Algorithms and Tools for Petri Nets - Proceedings of the Workshop AWPN 2017

Kindler, Ekkart; Bergenthum, Robin

Publication date:
2017

Document Version
Publisher's PDF, also known as Version of record

Technical University of Denmark

Algorithms and Tools for Petri Nets

Proceedings of the Workshop AWPN 2017, Kgs. Lyngby, Denmark

October 19–20, 2017

Robin Bergenthum and Ekkart Kindler (Eds.)

DTU Compute Technical Report 2017-06

Technical University of Denmark
Department of Applied Mathematics and Computer Science (DTU Compute)
Richard Petersens Plads, Building 324
DK-2800 Kgs. Lyngby, Denmark
Phone: +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

DTU Compute Technical Reports: ISSN 1601-2321

Contents

Preface

The Algorithms and Tools for Petri Nets (AWPN) workshop is organized by the Special Interest Group "Petri nets and related system models" of the German Gesellschaft für Informatik (GI) with the focus on algorithms and tools for Petri nets.

AWPN 2017 took place at the Technical University of Denmark in Lyngby near Copenhagen on October 19-20. The emphasis of the meeting was on the exchange of experiences and discussions.

Papers related to theoretical issues for analysis and simulation of Petri nets and on experiences with the implementation of visualization, analysis and simulation tools were presented at AWPN 2017.

Papers did not undergo a detailed reviewing process, but were inspected for relevance with respect to the topics of AWPN 2017. Ten papers were accepted for the workshop. Overall, the quality of the submitted papers was very good and all submissions matched the workshop goals very well. We thank the authors and the presenters for their contributions.

Enjoy reading the proceedings!

Robin Bergenthum and Ekkart Kindler
October 2017

Towards Compact Regions for Labeled Prime Event Structures

Robin Bergenthum[1], Johannes Metzger[2], Lev Sorokin[2], and Robert Lorenz[2]

[1] FernUniversität in Hagen, Germany
robin.bergenthum@fernuni-hagen.de
[2] University of Augsburg, Germany
first.lastname@informatik.uni-augsburg.de

Abstract. In this paper, we use the so-called synthesis based modeling approach to synthesize a Petri net from specified behavior. At first, we model a set of single executions of a process by means of a very intuitive modeling language. In a second step, we use a synthesis algorithm to automatically produce a related Petri net model.
Taking a look at the literature, compact regions define the state of the art algorithm to synthesize a Petri net from a set of executions given by a partial language. The partial language is modeled by means of a set of labeled Hasse diagrams. In this paper, we discuss the problem of lifting the notion of compact regions from Hasse diagrams to prime event structures.

1 Introduction

We model distributed systems by means of Petri nets [1, 6, 13, 14]. Petri nets have an intuitive graphical representation, formal semantics, and are able to express concurrency among the occurrence of actions. Furthermore, there is a rich body of algorithms and theory checking structural and behavioral properties of Petri net. However, modeling a Petri net form scratch is a costly and error-prone task [1, 12].

Fortunately, whenever modeling a system, there are often some associated descriptions or even specifications of the desired processes. There may be log-files of recorded behavior, example runs, or product specifications describing use cases. Such specifications can be formalized by a set of Hasse diagrams [3]. The main idea of a synthesis based modeling approach is to input a set of Hasse diagrams and get the related Petri net model for free using Petri net synthesis. The synthesis problem is to compute a process model so that: (A) the specification is a subset of the language of the generated model and (B) the generated model has minimal additional behavior. In a nutshell, it is easier to come up with a set of Hasse diagrams and synthesize a Petri net than to produce the Petri net model from scratch.

Right now, compact regions are the most efficient approach to synthesize a Petri net from a partial language [3]. In this paper, we discuss how to lift the notion of these compact regions from a set of Hasse diagrams to prime event structures. At a first glance, it seems that prime event structures are just a more compact representation of a set of Hasse diagrams. In this paper, we discuss the differences between both specification languages and the resulting implications for synthesis algorithms. Roughly speaking,

we compare compact regions on Hasse diagrams to compact regions on prime event structures.

The paper is organized as follows: Section 2 introduces Petri nets, their partial language, and the synthesis problem. In Section 3, we recall the concept of compact regions and describe the related synthesis algorithm. In Section 4, we introduce compact regions for prime event structures and state an example pinpointing the difference between both notions.

2 Preliminaries

Let f be a function and B be a subset of the domain of f. We write $f|_B$ to denote the restriction of f to B. We call a function $m : A \to \mathbb{N}$ a multiset and write $m = \sum_{a \in A} m(a) \cdot a$ to denote multiplicities of elements in m. Let $m' : A \to \mathbb{N}$ be another multiset. We write $m \geq m'$ if $\forall a \in A : m(a) \geq m'(a)$ holds. We denote the transitive closure of an acyclic and finite relation $<$ by $<^*$. We denote the skeleton of $<$ by $<^\diamond$. The skeleton of $<$ is the smallest relation $\lhd$ such that $\lhd^* = <^*$ holds. Let $(V, <)$ be some acyclic and finite graph, $(V, <^\diamond)$ is called its Hasse diagram. Furthermore, we model business processes by p/t-nets [7, 13, 14].

Definition 1. *A place/transition net (p/t-net) is a tuple (P, T, W) where P is a finite set of places, T is a finite set of transitions such that $P \cap T = \emptyset$ holds, and $W : (P \times T) \cup (T \times P) \to \mathbb{N}$ is a multiset of arcs. A marking of (P, T, W) is a multiset $m : P \to \mathbb{N}$. Let m_0 be a marking, we call the tuple $N = (P, T, W, m_0)$ a marked p/t-net and m_0 the initial marking of N.*

There is a simple firing rule for transitions of a p/t-net: let t be a transition of a marked p/t-net (P, T, W, m_0). We denote the weighted preset of t by $\circ t = \sum_{p \in P} W(p, t) \cdot p$. Likewise, we denote the weighted postset of t by $t\circ = \sum_{p \in P} W(t, p) \cdot p$. A transition t is enabled (can fire) at marking m if $m \geq \circ t$ holds. Once transition t fires, the marking changes from m to $m' = m - \circ t + t\circ$. Repeatedly processing the firing rule produces firing sequences. These firing sequences are the most basic behavioral model of Petri nets. Let N be a marked p/t-net, the set of all initially enabled firing sequences of N is the sequential language of N.

Petri nets are able to express concurrency of the occurrences of transitions. However, firing sequences are not able to capture or describe such behavior. The common behavioral model for partially ordered behavior of Petri nets are so-called process nets [11]. A process net is a Petri net modeling only one single partially ordered run of a marked p/t-net. For a formal definition of process nets we refer to [11, 14]. If we abstract from the places of a process net related to a p/t-net N, we have a set of events arranged in a partial order. Just like some valid firing sequence, this partially ordered set of events is enabled in the p/t-net. In other words, we can replay such a partial order by firing transitions of N where unordered parts of the partial order can fire concurrently. The set of labeled partial orders induced by all processes of N is the partial language of N.

Although process nets are the classical definition of the partial language of a Petri net, in [5, 4] there is an equivalent and more efficient characterization of this language,

namely compact tokenflows. We define compact tokenflows not on a labeled partial order, but on the underlying Hasse diagram.

Definition 2. *Let T be a set of labels. A labeled partial order is a triple $lpo = (V, <, l)$ where V is a finite set of events, $< \subseteq V \times V$ is a transitive and irreflexive relation, and the labeling function $l : V \to T$ assigns a label to every event. A triple $run = (V, <, l)$ is a labeled Hasse diagram if $(V, <^*, l)$ is a labeled partial order and $<^\diamond = <$ holds. Let $run = (V, <, l)$ be a labeled Hasse diagram, we define $run^* = (V, <^*, l)$.*

A Hasse diagram is in the language of a Petri net if there are valid compact tokenflows describing valid distributions of tokens along the arcs of such a diagram for every place of the net [5, 4].

Definition 3. *Let $N = (P, T, W, m_0)$ be a marked p/t-net and $run = (V, <, l)$ be a labeled Hasse diagram such that $l(V) \subseteq T$ holds. A compact tokenflow is a function $x : (V \cup <) \to \mathbb{N}$. x is compact valid for $p \in P$ iff the following conditions hold:*

(i) $\forall\, v \in V$: $x(v) + \sum_{v' <^\diamond v} x(v', v) \geq W(p, l(v))$,

(ii) $\forall\, v \in V$: $\sum_{v <^\diamond v'} x(v, v') \leq x(v) + \sum_{v' <^\diamond v} x(v', v) - W(p, l(v)) + W(l(v), p)$,

(iii) $\sum_{v \in V} x(v) \leq m_0(p)$.

run is compact valid for N iff there is a compact valid tokenflow for every $p \in P$.

The language of a marked p/t-net N is well-defined by the set of compact valid labeled Hasse diagrams [4, 5]. We write $L(N) = \{run^* |\ run \text{ is compact valid for } N\}$.

As we already pointed out in the introduction, our goal is to synthesize a p/t-net from a specification describing the behavior of a system.

Definition 4. *A finite set of labeled Hasse diagrams is a specification. Let N be a marked p/t-net and $S = \{run_1, \ldots, run_n\}$ be a specification. We write $S \subseteq L(N)$ iff $\{run_1^*, \ldots, run_n^*\} \subseteq L(N)$ holds.*

Finally, we are able to define the synthesis problem. The synthesis problem is to construct a p/t-net such that its behavior matches a specification. If there is no such p/t-net, we construct a p/t-net such that its behavior includes the specification and has minimal additional behavior.

Definition 5. *Let S be a specification, the synthesis problem is to compute a marked p/t-net N such that the following conditions hold: $S \subseteq L(N)$ and for all marked p/t-nets N' : $L(N) \backslash L(N') \neq \emptyset \Longrightarrow S \not\subseteq L(N')$.*

In this paper, we lift the notion of compact regions to labeled prime event structures. So instead of considering a specification as input for the synthesis problem, we consider a labeled prime event structure. The main idea is that if Hasse diagrams of a specification share common prefixes, these prefixes can be glued together to come up with a more compact representation of the same set of partial orders. To keep track of the shared and non-shared parts of sets of events of a prime event structure, every such structure has a so-called set of consistency sets.

Definition 6. *Let T be a set of labels. We define a labeled prime event structure as tuple $pes = (V, <, l, \Gamma)$ where $(V, <, l)$ is a labeled Hasse diagram and $\Gamma = \{C_1, \ldots, C_n\}$ is a set of subsets of V satisfying:*

(I) $\bigcup_{C \in \Gamma} C = V$ *and*

(II) $\forall\, C \in \Gamma, v \in C, v' \in V : (v' < v) \implies (v' \in C)$.

Let $v, v' \in V$ be two events, we write $v \# v'$ iff there is no $C \in \Gamma$ so that $\{v, v'\} \subseteq C$ holds. Every $C \in \Gamma$ is called a consistency set of pes.

If we model a specification, we can use a prime event structure instead of a set of Hasse diagrams. Roughly speaking, every consistency set of a prime event structure relates to one Hasse diagram of the specification.

Definition 7. *Let $pes = (V, <, l, \Gamma)$ be a labeled prime event structure. Then the set $H(pes) = \{(C, <|_{C \times C}, l|_C) \mid C \in \Gamma\}$ is a set of Hasse diagrams. We call $H(pes)$ the specification modeled by pes.*

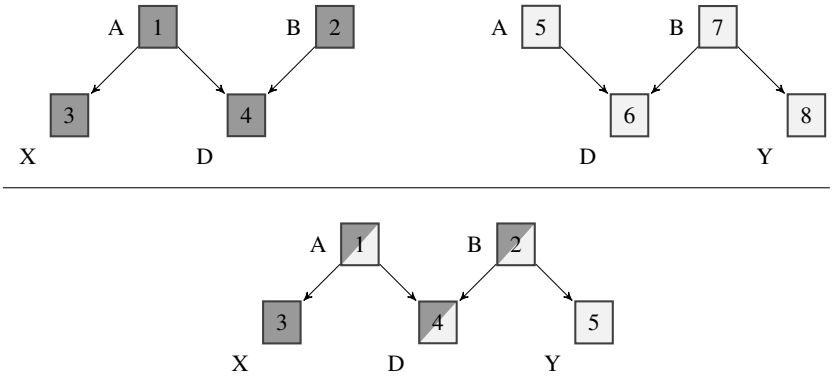


Fig. 1. Top: a specification with two Hasse diagrams. Bottom: a prime event structure modeling the same specification.

The upper part of Figure 1 depicts a specification. The lower part of Figure 1 depicts a prime event structure modeling the specification above. Different consistency sets of this prime event structure are indicated by different shades of gray. Both parts of the figure relate to the same partial language. Obviously, the number of nodes and arcs of the prime event structure is smaller than the number of nodes of the specification. We conclude this section by stating the synthesis problem for prime event structures.

Definition 8. *Let pes be a labeled prime event structure. The synthesis problem is to compute a marked p/t-net N such that the following conditions hold: $H(pes) \subseteq L(N)$ and for all marked p/t-nets N' : $L(N) \backslash L(N') \neq \emptyset \implies H(pes) \not\subseteq L(N')$.*

3 Compact Regions for Hasse Diagrams

The algorithm presented in this section is based on the theory of regions [9] and was introduced in [3]. The first step of every region-based approach is to construct a transition

for every label of the specification to get an initial p/t-net without places. The language of this net includes arbitrary behavior because all transitions have an empty preset and can fire in any order. Obviously, we need to add places and arcs to restrict the behavior of this initial net. To solve the synthesis problem, we are only allowed to add places and connected arcs that do not inhibit our specification.

