdc08x/MINERful`

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-319-58457-7_6

Clearly, *not co-existence(f,g)* and *precedence(f,b)* cannot co-exist in this model, as c can never be executed when g is prohibited from firing through *not co-existence(f,g)*. Therefore, one of these constraints is disregarded by the model checker to provide a final execution automaton for the model. According to the priority strategy, this is the case for *not co-existence(f,g)*.

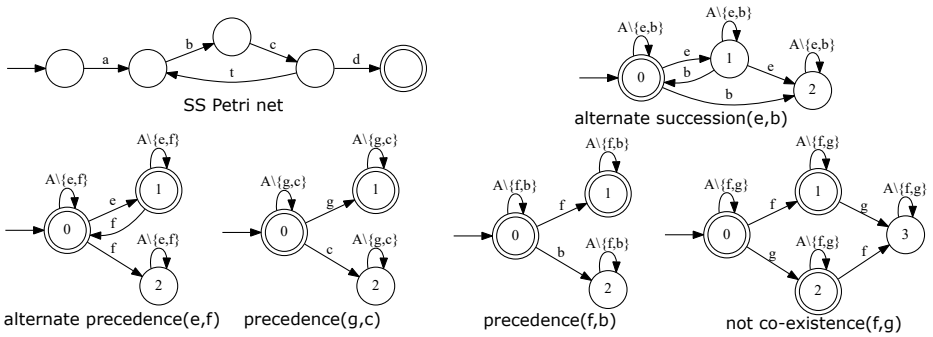

Fig. 2: The state space of the Petri net and the Declare constraints' automata from Figure 1.

4.2 Better Mining with Model Checking Iterations

Next to checking models, the model checking algorithm can also be used to guide the discovery algorithm towards a better solution in the following way. Since calculating the reachability graph can be computationally expensive, the algorithm can be adapted as follows. Starting from the initially assigned value, the entropy level is subsequently raised to increase the size of the declarative part versus the procedural one. In the extreme case, the algorithm resorts to mining solely a declarative model, for which the model checker is guaranteed to finish. Hence, the algorithm actually checks the amount of flexibility that is in the log and adapts itself accordingly to the amount of 'declarativeness' that is needed. We call this the *self-learning* capability of the approach.

In order to achieve this, the Fusion Miner algorithm (as documented in [11]) is adapted to check whether the reachability graph of the procedural model can be calculated, containing less than a certain number of states (Algorithm 2, line 8). This number is calculated based on a threshold n, a model size multiplication coefficient, and the size of the procedural model, i.e., $n \times |S^{PN}| \times 1,000$. If the graph cannot be calculated, the entropy measure is raised by another threshold called *resilience coefficient*, r, to reiterate the process towards a model consisting of a bigger declarative part and a smaller procedural one (line 20).

Example Consider the procedural model produced by Fusion Miner depticted in Figure 3. The procedural part of the model has to take into account many different ways of enabling d in between the other activities, introducing many

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-319-58457-7_6

Algorithm 2 FusionMINERFul algorithm.

Output: Φ_{MPM}

procedure $calculateModel(T, e, n, r)$ ▷ Input: set of traces T, entropy measure e and
  $\Phi_{MPM} \leftarrow \emptyset$ ▷ resilience measure r
  while $\Phi_{MPM} = \emptyset$ **do**
    $D \leftarrow getEntropicActivities(T, e)$
    $DM \leftarrow MINERful(T, D)$
    $PN \leftarrow HeuristicsMiner(T, A \setminus D)$
    $\Phi_{PN} \leftarrow calculateReachabilityGraph(PN, n)$ ▷ Returns an empty set after
    $V \leftarrow \emptyset$, $b \leftarrow 0$ ▷ $n \times |S| \times 1,000$ states
    if $\Phi_{PN} \neq \emptyset$ **then**
      for $f \in F_D$ **do**
        $\Phi_T \leftarrow checkConstraintForConflicts(\Phi_{PN}, F_D, f, V)$
        if $V = F_D$ **then**
          $\Phi_{MPM} \leftarrow \Phi_T$
          break
        end if
        if $|V| > b$ **then**
          $b \leftarrow |V|$
          $\Phi_{MPM} \leftarrow \Phi_T$
        end if
      end for
    else
      $e \leftarrow e + r$
    end if
  end while
  return Φ_{MPM}
end procedure

