Discovery, Verification and Conformance of Workflows with Cancellation

W.M.P. van der Aalst

Department of Mathematics and Computer Science,
Eindhoven University of Technology
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
w.m.p.v.d.aalst@tue.nl

Abstract. Petri nets are frequently used for the modeling and analysis of workflows. Their graphical nature, well-defined semantics, and analysis techniques are attractive as information systems become more "process-aware". Unfortunately, the classical Petri net has problems modeling cancellation in a succinct and direct manner. Modeling cancellation regions in a classical net is impossible or results in a "spaghetti-like" net. Cancellation regions are supported by many workflow management systems, but these systems do not support advanced analysis techniques (process mining, verification, performance analysis, etc.). This paper proposes to use *reset workflow nets* (RWF-nets) and discusses (1) the *discovery* of RWF-nets (i.e., extracting information from event logs to construct such models), (2) the *verification* of RWF-nets (i.e., checking whether a workflow process has deadlocks, livelocks, etc.), and (3) the *conformance* of an event log with respect to a RWF-net (i.e., comparing real with modeled behavior).

Keywords: Petri Nets, Reset Nets, Soundness, Verification, and Process Mining.

1 Introduction

Information systems have become "process-aware", i.e., they are driven by process models [26]. Often the goal is to automatically configure systems based on process models rather than coding the control-flow logic using some conventional programming language. Early examples of process-aware information systems were called WorkFlow Management (WFM) systems [5, 30, 36, 50]. In more recent years, vendors prefer the term Business Process Management (BPM) systems. BPM systems have a wider scope than the classical WFM systems and are not just focusing on process automation. BPM systems tend to provide more support for various forms of analysis and management support. Both WFM and BPM aim to support operational processes that we refer to as "workflow processes" or simply "workflows".

The flow-oriented nature of workflow processes makes the Petri net formalism a natural candidate for the modeling and analysis of workflows. Figure 1 shows a so-called *workflow net* (WF-net), i.e., a Petri net with a start place and an

end place such that all nodes are on some path from start to end. WF-nets were introduced in [1, 2].[1]

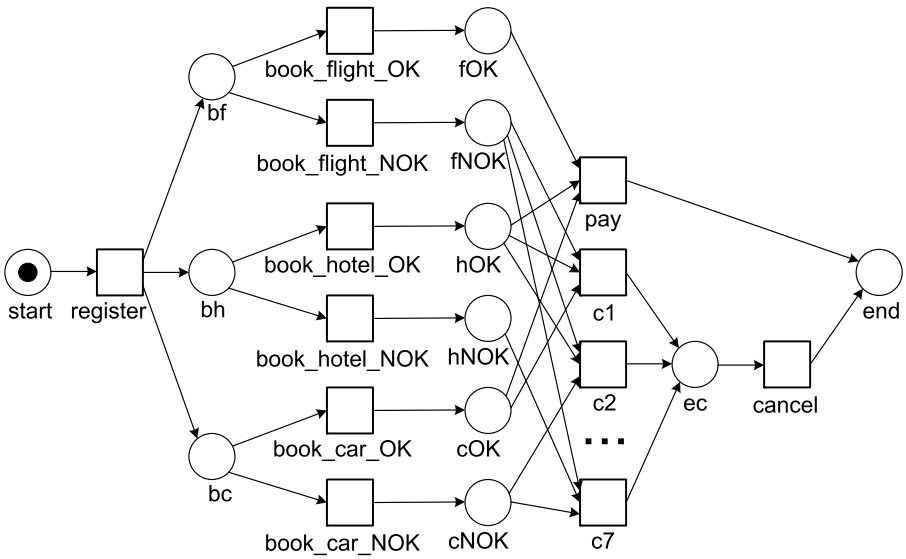

Fig. 1. A WF-net N_1 modeling the booking of trips. Note that after one "NOK" the trip will be cancelled eventually. There are $2^3 - 1 = 7$ situations modeled by transitions $c1, c2, \ldots, , c7$. Although the model is already complicated it fails to model that there should not be any booking activities after the first "NOK", because the trip will be cancelled anyway.

The WF-net in Figure 1 models the booking of trips. After registration a flight, a hotel, and a car are booked. Each of these booking activities can succeed ("OK") or fail ("NOK"). For reasons of simplicity, only the successful or unsuccessful completion of these activities is shown in Figure 1 (i.e., activities are considered to be atomic). If all booking activities succeed, then a payment follows. If one of them fails, a cancellation activity follows. Since each of the 3 booking activities can succeed or fail, there are $2^3 = 8$ scenarios. Only for one of these eight scenarios, payment is executed. For all other seven scenarios, the trip is cancelled.

Figure 1 is already rather complex but fails to model that there should not be any unnecessary work, i.e., after the first failure ("NOK"), no other booking activities should be executed as cancellation is inevitable. To model this, the seven c-transitions $(c1, c2, \ldots, , c7)$ are not adequate as there are $3^3 - 2^3 = 19$

[1] According to Google Scholar (visited on April 23rd, 2008), [2] got more than thousand references illustrating the interest in the topic. In fact, [2] is the second most cited workflow paper after [30].

possible states after registration and before payment/cancellation in which there is at least one "NOK". Hence, $19 - 7 = 12$ additional c-transitions are needed to capture this. Moreover, we simplified the model by assuming that the activities are atomic. This is not realistic because the booking of the flight, the hotel, and the car may happen in parallel and if one of them fails the other ones need to be withdrawn. If we incorporate this in the model, there there are $4^3 - 3^3 = 37$ states after registration and before payment/cancellation in which there is at least one "NOK".[2] This implies that to fully model the example 37 c-transitions are needed to remove the tokens from the right places. This illustrates that cancellation is difficult to model in WF-nets. Therefore, we propose to use *reset arcs* [24, 25, 29]. A reset arc removes tokens from a place but does not block the corresponding transition if the place is empty. This is a very useful construct that allows for the modeling of various cancellation operations supported by contemporary workflow languages, e.g., the withdraw construct of Staffware, the cancellation region of YAWL, the cancel event of BPMN, etc. In our example, the 37 c-transitions that are needed to remove the tokens from the right places, can be replaced by a single transition with reset arcs. This illustrates the usefulness and relevance of reset arcs. Therefore, we introduce the so-called *Reset WorkFlow nets* (RWF-nets) as an extension to the classical WF-nets [1, 2].

Taking RWF-nets as a starting point we explore a spectrum of analysis questions. Concretely, we investigate the following three challenges:

- *Discovering RWF-Nets.* Given an event log extracted from some database, transaction log, or set of use cases/audit trails, we want to automatically infer a process model. Since cancellation is important when modeling workflows, it is also important to discover cancellations when observing systems and processes.
- *Verification of RWF-Nets.* Many of the modern workflow languages offer some form of cancellation. Hence, it is important to be able to verify such models and point out deadlocks, livelocks, etc.
- *Conformance with respect to a RWF-Net.* The alignment of model and systems on the one hand and real-life processes on the other often leaves much to be desired. Therefore, it is important to be able to compare event logs with models. Since real-live processes and their models exhibit cancellation, it is important to take this into account when checking conformance of event logs and models.

The remainder of this paper is organized as follows. First, we present reset workflow nets and introduce three challenges (discovery, verification, and conformance), followed by Section 3 which introduces the concept of event logs. Section 4 focuses on the verification of workflows with cancellation. Section 5 shows that conformance can be checked by "playing the token game" based on the event log. Section 6 presents the challenge to discover reset workflow nets.

[2] Each of the booking activities has 4 states: enabled, running, succeeded (i.e., "OK"), and failed (i.e., "NOK"). Therefore, there are $4^3 = 64$ possible states and $3^3 = 27$ of these states are non-failure states.

An overview of related work is provided in Section 7 and Section 8 concludes the paper.