Definition 9. *Let S be a specification and $N = (P, T, W, m_0)$ be a marked p/t-net. A place $p \in P$ is called feasible for S iff $S \subseteq L((\{p\}, T, W|_{(\{p\}\times T)\cup(T\times\{p\})}, m_0(p)))$ holds. Let S be a specification and $N = (\{p\}, T, W, m_0)$ be a marked one-place p/t-net. We call N feasible for S iff p is feasible for S.*

If we are able to identify feasible places, we can add these to our initially placeless p/t-net. These places restrict the behavior, yet such a net will still be able to execute all the labeled Hasse diagrams of the specification.

Remark 1. *Let S be a specification and let a set of p/t-nets $\{(\{p_1\}, T, W_1, m_1), \ldots, (\{p_n\}, T, W_n, m_n)\}$ be feasible for S. Let $N = (\bigcup_i \{p_i\}, T, \sum_i W_i, \sum_i m_i)$ be the union of all feasible nets, every place of N is feasible and $S \subseteq L(N)$ holds.*

Theoretically, we could restrict the behavior of the initial p/t-net by adding the set of all feasible places. This would guarantee that the behavior of the net cannot be restricted further without excluding some executions of the specification. This is a fundamental theorem of region theory (see for example [2]). Practically, we need to construct a finite p/t-net with the same behavior as the union-of-all-feasible-places p/t-net. We use a technique of so-called wrong continuations. Roughly speaking, the set of wrong continuations is the border between the specified and all other behaviors. The set of wrong continuations is finite as long as the specification is finite as well. For each wrong continuation, we add one feasible place, thus excluding the wrong continuation from the language of the constructed net. The resulting finite p/t-net solves the synthesis problem.

The only puzzle piece that is missing, is a characterization of the set of all feasible places. We get such a characterization by taking advantage of compact tokenflows to define the notion of compact regions [3] for Hasse diagrams.

Definition 10. *Let $S = \{(V_1, <_1, l_1), \ldots, (V_n, <_n, l_n)\}$ be a specification, T be its set of labels, and p be a place. The set of events with an empty prefix in $(V_i, <_i, l_i)$ is denoted by V_i'. A function $r : (\bigcup_i (V_i' \cup <_i) \cup (T \times \{p\}) \cup (\{p\} \times T) \cup \{p\}) \to \mathbb{N}$ is a compact region for S iff $\forall i \in \{1, ..., n\} : r|_{\{V_i' \cup <_i\}}$ is compact valid for p in $(\{p\}, T, r|_{(T\times\{p\})\cup(\{p\}\times T)}, r(p))$.*

Every region is kind of a vector of numbers respecting the conditions (i), (ii), and (iii) of Definition 3. With this in mind, we are able to express all feasible p/t-nets by a single inequality system. Again, in this system, there is an unknown for every element in the domain of a compact region, i.e. one unknown for every event with an empty prefix, another unknown for every arc, two unknowns for every label, and a single unknown for the initial marking. The inequality system is built from the inequalities defined in Definition 3. According to (i) and (ii), there are two inequalities for every event of the specification. According to (iii), there is another inequality for every labeled Hasse

diagram. The set of positive integer solutions of this inequality system is the set of all feasible nets. We call this inequality system the compact region inequality system. Every solution of this system defines one feasible place [3].

Theorem 1. *Let S be a specification and T be its set of labels. Every compact region r for S defines a feasible p/t-net $N_r = (\{p\}, T, W, m_0)$ and vice versa.*

Finally, we sum up the complete synthesis algorithm using compact regions, input is a set of labeled Hasse diagrams. At the beginning, we construct a Petri net starting with an empty set of places and a transition for every label. We then set up the compact region inequality system and the set of wrong continuations. For every wrong continuation c, we check if it is still executable in the net constructed so far. If it is executable, we need to exclude the wrong continuation from the behavior of the net. This is done by adding a feasible place, i.e. a compact region. We encode the non-executability of c in an additional inequality for the compact region inequality system. Every solution of this extended system is a region and excludes c. If this system has a solution, we add the related one-place net to our initially constructed set of transitions. If the extended compact region inequality system has no solution, the wrong continuation c cannot be excluded. We assure that the constructed net is a best approximation by expanding the set of wrong continuations accordingly.

4 Compact Regions for Prime Event Structures

In this section, we apply compact tokenflows to prime event structures. The goal is to reduce the number of events and arcs of a specification by modeling a specification in terms of a prime event structure. Less arcs and nodes lead to less variables in the compact region inequality system and thus to faster synthesis results. We will show that, although every compact region of a prime event structure defines a feasible place, the opposite direction does not hold. First, we lift the notion of compact tokenflows to prime event structures.

Definition 11. *Let $N = (P, T, W, m_0)$ be a marked p/t-net and $pes = (V, <, l, \Gamma)$ be a labeled prime event structure such that $l(V) \subseteq T$ holds. A compact tokenflow is a function $x : (V \cup <) \to \mathbb{N}$.*
x is compact valid for $p \in P$ iff $x|_{(C \cup <|_{(C \times C)})}$ is compact valid for $(C, <|_{(C \times C)}, l|_C)$ for all $C \in \Gamma$. A prime event structure pes is compact valid for N iff there is a compact valid tokenflow for every $p \in P$.

If we consider a prime event structure with only one consistency set, of course, both notions of compact tokenflows coincide. The difference between both notions becomes more clear if we think of two consistency sets with a common prefix. Both projections of the tokenflow to each consistency set must be compact valid. Furthermore, the distribution of tokens on the common prefix is always the same for both consistency sets. On the one hand, this reduces the amount of variables of a compact tokenflow, on the other hand, it might be possible that there are two valid compact tokenflows for two separated Hasse diagrams, but not a single compact valid tokenflow on the merged prime event structure.

Nevertheless, following the ideas of the previous section, we define compact regions for prime event structures.

Definition 12. *Let $pes = (V, <, l, \Gamma)$ be a labeled prime event structure, T be its set of labels, and p be a place. The set of events with an empty prefix in pes is denoted by V'. A function $r : (V' \cup <) \cup (T \times \{p\}) \cup (\{p\} \times T) \cup \{p\}) \to \mathbb{N}$ is a compact region for pes iff $r|_{(V' \cup <)}$ is compact valid for p in $(\{p\}, T, r|_{(T \times \{p\}) \cup (\{p\} \times T)}, r(p))$.*

Finally, we state the main contribution of this paper.

Theorem 2. *Let pes be a labeled prime event structure and T be its set of labels. Every compact region r for pes defines a feasible p/t-net $N_r = (\{p\}, T, W, m_0)$ but the opposite direction does not hold.*

Proof. Let r be a compact region. Obviously, $r|_{\{V' \cup <\}}$ is a valid compact tokenflow of p in $N_r = (\{p\}, T, r|_{(T \times \{p\}) \cup (\{p\} \times T)}, r(p))$ so that $H(pes) \subseteq L(N_r)$ holds. N_r is feasible for $H(pes)$.

Let N be the Petri net depicted in Figure 3, let p be the place of N, and S be the specification depicted in Figure 1. The place p of N is feasible for S, because there is a valid compact tokenflow (see Figure 2). Let pes be the prime event structure depicted in Figure 1. Obviously, $H(pes) = S$ holds. We prove by contradiction and assume there is a compact valid tokenflow x for p in pes. Obviously, $x(v_1, v_3) \geq W(p, l(v_3)) = 1$ and $x(v_2, v_5) \geq W(p, l(v_5)) = 1$ holds to satisfy condition (i) of Definition 3. Therefore, $x(v_1, v_4) = 0$ and $x(v_2, v_4) = 0$ hold to satisfy condition (ii) for both events v_1, v_2. So, $0 = x(v_1, v_4) + x(v_2, v_4) < W(p, l(v_4)) = 1$ holds which contradicts (i). Altogether, there is no valid compact tokenflow in pes for p and no compact region defining p. □

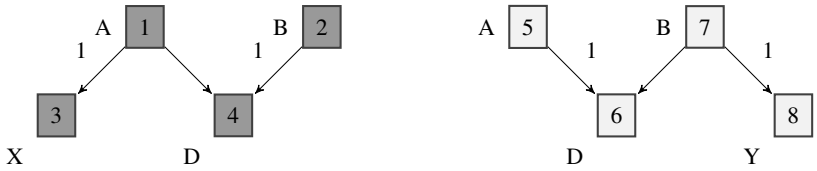


Fig. 2. Two Hasse diagrams with compact valid tokenflows. A zero-valued tokenflow at arcs is not depicted.

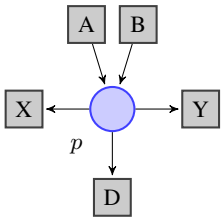


Fig. 3. One-place p/t-net N_r.

5 Conclusion

In this paper, we presented compact tokenflows and compact regions for prime event structures. We proved that every compact region defines a feasible p/t-net, but the opposite direction does not hold. Using prime event structures, we sketched a synthesis algorithm using less variables and constraints compared to the former approach using Hasse diagrams.

It remains to prove, that there is a subclass of all prime event structures for which both directions of Theorem 2 hold. For this subclass the presented algorithms will be much faster than the algorithm using Hasse diagrams.

References

[1] van der Aalst, W. M. P.; van Dongen, B. F.: *Discovering Petri Nets from Event Logs*. ToPNoC VII, LNCS 7480, Springer, 2013, 372–422.

[2] Badouel, E.; Bernardinello, L.; Darondeau, P.: *Petri Net Synthesis*. Texts in Theoretical Computer Science, Springer, 2015.

[3] Bergenthum, R.: *Synthesizing Petri Nets from Hasse Diagrams*. Business Process Management 2017. LNCS 10445. Springer, 22-39.

[4] Bergenthum, R.; Lorenz, R.: *Verification of Scenarios in Petri Nets Using Compact Tokenflows*. Fundamenta Informaticae 137, IOS Press, 2015, 117–142.

[5] Bergenthum, R.: *Faster Verification of Partially Ordered Runs in Petri Nets Using Compact Tokenflows*. Petri Nets 2013, LNCS 7927, Springer, 2013, 330–348.

[6] Desel, J.; Juhás, G.: *"What is a Petri Net?"*. Unifying Petri Nets, Advances in Petri Nets, LNCS 2128, Springer, 2001, 1–25.

[7] Desel, J.; Reisig, W.: *Place/Transition Petri Nets*. Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, LNCS 1491, Springer, 1998, 122–173.

[8] Dumas, M.; Garca-Bauelos, L.: *Process Mining Reloaded: Event Structures as a Unified Representation of Process Models and Event Logs*. Petri Nets 2015, LNCS 9115, Springer, 2015, 33–48.

[9] Ehrenfeucht, A.; Rozenberg, G.: *Partial (Set) 2-Structures. Part I: Basic Notions and the Representation Problem, Part II: State Spaces of Concurrent Systems*. Acta Inf. 27(4), 1990, 315–368.

[10] Grabowski, J.: *On partial languages*. Fundamenta Informaticae 4, IOS Press, 1981, 427–498.

[11] Goltz, U.; Reisig, W.: *Processes of Place/Transition-Nets*. Automata Languages and Programming 154, Springer, 1983, 264–277.

[12] Mayr, H. C.; Kop, C.; Esberger, D.: *Business Process Modeling and Requirements Modeling*. ICDS 2007, Computer Society, IEEE, 2007, 8-14.

[13] Peterson, J.L.: *Petri Net Theory and the Modeling of Systems*. Prentice-Hall (Englewood Cliffs), 1981.

[14] Reisig, W.: *Understanding Petri Nets - Modeling Techniques, Analysis Methods, Case Studies*. Springer, 2013.

Verification of Nested Petri Nets Using an Unfolding Approach

Vera O. Ermakova, and Irina A. Lomazova

National Research University Higher School of Economics,
Myasnitskaya ul. 20, 101000 Moscow, Russia
voermakova@edu.hse.ru ilomazova@hse.ru

Abstract. Nested Petri nets (NP-nets) is an extension of the Petri nets formalism within the "nets-within-nets" approach, allowing to model systems of interacting dynamic agents in a natural way. One of the main problems in verifying of such systems is the State Explosion Problem. To tackle this problem for highly concurrent systems the unfolding method has proved to be very helpful. In this paper we continue our research on applying unfoldings for NP-nets verification and compare unfolding of NP-net translated into classical Petri net with direct component-wise unfolding.

Keywords: Multi-agent systems, verification, Petri nets, nested Petri nets, unfoldings.

1 Introduction

Multi-agent systems have been studied explicitly for the last decades and can be regarded as one of the most advanced research and development area in computer science today. They are used in various practical fields and areas, such as artificial intelligence, cloud services, grid systems, augmented reality systems with interactive environment objects, information gathering, mobile agent cooperation, sensor information and communication.

Petri nets have been proved to be one of the best formalisms for modeling and analysis of distributed systems. However, due to the flat structure of classical Petri nets, they are not so good for modeling complex multi-agent systems. For such systems a special extension of Petri nets, called nested Petri nets [1], can be used. Nested Petri nets follow 'nets-within-nets' approach [2] and naturally represent multi-agent systems structure, because tokens in the main system net are Petri nets themselves, and can have their own behavior.

To check nested Petri net model properties one of the most popular verification method, *model checking*, could be used. The basic idea of model checking is to build a reachability (transition) graph and check properties on this graph. However, there is a crucial problem for verification of highly concurrent systems using model checking approach — a large number of interleavings of concurrent processes. This leads to the so-called state-space explosion problem.

To tackle this problem unfolding theory [3,4] was introduced. In [5] applicability of unfoldings for nested Petri nets was studied and the method for constructing unfoldings for safe conservative nested Petri nets was proposed. It was proven there, that unfoldings for nested Petri nets satisfy the unfoldings fundamental property, and thus can be used for verification of conservative nested Petri nets similar to the classical unfoldings methods.