silent transitions. Calculating the reachability graph will yield an enormous automaton, requiring computationally expensive conjoining operations. Because the reachability graph calculation is stopped after 32,000 ($2 \times 16 \times 1000$ when $n = 2$) states, the algorithm repeats its main procedure with an entropy level e which is increased by r. For $e = 0.4$ and $r = 0.1$, the resulting model eliminates the need for invisible transitions by removing d from the procedural workflow, as depicted in Figure 4.

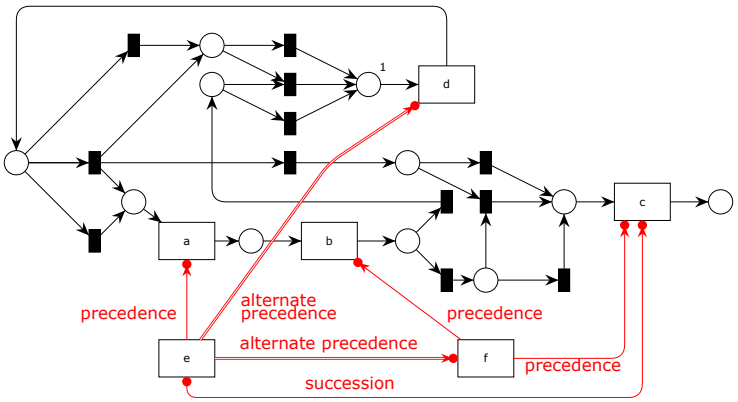

Fig. 3: Mixed-paradigm output of FusionMINERFul for an entropy level of 0.4.

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-319-58457-7_6

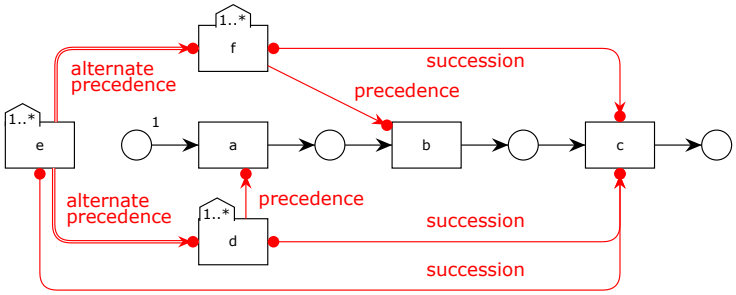

Fig. 4: Mixed-paradigm output of FusionMINERFul for the same event log after the entropy level was raised to 0.5.

5 Implementation and Evaluation

In this section, the implementation in FusionMINERFul is introduced. Next, this process mining tool is used to evaluate the approach on the 2012 BPI Challenge[4] event log.

5.1 Implementation

Mining a mixed-paradigm model with intertwined state spaces was introduced by Fusion Miner [11]. This mining algorithm uses the notion of entropy to find activity types in the log that do not fit a strict workflow well, based on the dependency information of Heuristics Miner [2]. Activities are divided into D, S, DD, and SS. F_D, F_{DS}, and F_S are mined, while F_{SD} is not considered to avoid too convoluted models that have a high risk of inconsistencies. For this work, a new version called FusionMINERFul is used. This algorithm uses MINERFul [6] to derive F_D and F_{DS} and Heuristics Miner to mine F_S.The technique also relies on the state space analysis tools which can be found in ProM[5]. The implementation is compatible with ProM and can be found at `http://www.processmining.be/fusionminerful/`, together with high resolution versions of the figures in this paper. The final output model is represented as a dependency graph with Declare constraints [11], which can be converted to a Petri net with Declare constraints. This serves as the basis for the model checking approach. In the output, removed constraints are colored differently, and the implementation also include the self-learning capability. Blue arcs comprise the procedural model, while black annotated arcs contain Declare constraints. Negative constraints are yellow, while constraints removed during verification are red. Declarative activities use dashed outlines, and gray and red coloring indicates *existence(A,1)* and *exactly(A,1)* respectively.