2 Reset Workflow Nets

The WF-net in Figure 1 is a nice illustration of the inability of classical Petri nets to model cancellation. Therefore, we use *reset nets*, i.e., classical Petri net extended with reset arcs.

Definition 1 (Reset net). *A* reset net *is a tuple* (P, T, F, W, R)*, where:*

- (P, T, F) *is a classical Petri net with places* P*, transitions* T*, and flow relation* $F \subseteq (P \times T) \cup (T \times P)$*,*
- $W \in F \to \mathbb{N} \setminus \{0\}$ *is an (arc) weight function, and*
- $R \in T \to 2^P$ *is a function defining reset arcs.*

A reset net extends the classical Petri net with reset arcs. These are used to remove all tokens from a place independent of the number of tokens. $R(t)$ is the set of places that are emptied when firing t. Also note that we are using arc weights. Arc weights specify the number of tokens to be consumed or produced. $W(p,t)$ is the number of tokens transition t consumes from input place p and $W(t,p)$ is the number of tokens transition t produces for output place p.

Figure 2 shows a reset net. In this example all arc weights are 1, i.e., $W(x,y) = 1$ for $(x,y) \in F$. Transition c has seven reset arcs connected to it. When c fires all tokens are removed from places fOK, hOK, cOK, NOK, bf, bh, and bc. For the enabling of c these reset arcs is irrelevant, i.e., c is enabled if and only if there is a token in place NOK.

The state of a reset net, also referred to as *marking*, is described as a multiset. Therefore, we introduce some notation. Let A be a set, e.g., the set of places P. $\mathbb{B}(A) = A \to \mathbb{N}$ is the *set of multi-sets* (bags) over A, i.e., $X \in \mathbb{B}(A)$ is a multi-set where for each $a \in A$: $X(a)$ denotes the number of times a is included in the multi-set. The sum of two multi-sets $(X+Y)$, the difference $(X-Y)$, the presence of an element in a multi-set $(x \in X)$, and the notion of sub-multi-set $(X \leq Y)$ are defined in a straightforward way and they can handle a mixture of sets and multi-sets. $\pi_{A'}(X)$ is the projection of X onto $A' \subseteq A$, i.e., $(\pi_{A'}(X))(a) = X(a)$ if $a \in A'$ and $(\pi_{A'}(X))(a) = 0$ if $a \notin A'$.

To represent a concrete multi-set we use square brackets, e.g., $[fOK, hOK, cOK]$ is the marking with a token in each of the "OK places" and $[NOK^3]$ is the marking with three tokens in place NOK.

Because of the arc weights the classical preset and postset operators return bags rather than sets: $\bullet a = [x^{W(x,y)} \mid (x,y) \in F \ \wedge \ a = y]$ and $a\bullet = [y^{W(x,y)} \mid (x,y) \in F \ \wedge \ a = x]$. For example, $\bullet pay = [fOK, hOK, cOK]$ is the bag of input places of pay and $pay\bullet = [end]$ is the bag of output places of pay.

Now we can formalize the notions of enabling and firing.

Definition 2 (Firing rule). *Let* $N = (P, T, F, W, R)$ *be a reset net and* $M \in \mathbb{B}(P)$ *be a marking.*

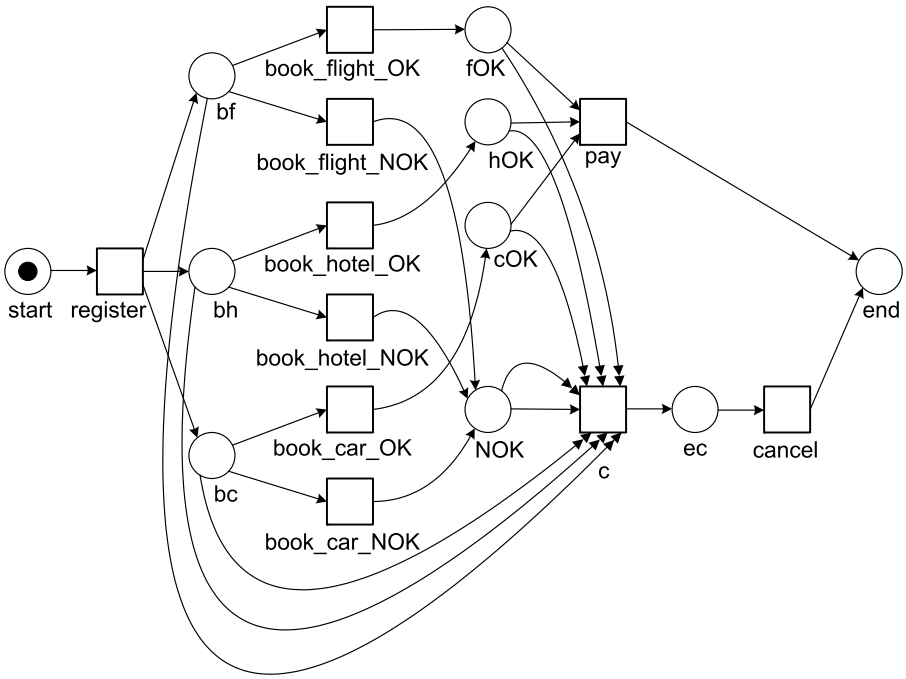

Fig. 2. A RWF-net N_2 modeling the booking of trips. Note that unlike the WF-net in Figure 1, unnecessary work is avoided. Moreover, the number of nodes to handle the cancellation is constant and the number of arcs is linear in the number of booking activities (flight, hotel, car, etc.) and activity states (enabled, running, succeeded, failed, etc.).

- *A transition $t \in T$ is enabled, notation $(N, M)[t\rangle$, if and only if, $M \geq {\bullet t}$.*
- *An enabled transition t can fire while changing the state to M', notation $(N, M)[t\rangle(N, M')$, if and only if, $M' = \pi_{P \setminus R(t)}(M - {\bullet t}) + t{\bullet}$.*

The resulting marking $M' = \pi_{P \setminus R(t)}(M - {\bullet t}) + t{\bullet}$ is obtained by first removing the tokens required for enabling: $M - {\bullet t}$. Then all tokens are removed from the reset places of t using projection. Applying function $\pi_{P \setminus R(t)}$ removes all tokens except the ones in the non-reset places $P \setminus R(t)$. Finally, the specified numbers of tokens are added to the output places. Note that $t{\bullet}$ is a *bag* of tokens

$(N, M)[t\rangle(N, M')$ defines how a Petri net can move from one marking to another by firing a transition. We can extend this notion to firing sequences. Suppose $\sigma = \langle t_1, t_2, \ldots, t_n \rangle$ is a sequence of transitions present in some Petri net N with initial marking M. $(N, M)[\sigma\rangle(N, M')$ means that there exists a sequence of markings $\langle M_0, M_1, \ldots, M_n \rangle$ where $M_0 = M$, $M_n = M'$, such that for any $0 \leq i < n$: $(N, M_i)[t_{i+1}\rangle(N, M_{i+1})$. Using this notation we define the set of reachable markings $R(N, M)$ as follows: $R(N, M) = \{M' \in \mathbb{B}(P) \mid \exists_\sigma (N, M)[\sigma\rangle(N, M')\}$.

Note that by definition $M \in R(N, M)$ because the initial marking M is trivially reachable via the empty sequence ($n = 0$).

We would like to emphasize that any reset net with arc weights can be transformed into a reset net without arc weights, i.e., all arcs have weight 1. Therefore, in proofs can assume arc weights of 1 when convenient and still use them in constructs. See [6] for a construction.

The idea of a workflow process is that many *cases* (also called *process instances*) are handled in a uniform manner. The workflow definition describes the ordering of *activities* to be executed for each case including a clear *start state* and *end state*. These basic assumptions lead to the notion of a *WorkFlow net* (WF-net) [1, 2] which can easily be extended in the presence of reset arcs.