Classical unfoldings are defined for P/T nets, but in this paper we deal with a restricted subclass of nested Petri nets — *conservative safe nested Petri nets*. This means that net tokens, representing agents, cannot be destroyed or created, but can change their location in the system net and can change their inner states. Thus, the number of agents is constant and each agent can be identified. It was shown in [5] that for conservative safe nested Petri nets unfoldings can be constructed in a component-wise manner, what makes practical verification of such models feasible.

However, safe conservative nested Petri nets are bounded. So, for such net it is possible to construct a P/T net with equivalent behavior, for which the standard unfolding techniques can be applied. Then the question is whether direct unfolding proposed in [5] is really better than constructing unfoldings via translation of nested Petri nets into safe P/T nets in terms of time complexity.

In this paper we study this question. For that we develop an algorithm for translating a safe conservative NP-net into a behaviorally equivalent P/T net. We prove that the reachability graphs of a source NP-net and the obtained P/T net are isomorphic, and hence both unfolding methods give the same (up to isomorphism) result. From general considerations translating an NP-net into a P/T net and then constructing unfoldings will be more time consuming, than constructing unfoldings directly. To check whether this time gap reveals itself in practice we implement all the algorithms and compare both methods experimentally.

2 Related Work

Nested Petri nets (NP-nets) are widely used in modeling of distributed systems [6,7,8], serial or reconfigurable systems [9,10], protocol verification [11], coordination of sensor networks with mobile agents [12], innovative space system architectures [13], grid computing [14].

Several methods for NP-nets behavioral analysis were proposed in the literature, among them compositional methods for checking boundedness and liveness for nested Petri nets [15], translation of NP-nets into Colored Petri nets in order to verify them with CPNtools [16], verification of a subclass of recursive NP-nets with SPIN [17].

Unfolding approach and state-space explosion problem are explicitly studied in the literature. The original development in *unfoldings* (of P/T-nets) is due to [18]. McMillan [3] was the first to use unfoldings for verification. He introduced the concept of *complete finite prefixes* of unfoldings, and demonstrated the applicability of this approach to the verification of asynchronous circuits.

The original McMillan's algorithm was used to solve the *executability* problem — to check whether a given transition can fire in the net. This algorithm can be used also for checking deadlock-freedom and for solving some other problems. Later, numerous improvements to the algorithm have been proposed ([19,20] to name a few); and the approach has been applied to high-level Petri nets [21], process algebras [22] and M-nets [21].

The general method for truncating unfoldings, which abstracts from the information one wants to preserve in the finite prefix of the unfolding, was proposed in [23,24]. This method is based on the notion of a *cutting context*. We use this approach for defining branching processes and unfoldings of conservative nested Petri nets.

3 Two ways of nested Petri net unfolding

3.1 Translation of Nested Petri Nets into Classical Petri Nets

As reachability graph of the unfolding is isomorphic to the reachability graph of the P/T-net, unfoldings can be used in verification. Since safe conservative nested Petri nets have finite number of states, it will be apparent to assume, that they can be translated into classical Petri nets and then can be unfolded according to the classical unfolding rules for further verification. The problem of conservative safe nested Petri nets to classical Petri nets has not been studied before.

To make a correct translation we have to set a number of requirements for a translation. The main goal for building a model is the possibility to make a simulation. Simulation implies behavioral equivalence: a possibility to repeat all possible moves of one model on another model. Behavioral equivalence is guaranteed by establishing strong bisimulation equivalence between states of two models. The second requirement is about constructing a reachability graph. It means that we need exact correspondence between nodes (states) of our model. If these two requirements are met, we can build a translation algorithm which allows us to use target model having the same behavioral properties like original for verification and analysis.

3.2 Comparing two ways of nested Petri net unfolding

As each conservative safe NP-net can be converted into a behaviorally equivalent classical Petri net, namely their reachability graphs are isomorphic. So, to construct unfoldings for a NP-net we can either translate it into a P/T net and then apply the classical P/T net unfolding procedure, or directly construct NP-nets unfoldings, as it is described in [5].

The fundamental property of unfoldings states that the reachability graph of the unfolding is isomorphic to the reachability graph of the initial net. Since the fundamental property holds both for Petri net unfoldings and for NP-net unfoldings, we can immediately conclude that both approaches give the same (up to isomorphism) result.

The difference is in the complexity of these two solutions. It is easy to see, that when there are several net tokens of the same type in the initial marking, the translation leads to a significant net growth. Thus e.g. for a system net transition t with n input places of the same type and k tokens of this type in the initial marking we are to construct k^n copies of this transition in the target P/T net, corresponding to different bindings for t-firings. And it is rather clear, that we cannot avoid this, since we are to distinguish markings of net tokens residing in different places, and hence to construct a separate P/T net transition for each mode of a system net transition firing.

To check the advantage of the direct unfolding method w.r.t. time complexity for concrete examples we've developed a software application which allows:

1. translation of a conservative safe nested Petri net into a P/T net and then building an unfolding for it;
2. building an unfolding directly for a nested Petri net.

We expected that a large number of net tokens cause significant net growth during translation. To get representative results, we conducted experiments on nets having similar structure, but different number of element nets with different types.

Even experiments with rather small models confirmed our assumptions. If we are dealing with a system, which consists of a large number of net tokens and incoming arcs, after applying translation of a nested Petri net into a P/T net the net graph will increase strongly. Since we do not know in advance, which transitions will be used in the unfolding, we should create an intermediate graph with a lot of transitions unnecessary for unfolding.

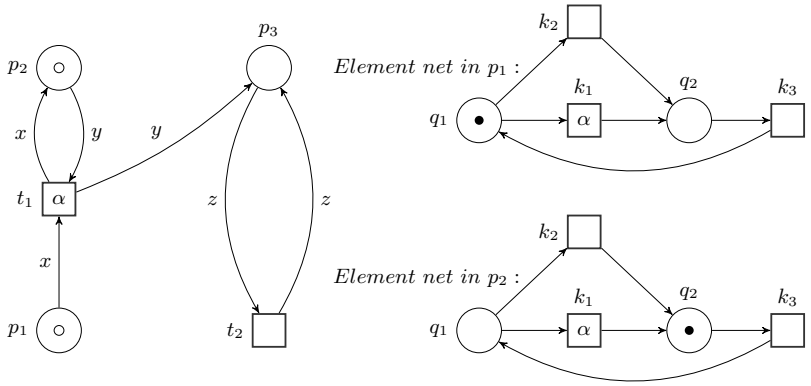


Fig. 1. NP-net NP_1

So, we conducted experiments on nets having similar structure, but different number of element nets with different types. We've done a series of experiments with rather small models, which confirm our assumptions. Thus for our example net NP_1 we've got 0.38 ms. for the direct unfolding, and 0.54 ms. for unfolding via the translation into a P/T net. So, even in the case of two net tokens we get a noticeable difference in time.

To get representative experiment results we have done more experiments with larger models having common patterns, for example: cyclic element net, net with a few branching processes, with few element nets, with a deadlock. In all mentioned cases direct unfolding time was either less or almost equal to translation and unfolding, depending on structure of the net.

We plan to conduct more experiments to get statistically full results.

4 Conclusion and Future Work

In this paper we've proposed and compared two ways of unfolding for safe conservative nested Petri nets. The first method is based on equivalent translation of NP-nets into safe P/T nets and then applying standard unfolding procedure described in the literature. The second method is a direct unfolding, proposed and justified earlier in [5].

For that we've developed and justified an algorithm for translation of a safe conservative NP-net into an equivalent P/T net. Direct analysis of the algorithm complexity allows us to conclude that the direct unfolding has a distinct advantage in time complexity. To check this advantage with practical examples we've implemented the algorithms for translation and unfolding. Experiments on small nets have demonstrated the anticipated benefits of direct unfolding.

So having these results in mind, we plan:

1. to develop unfolding-based specific methods for verification of concrete crucial behavioral properties for nested Petri nets;
2. to extend the developed approach for more wide classes of nested Petri nets;
3. to develop a software tool (or a plug-in for an existing software) that implements the above methods.

References

1. Lomazova, I.A.: Nested Petri nets—a formalism for specification and verification of multi-agent distributed systems. Fundamenta Informaticae **43**(1) (2000) 195–214
2. Valk, R.: Object petri nets. In: Lectures on Concurrency and Petri Nets. Springer (2004) 819–848
3. McMillan, K.L.: Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In: Computer Aided Verification, Springer (1992) 164–177
4. Nielsen, M., Plotkin, G., Winskel, G.: Petri nets, event structures and domains, part i. Theoretical Computer Science **13**(1) (1981) 85–108

5. Frumin, D., Lomazova, I.A.: Branching processes of conservative nested Petri nets. In: VPT@ CAV. (2014) 19–35
6. Lomazova, I.A., van Hee, K.M., Oanea, O., Serebrenik, A., Sidorova, N., Voorhoeve, M.: Nested nets for adaptive systems. Application and Theory of Petri Nets and Other Models of Concurrency, LNCS (2006) 241–260
7. Lomazova, I.A.: Modeling dynamic objects in distributed systems with nested petri nets. Fundamenta Informaticae **51**(1-2) (2002) 121–133
8. Lomazova, I.A.: Nested petri nets for adaptive process modeling. In: Pillars of computer science. Springer (2008) 460–474
9. López-Mellado, E., Villanueva-Paredes, N., Almeyda-Canepa, H.: Modelling of batch production systems using Petri nets with dynamic tokens. Mathematics and Computers in Simulation **67**(6) (2005) 541–558
10. Zhang, L., Rodrigues, B.: Nested coloured timed Petri nets for production configuration of product families. International journal of production research **48**(6) (2010) 1805–1833
11. Venero, M.L.F., da Silva, F.S.C.: Modeling and simulating interaction protocols using nested Petri nets. In: Software Engineering and Formal Methods. Springer (2013) 135–150
12. Chang, L., He, X., Lian, J., Shatz, S.: Applying a nested Petri net modeling paradigm to coordination of sensor networks with mobile agents. In: Proc. of Workshop on Petri Nets and Distributed Systems, Xian, China. (2008) 132–145
13. Cristini, F., Tessier, C.: Nets-within-nets to model innovative space system architectures. In: Application and Theory of Petri Nets. Springer (2012) 348–367
14. Mascheroni, M., Farina, F.: Nets-within-nets paradigm and grid computing. In: Transactions on Petri Nets and Other Models of Concurrency V. Springer (2012) 201–220
15. Dworzański, L.W., Lomazova, I.A.: On compositionality of boundedness and liveness for nested Petri nets. Fundamenta Informaticae **120**(3-4) (2012) 275–293
16. Dworzański, L., Lomazova, I.A.: Cpn tools-assisted simulation and verification of nested petri nets. Automatic Control and Computer Sciences **47**(7) (2013) 393–402
17. Venero, M.L.F.: Verifying cross-organizational workflows over multi-agent based environments. In: Enterprise and Organizational Modeling and Simulation. Springer (2014) 38–58
18. Winskel, G.: Event structures. Springer (1986)
19. Bonet, B., Haslum, P., Hickmott, S., Thiébaux, S.: Directed unfolding of petri nets. In: Transactions on Petri Nets and Other Models of Concurrency I. Springer (2008) 172–198
20. McMillan, K.L.: A technique of state space search based on unfolding. Form. Methods Syst. Des. **6**(1) (1995) 45–65
21. Khomenko, V., Koutny, M.: Branching processes of high-level Petri nets. In Garavel, H., Hatcliff, J., eds.: Tools and Algorithms for the Construction and Analysis of Systems. Volume 2619 of Lecture Notes in Computer Science. Springer (2003) 458–472
22. Langerak, R., Brinksma, E.: A complete finite prefix for process algebra. In: Computer Aided Verification, Springer (1999) 184–195
23. Khomenko, V., Koutny, M., Vogler, W.: Canonical prefixes of Petri net unfoldings. Acta Informatica **40**(2) (2003) 95–118
24. Khomenko, V.: Model Checking Based on Prefixes of Petri Net Unfoldings. Ph.D. Thesis, School of Computing Science, Newcastle University (2003)

Petrixx - Petri Nets for OMILAB based on ADOxx

Mustafa Ghani[1,2]

[1] Technical University of Berlin
Department of Electrical Engineering and Computer Science
Complex and Distributed IT-Systems
[2] Humboldt-Universität zu Berlin
Faculty for Mathematics and Natural Sciences
Theory of Programming

Abstract The construction of a modeling tool is a sophisticated and time-consuming task. The Open Models Initiative Laboratory (OMILAB) is a dedicated research and experimentation space for the engineering of different modeling methods. It provides techniques and resources to support the development of a modeling method plus its construction into full-fledged modeling tools with comparatively few effort. By means of this resources a modeling tool for Petri Nets, "Petrixx", was implemented and aims to generate a symbiosis with existing modeling methods within OMILAB. Further, new algorithms can be easily implemented by any other member of OMILAB without software maintenance.

Keywords: Petrixx, Petri Nets, OMILAB, ADOxx

1 Introduction

The Open Models Initiative Laboratory (OMILAB) [1] is a modeling community which brings more than 20 modeling methods together. The majority of them aims at describing enterprise information systems or business processes which are characterized by a semi-formal [2] nature. OMILAB provides resources to develop an own modeling method and to build a modeling tool (i.e. for the created method or an existing one). One of these ressources is ADOxx, a metamodeling platform [3]. Purpose of this standardized platform is among others to enable an interplay between different modeling methods [4]. Further, to make each tool accessible for any community member they share the same technical platform (i.e. every modeling tool developed within OMILAB is based on ADOxx). Yet, there is none for Petri Nets [5] based on ADOxx. Thereby, there is no formal modeling method represented within OMILAB. This paper at hand aims at closing the gap and presents a Petri Net modeling tool based on ADOxx which is called Petrixx [3] as shown in Figure 1.