[4] DOI: `10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f`
[5] `http://www.promtools.org/`

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-319-58457-7_6

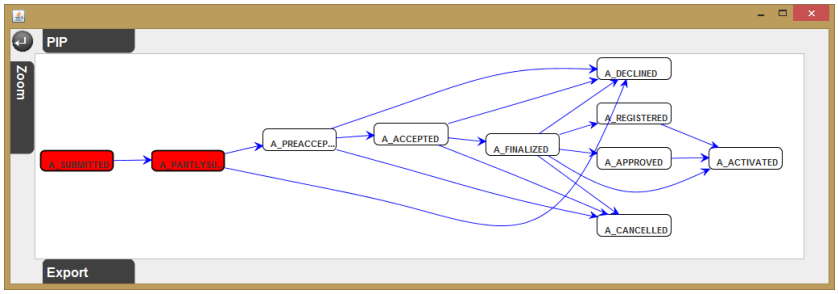

Fig. 5: Output of FusionMINERFul for the X subprocess of BPIC 2012. The set-up is: initial entropy level e of 0, resilience r of 0.1, and model size multiplicator n of 10. The entropy-level remains unchanged, indicating the log to contain procedural behavior.

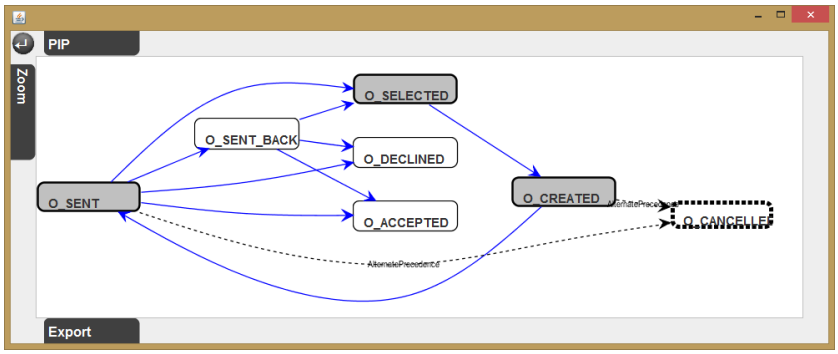

Fig. 6: Output of FusionMINERFul for the Y subprocess of the BPIC 2012. The set-up is: $e = 0$, eventually raised to 0.2, $r = 0.1$, and $n = 10$.

5.2 Application to BPIC 2012

The approach of model checking with FusionMINERFul has been tested on the 2012 BPI Challenge event log (BPIC 2012). This log consists of three distinct subprocesses which will be treated separately, in analogy with the approach followed in [22]. The goal is to find out whether the model that is discovered is sound, and how well FusionMINERFul can determine the level of flexibility that is needed for mining an informative process model. To test the self-learning capabilities of the algorithm, the initial entropy level is always kept at 0, giving the algorithm the chance to adapt itself according to whether a procedural finite state space can be constructed. The model size multiplicator level was set to 10, and the resilience measure at 0.1.

The first subprocess, X, does not contain any behavior that is too unstructured to handle in a procedural model, hence the algorithm does not raise the entropy level. In this case, the full model of Heuristics Miner is outputted, as can be seen in Figure 5. The second subprocess, subprocess Y, shows a low level of flexibility. In this case, two iterations finally churned out the process that can