Definition 3 (RWF-net). *An reset net $N = (P, T, F, W, R)$ is a Reset WorkFlow net (RWF-net) if and only if*

- *There is a single source place i, i.e., $\{p \in P \mid \bullet p = \emptyset\} = \{i\}$.*
- *There is a single sink place o, i.e., $\{p \in P \mid p\bullet = \emptyset\} = \{o\}$.*
- *Every node is on a path from i to o, i.e., for any $n \in P \cup T$: $(i, n) \in F^*$ and $(n, o) \in F^*$.*
- *There is no reset arc connected to the sink place, i.e., $\forall_{t \in T}\ o \notin R(t)$.*

Figures 1 and 2 both show a RWF-net. The requirement that $\forall_{t \in T}\ o \notin R(t)$ has been added to emphasize that termination should be irreversible, i.e., it is not allowed to complete (put a token in o) and then undo this completion (remove the token from o).

Let us now compare figures 1 and 2 showing RWF-nets N_1 and N_2 respectively. In the the original net without reset arcs (N_1) the number of cancellation transitions is exponential in the number of bookings while in the second net (N_2) there is just one cancellation transition and the number of reset arcs is linear in the number of bookings. Also note that in N_2 tokens are also removed from the input places of the booking activities to make sure than no unnecessary work is conducted. Extending N_1 to obtain the same behavior requires the addition of 12 more cancellation transitions. This clearly shows the benefits of using reset arcs. Moreover, figures 1 and 2 also illustrate the need for the modeling of cancellations in real-life workflow processes.

3 Event Logs

Traditionally, the focus of workflow analysis at design-time has been on model-based verification and simulation while at run-time the focus has been on measuring simple key performance indicators such as flow times, service levels, etc. Because more and more information about processes is recorded by information systems in the form of so-called "event logs", it seems vital to also use this information while analyzing processes. A wide variety of process-aware information systems [26] is recording excellent data on actual events taking place. ERP (Enterprise Resource Planning), WFM (WorkFlow Management), CRM

(Customer Relationship Management), SCM (Supply Chain Management), and PDM (Product Data Management) systems are examples of such systems. Despite the omnipresence and richness of these event logs, most software vendors have been focusing on relatively simple questions *under the assumption that the process is fixed and known*, e.g., the calculation of simple performance metrics like utilization and flow time. However, in many domains processes are evolving and people typically have an oversimplified and incorrect view of the actual business processes. Therefore, *process mining* techniques attempt to extract non-trivial and useful information from event logs. One aspect of process mining is *control-flow discovery*, i.e., automatically constructing a process model (e.g., a Petri net) describing the causal dependencies between activities [11, 12, 17, 20].

Later in this paper, we discuss process discovery and conformance checking using RWF-nets. These are particular process mining techniques that require event logs as input. Therefore, we define the notion of an event log.

Definition 4 (Event log). *Let A be a set of activities. A* trace *σ can be described as a sequence of activities, i.e., $\sigma \in A^*$. An event log L is a multiset of traces, i.e., $L \in \mathbb{B}(A^*)$.*

A trace can be considered as the execution path of a single process instance (case). Note that this is a rather simplified view, i.e., in real life events have timestamps (When did the activity happen?), resource information (Who executed the activity?), data (What information was used and produced?), etc. However, for this paper we focus on the control-flow only. A trace possible according to Figure 2 is $\sigma = \langle \mathit{register}, \mathit{book\_flight\_NOK}, c, \mathit{cancel} \rangle$. An example event log consisting of 5 traces is $L = [\langle \mathit{register}, \mathit{book\_hotel\_NOK}, c, \mathit{cancel} \rangle^3, \langle \mathit{register}, \mathit{book\_hotel\_OK}, \mathit{book\_car\_OK}, \mathit{book\_flight\_OK}, \mathit{pay} \rangle^2]$.

As already indicated in Section 1, this paper focuses on three challenges:

- *Discovering RWF-Nets.* Given an event log L, we want to infer a RWF-net N.
- *Verification of RWF-Nets.* Given a RWF-net N, we want to discover errors such as deadlocks, etc.
- *Conformance with respect to a RWF-Net.* Given an event log L and a RWF-net N, we want to discover discrepancies between L and N.

The remainder of this paper will focus on these three challenges. We start by elaborating on the verification of RWF-nets.

4 Verification of RWF-Nets

Based on the notion of RWF-nets we now investigate the fundamental question: "Is the workflow correct?". If one has domain knowledge, this question can be answered in many different ways. However, without domain knowledge one can only resort to generic questions such as: "Does the workflow terminate?", "Are there any deadlocks?", "Is it possible to execute activity A?", etc. Such kinds of generic questions triggered the definition of *soundness* [1, 2]. Different soundness

notions have been proposed, e.g., k-soundness [32, 33], weak soundness [39], generalized soundness [32, 33], relaxed soundness [21], etc. However, here we focus on the original definition given in [1].

Definition 5 (Classical soundness [1, 2]). *Let $N = (P, T, F, W, R)$ be a RWF-net. N is sound if and only if the following three requirements are satisfied:*

- *Option to complete:* $\forall_{M \in R(N,[i])}\ [o] \in R(N, M)$.
- *Proper completion:* $\forall_{M \in R(N,[i])}\ (M \geq [o]) \Rightarrow (M = [o])$.
- *No dead transitions:* $\forall_{t \in T}\ \exists_{M \in R(N,[i])}\ (N, M)[t\rangle$.

A RWF-net such as the one sketched in Figure 1 is sound if and only if the following three requirements are satisfied: (1) *option to complete*: for each case it is always still possible to reach the state which just marks place *end*, (2) *proper completion*: if place *end* is marked all other places are empty for a given case, and (3) *no dead transitions*: it should be possible to execute an arbitrary activity by following the appropriate route through the RWF-net. It is easy to see that N_1 and N_2 (figures 1 and 2) are sound.

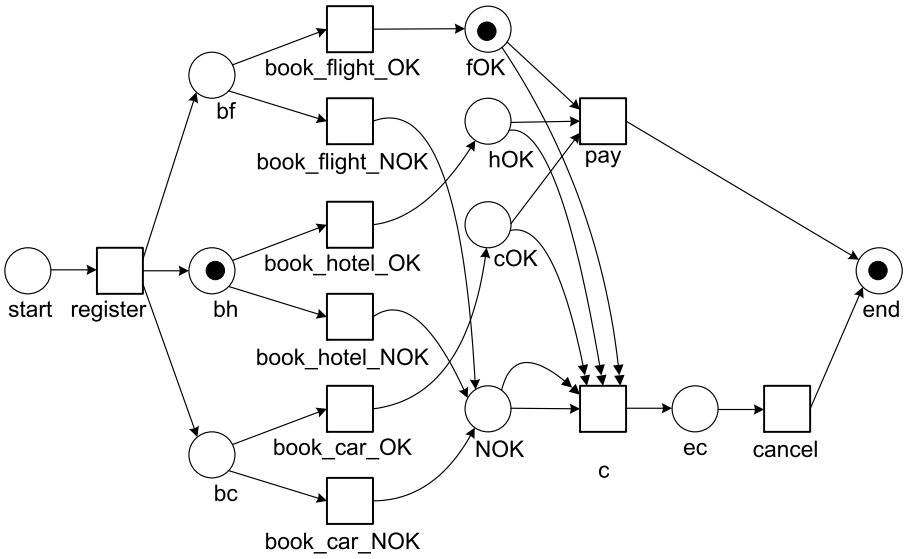

Fig. 3. A RWF-net N_3 that is not sound. From the initial marking $[start]$ e.g. the marking shown (i.e., $[bh, fOK, end]$) is reachable. This shows that the first two requirements stated in Definition 5 do not hold.

RWF-net N_3 shown in Figure 3 is an example of a workflow that is not sound. Since c does not remove tokens from the places before the booking activities,

tokens may be left behind. In fact, it is still possible to book a hotel after transition *cancel* has put a token in *end* (cf. Figure 3). This example shows that it is easy to make errors when modeling workflows with cancellation.