[3] http://austria.omilab.org/psm/content/petrinetstool/info

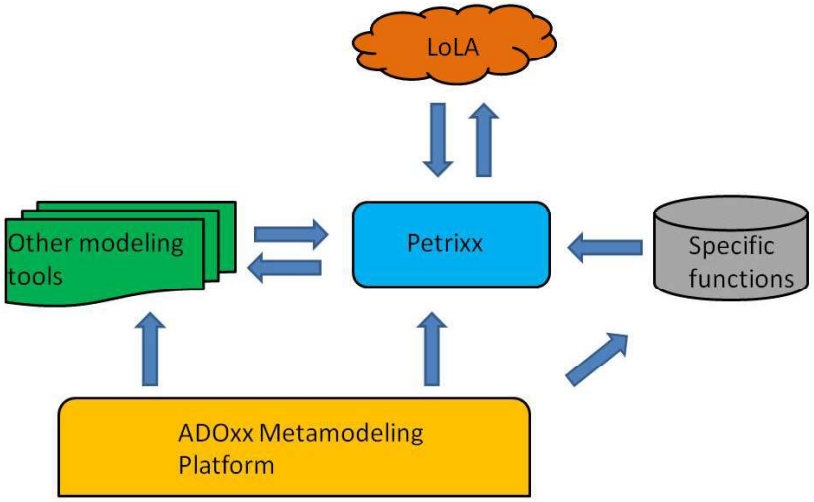


Figure 1. Petrixx in Action

It aims at generating a symbiosis with the existing modeling methods and their tools. Those simply need a function which transforms their domain-specific models [6] into Petri Nets in order to make use of this formalism. Furthermore, Petrixx is intended to make its contribution to the "Model Value" concept described in [4]. According to this idea, any process model is a knowledge base for various purposes such as providing input for simulations or to gather intelligence. To this end, the Low-Level Petri Net Analyzer (LoLA) [7] is connected to Petrixx by using a Web service. LoLA can be utilised to the highest possible extent since Petrixx offers a converter as a specific function which transforms any Petri Net to the format required by LoLA. Petrixx targets to be flexible in terms of implementing additional specific functions in the form of a script or to make adjustments to existing functions on behalf on any OMILAB member. Thus, its Metamodel which will be introduced in section 2.4 needs to be relaxed plus generalized in order to not constrain future work and at the same time precise in order to be operable. More specific functions will be introduced in section 2.1.

2 Software Design

In this work the notion of Functional and non-functional Requirements [8] are adopted.

2.1 Functional Requirements

Each change of a state in a Petri Net creates a number of algorithmic capabilities [9] thus this calls for a functional requirement. It is the ability to analyze any property of a net in a software-controlled manner. Due to the firing rule, Petri nets are preseeded for an interactive, token-based simulation. Thus, a third functional requirement can be inferred, the correct execution of the firing rule. In addition, the respective state of a Model, which can be seen by the current distribution of the tokens, needs to be generated as a graph while pushing a button and displayed. It is useful to drag and drop the graph by means of copy-and-paste for further purposes. The graph described in this work is referred to as a "state graph". This kind of determination of the respective state, in particular in the case of larger models, allows for an investigation if a particular state occurs or not. A state explosion [10] is thus prevented since not every possible state is calculated but rather every single state of interest separately. Further, a function which counts the number of any element of a Petri Net is required and called Info. As the last functional requirement, the composition operator described in [11] was chosen. The composition of Transition and Places are required with a restriction only two objects of the same kind (i.e. Transition - Transition/ Place - Place) can be merged at the same time. Structural characteristics such as the number of tokens in a place are considered while merging.

2.2 Non-Functional Requirements

Creating syntactically correct models is the right-of-way precedence for a modeling tool. Hence, one non-functional requirement is to provide model consistency. In the form of functionalities which ensures the strict observance of the Petri net rules such as its flow relation. However, that implies for this case the necessity of the corresponding Metamodel since OMILAB pursues a Metamodel approach which will be described in section 2.4 in more detail. Maintainability is the second non-functional requirement. The maintainability of a software is assessed by means of key figures [12]. In this context, a key figure is the modularization of Petrixx. Another one is the number of global variables in the source code. As thus, the request is derived from the fact that Petrixx needs to be composed of modules. This design allows the necessary flexibility. In doing so, the individual modules should contain global variables just in case.

2.3 Synergy effects of software components

Petrixx is implemented on the basis of ADOxx. Therefore, ADOxx is a component from Petrixx. LoLA is another component. Added value can be generated by means of analyzing models which are represented in a semi-formal modeling notation in a quicker, more formal and more extensive manner by transforming into Petri Nets and in combination with Petrixx. Thus, Petri Nets are accessible for OMILAB and its modeling methods for multi-purpose. Many modeling tools for Petri Nets contain LoLA and its format [13] [14]. Hence, these modeling

tools support this kind of format. In fact, Petrixx is able to transform a net into that format. According to the model value approach process models generate additional value not only by serving as a base for software development. Beyond representing complex proceedings and enabling corresponding analysis, they act as a knowledge base in terms of providing inputs for simulation or solutions for queries and can be executed automatically. Moreover, Petri Nets can be established as an exchange format for transferring models of other notations by use of Petrixx. Furthermore, the model value concept can be amplified since all models represented in the LoLA format can be integrated into OMILAB while Petrixx operates as a gatekeeper.

2.4 Metamodel of Petrixx

Working with the ADOxx platform requires a Metamodel. This needs to be an instance of the Meta-Metamodel of ADOxx in order to inherit predefined functions which are independent of a specific modeling method such as printing a created model as a pdf file [3].

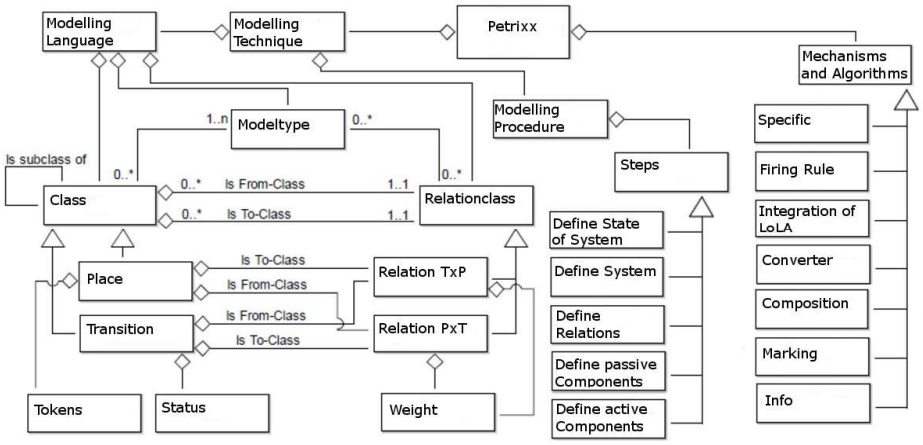


Figure 2. Metamodel of Petrixx

An inheritance from the Meta-Metamodel is possible by the two building blocks "Class" and "Relations Class" which serves as the basement. These two classes enable any end user to create objects (i.e. Transitions and Places) and connecting them according to the flow relation of Petri Nets. Additionally, each class can be enriched by specific attributes where each attribute can also be specified with constraints. For instance, as Figure 2 describes, each object of Class "Place" can have a set of "Tokens" which represents the behavior of the net by positive values including null. The relation "$T\times$ P" and "$P\times$ T" ensures the correct compliance of the flow relation whereas both classes are equipped with "Weight" as an

attribute. Class "Transition" has a "Status" which means in this work it can only be cold or hot. In comparison to proposed Metamodeling approaches [15] [16] [17] "Mechanisms and Algorithms" can be seen as an extension. Specific functions for modeling with Petri Nets can be implemented and integrated into the metamodel. Petrixx contains the firing rule, integration of LoLA for analysis purposes and a converter for transforming a given net into the required format. Furthermore, it is possible to compose exactly two objects of the same kind, see the current distribution of tokens and an overview of the actual elements of a net. Another extension is the modeling procedure which is part of the modeling technique and describes the steps in order to create a model as a Petri Net.

3 Implementation

ADOxx has its own programming language, namely Adoscript [18]. It allows to configure the Metamodel (i.e. setting constraints) and implementing any kind of Mechanisms and Algorithms as described in the previous section. Each function except Info does not automatically consider every element of a net. Instead, every element of relevance needs to be selected by the end user due to the nature of Petri Nets, consisting of subnets [19] where in some cases only a fragment of a net is taken into account. If there are not any elements selected a pop-up window will occur which asks the end user to select a (sub-)net. LoLA is integrated by a web service and a Java-wrapper. This was made by the ADOxx developer team and is partly undertaken as well as customized. The source code of Petrixx is available at its website[4] and contains two global variables within one script. Nevertheless, they influence neither the performance nor the maintainability since Petrixx is made of scripts which are running independently.

4 Conclusion

This work presents Petrixx which is a unique modeling tool for Petri Nets due to its setting. It is based on ADOxx, the central platform from OMILAB which is a modeling community with more than 20 modeling methods and tools. Petrixx provides OMILAB with Petri Nets in order to execute formal analysis. Moreover, Petrixx is an open source project and can be easily updated with new algorithms by members of OMILAB. Several case studies [5] were undertaken and document the performance capability of Petrixx.

Acknowledgments This work would not be possible without Wolfgang Reisig, Dimitris Karagiannis, Karsten Wolf, Dominik Bork, Odej Kao and many other people [your place goes here...]. The author dedicates Petrixx to his mother. Thank you!

[4] http://austria.omilab.org/psm/content/petrinetstool/download?view=download
[5] http://austria.omilab.org/psm/content/petrinetstool/info?view=home

References

1. Karagiannis, Dimitris, Robert Andrei Buchmann, Patrik Burzynski, Ulrich Reimer and Michael Walch: *Fundamental Conceptual Modeling Languages in OMiLAB.* In *Domain-Specific Conceptual Modeling*, pages 3–30. Springer, 2016.
2. Desel, Jörg: *Formaler Umgang mit semi-formalen Ablaufbeschreibungen*, pages 45–60. Vieweg+Teubner Verlag, Wiesbaden, 1999.
3. Kühn, H: *The adoxx® metamodelling platform.* In *Workshop on Methods as Plug-Ins for Meta-Modelling, Klagenfurt, Austria*, 2010.
4. Fill, Hans-Georg and Dimitris Karagiannis: *On the conceptualisation of modelling methods using the ADOxx meta modelling platform.* Enterprise Modelling and Information Systems Architectures, 8(1):4–25, 2015.
5. Reisig, Wolfgang: *Petrinetze: Modellierungstechnik, Analysemethoden, Fallstudien.* Vieweg + Teubner, Wiesbaden, 2010.
6. Karagiannis, Dimitris, Heinrich C. Mayr and John Mylopoulos: *Domain-Specific Conceptual Modeling: Concepts, Methods and Tools.* Springer Publishing Company, Incorporated, 1st edition, 2016.
7. Schmidt, Karsten: *LoLA a Low Level Petri net Analyzer.* Humbolt Universitat zu Berlin–Institut fur Informatik, 2000.
8. Chung, Lawrence, Brian A Nixon, Eric Yu and John Mylopoulos: *Non-functional requirements in software engineering*, volume 5. Springer Science & Business Media, 2012.
9. Starke, Peter H: *Analyse von Petri-netz-modellen*, volume 6. Springer, 1990.
10. Burch, Jerry R, Edmund M Clarke, Kenneth L McMillan, David L Dill and Lain-Jinn Hwang: *Symbolic model checking: 1020 states and beyond.* Information and computation, 98(2):142–170, 1992.
11. Reisig, Wolfgang: *Simple Composition of Nets.* In Franceschinis, Giuliana and Karsten Wolf (editors): *Proceedings of the 30th International Conference on Petri Nets and Other Models Of Concurrency*, volume 5606 of *Lecture Notes in Computer Science*, pages 23–42, Paris, France, June 2009. Springer-Verlag.
12. Bommer, Christoph, Markus Spindler and Volkert Barr: *Softwarewartung: Grundlagen, Management und Wartungstechniken.* BoD–Books on Demand, 2017.
13. Lohmann, Niels and Karsten Wolf: *How to implement a theory of correctness in the area of business processes and services.* In *International Conference on Business Process Management*, pages 61–77. Springer, 2010.
14. Kindler, Ekkart and Michael Weber: *The petri net kernel an infrastructure for building petri net tools.* International Journal on Software Tools for Technology Transfer (STTT), 3(4):486–497, 2001.
15. Harel, D. and B. Rumpe: *Modeling Languages: Syntax, Semantics and All That Stuff, Part I: The Basic Stuff.* Technical Report, Jerusalem, Israel, Israel, 2000.
16. Schmidt, Douglas C: *Model-driven engineering.* COMPUTER-IEEE COMPUTER SOCIETY-, 39(2):25, 2006.
17. Jeusfeld, Manfred A., Matthias Jarke and John Mylopoulos: *Metamodeling for Method Engineering.* The MIT Press, 2009.
18. Junginger, Stefan, Harald Kühn, Robert Strobl and Dimitris Karagiannis: *Ein geschäftsprozessmanagement-werkzeug der nächsten generation—ADONIS: konzeption und anwendungen.* Wirtschaftsinformatik, 42(5):392–401, 2000.
19. Reisig, Wolfgang: *Associative Composition of Reactive Systems*, 2017.