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-319-58457-7_6

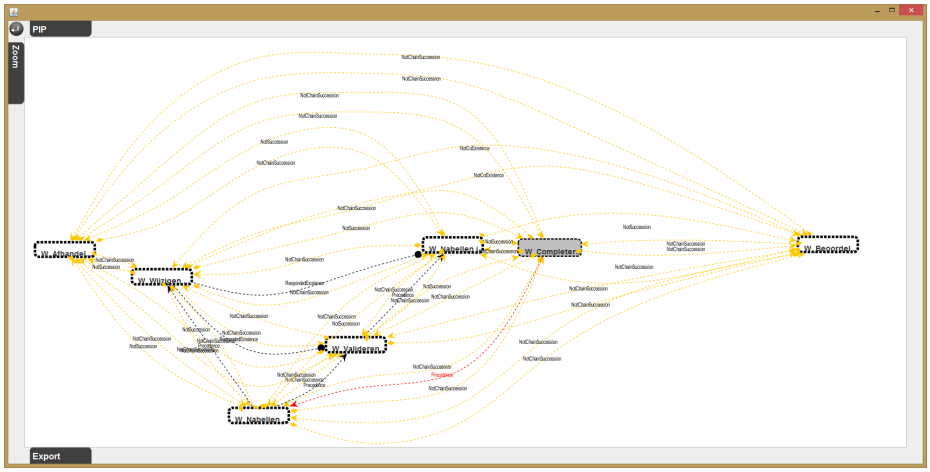

Fig. 7: Output of FusionMINERFul for the Z subprocess of BPIC 2012. The set-up is: $e = 0$, eventually raised to 1.0, $r = 0.1$, and $n = 10$.

be seen in Figure 6. Finally, the last subprocess, Z, reaches the maximal entropy level of 1, indicating that the process can be considered very unstructured. Only the Declare model is outputted, as can be seen in Figure 7, which contains many negative constraints and one conflicting constraint which is removed from the model.

Overall, the approach is capable of detecting inconsistencies, although few appeared. This is in line with the intuition established in [16]. Furthermore, FusionMINERFul is also capable of finding different levels of flexibility which approaches the results from Heuristics Miner and ILP Miner. These mining techniques also need to resort to imprecise many-to-many connections to reflect the unstructuredness that is present in the event logs, especially for Y and Z.

6 Conclusion

In this paper, a model checking approach for mixed-paradigm models was proposed based on automaton multiplication. Furthermore, it was shown how this notion can be used in a mining environment to find a fit between procedural and declarative models to achieve models that accurately describe the log with the right level of granularity in terms of flexibility.

For future work, we envision to use the global automaton to apply conformance checking in order to show in which areas mixed-paradigm models can excel not only in terms of dealing with mixed-flexibility, but in terms of recall and precision. Secondly, it will be investigated how coverability graphs can be used instead of reachability graphs, in order to deal with unbounded behavior that can occur in procedural models. Finally, Heuristics Miner cannot guarantee to churn out a deadlock-free model. It is therefore in our plans to integrate the tool with other procedural process mining algorithms.