In [1, 2] it was shown that soundness is decidable for WF-nets, i.e., RWF-nets without reset arcs. A WF-net $N = (P, T, F)$ (without reset arcs and arc weights) is sound if and only if the short-circuited net $(\overline{N}, [i])$ with $\overline{N} = (P, T \cup \{t^*\}, F \cup \{(o, t^*), (t^*, i)\})$ is live and bounded. Since liveness and boundedness are both decidable, soundness is also decidable. For some subclasses (e.g., free-choice nets), this is even decidable in polynomial time [1, 2].

Since the mid-nineties many people have been looking at the verification of workflows. These papers all assume some underlying model (e.g., WF-nets) and some correctness criterion (e.g., soundness). However, in many cases a rather simple model is used (WF-nets or even less expressive) and practical features such as *cancellation* are missing. Many practical languages have a cancellation feature, e.g., Staffware has a withdraw construct, YAWL has a cancellation region, BPMN has cancel, compensate, and error events, etc. Therefore, it is interesting to investigate the notion of soundness in the context of RWF-nets, i.e., WF-nets with reset arcs [24, 25, 29]. Unfortunately, soundness is *not* decidable for RWF-nets with reset arcs.

Theorem 1 (Undecidability of soundness). *Soundness is undecidable for RWF-nets.*

For a proof we refer to [6]. Although far from trivial, it is possible to construct a RWF-net N' given an arbitrary reset net N such that N' is sound if and only if M' is *not* reachable from M in N. Since reachability is undecidable for reset nets [24, 25, 29], this implies that soundness is also undecidable for RWF-nets.

Theorem 1 is non-trivial because properties such as coverability (Is it possible to reach a marking M' that covers M, i.e., $M' \geq M$?) are decidable for reset nets.

Note that although soundness is undecidable for RWF-nets, for many representatives of this class, it may still be possible to conclude soundness or non-soundness. There may be rules of the form "If WF-net N has property X, then N is sound" or "If WF-net N has property Y, then N is not sound". As shown in [41] it is possible to find many errors using such an approach. In [41] a set of more than 2000 process models from practice (including more than 600 processes from the SAP reference model) was analyzed. It could be shown that at least 10 percent of these models is not sound. These examples show that *even if soundness is undecidable, errors can be discovered.* Similarly, for many models it is still possible to guarantee soundness even if the general verification problem is undecidable.

In the related work section, we provide some pointers to analysis techniques using brute force (e.g. coverability graphs), structural techniques (invariants), and/or reduction rules. For example, RWF-nets can be reduced using the reduction rules presented in [48] to speed-up analysis and improve diagnostics.

5 Conformance with Respect to a RWF-Net

As indicated in Section 3, lion's share of analysis efforts has been devoted to model-based analysis (verification, simulation, etc.) and measuring simple performance indicators. However, given the abundance of event logs it is interesting to "discover models" based on event logs (see Section 6) or to measure the conformance of existing models based on the real behavior recorded in logs. In this section, we focus on the latter question, i.e., "Do the model and the log *conform* to each other?". Conformance checking aims at the detection of inconsistencies between a process model N and its corresponding execution log L, and their quantification by the formation of metrics. In [4, 42, 43] two basic conformance notions have been identified (fitness and appropriateness). First of all, the *fitness* between the log and the model is measured (i.e., "Does the observed process comply with the control flow specified by the process model?"). Second, the *appropriateness* of the model can be analyzed with respect to the log (i.e., "Does the model describe the observed process in a suitable way?"). Appropriateness can be evaluated from both a *structural* and a *behavioral* perspective [43].

In this paper, we only consider fitness. However, it is important to stress that a model with good fitness may not be appropriate. For example, the model with just a single place that serves as a self-loop for all transitions T is able to parse any trace in T^* [4, 42, 43].

One way to measure the fit between event logs and process models is to "replay" the log in the model and somehow measure the mismatch. The replay of every trace starts with the marking of the initial place in the model, e.g., $[start]$ in Figure 1. Then, the transitions that belong to the logged events in the trace are fired one after another. While replay progresses, we count the number of tokens that had to be created artificially (i.e., the transition belonging to the logged event was not enabled and therefore could not be *successfully executed*) and the number of tokens that were left in the model, which indicate that the process was not *properly completed*. Based on counting the number of missing tokens during replay and the number of remaining tokens after replay, we define a function f that has a value between 0 (=poor fitness) and 1 (=good fitness).

Definition 6 (Fitness). *Let $N = (P, T, F, W, R)$ be a RWF-net and let $L \in \mathbb{B}(A^*)$ be an event log where we assume that $T = A$. Let k be the number of traces in event log L. For each log trace i ($1 \leq i \leq k$), m_i is the number of missing tokens, r_i is the number of remaining tokens, c_i is the number of consumed tokens, and p_i is the number of produced tokens during log replay of trace i. The token-based fitness metric f is defined as follows:*

$$f(N, L) = \frac{1}{2}(1 - \frac{\sum_{i=1}^{k} m_i}{\sum_{i=1}^{k} c_i}) + \frac{1}{2}(1 - \frac{\sum_{i=1}^{k} r_i}{\sum_{i=1}^{k} p_i})$$

Note that the above definition is rather informal. In [42, 43] this metric was defined for WF-nets, i.e., workflows without cancellation. However, as shown here the metric can easily be extended for RWF-nets. Let us consider some

trace i consisting of a sequence of n events $\sigma_i = \langle e_1, e_2, \ldots, e_n \rangle \in T^*$. The initial state of N is $[i]$ and the desired end state is $[o]$. If $(N, [i])[\sigma_i\rangle(N, [o])$, then there is a perfect fit, i.e., the trace can be replayed without leaving tokens behind. So, $(N, [i])[\sigma_i\rangle(N, [o])$ if and only if $f(N, [\sigma_i]) = 1$. Let $\sigma_1 = \langle \mathit{register}, \mathit{book\_hotel\_NOK}, c, \mathit{cancel} \rangle$. It is easy to see that $f(N_2, [\sigma_1]) = 1$, i.e., the trace can be replayed without missing or remaining tokens. (Recall that N_2 is the RWF-net shown in Figure 2.) Now consider $\sigma_2 = \langle \mathit{register}, \mathit{book\_hotel\_OK}, c, \mathit{pay} \rangle$. It is possible to execute the partial trace $\langle \mathit{register}, \mathit{book\_hotel\_OK} \rangle$. However, to execute c, there has to be a token in NOK (i.e., one token is missing). If we force c to fire anyway, the resulting state is $[ec]$. In this state, we cannot fire *pay* as there are three missing tokens. This brings the number of missing tokens to 4. After forcing *pay* to fire, the resulting state is $[ec, \mathit{end}]$. Hence one token remains in place *ec*.

We can show the calculation of the values m_2, r_2, c_2, and p_2 step-by-step using four temporary variables. Initially, $m = 0$ (no missing tokens), $r = 0$ (no remaining tokens), $c = 0$ (no consumed tokens), and $p = 1$ (prior to the execution of *register* the net is in state $[\mathit{start}]$, so the environment already put a token in the initial place). After *start* fires state $[bf, bh, bc]$ is obtained and $m = 0$, $c = 0 + 1 = 1$, and $p = 1 + 3 = 4$. After *book_hotel_OK* fires marking $[bf, bc, hOK]$ is reached and $m = 0$, $c = 1+1 = 2$, and $p = 4+1 = 5$. After c fires state $[ec]$ is obtained and $m = 0 + 1 = 1$ (missing token in NOK), $c = 2 + 4 = 6$ (four tokens are removed, one by a "normal" arc and three by reset arcs), and $p = 5 + 1 = 6$. After *pay* fires state $[ec, \mathit{end}]$ is reached and $m = 1 + 3 = 4$, $c = 6 + 3 = 9$, and $p = 6 + 1 = 7$. Finally, the token is removed from *end* and the remaining token is recorded, i.e., $c = 9 + 1 = 10$ and $r = 1$. Note that in the calculation the marking of the source place is considered to be a production step while the removal of the token from the sink place is considered to be a consumption step. Also note that the removal of tokens through reset arcs is calculated as a consumption step. Hence, $m_2 = 4$, $r_2 = 1$, $c_2 = 10$, and $p_2 = 7$. Therefore, $f(N_2, [\sigma_2]) = \frac{1}{2}(1 - \frac{4}{10}) + \frac{1}{2}(1 - \frac{1}{7}) = \frac{51}{70} \cong 0.73$. The fitness of $f(N_2, [\sigma_1, \sigma_2]) = \frac{1}{2}(1 - \frac{0+4}{7+10}) + \frac{1}{2}(1 - \frac{0+1}{7+7}) = \frac{403}{476} \cong 0.85$.