OPOA - a Petri Net Generation Algorithm for Molecular Dynamics Analysis

Anna Gogolińska[1] and Wiesław Nowak[2]

[1] Faculty of Mathematics and Computer Science, Nicolaus Copernicus University, Toruń, Poland

[2] Faculty of Physics, Astronomy and Informatics, Nicolaus Copernicus University, Toruń, Poland

anna.gogolinska@mat.umk.pl

Abstract. Molecular dynamics (MD) simulations are an issue of high importance in computational biology, chemistry and physics. They allow to study biological components like proteins or nucleic acids, observe their behavior, analyze their active sites, compare native and mutated structures. Unfortunately, MD simulations produce enormous amount of data. The main problem is an effective analysis of this structural and dynamic information. We designed the complex framework to analyze MD data using Petri nets (PNs). In this paper a part of the framework - the One Place One Atom (OPOA) algorithm will be described in more detail. The OPOA algorithm is one of designed by us algorithms, that generates Petri nets based on the MD trajectory. In OPOA one place represents one point in the space and transitions correspond to movements of atoms. Some more complex aspects of the algorithm will be also mentioned and described.

Keywords: Petri nets, Molecular Dynamics Simulations, MD analysis, PN generation

1 Introduction

Molecular dynamics (MD) simulations [6] are computer calculations of the trajectory of the motion of every atom from a given input set, for example proteins. The theoretical basis of the MD simulation is straightforward: the localization of every atom in each time step t_{i+1} is calculated by solving the Newton equation of motion using, for example, the Verlet algorithm. The force values necessary for the computations is calculated on the fly based on the positions of all interacting atoms and predefined potentials V. A set of analytical formulas together with appropriate parameters to calculate V is called a force field [5]. The results of the simulations are spatial structures of the studied system, which present time

evolution of the system. Output files from the MD simulations contain frames (snapshots) which describe localization of every atom from the system studied. Information about physical states of the system is hidden in those output files. They are called trajectories, because they represent a motion of the system in the phase space.

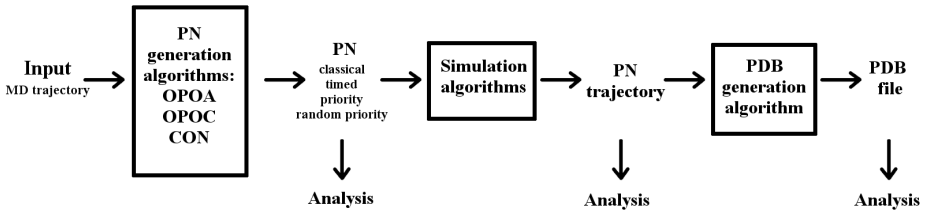


Fig. 1. The workflow scheme of the MD simulations analysis using PN formalism.

The main problem with MD simulations is an effective analysis of this structural and dynamics information. They produce enormous amount of data, easily gigabytes or even terabytes, which have to be analyzed mostly by hand. Various methods of analysis are used [2], but there is still huge demand for new ones. In our opinion, PNs have a potential to be used in MD simulations and we developed the whole framework of MD analysis with the use of PNs. The idea of the framework is presented in Figure 1.

The One Place One Atom (OPOA) algorithm is one of the PN generations algorithms, others are One Place One Conformation (OPOC) [1] and Contacts (CON) algorithms. The structure of generated PN can be analyzed or its dynamic can be study. In the next step the PN dynamic can be analyzed separately or use to generate PDB trajectory (similarly to MD simulations outputs).

2 Petri Nets

The idea of Petri net was proposed by Carl Adam Petri [4].

Definition 1. *A Petri net graph is a 4-tuple* (P, T, F, W)*, where:*

- P *is a finite set of places.*
- T *is a finite set of transitions (or actions), such that* $P \cap T = \emptyset$*.*
- F *is a set of directed arcs, satisfying:* $F \cap (P \times P) = F \cap (T \times T) = \emptyset$ *(a place may be connected with the transition or the transition with the place; two places or two transitions cannot be connected).*

– $W : F \to \{1, 2, 3, ...\}$ *is a weight function assigned to arcs. As a default the weight of one is assigned to an arc.*

Definition 2. *A marking M of a net N is a mapping $M : P \to \{0, 1, 2, 3, ...\}$.*

Definition 3. *Petri net is a set (P, T, F, W, M_0) where M_0 is an initial marking, and (P, T, F, W) is a Petri net graph (see Definition 1.*

Definition 4. *For each element $t \in T$ we define a set of input places $\bullet t = \{p \in P; (p, t) \in P\}$ and a set of output places $t\bullet = \{p \in P; (t, p) \in P\}$.*

Definition 5. *A transition t may fire (such transition is called enabled) in a marking M if the number of tokens in every input place p of the transition t is equal or greater than the weight $W(p)$ assigned to an arc between the place p and the transition t in the marking M.*

During firing the transition t consumes tokens from the input places and puts them into the output places - the number of tokens transferred is determined by the weights of arcs.

3 One Place One Atom algorithm

In the OPOA algorithm a single place in the Petri net represents the position of one atom from the system studied by MD simulation. A transition represents the movement of the atom. This transition connects the previous localization of the atom (the input place) and the new localization of the same atom (the output place). Every transition has to be labeled not only by the initial and final localizations of the corresponding atom, but by the description of this atom as well. It can be obtained for example by additional annotations or by special places to which transitions will be connected by loops.

Definition 6. *The set of transitions which describe the trajectory of one specific atom is called atom transitions set (ATS).*

One can image that a PN generated by the OPOA algorithm for even small proteins can be enormous. Big networks are extremely hard to analyze, so to reduce the number of places and transitions two methods were used: a coarse grain representation and a discretization of the $\mathbb{R}^3$ space. The coarse grain representation is standard and widely used method of reduction complexity of the model [3]. It involves a representation of a group of atoms, by a smaller group, usually by just one atom. There are

different types of a such reduction, one is to represent an amino acid by its C_α atom. This method was used in our work. The discretization of $\mathbb{R}^3$ space in this paper means that in order to reduce the number of points used to represent atoms' position, the three dimensional grid is laid over $\mathbb{R}^3$ space and it divides the space into cubes. The edge of each cube is equal to the resolution of the grid. Each cube represents a new point in the new three dimensional discrete space and every atom which belong to the same cube in the $\mathbb{R}^3$ is localized in one point in the new space.

The basic pseudo code of the OPOA algorithm is listed below. It follows the general description of the algorithm. The main part of the algorithm is related to checking whether required places or transitions are already present in the PN.

One Place One Atom algorithm (OPOA)
Input: Files in Protein Data Bank (PDB) format containing the MD trajectories.
Output: Petri net described by lists of places, transitions and arcs.
Steps:

```
1:  for (every PDB file) do
2:      frame ← readFrame()
3:      for (every C_α atom in frame) do
4:          if (position(C_α, frame) ≠ (position(C_α, previousFrame) then
5:              prevPlace ← places.get(C_α, position(C_α, previousFrame))
6:              if (!places.find(position(C_α, frame))) then
7:                  newPlace ← createPlace(position(C_α, frame))
8:                  places.add(place)
9:                  transition ← createTransition(C_α)
10:                 transition.add(transition)
11:                 createArcs(newPlace, transition, prevPlace, C_α)
12:             else
13:                 currentPlace ← places.get(position(C_α, frame))
14:                 if (!transitions.find(prevPlace, currentPlace)) then
15:                     transition ← createPlace(position(C_α, frame))
16:                     transition.add(transition)
17:                     createArcs((currentPlace, transition, prevPlace, C_α)
18:                 end if
19:             end if
20:         end if
21:     end for
22: end for
```

In the OPOA algorithm the presence of a token describes the current location of the C_α atom. There is one token per one C_α atom. However, the idea described previously has a disadvantage - the "stealing problem". Such a situation occurs when a few C_α atoms are allowed to be in the same point in the space (for example not necessary at the same time during MD simulation). In such a case two (or more) tokens can be present in one place, each of them has to be put there by transitions from different atom transitions sets. However, the PN potentially allows a transition t_k from ATS of C_α atom to consume all the tokens from exact place. After such a situation, the PN marking will correspond to the situation when we would have two (or more) instances of a single atom and none instances of other C_α atoms, their tokens have been present in the considered place and "stolen". It is an undesirable situation. Two mechanisms to avoid the "stealing problem" was developed, they resulted in two versions of the OPOA algorithm. In the first version, separated places are generated for every atom. In the second one, additional construction is added to the PN to prevent the "stealing problem".

4 Results and conclusions

Both versions of the OPOA algorithm were used to analyze sixteen MD simulations of a small chemokine MCP-1. Analyzing a Petri net generated by the OPOA version 1 algorithm, one may easily find amino acids which are more flexible than others by a simple checking number of places in every ATS. Such an analysis has been performed for MCP-1 and its results are presented in Figure 2.

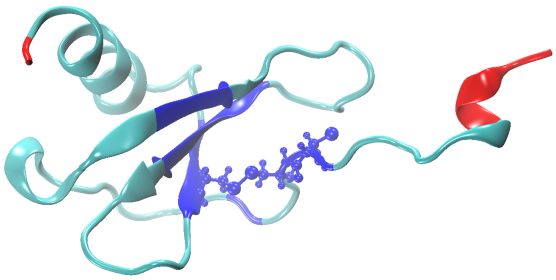


Fig. 2. MCP-1 with highlighted amino acids with a large number of places in their ATS (red) and the small number (blue).

The selected amino acids with relatively large number of places in their ATSs are highlighted in red, those with small in blue. The "red" regions contain the most flexible amino acids indeed. They are located at the terminal loops. On the other hand the "blue" amino acids should be located at the most stable parts of the protein. It is also true because those "blue" parts correspond to β-sheet or regions stabilized by H-bonds with β-sheet. The method is sensitive enough to detect such features, even regions stabilized by H-bonds.

Petri nets created with OPOA algorithms allow also to study dynamical properties of every amino acid. Points/places frequently visited by each C_α atom can be easily identified and may be analyzed. Moreover, the usage of PN allows deeper data mining: one can extract all the paths leading to/from the most frequently visited place. The general characteristic of C_α movements can be analyzed.

The PN generated by the OPOA algorithm can be a useful tools to improve analysis of MD trajectories. Moreover, the algorithm is a part of bigger PN framework. This paper shows that PNs are very flexible and universal modeling method, which is suitable not only for strict mathematical uses but also for MD studies.

References

1. Anna Gogolinska, Rafal Jakubowski, and Wieslaw Nowak. Petri nets formalism facilitates analysis of complex biomolecular structural data. *RAIRO-Operations Research*, 50(2):401–411, 2016.
2. William Humphrey, Andrew Dalke, and Klaus Schulten. Vmd: visual molecular dynamics. *Journal of molecular graphics*, 14(1):33–38, 1996.
3. Siewert J Marrink, H Jelger Risselada, Serge Yefimov, D Peter Tieleman, and Alex H De Vries. The martini force field: coarse grained model for biomolecular simulations. *The journal of physical chemistry B*, 111(27):7812–7824, 2007.
4. Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.
5. Wieslaw Nowak. Applications of computational methods to simulations of proteins dynamics. In *Handbook of Computational Chemistry*, pages 1127–1153. Springer, 2012.
6. James C Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D Skeel, Laxmikant Kale, and Klaus Schulten. Scalable molecular dynamics with namd. *Journal of computational chemistry*, 26(16):1781–1802, 2005.

ePNK Applications and Annotations: A Simulator for YAWL Nets

Ekkart Kindler

DTU Compute, Technical University of Denmark
ekki@dtu.dk

1 Introduction

The *ePNK* is an Eclipse based framework and platform for developing and integrating Petri net tools and applications. One of its core features is that new types of Petri nets can be realized and plugged into the ePNK without any programming by providing a model of the concepts of the new type, the so-called *Petri net type definition* (*PNTD*). Moreover, the ePNK allows developers customizing the graphical appearance of the features of a new Petri net type.

The main idea and features of the ePNK have been presented before [1, 2]. One important aspect of the ePNK, however, has not been discussed yet: realizing new *applications* for the ePNK and, in particular, visualizing the result of an application in the graphical editor of the ePNK by using *annotations*, and interacting with the end user using these annotations.

In this paper, we give an overview of the concepts of ePNK applications by discussing the implementation of a simulator for YAWL nets [3].

2 The Result

Before we discuss how to develop the YAWL simulator for the ePNK, we give a brief overview of the final result. Figure 1 shows a YAWL net with two YAWL simulators running on it, where the current state of the simulator selected in the *ePNK application view* are shown highlighted on top of the net in the graphical net editor. The blue numbers at the top right corner of a place indicate the current marking of the net, and the blue overlays of a transition indicate the currently enabled transitions. The red overlays for places, arcs and transitions indicate a path on which a token might arrive on place `c4`; this serves as a warning that the user should not yet fire transition `a6`, since it is an OR-join that is supposed to wait for all possibly incoming tokens.

In Fig. 1, you can also see that a YAWL net has exactly one *start place* and exactly one *finish place*, indicated by corresponding icons. In our example, there is one OR-split transition (`a1`) and one OR-join transition (`a6`), indicated by the corresponding graphical symbols. YAWL nets have some additional features like *AND-* and *XOR-joins* and *-splits*, and *reset arcs*. But, our example from Fig. 1 does not show them.