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-319-58457-7_6

References

1. van der Aalst, W.M.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. Weijters, A., van der Aalst, W.M., De Medeiros, A.A.: Process mining with the heuristics miner-algorithm. TU Eindhoven, Tech. Rep. WP **166** (2006)
3. van der Aalst, W.M., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**(9) (2004) 1128–1142
4. van der Werf, J.M.E., van Dongen, B.F., Hurkens, C.A., Serebrenik, A.: Process discovery using integer linear programming. In: Petri Nets. Springer (2008) 368–387
5. Maggi, F.M., Mooij, A.J., van der Aalst, W.M.: User-guided discovery of declarative process models. In: CIDM, IEEE (2011) 192–199
6. Di Ciccio, C., Mecella, M.: A two-step fast algorithm for the automated discovery of declarative workflows. In: CIDM, IEEE (2013) 135–142
7. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE **77**(4) (1989) 541–580
8. Pesic, M., Schonenberg, H., van der Aalst, W.M.: Declare: Full support for loosely-structured processes. In: EDOC, IEEE (2007) 287–287
9. Maggi, F.M., Reijers, H.A., Slaats, T.: The automated discovery of hybrid processes. In: BPM. Springer (2014) 392–399
10. De Smedt, J., De Weerdt, J., Vanthienen, J.: Multi-paradigm process mining: retrieving better models by combining rules and sequences. In: OTM Conferences, Springer (2014) 446–453
11. De Smedt, J., De Weerdt, J., Vanthienen, J.: Fusion miner: Process discovery for mixed-paradigm models. Decision Support Systems **77** (2015) 123–136
12. Westergaard, M.: CPN Tools 4: multi-formalism and extensibility. In: Application and Theory of Petri Nets and Concurrency. Springer (2013) 400–409
13. Pesic, M., van der Aalst, W.M.: A declarative approach for flexible business processes management. In: BPM Workshops, Springer (2006) 169–180
14. De Smedt, J., De Weerdt, J., Vanthienen, J., Poels, G.: Mixed-paradigm process modeling with intertwined state spaces. Bus. & Inf. Systems Eng. (2016) 19–29
15. Westergaard, M., Slaats, T.: Mixing paradigms for more comprehensible models. In: BPM. Springer (2013) 283–290
16. Di Ciccio, C., Maggi, F.M., Montali, M., Mendling, J.: Ensuring model consistency in declarative process discovery. In: BPM. Springer (2015) 144–159
17. Prescher, J., Di Ciccio, C., Mendling, J.: From declarative processes to imperative models. In: SIMPDA. (2014) 162–173
18. Di Ciccio, C., Mecella, M.: On the discovery of declarative control flows for artful processes. ACM Trans. Manage. Inf. Syst. **5**(4) (January 2015) 24:1–24:37
19. Westergaard, M., Stahl, C., Reijers, H.A.: Unconstrainedminer: Efficient discovery of generalized declarative process models. Technical report, BPMcenter (2013)
20. Desel, J., Reisig, W.: Place/transition petri nets. In: Lectures on Petri Nets I: Basic Models. Springer (1998) 122–173
21. De Smedt, J., De Weerdt, J., Serral, E., Vanthienen, J.: Improving understandability of declarative process models by revealing hidden dependencies. In: International Conference on Advanced Information Systems Engineering, Springer (2016) 83–98
22. Adriansyah, A., Buijs, J.C.: Mining process performance from event logs. In: BPM Workshops, Springer (2012) 217–218

Pre-print copy of the manuscript published by Springer (available at link.springer.com)
identified by DOI: 10.1007/978-3-319-58457-7_6

This document is a pre-print copy of the manuscript
(De Smedt et al. 2017)
published by Springer (available at link.springer.com).

The final version of the paper is identified by DOI: 10.1007/978-3-319-58457-7_6

References

De Smedt, Johannes, Claudio Di Ciccio, Jan Vanthienen, and Jan Mendling (2017). "Model Checking of Mixed-Paradigm Process Models in a Discovery Context - Finding the Fit Between Declarative and Procedural". In: *BPM workshops*. Ed. by Marlon Dumas and Marcelo Fantinato. Vol. 281. Lecture Notes in Business Information Processing. Springer, pp. 74–86. ISBN: 978-3-319-58456-0. DOI: 10.1007/978-3-319-58457-7_6.

BibTeX

```
@InProceedings{   DeSmedt.etal/BPI2016:ModelCheckingofMixedParadigmDiscovery,
  author        = {De Smedt, Johannes and Di Ciccio, Claudio and Jan
                   Vanthienen and Jan Mendling},
  title         = {Model Checking of Mixed-Paradigm Process Models in a
                   Discovery Context - Finding the Fit Between Declarative and
                   Procedural},
  booktitle     = {BPM workshops},
  year          = {2017},
  pages         = {74--86},
  crossref      = {BPM2016Workshops},
  doi           = {10.1007/978-3-319-58457-7_6}
}
@Proceedings{     BPM2016Workshops,
  title         = {Business Process Management Workshops - {BPM} 2016
                   International Workshops, Rio de Janeiro, Brazil, September
                   19, 2016, Revised Papers},
  year          = {2017},
  editor        = {Dumas, Marlon and Fantinato, Marcelo},
  volume        = {281},
  series        = {Lecture Notes in Business Information Processing},
  publisher     = {Springer},
  isbn          = {978-3-319-58456-0},
  doi           = {10.1007/978-3-319-58457-7}
}
```