Several definitions of fitness are possible. For example, Definition 6 gives equal weights to missing tokens and remaining tokens. By replacing the weights $\frac{1}{2}$ by e.g. weight $\frac{3}{4}$ and weight $\frac{1}{4}$ in Definition 6, more emphasis is put on problems related to missing tokens and less on proper termination.

Several metrics for fitness and various appropriateness notions have been implemented in ProM for WF-nets [42, 43]. As shown in this section, these metrics can be adapted for RWF-nets in a straightforward manner. Figure 4 illustrates the current functionality of ProM. For a specific log, the fitness is calculated with respect to the WF-net shown in Figure 1. As Figure 4 shows the fitness is 0.80908644 (metric is shown in top right corner). Several places are annotated with one or two numbers. Positive numbers refer to remaining tokens (i.e., $\sum_{i=1}^{k} r_i$ for a particular place) and negative tokens refer to missing tokens (i.e., $\sum_{i=1}^{k} m_i$ for a particular place). The input place of *cancel* (i.e., place *ec* in Figure 1) has the number "−25" indicating that in the whole log there were 25

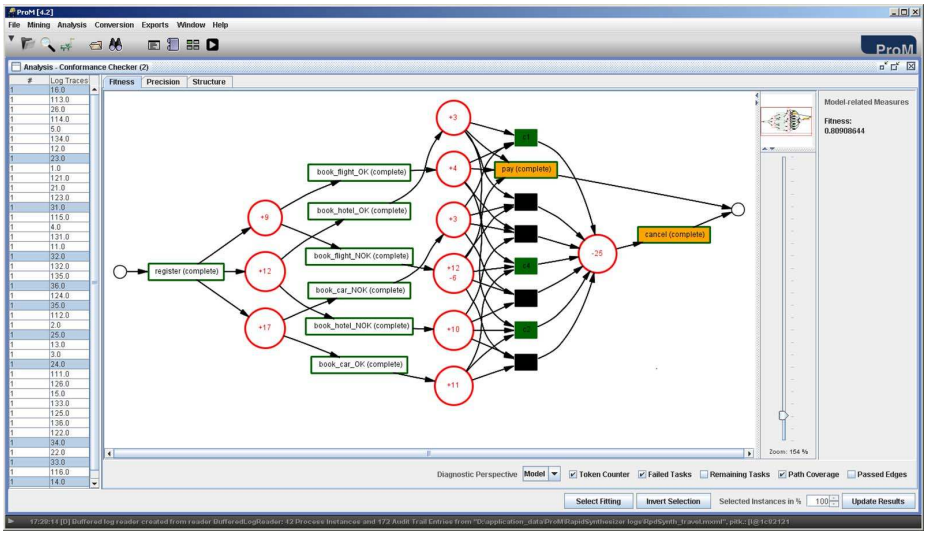

Fig. 4. The conformance checker in ProM applied to the WF-net shown in Figure 1 and an event log containing 42 cases and 172 events.

situations where according to the log *cancel* had to fire while according to the model this was not possible because *ec* was empty. As shown in Figure 4, nice diagnostics can be given showing were in the model mismatches occur and how severe they are.

Note that the transitions $c1, c2, \ldots, c7$ in Figure 1 are depicted differently in the conformance checker (cf. Figure 4). The reason is that there are no events related to $c1, c2, \ldots, c7$ in the log, i.e., these are "silent transitions" and cannot be observed. The conformance checker in ProM can deal with such situations using state-space analysis. The same technique is used to deal with "duplicates", i.e., two transitions having the same label. See [43] for details. Interestingly, all ideas from [43] can be generalized to workflows with cancellation.

6 Discovering RWF-Nets

The last challenge addressed in this paper is the discovery of workflows with cancellation, i.e., based on some event log L we want to automatically construct a RWF-net N that "captures" the behavior seen in the log. Many techniques have been proposed in literature [11, 8, 12, 17, 20, 22, 23, 49]. However, *none of these techniques discovers workflow models with cancellation features.*

Figure 5 illustrates the concept of process discovery. Given an event log without any explicit process information, we want to discover a process model. On the right-hand side of Figure 5, a fragment of a larger event log is shown. As discussed in Section 3, event logs tend to have more information (e.g., timestamps,

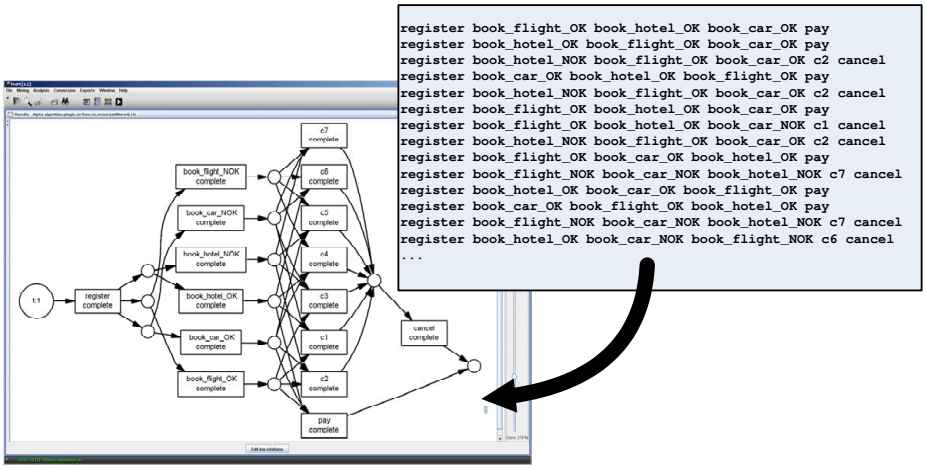

Fig. 5. Based on a complete event log, the α-algorithm [11] can discover the WF-net shown in Figure 1.

data, etc.), but here was assume that an event log is simply a multiset of traces. For example, the first trace in Figure 5 refers to a scenario were all bookings succeeded. The α-algorithm [11] is a very basic process mining algorithm that is able to discover the model shown in Figure 5. Since the traces shown correspond to possible traces of the WF-net N_1 shown in Figure 1, it is nice to see that the α-algorithm is actually able to discover N_1 (modulo renaming of places). The α-algorithm [11] is very simple but not very suitable for real-life applications. The algorithm makes strong assumptions about the routing constructs to be used and the completeness of the log. For example, it is not realistic that one actually observes the routing transitions $c1, c2, \ldots, , c7$. Unlike transition *cancel* which is a real activity, $c1, c2, \ldots, , c7$ have only been added for routing purposes. Fortunately, better process mining algorithms are available today (see Section 7). However, these *do not capture cancellation as the underlying models do not allow for a direct representation of such constructs.*

The goal is to develop process mining algorithms that discover cancellations in event logs and treat cancellation as a basic construct. Concretely, we want to *discover a RWF-net N with "suitable" reset arcs based on some event log L.* Since we do not what to develop a process mining algorithm from scratch, we try to extend existing techniques.