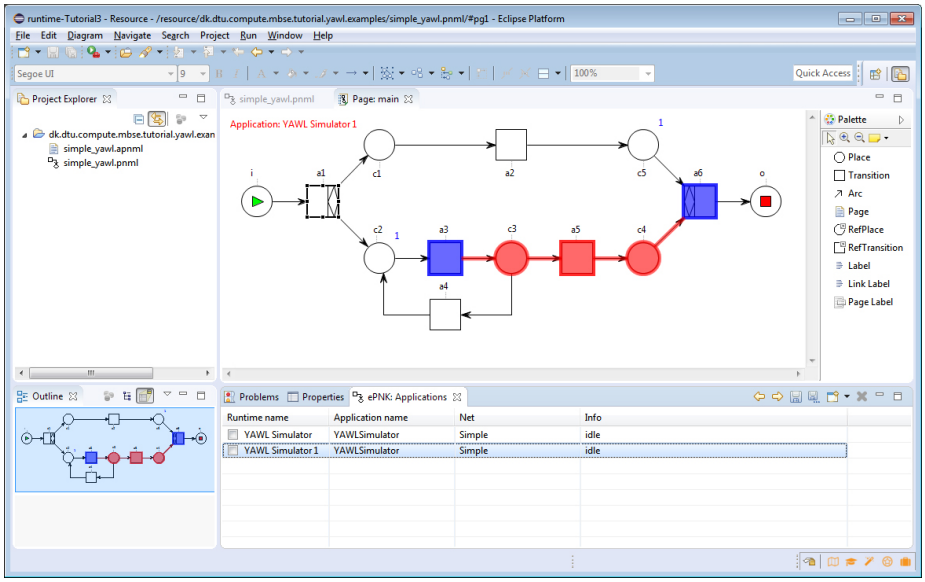


Fig. 1. ePNK with two YAWL simulators running

Once the simulator is running and selected, the user can interact with it by double clicking on the enabled transitions. This will fire the respective transition and show the successor marking. And the user can go back and forth to the previous or next marking by the respective buttons in the toolbar of the ePNK *application view*. From there, it is also possible to save the current state of the simulator with the save button, to start new applications via a drop down menu, which will show all applications registered for the net in the currently active editor. Moreover, the user, can shut down applications or load an application from a state that was saved earlier. As long as the developer follows the guidelines of the ePNK, these features do not need any extra programming.

3 The YAWL PNTD

Before actually realizing an application for YAWL nets in the ePNK, we need to realize YAWL nets in the ePNK. This is done by providing a class diagram, which defines all the concepts of YAWL. This is called a *Peti net type definition* (*PNTD*). Figure 2 shows the PNTD for YAWL nets: Places are extended to *Conditions*, YAWL's name for places, which have an *attribute* type, saying whether it is a *normal*, a *start* or a *finish* place. Transitions are extended to *Actions*, YAWL's name for transitions, which have two attributes, defining the *join*- and the *split*-type of the transition; they can have the values *AND*, *XOR* and *OR*. Likewise for arcs, there is an attribute defining the type of the arc, which can be *NORMAL* or *RESET*.

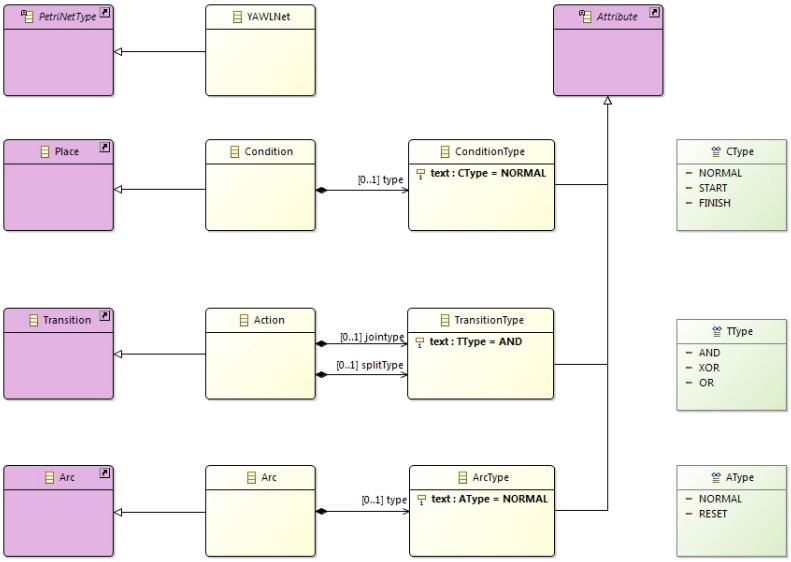


Fig. 2. PNTD for YAWL

In addition to the concepts defined in the model, there are some constrains, which we do not discuss here. In contrast to earlier versions of the ePNK, which required some minor programming, version 1.2 of the ePNK does not need any programming for plugging in a new PNTD; the model from Fig. 2 and the code generated from it by EMF can be plugged in directly to the ePNK.

In addition to the PNTD, we would also need to customize the graphical appearance for *start* and *end places*, *reset arcs* and the split- and join-types of transitions by some minor programming. But, we do not discuss this here.

Then, the graphical editor of the ePNK would be able to show and edit YAWL nets as shown in Fig. 1.

4 The YAWL Simulator

Next, we discuss the concepts of ePNK applications by the example of a simulator for YAWL nets.

4.1 The Annotations

The first step is to define the *annotations* that represent the *runtime information* of an application; in the example of our YAWL simulator, this is a sequence of markings of the simulator and the current marking.

Figure 3 shows the annotations defined for the YAWL simulator. The three classes at the bottom define the annotations EnabledTransition, SelectArc, and

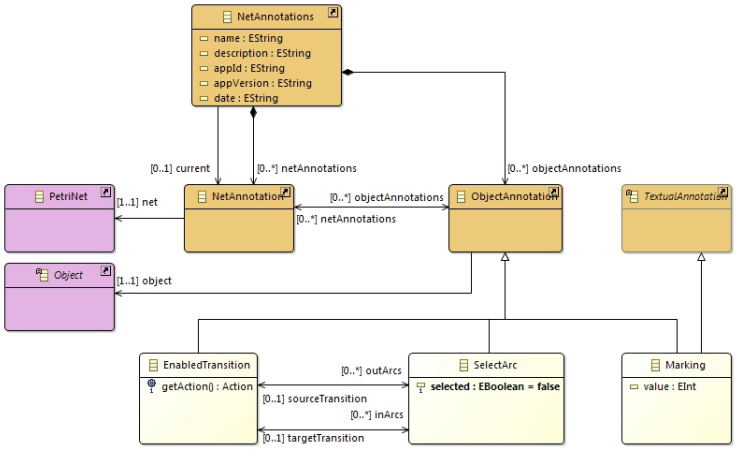


Fig. 3. YAWL annotations

Marking, which we have seen in use already in Fig. 1. Note that we do not define new annotations for highlighting the path of possible tokens arriving at an OR-join here, since we can reuse ObjectAnnotation, which is defined by the ePNK already. The top part of Fig. 3, actually, shows the concepts of annotations as defined by the ePNK.

Let us have a brief look at these concepts: The two concepts PetriNet and Object on the left and shown in magenta come from the *PMNL core model* [4] and represent concepts of *Petri nets* themselves along with their *objects*: places, transitions, and arcs; but also pages, and reference transitions and reference places. The orange classes at the top define ePNK's concepts of annotations. The ObjectAnnotation annotates exactly one object, which is represented by the reference object. Note that it is crucial that a Petri net object itself does not know anything about its object annotations at all. A NetAnnotation refers to one Petri net and consist of many object annotations. The class NetAnnotations comprises all the annotations of a running application, a set of net annotations—one of which is pointed out as the *current* one. The interpretation of this structure is up to the concrete application, but the default one is a sequence of net annotations. The class NetAnnotations has some additional attributes, which is relevant when an application—actually its state—is saved. In particular, the appId is used for starting the respective application, from which the state was saved, again.

The ePNK defines one abstract class TextualAnnotation. An ePNK application, by default, presents annotations inheriting from TextualAnnotation as textual labels at the top left of the object, showing the value of its value attribute. In our YAWL simulator, the Marking is an example of a textual annotation. All other annotations are, by default, shown as red overlays of the respective object. But, we will see later that an application can actually customize how an annotation is graphically presented.

As mentioned already, we define three annotations for our YAWL simulator. The annotation SelectArc is used for indicating which arcs of an enabled transition can be selected by the user, and which of the are currently selected, represented by the attribute selected. In order to define the logic of the arc selection (see Sect. 4.2 for details), the SelectArc annotations are related to the respective EnabledTransition annotation.

4.2 The Application

The three new classes from Fig. 3 represent the state of the running simulator with a NetAnnotations object as its root. We call this the *runtime information* of the simulator. Each NetAnnotation represents a marking (plus the enabled transitions and the selected arcs), current representing the current marking.

In addition to defining its runtime information, an application must implement three different things: the *initialization*, some *presentation handlers*, and some *action handlers*. The *initialization* needs to compute the initial net annotations, representing the state initial state (the initial marking in our case); the *presentation handlers* define how the different object annotations should be presented; the *action handlers* define what should happen when the user interacts with an annotation (or actually its presentation in the graphical editor).

Separating the definition of the runtime information, the presentation handlers and action handlers in applications follows the architectural pattern of *model-view-controller* (*MVC*).

Here, we cannot discuss all the details of implementing the action handlers and the presentation handlers. But we give a brief overview. The YAWL simulator has two action handlers: one for firing the transition when the user double clicks on its enabled transition annotation, and one for selecting or unselecting arcs when the user clicks on an select arc annotation. The *enabled transition* handler, basically, adds a new net annotation to the state of the simulator with the annotations representing the new marking and makes this new net annotation the current one. The *select arc* handler, basically, toggles the selected attribute taking the semantics of the respective split or join into account.

The presentation handlers, basically, returns an overlay figure for the respective graphical figure representing the annotated object in the graphical editor. In our case, it returns a blue overlay for an enabled transition and, dependent on the selected attribute, a blue or grey overlay for a selected arc annotation. For all other annotations, ePNK's default presentation handler kicks in, returning a red overlay.

5 Conclusion

By the example of YAWL nets and the YAWL simulator, we have discussed the main ideas of ePNK applications and how they are realized based on an annotation model. Like the definition of YAWL nets themselves, the runtime information of the simulator is defined by a model. Here, we could not discuss

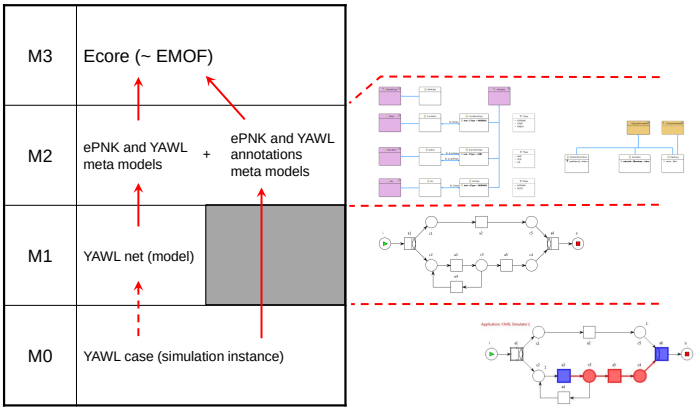


Fig. 4. MOF levels: YAWL nets and simulator

any technical details, for which we refer to the revised manual of version 1.2 of the ePNK, which will be published shortly [5].

Modelling the runtime information following the MVC-pattern has many advantages: among other things, allowing saving and loading the state of an application in a uniform way without any additional programming by the developer of an application. Moreover, it clarifies which parts of the models and meta models belong to which level in MOF [6] as shown in Fig. 4, where the red arrows represent an *is instance* relation between different models, and the dashed arrow represents that a model is a*conceptual* instance of another model like a simulation being an instance of a YAWL net.

References

1. Kindler, E.: The ePNK: An extensible Petri net tool for PNML. In: Applications and Theory of Petri Nets - 32^{nd} International Conference, Proceedings. Volume 6709 of LNCS., Springer (2011) 318–327
2. Kindler, E.: The ePNK: A generic PNML tool - users' and developers' guide for version 1.0.0. Technical Report IMM-Technical Report-2012-14, DTU Informatics, Kgs. Lyngby, Denmark (2012) URL `http://www2.imm.dtu.dk/~ekki/projects/ePNK/PDF/ePNK-manual-1.0.0.pdf`.
3. van der Aalst, W., ter Hofstede, A.: YAWL: Yet another workflow language. Technical Report QUT Technical report, FIT-TR-2002-06, Queensland University of Technology, Brisbane (2002)
4. Hillah, L., Kindler, E., Kordon, F., Petrucci, L., Treves, N.: A primer on the Petri Net Markup Language and ISO/IEC 15909-2. In Jensen, K., ed.: 10^{th} Workshop on Coloured Petri Nets (CPN 09). (2009) 101–120
5. Kindler, E.: The ePNK: A generic PNML tool - users' and developers' guide for version 1.2.0. Technical Report 2017-??, DTU Compute, Kgs. Lyngby, Denmark (2017 to appear)
6. OMG: Meta Object Facility (MOF) specification, version 1.4.1. Technical Report formal/05-05-05, The Object Management Group, Inc. (2005)

Detecting Security Attacks by Process Mining

Sebastian Mauser and Tobias Eggendorfer

Hochschule Ravensburg-Weingarten, Doggenriedstr., 88250 Weingarten, Germany
{sebastian.mauser,tobias.eggendorfer}@hs-weingarten.de

Abstract. Software attacks are often used to exfiltrate data or inject arbitrary commands into systems in order to gain control over them. Most of these attacks rely on changing a process' execution path. Process mining is a method to identify a model of the process underlying a system. By monitoring process executions and comparing them to such model of the normal process, security issues could be identified. This paper describes how this is achieved.

Keywords: Process Mining, Security Attacks, Process Monitoring, Petri Nets.

1 Introduction

Attacking software often means to either inject new code through a vulnerability such as a buffer overflow or alter the execution path of a process, e.g. by a „return-to-libc“ attack. However these attacks are hard to identify, since according to Rice's theorem it is impossible for a program to tell what another program is meant for. Social engineering attacks, where users are tricked to use a program in another way than it was intended by either the developer or the company policies, are even harder to detect by other programs. The same issues arise with insider attacks where authenticated users exfiltrate or manipulate data. However all these attacks share that the process' execution path is altered and thus changes from the usual behavior of the program. In fact, most attack vectors tend to change the normal process flow of a system by differing from typical usage patterns, skipping parts of a program, injecting new program behavior or forcing anomalous program executions.