The basic idea behind most of the existing process mining algorithms is to add a causal dependency based on an analysis of the log. For example, $a >_L b$ iff there is a trace in L where a is directly followed by b and $a \rightarrow_W b$ iff $a >_W b$ and $b \not> _W a$. Using such information places are added, e.g., a and b are connected through some place if $a \rightarrow_W b$. Hence, the places provide information about one activity triggering another activity. However, there is no explicit information in

the log on disabling events (i.e., there is no "negative information" in the log). Therefore, we suggest to use existing algorithms and do some post-processing using reset arcs as a cleanup. Below is an informal sketch of the basic idea:

Step 1 Given an event log L construct a RWF-net $N = (P, T, F, W, R)$ using conventional process mining techniques (initially $R(t) = \emptyset$ for all $t \in T$). It is best to use a technique that avoids blocking transitions, i.e., no missing tokens (m_i) in the sense of Definition 6.

Step 2 Construct a relation $\not\gg_L \subseteq T \times T$ such that $a \not\gg_L b$ if and only if a is never followed by b.

Step 3 Replay the log in $N = (P, T, F, W, R)$ and record places with remaining tokens and calculate the fitness. If there are no remaining tokens or all alternatives below have been tried, return N.

Step 4 Pick a place p that has the most remaining tokens. Let T_p be the set of output transitions of p, i.e., $T_p = p\bullet$.

Step 5 $T' = \{t' \in T \setminus T_p \mid \forall_{t \in T_P}\ t' \not\gg_L t\ \wedge\ p \notin R(t')\}$, i.e., transitions that "seem" to disable T_p transitions but do not actually disable these transitions yet. If $T' = \emptyset$, then go to Step 3, otherwise pick a $t_r \in T'$. Take the "earliest" transition in T', e.g., using a relation similar to $\gg_L$.

Step 6 Add a reset arc to N connecting p and t_r, i.e., $N' = (P, T, F, W, R')$ where $R'(t_r) = R(t_r) \cup \{p\}$ and $R'(t) = R(t)$ for all other t.

Step 7 Return to Step 3 using $N = N'$.

Note that the above is *not* indented to be a concrete algorithm. It is merely a solution approach that needs to be made specific in the context of a concrete process mining algorithm. To illustrate this let us use a log L_2 that contains all possible behaviors of the RWF-net shown in Figure 2. In the log transition c is not visible as it is just there for routing purposes, i.e., an example trace in L_2 is $\langle$*register*, *book_flight_NOK*, *cancel*$\rangle$. Applying the α-algorithm to L_2 gives an incorrect and rather meaningless result because of the invisible routing activity c and the cancellation construct. If we apply the region-based approach presented in [9, 44], then we obtain the Petri net shown in Figure 6. The region-based approach guarantees that it is possible to replay all traces in the log without missing tokens, but there may be remaining tokens. In terms of Definition 6, this means $m_i = 0$ and $r_i \geq 0$ for any trace i. This makes the technique of [9, 44] suitable for the post-processing mentioned above. Note that there are six places where tokens may remain.

The Petri net in Figure 6 is able to execute the sequence $\langle$*register*, *book_flight_NOK*, *cancel*$\rangle$ but leaves two tokens behind. Note that the central place in-between *register* and *pay* acts as a mutex place blocking all activities after the first "NOK". Also note that there is a not a sink place in Figure 6, i.e., it is not a WF-net. This is due to the implementation of the plug-in in ProM and is merely a technicality that can be resolved easily (e.g., by adding a dummy end transition). Using the 7 steps described above reset arcs are added from the six places with remaining tokens to transition *cancel*. However, also superfluous reset arcs are added, e.g., from some of places with remaining tokens to *pay*. This can be optimized in various ways. First of all, additions that do not improve

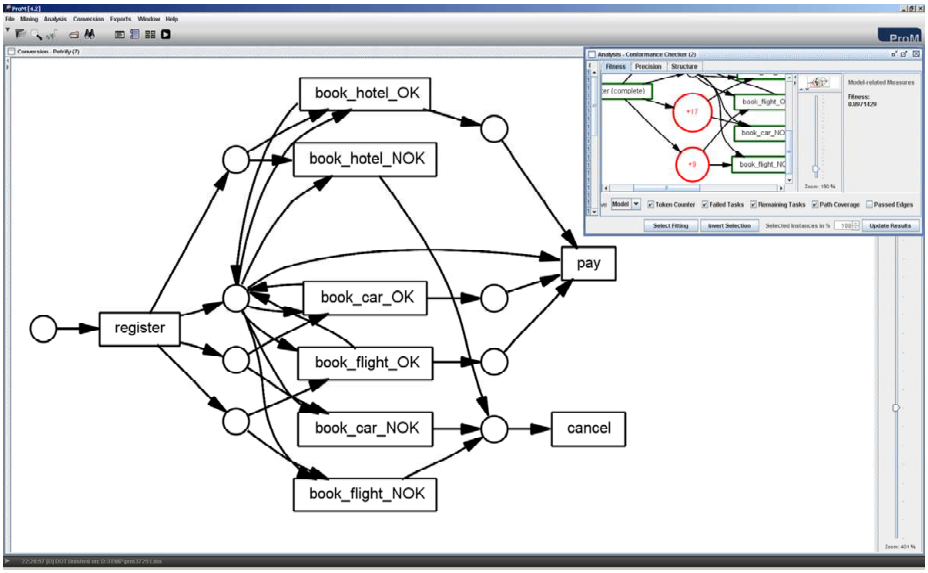

Fig. 6. Using regions, a Petri net is discovered that captures the behavior but where tokens are left behind. The fitness is 0.897 as shown in top right corner.

fitness can be discarded. Second, improving T' to filter out transitions that do not appear in traces that have problems (i.e., if there is no problem related to sequences where transition t' appears, then no reset of t' on p is needed). Both optimizations would get rid of the superfluous reset arcs. The resulting model nicely captures the behavior recorded in the log including the cancellation after the first "NOK" result.

There are alternatives to the post-processing approach described above. First of all, it would be relatively easy to extend the genetic miner in ProM [7, 40] to deal with reset arcs. For genetic mining basically only a good representation (RWF-nets) and fitness function (Definition 6) are needed [7, 40]. Second, it would be interesting to extend mining approaches based on language-based regions [45] to come up with special arcs. Figure 7 shows the application of the language-based region miner in ProM using a very conservative setting for log L_2, i.e., the same log as used to construct Figure 6. Because of the conservative setting just a few places were added, however, a correct characterization of the behavior is given. This shows the basic principle that a Petri net without any places can parse any log and that adding places corresponds to adding constraints. Since this approach uses integer linear programming as a basis, it is versatile and seems to be a good platform to add special types of arcs such as reset arcs, inhibitor arcs, etc.

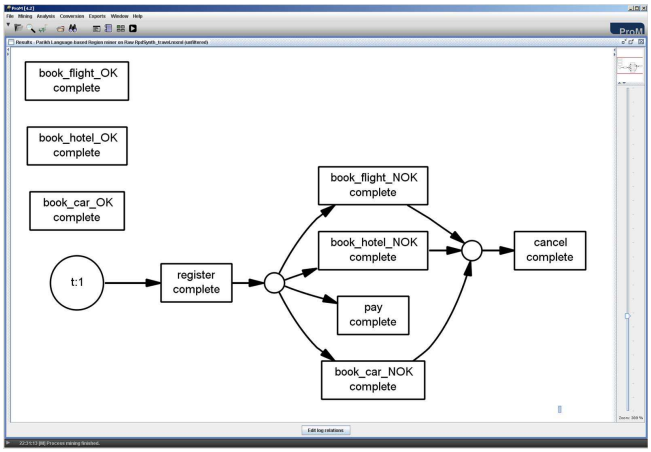

Fig. 7. A Petri net constructed using language-based regions theory [45].

7 Related Work

Since the mid nineties, many researchers have been working on workflow verification techniques. It is impossible to give a complete overview here. Moreover, most of the papers on workflow verification focus on rather simple languages, e.g., AND/XOR-graphs which are even less expressive than classical Petri nets. Therefore, we only mention the work directly relevant to this paper.

The use of Petri nets in workflow verification has been studied extensively. In [1, 2] the foundational notions of WF-nets and soundness are introduced. In [32, 33] two alterative notions of soundness are introduced: k-soundness and generalized soundness. These notions allow for dead parts in the workflow but address problems related to multiple instantiation. In [39] the notion of weak soundness is proposed. This notion allows for dead transitions. The notion of relaxed soundness is introduced in [21]. This notion allows for potential deadlocks and livelocks, however, for each transition there should be at least one proper execution.

Most soundness notions (except generalized soundness [32, 33]) can be investigated using classical model checking techniques that explore the state space. However, such approaches can be intractable or even impossible because the state-space may be infinite. Therefore, alternative approaches that avoid constructing the (full) state space have been proposed. [3] describes how structural properties of a workflow net can be used to detect the soundness property. [46, 47] presents an alternative approach for deciding relaxed soundness in the presence of OR-joins using invariants. The approach taken results in the approximation of OR-join semantics and transformation of YAWL nets into Petri nets with inhibitor arcs. In [51] it is shown that the backward reachability graph can be used to determine the enabling of OR-joins in the context of cancellation. In the

general area of reset nets, Dufourd et al.'s work has provided valuable insights into the decidability status of various properties of reset nets including reachability, boundedness and coverability [24, 25, 29]. Moreover, in [48] it is shown that reduction rules can be applied to reset nets (and even to inhibitor nets) to speed-up analysis and improve diagnostics.

Since the mid-nineties several groups have been working on techniques for process mining [11, 8, 12, 17, 20, 22, 23, 49], i.e., discovering process models based on observed events. In [10] an overview is given of the early work in this domain. The idea to apply process mining in the context of workflow management systems was introduced in [12]. In parallel, Datta [20] looked at the discovery of business process models. Cook et al. investigated similar issues in the context of software engineering processes [17]. Herbst [34] was one of the first to tackle more complicated processes, e.g., processes containing duplicate tasks.

Most of the classical approaches have problems dealing with concurrency. The α-algorithm [11] is an example of a simple technique that takes concurrency as a starting point. However, this simple algorithm has problems dealing with complicated routing constructs and noise (like most of the other approaches described in literature). In [22, 23] a more robust but less precise approach is presented. The classical "theory of regions" [13, 14, 18, 19, 27] can also be used to discover Petri-net-based models as shown in [9, 44]. Recently, some work on language-based regions theory appeared [16, 45, 37, 38]. In [16, 45] it is shown how this can be applied to process mining.

In this paper we do not consider issues such as noise. Heuristics [49] or genetic algorithms [7, 40] have been proposed to deal with issues such as noise.

For an overview of related work with respect to conformance checking we refer to [4, 42, 43]. Note that so far no process mining techniques (discovery and/or conformance) have been proposed for models with cancellation such as RWF-nets.

To conclude this related work section, we provide some pointers to the relationships between Petri nets and graph grammars/transformations [28, 31]. The relationships between Petri nets and graph grammars have been well known for quite some time [35]. In fact, graph grammars can be seen as a proper generalization of Petri nets. The firing of a transition corresponds to applying a "production" to a graph while firing sequences correspond to graph derivations. Reset arcs can easily be encoded in terms of graph grammars. In Section 3.2 of [31], extensions of graph rewriting using multi objects are discussed, i.e., universally quantified operations are used to remove all objects of a particular type in one go. Such ideas directly apply to reset nets. In [15] the relation between "extended Petri nets" and graph rewriting is investigated in detail. In this paper, Petri nets having read, inhibitor and reset arcs are mapped onto graph grammars. Thus far little work has been done on the relation between graph grammars/transformations on the one hand and workflow verification and process discovery on the other. It would be interesting to explore this further.

8 Conclusion

In this paper we explored various analysis questions related to workflows with cancellations. As a modeling language we used *reset workflow nets* (RWF-nets). Taking RWF-nets as a starting point we explored challenges related to *discovery* (process mining), *verification*, and *conformance*. For example, it was shown that soundness is undecidable for RWF-nets. However, despite this result, analysis is possible in most cases using e.g. reduction rules and structural techniques as shown in [48]. Conformance checking can be done in a straightforward manner by adapting the techniques described in [42, 43] to RWF-nets. From a process mining viewpoint, no prior work has been done on the discovery of processes with cancellations. In this paper we made some initial suggestions to develop process discovery algorithms for RWF-nets. Given the importance of cancellation in workflows, it is interesting to develop techniques and tools to further address the challenges mentioned in this paper.

References

1. W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer-Verlag, Berlin, 1997.
2. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
3. W.M.P. van der Aalst. Workflow Verification: Finding Control-Flow Errors using Petri-net-based Techniques. In W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors, *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 161–183. Springer-Verlag, Berlin, 2000.
4. W.M.P. van der Aalst. Business Alignment: Using Process Mining as a Tool for Delta Analysis and Conformance Testing. *Requirements Engineering Journal*, 10(3):198–211, 2005.
5. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2004.
6. W.M.P. van der Aalst, K.M. van Hee, A.H.M. ter Hofstede, N. Sidorova, H.M.W. Verbeek, M. Voorhoeve, and M.T. Wynn. Soundness of Workflow Nets: Classification, Decidability, and Analysis. BPM Center Report BPM-08-02, BPMcenter.org, 2008.
7. W.M.P. van der Aalst, A.K. Alves de Medeiros, and A.J.M.M. Weijters. Genetic Process Mining. In G. Ciardo and P. Darondeau, editors, *Applications and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 48–69. Springer-Verlag, Berlin, 2005.
8. W.M.P. van der Aalst, H.A. Reijers, A.J.M.M. Weijters, B.F. van Dongen, A.K. Alves de Medeiros, M. Song, and H.M.W. Verbeek. Business Process Mining: An Industrial Application. *Information Systems*, 32(5):713–732, 2007.
9. W.M.P. van der Aalst, V. Rubin, B.F. van Dongen, E. Kindler, and C.W. Günther. Process Mining: A Two-Step Approach using Transition Systems and Regions. BPM Center Report BPM-06-30, BPMcenter.org, 2006.

10. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.
11. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.
12. R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Sixth International Conference on Extending Database Technology*, pages 469–483, 1998.
13. E. Badouel, L. Bernardinello, and P. Darondeau. The Synthesis Problem for Elementary Net Systems is NP-complete. *Theoretical Computer Science*, 186(1-2):107–134, 1997.
14. E. Badouel and P. Darondeau. Theory of regions. In W. Reisig and G. Rozenberg, editors, *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 529–586. Springer-Verlag, Berlin, 1998.
15. P. Baldan, A. Corradini, and U. Montanari. Relating SPO and DPO Graph Rewriting with Petri nets having Read, Inhibitor and Reset Arcs. *Electronic Notes in Theoretical Computer Science*, 127(2):5–28, 2005.
16. R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Process Mining Based on Regions of Languages. In G. Alonso, P. Dadam, and M. Rosemann, editors, *International Conference on Business Process Management (BPM 2007)*, volume 4714 of *Lecture Notes in Computer Science*, pages 375–383. Springer-Verlag, Berlin, 2007.
17. J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.
18. J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Synthesizing Petri Nets from State-Based Models. In *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design (ICCAD '95)*, pages 164–171. IEEE Computer Society, 1995.
19. J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Deriving Petri Nets from Finite Transition Systems. *IEEE Transactions on Computers*, 47(8):859–882, August 1998.
20. A. Datta. Automating the Discovery of As-Is Business Process Models: Probabilistic and Algorithmic Approaches. *Information Systems Research*, 9(3):275–301, 1998.
21. J. Dehnert and W.M.P. van der Aalst. Bridging the Gap Between Business Models and Workflow Specifications. *International Journal of Cooperative Information Systems*, 13(3):289–332, 2004.
22. B.F. van Dongen and W.M.P. van der Aalst. Multi-Phase Process Mining: Building Instance Graphs. In P. Atzeni, W. Chu, H. Lu, S. Zhou, and T.W. Ling, editors, *International Conference on Conceptual Modeling (ER 2004)*, volume 3288 of *Lecture Notes in Computer Science*, pages 362–376. Springer-Verlag, Berlin, 2004.
23. B.F. van Dongen and W.M.P. van der Aalst. Multi-Phase Mining: Aggregating Instances Graphs into EPCs and Petri Nets. In D. Marinescu, editor, *Proceedings of the Second International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management*, pages 35–58. Florida International University, Miami, Florida, USA, 2005.
24. C. Dufourd, A. Finkel, and Ph. Schnoebelen. Reset Nets Between Decidability and Undecidability. In K. Larsen, S. Skyum, and G. Winskel, editors, *Proceedings of the*