Therefore, this paper proposes an approach of automatically observing process executions of computer programs by monitoring usage data. This way anomalous execution paths can be detected in order to identify and prevent security attacks. To develop appropriate techniques for analyzing process executions and distinguishing between normal and suspicious system behavior, we use methods from the research field called process mining [1, 2]. As the above examples show, such approach might provide an effective security mechanism for some types of attacks, e.g. insider attacks, which so far are almost impossible to detect. Moreover, for most other attack vectors our approach can be considered as an additional line of defense complementing existing security measures of e.g. intrusion detection systems and firewalls. The long term goal of our work is to develop a new process-based generic security system which can be applied to any kind of software system.

We have conducted first research on the topic and created a simple prototype for demonstration purposes [7]. Our approach is based on the preliminary research on security and process mining by van der Aalst et al [2]. The latter paper discusses the topic on a high level of abstraction focusing more on auditing. It has barely been continued specifically in the direction of security attacks and intrusion detection which is the purpose of our research. In contrast to most other approaches on security attacks and process monitoring which focus on pattern analysis, e.g. [3], our method is based on the overall process flow of a system by using process mining techniques.

2 Motivation by an Industrial Case Study

A special motivation to work on this topic has been personal experience in the financial industry. In this area, data security and confidentiality are of particular importance. However, the existing software landscape in finance companies often consists of many legacy systems. These highly business critical systems are far from meeting state-of-the-art software security requirements.

One of the authors has participated in a multi-million Euro project to significantly improve the security of a legacy financial system. While the project was partly successful, some security problems cannot be solved adequately for legacy systems. Moreover, the costs of the project exploded and the overall cost-benefit ratio was very poor. Due to this unsatisfactory situation which is valid for many legacy systems, there is a growing need to find new generic solutions for improving the security of such systems. In this context, the authors came up with the idea of a generic system to detect security attacks using process mining techniques as presented in this paper. We conducted a first case study on a real world financial system which will briefly be sketched in this section.

In this study a web application which is used to process financial trades has been investigated. Every process activity of a user of the system is stored in a so called event log together with a time stamp, the trade-id of the processed trade and some additional information.[1] The sequence of activities executed for one trade defines one execution of the system's trading process. Such process execution is called a case of the process [1]. All cases of the process can be identified by ordering the recorded activities related to a certain trade-id according to their time stamps. These cases show what really happened in the system, i.e. the true trading process. One main idea of process mining is to generate a model of this process from all the cases stored in the event log [1]. This is called process discovery. In literature, numerous process discovery algorithms have been proposed and many of them are implemented in the tool ProM developed at Eindhoven Technical University (http://www.promtools.org).

The event log of the trading software used here covers a three-month period of system usage with 62380 events, 10913 cases (trades) and 10 different activities. This log has been imported to ProM. For process discovery, the heuristic miner of ProM has been used, which implements the heuristic process mining algorithm described in [5].

[1] Note that the system also contains user activities which are not connected to a trade. To only focus on the trading process, these activities have been removed from the event log.

This popular process discovery algorithm is particularly suitable to express the main behavior of a process and to deal with noise. The degree of abstraction of the resulting process model can be configured by various settings of the algorithm. The algorithm generates a process model in the form of a dependency graph which shows the dependencies of the process activities together with the frequencies of activities and ordering relations in the log. For the given log we used the standard ProM-settings of the heuristic miner despite of the threshold for "length-two-loops" which was set to 0 (since these loops are important in the trading process). The dependency graph generated from the log is shown in Fig. 1. As confirmed by domain experts, this graph is a good representation of the real trading process. Note that for confidentiality reasons the real activities of the process like submit, acknowledge or cancel a trade are not disclosed in the diagrams of this paper.

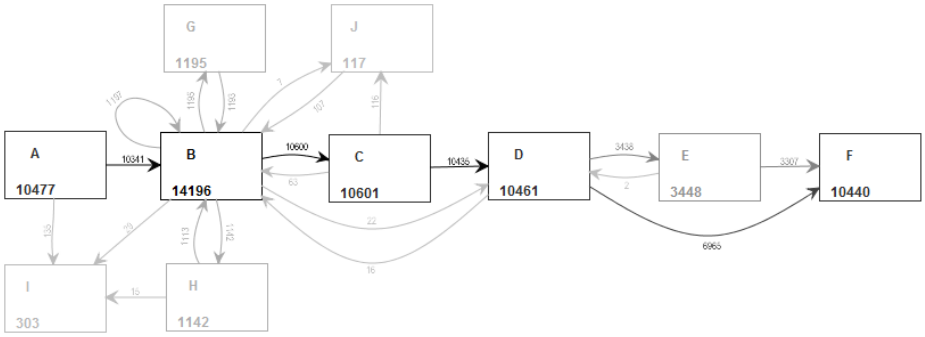


Fig. 1. Dependeny graph model of trading process generated by heuristic miner.

Now having a valid model of the process and a corresponding event log, we are able to set up a monitoring system that observes the live event data in the event log and detects deviations from the normal business process given by the dependency graph. The crucial question here is if this helps to identify security attacks of hackers.

In our example, an attacker would of course try to unauthorizedly process a trade. Since the trading activities in the system require different roles and thus must not all be done by the same person, the attacker would perhaps try to skip some of the steps or execute all of the steps with the same system user or manipulate a canceled trade or something like that. For this purpose, he will play around with the possibilities offered by the user interface and do things a normal user would never do. Moreover, he would try to manipulate the system in order to force behavior that is not possible when normally interacting with the user interface. In any case, this would result in a process execution that does not correspond to normal process behavior and would therefore be detected by a respective monitoring system.

Results from a penetration test of the trading software done by a security testing company gave evidence for these assumptions. The penetration test used a combination of several attack patterns. The results of the penetration test clearly showed that the attacks led to deviations from the normal process. In fact, most security attacks require playing around with the system resulting in anomalous process executions.

3 Algorithmic Aspects

Finally, we discuss some algorithmic and technical details of using process mining to detect security attacks.

Process Model. For such approach a reference model representing normal process executions has to be generated. The reference model serves as a basis to decide whether newly observed execution paths of the process are potential security violations or not. In Section 2 we used a dependency graph as a references model which worked well in this case. However, the example process includes no concurrency and only local dependencies of activities. Although in [5] it is shown how the heuristic miner can deal with these aspects, in our opinion the resulting models are not any more intuitively understandable. For modeling concurrency additional modeling elements have to be added such as e.g. in c-nets [1, 5]. Moreover, long-distance dependencies in dependency graphs can hardly be distinguished from short-cuts such as skipping the activity E in Fig. 1. For a general process including concurrency and long-distance dependencies, a Petri net based model is a more natural choice since Petri nets can intuitively represent the fine interplay of concurrency and non-determinism. In Fig. 2 a Petri net is shown which was automatically generated by ProM from the dependency graph in Fig. 1.

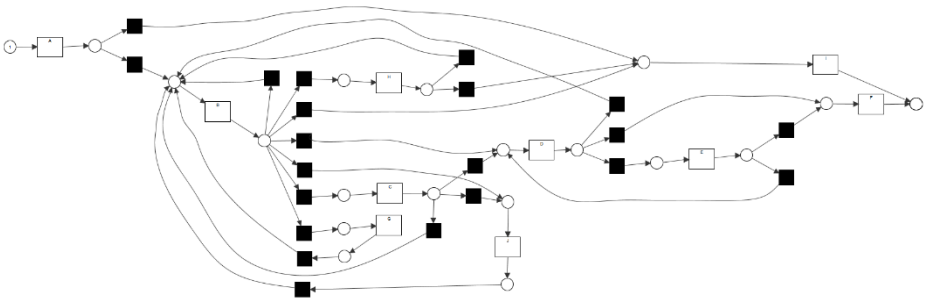


Fig. 2. Petri net model of trading process generated from the graph in Fig. 1.

Event Log. To generate such Petri net model using process mining we first have to find an adequate event log of the observed system, e.g. an audit trail containing information about the underlying business process as in the example of Section 2 or a low level log file recording technical events about process executions. Both low level and high level data are suitable for our approach since they both can be used to find process deviations caused by security attacks. Today's software systems often provide such logging data. For instance, in the financial domain mentioned before, a vast amount of logging information is usually available ranging from low level to high level system events. Given an appropriate event log, a major challenge is how to identify the cases in the log, e.g. in the example this has been done via the trade-id. As our first practical experiments showed, the information available in typical logging data can very well be used for our security monitoring approach. Still, when this is not the

case for some system, it is often possible to generate adequate usage data by applying a generic tracing mechanism and potentially memory layout analysis.

However, a problem is that the event log of a system may already contain security violations from past attacks which will then possibly be classified as correct process executions. This problem can either be solved by some filtering technique, e.g. the process discovery algorithm ignores process executions with low probability as it is the case for the heuristic miner used in our example, or by specifically supervising the learning phase when the event log is created, e.g. in this time only test users work with the software in some offline setup.

Process Discovery. The main algorithmic challenge is to generate a Petri net process model from a given event log such that the model is suitable for security monitoring. In this context an evaluation of existing process discovery approaches and possibly the development of a new algorithm is necessary. Typical questions of process discovery such as probabilities, timing, noise, unobserved normal behavior, simplicity, fitness, precision, generalization etc. [1] have to be handled in an appropriate way for a security monitoring system.

We illustrate the difficulties of process discovery by a short example. By playing around with the example log of the trading software we found some exceptional behavior. For instance, for three trades activity F has been executed twice. This behavior is not included in the model of Fig. 1 since it has been filtered out by the heuristic miner. It seems appropriate to not include such behavior in the reference process model because on the one hand the system has not been correctly used in these three cases and on the other hand the model might become too complex yielding a useless spaghetti-model when including all such exceptional behavior. However, it can also be argued that omitting such exceptional behavior from the reference model might cause a false alarm the next time the same misuse occurs by a regular user.

Process Monitoring. Finally, a monitoring mechanism which checks the live logging data of the observed system for suspicious process' execution paths has to be implemented. Based on a comparison to the reference process model, the probability that a new case is a security violation has to be evaluated. In case of a potential attack, the security system can take appropriate measures, e.g. generate a warning message for administrators, block a further execution of the process, log out the user, etc.

A simple evaluation method is to check if the new case can be executed in the reference Petri net by playing the token game. However, this might be too strict yielding a lot of false alarms because as shown before regular users sometimes slightly deviate from the normal process. A more appropriate approach is using techniques from the area of conformance checking in process mining [1]. While in general conformance checking compares the process model to a whole event log, Petri net based approaches often consider the so called fitness of single cases. In [4] the fitness of a case is defined by the number of tokens which have to be added to the net such that the case can successfully be replayed in the net. In [6] the cost of "movements" such that the case can be executed in the net is computed. To define an appropriate notion of fitness of cases in the context of security monitoring we can also regard a "nearest neighbor" approach, probabilities, resources etc. Then, we can use a threshold value for the fitness of a case to decide whether a new case is classified as a security violation.

As an example consider the standard case A-B-C-D-F of the process shown in Fig. 2. Of course this case can be executed in the Petri net and thus will not be classified as a security violation. The same holds e.g. for the case A-B-G-B-C-D-E-F. However, for the exceptional behavior of some users including slight deviations from the normal process, e.g. executing the activity F twice as mentioned before, this is not true. For instance the case A-B-C-D-F-F cannot be executed in the net. However, for a successful replay of the case only one extra token has to be added to the input place of the F-activity. That means, according to the approach in [4] only one token is missing for successful replay compared to ten tokens which are produced in the net during execution of the case. This yields a relation of missing tokens to produced tokens of 0.1 and a fitness of 1 – 0.1 = 90% when only considering missing tokens (in [4] also remaining tokens are regarded). Therefore, given a threshold value for fitness of e.g. 80% (< 90%), the exceptional case would not be classified as a security attack.

4 Conclusion

We already discussed the approach presented in this paper on a security workshop [7] and the feedback from the security experts was positive. With this paper, we now want to also discuss the approach with the experts in Petri nets and process mining.

In [7] a first prototype implementation has been developed. It has been tested against a simple web shop application for a first analysis. The results of our initial research are promising. We were in this paper able to theoretically and practically show that our approach is capable of effectively detecting security violations. In the future we plan to extend our work along the lines sketched in this paper.

References

1. Aalst, W..: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer, Berlin (2011).
2. Aalst, W., Medeiros, A.: Process Mining and Security: Detecting Anomalous Process Executions and Checking Process Conformance. Electronic Notes in Theoretical Computer Science, 121:3-21 (2005).
3. Forrest, S., Perelson, A., Allen, L., Cherukuri. R.: Self-Nonself Discrimination in a Computer. In: Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy, pp. 202–212. IEEE Computer Society Press, Los Alamitos (1994).
4. Aalst, W., Rozinat, A.: Conformance Checking of Processes Based on Monitoring Real Behavior. Information Systems, 33:64-95 (2008).
5. Weijters, A, Aalst, W., Medeiros, A.: Process Mining with the Heuristics Miner-algorithm. BETA Working Paper Series, WP 166. Eindhoven University of Technology (2006).
6. Aalst, W., Adriansyah, A., Dongen, B.: Replaying History on Process Models for Conformance Checking and Performance Analysis. Wiley Int. Rev. Data Min. and Knowl. Disc., 2, 2:182–192 (2012).
7. Mauser, S., Eggendorfer, T., Wichert, D.: Using process mining to identify attacks. In: Proceedings of the 3rd Interdisciplinary Cyber Research Workshop, pp. 36-37. Tallinn University of Technology (2017).