25th International Colloquium on Automata, Languages and Programming, volume 1443 of *Lecture Notes in Computer Science*, pages 103–115, Aalborg, Denmark, July 1998. Springer-Verlag.
25. C. Dufourd, P. Jančar, and Ph. Schnoebelen. Boundedness of Reset P/T Nets. In J. Wiedermann, P. van Emde Boas, and M. Nielsen, editors, *Lectures on Concurrency and Petri Nets*, volume 1644 of *Lecture Notes in Computer Science*, pages 301–310, Prague, Czech Republic, July 1999. Springer-Verlag.
26. M. Dumas, W.M.P. van der Aalst, and A.H.M. ter Hofstede. *Process-Aware Information Systems: Bridging People and Software through Process Technology*. Wiley & Sons, 2005.
27. A. Ehrenfeucht and G. Rozenberg. Partial (Set) 2-Structures - Part 1 and Part 2. *Acta Informatica*, 27(4):315–368, 1989.
28. H. Ehrig, H.J. Kreowski, U. Montanari, and G. Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 3: Concurrency, Parallelism, and Distribution*. World Scientific, 1999.
29. A. Finkel and Ph. Schnoebelen. Well-structured Transition Systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, April 2001.
30. D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3:119–153, 1995.
31. R. Heckel. Graph Transformation in a Nutshell. *Electronic Notes in Theoretical Computer Science*, 148(1):187–198, 2006.
32. K.M. van Hee, N. Sidorova, and M. Voorhoeve. Soundness and Separability of Workflow Nets in the Stepwise Refinement Approach. In W.M.P. van der Aalst and E. Best, editors, *Application and Theory of Petri Nets 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 335–354. Springer-Verlag, Berlin, 2003.
33. K.M. van Hee, N. Sidorova, and M. Voorhoeve. Generalised Soundness of Workflow Nets Is Decidable. In J. Cortadella and W. Reisig, editors, *Application and Theory of Petri Nets 2004*, volume 3099 of *Lecture Notes in Computer Science*, pages 197–215. Springer-Verlag, Berlin, 2004.
34. J. Herbst. A Machine Learning Approach to Workflow Management. In *Proceedings 11th European Conference on Machine Learning*, volume 1810 of *Lecture Notes in Computer Science*, pages 183–194. Springer-Verlag, Berlin, 2000.
35. H.J. Kreowski. A Comparison Between Petri-Nets and Graph Grammars. In H. Noltemeier, editor, *Graph Theoretic Concepts in Computer Science*, volume 100 of *Lecture Notes in Computer Science*, pages 306–317. Springer-Verlag, Berlin, 1981.
36. F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Prentice-Hall PTR, Upper Saddle River, New Jersey, USA, 1999.
37. R. Lorenz, R. Bergenthum, J. Desel, and S. Mauser. Synthesis of Petri Nets from Finite Partial Languages. In T. Basten, G. Juhás, and S.K. Shukla, editors, *International Conference on Application of Concurrency to System Design (ACSD 2007)*, pages 157–166. IEEE Computer Society, 2007.
38. R. Lorenz and G. Juhás. How to Synthesize Nets from Languages: A Survey. In S.G. Henderson, B. Biller, M. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, editors, *Proceedings of the Wintersimulation Conference (WSC 2007)*, pages 637–647. IEEE Computer Society, 2007.
39. A. Martens. Analyzing Web Service Based Business Processes. In M. Cerioli, editor, *Proceedings of the 8th International Conference on Fundamental Approaches to Software Engineering (FASE 2005)*, volume 3442 of *Lecture Notes in Computer Science*, pages 19–33. Springer-Verlag, Berlin, 2005.

40. A.K.A. de Medeiros. *Genetic Process Mining*. PhD thesis, Eindhoven University of Technology, Eindhoven, 2006.
41. J. Mendling, G. Neumann, and W.M.P. van der Aalst. Understanding the Occurrence of Errors in Process Models Based on Metrics. In F. Curbera, F. Leymann, and M. Weske, editors, *Proceedings of the OTM Conference on Cooperative information Systems (CoopIS 2007)*, volume 4803 of *Lecture Notes in Computer Science*, pages 113–130. Springer-Verlag, Berlin, 2007.
42. A. Rozinat and W.M.P. van der Aalst. Conformance Testing: Measuring the Fit and Appropriateness of Event Logs and Process Models. In C. Bussler et al., editor, *BPM 2005 Workshops (Workshop on Business Process Intelligence)*, volume 3812 of *Lecture Notes in Computer Science*, pages 163–176. Springer-Verlag, Berlin, 2006.
43. A. Rozinat and W.M.P. van der Aalst. Conformance Checking of Processes Based on Monitoring Real Behavior. *Information Systems*, 33(1):64–95, 2008.
44. V. Rubin, C.W. Günther, W.M.P. van der Aalst, E. Kindler, B.F. van Dongen, and W. Schäfer. Process Mining Framework for Software Processes. In Q. Wang, D. Pfahl, and D.M. Raffo, editors, *International Conference on Software Process, Software Process Dynamics and Agility (ICSP 2007)*, volume 4470 of *Lecture Notes in Computer Science*, pages 169–181. Springer-Verlag, Berlin, 2007.
45. J.M.E.M. van der Werf, B.F. van Dongen, C.A.J. Hurkens, , and A. Serebrenik. Process Discovery using Integer Linear Programming. Computer Science Report (08-04), Eindhoven University of Technology, Eindhoven, The Netherlands, 2008.
46. H.M.W. Verbeek and W.M.P. van der Aalst. Analyzing BPEL Processes using Petri Nets. In D. Marinescu, editor, *Proceedings of the Second International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management*, pages 59–78. Florida International University, Miami, Florida, USA, 2005.
47. H.M.W. Verbeek, W.M.P. van der Aalst, and A.H.M. ter Hofstede. Verifying Workflows with Cancellation Regions and OR-joins: An Approach Based on Relaxed Soundness and Invariants. *The Computer Journal*, 50(3):294–314, 2007.
48. H.M.W. Verbeek, M.T. Wynn, W.M.P. van der Aalst, and A.H.M. ter Hofstede. Reduction Rules for Reset/Inhibitor Nets. BPM Center Report BPM-07-13, BPMcenter.org, 2007.
49. A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering Workflow Models from Event-Based Data using Little Thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003.
50. M. Weske. *Business Process Management: Concepts, Languages, Architectures* . Springer-Verlag, Berlin, 2007.
51. M.T. Wynn, W.M.P. van der Aalst, A.H.M. ter Hofstede, and D. Edmond. Verifying Workflows with Cancellation Regions and OR-joins: An Approach Based on Reset Nets and Reachability Analysis. In S. Dustdar, J.L. Faideiro, and A. Sheth, editors, *International Conference on Business Process Management (BPM 2006)*, volume 4102 of *Lecture Notes in Computer Science*, pages 389–394. Springer-Verlag, Berlin, 2006.