Netgrif Workflow Management System based on Petriflow language

Juraj Mažári[1,2], Gabriel Juhás[1,2,3], Milan Mladoniczky[1,2], Tomáš Gažo[2], and Martin Makáň[2]

[1] Faculty of Electrical Engineering and Information Technology
Slovak University of Technology in Bratislava,
Ilkovičova 3, 812 19 Bratislava, Slovakia,
[2] NETGRIF, s.r.o.,
Blumentálska 12, 811 07 Bratislava, Slovakia
netgrif@netgrif.com
Home page: http://www.netgrif.com
[3] BIREGAL s. r. o.,
Klincova 37/B, 821 08 Bratislava, Slovakia
biregal@biregal.sk
Home page: http://www.biregal.com

Abstract. Netgrif Workflow Management System (Netgrif WMS) is JVM based application build on Spring framework. Netgrif WMS can be used to import Petriflow workflows in form of an XML file. Process roles are then assigned to other users by authorized user. Users can create new cases (instances) of imported workflows. New tasks are automatically generated for each new case. Users with corresponding roles are able to assign, reassign, finish or cancel task. Tasks data fields can be edited by assigned user and are automatically validated and saved.

Keywords: Petri nets, workflow, Spring Boot

1 Introduction

Modern companies need workflow management systems that are able to frequently change their workflows according to changes in bussiness. Petriflow language aims to provide modeling capabilities to satisfy this needs [1]. Netgrif Worfklow Management System is build as a tool for execution of Petriflow based workflows. Petriflow workflows are place/transition nets extended by roles for assigning and executing tasks represented by the net transitions and by data variables, which can be assigned to transitions. Data variables can be assigned to several transitions and for each transition a data variable can have different atributes, which specify for example whether the value for the variable is required, optional etc. Netgrif WMS enable to import Petriflow workflows and create new cases of them. Assigning, finishing and canceling tasks of created cases is available for registered users with process roles associated with the tasks. User management is available for users with admin system role.

2 Architecture

Netgrif WMS is build as a three-layer client-server application. AngularJS framework is used on presentation layer, Spring Boot framework on application layer and multiple databases on data layer. Figure 1 depicts three-layer architecture of Netgrif WMS.

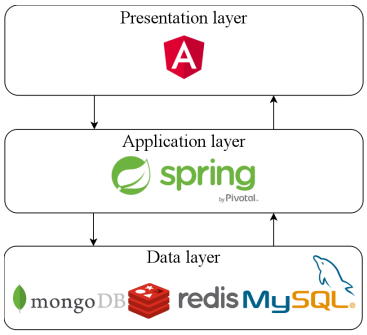


Fig. 1. Netgrif WMS three-layer architecture

2.1 Presentation layer

AngularJS framework is base stone of the presentation layer. AngularJS uses the MVC design pattern. The model, view and controller are well defined in AngularJS and serve to simplify the development process. Figure 2 shows communication between those components, web browsers and application layer [2].

The main advantage of AngularJS is that it enabls to easily create AngularJS services that communicates with multiple web-services. These services can be used on many places in order to optimize the size of the code. Along with AngularJS, Netgrif WMS also uses AngularJS Material framework. It provides a set of reusable UI components based on Google's Material Design. This gives Netgrif WMS modern look on which many users are accustomed. Therefore navigation and UI components are user friendly.

2.2 Application layer

Application layer of WMS is build on **Spring Boot** framework. In accordance with rapid development, Spring Boot makes it easy to create stand-alone applications that can be started by simple `java -jar` command thanks to embedded Tomcat [3]. Spring framework also introduces **dependecy injection**, which enables autowiring of components.

Application layer is divided into multiple modules separating their concerns. Each module consists of three packages - domain, service and web. Domain

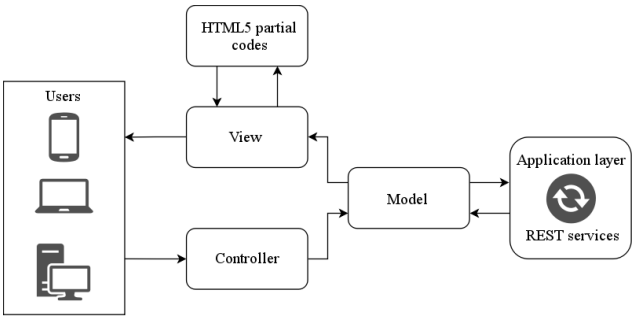


Fig. 2. AngularJS internal and external communication

package contains entity classes which are persisted into database. Bussines logic is kept in service packages. According to Spring customs, interface is created and used to autowire each service. Web package keeps request and response body classes and REST controllers, which calls methods on autowired services.

2.3 Data layer

Data layer leverages benefits of different database systems. Structured data which are rarely changed, such as user credentials, are stored in **MySQL** database. User sessions are stored in in-memory database **Redis**. Tasks, cases, imported workflows and other unstructured and constantly changing data are stored in **MongoDB** database.

Each of these databases is easily accessible via Spring Data project. It enables to use databases without any configuration. Spring Data also reduce the amount of boilerplate code required to implement data access layers for various persistence stores.

2.4 Presentation-application layer communication

Presentation and application layer are connected via REST web-services. **HATEOAS** constraints are applied on those web-services. Main advantage of HATEOAS principle is that presentation layer does not have to know all web-services URLs. HATEOAS response provides data and all accessible web-services for the given resource.

3 Implementation

Netgrif WMS is implemented as a Spring Boot project build by **Maven**. Open source automation server **Jenkins** is used for building and deploying. This tool automatically builds and deploys new version of Netgrif WMS after each commit to git repository. In this way it is secured that latest version is always tested and deployed if all tests passed.

Classes extending Spring Boot's `CommandLineRunner` interface are runned at startup. This is used to create dummy data for development. Sample user accounts and cases are easily generated this way. Using spring boot property `spring.jpa.hibernate.ddl-auto` set to `create-drop` for development use allows to automatically create new version of data at startup and delete existing data on application end.

Import of a workflow net has gone through many changes. Best option comes to be use of **JAXB API**. Java classes are generated from Petriflow XSD definition. Those classes are used to unmarshall XML file. Unmarshalled file is processed by importer and persisted into MongoDB database as domain objects which are used by Netgrif WMS. Original XML file is also saved to local directory. Later it can be used to get a snapshot of a state of any case in the system. This is very useful for debugging during development phase. It is also possible to see a complete list of all data fields and their values for any case. These features are only available for development environment.

Generated tasks of a case can be assigned by user himself or delegated by another user. This allows both pull and push control [4] of tasks assignement. Both actions are possible only for users with assign and perform roles assigned to the task.

Netgrif WMS supports three types of triggers, namely user, automatic and time triggers. Tasks with automatic trigger are finished by the system immediately. Potential conflicts of automatically triggered tasks (transitions) are resolved by priority according to the order of the transitions in the original XML Petriflow file. Time triggers can be set to specific date and time or specific delay from the current time.

4 User interface

Users can be invited by admin users through **admin console**. Admin can define users email, organizations and process roles for imported processes. This will send email invite to specified address. Invite contains link to registration form in which user have to enter his name and password. Users email is used as a login. In the next tab, admin can manage process roles of already registered users according to the definition of roles in the Petriflow file.

Netgrif WMS provides two different views for managing tasks. **Task view** displays all available and assigned tasks for logged in user. Task panel is expanded upon clicking on it. Expanded task panel shows all visible data fields. User can change each editable data field and new value is automatically validated and saved. Invalid values are highlighted and task containing invalid data fields can not be finished.

Second view is named **case view**. Case view displays all available cases. After clicking on case panel a new tab is opened. Tasks belonging to selected case are displayed in that tab. Each task panel behaves the same way as in task view. This enables users to work with multiple cases simultaneously. User can create

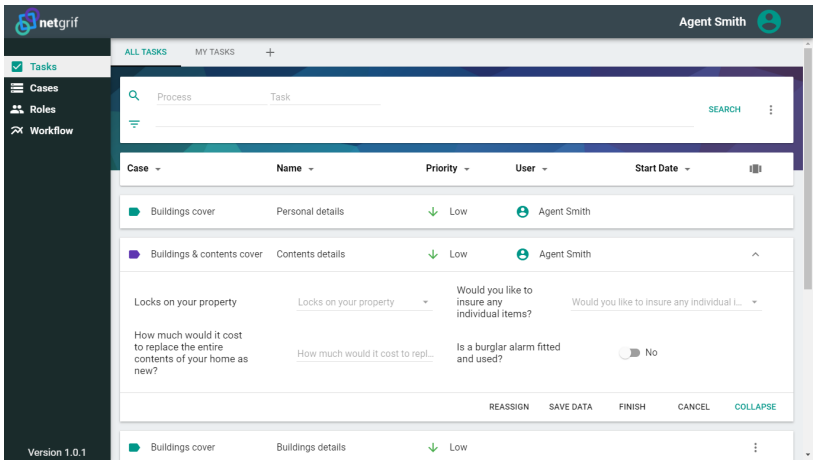


Fig. 3. Example of task view in Netgrif WMS

new case by clicking on a + button in tabs panel. This opens a dialog which enables user to select desired workflow, case title and label color.

5 Summary

Netgrif WMS is a light version of a workflow engine based on the Petriflow processes, which are basically Petri nets enriched by roles and data variables associated with transitions. A user registered in Netgrif WMS is able to upload processes and thus becoming their owner. Netgrif WMS enables to create new cases for a given Petriflow process and to control the cases processing according to the business logic given by the Petri net modelling the Petriflow process. For a case, the new copy of the underlying Petriflow process is created. The owner of the process can associate other users to assign and execute roles defined in the process. Once a transition of the case is enabled to fire, only users associated to an assign role of the transition can assign the corresponding task of a case modelled by the transition to the users associated with the execute role of the transition. By assigning a user to the transition, the tokens from pre-places of the transition are consumed. The user from the execute role assigned to the transition is able to manipulate with the values of data variables associated with the transition according to the policy defined by the Petriflow process model e.g. to fill the values of data fields specified as required for the transition, or to compute the values according to some formula over other data variables of the case etc. After all required data fields are filled, the assigned user from the execute role can finish the task execution. By finishing the task, the tokens are produced to the post-places of the transition.

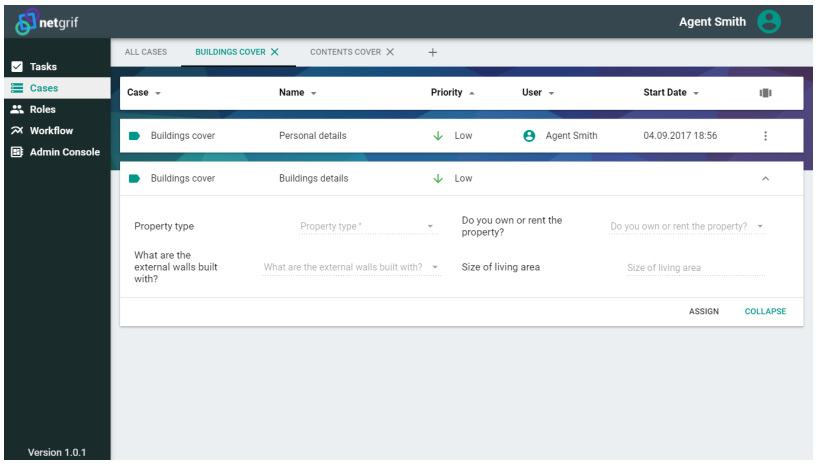


Fig. 4. Example of case view in Netgrif WMS

6 Conclusion

In this paper we introduced Netgrif Workflow Management System application, which is based on modern principles and technologies. It leverages main advantages of Petriflow language and supports rapid deployment of frequently changing workflows. From real world experiences we have learned this feature is highly valued and required by many companies. Future development includes new architecture. With Spring Boot 2.0 coming in December 2017 we plan to go **reactive** with Netgrif WMS. We are going to break the whole project into separate and independent **microservices**. Spring Boot 2.0 will introduce support for reactive programming. This will give us non-blocking, event-driven and easily scalable application. Presentation layer will also undergo changes. We are going to create UI service that will put together UI client from UI components based on workflow definition. We plan to use event sourcing and Command Query Responsibility Segregation for some microservices. This will provide us with a complete history of events and changes in any case.

References

1. Mladoniczky, M., Juhás, G., Mažari, J., Gažo, T. and Makáň, M.: Petriflow: Rapid language for modelling Petri nets with roles and data fields. Proceedings of the Workshop Algorithms and Tools for Petri nets 2017, October 19-20, 2017, Technical University of Denmark, Kgs. Lyngby, Denmark, (2017)
2. Williamson, K.: Learning AngularJS: A Guide to AngularJS Development O'Reilly Media, Inc., (2015)
3. Walls, C.: Spring Boot in action Manning Publications Co., (2016)
4. Van der Aalst, W. M.: The application of Petri nets to workflow management. Journal of circuits, systems, and computers, 8.01, 21-66 (1998)

Petriflow: Rapid language for modelling Petri nets with roles and data fields

Milan Mladoniczky[1,2], Gabriel Juhás[1,2,3], Juraj Mažári[1,2], Tomáš Gažo[2], and Martin Makáň[2]

[1] Faculty of Electrical Engineering and Information Technology
Slovak University of Technology in Bratislava,
Ilkovičova 3, 812 19 Bratislava, Slovakia,
[2] NETGRIF, s.r.o., Blumentálska 12, 811 07 Bratislava, Slovakia
netgrif@netgrif.com, Home page: http://www.netgrif.com
[3] BIREGAL s. r. o., Klincova 37/B, 821 08 Bratislava, Slovakia
biregal@biregal.sk, Home page: http://www.biregal.com

Abstract. Petri nets are the right tool for modelling of control flow of workflow processes. For more accurate reflection of reality it is necessary to extend Petri nets by more components. For this purpose, modelling language Petriflow was created, that adds roles and data variables to Petri nets and maps roles and data variables to transitions. Development of the language is deeply influenced by real requests incoming from customers or developers modelling complex processes. Every feature of the language is based on real life needs of modelling more robust and complex processes. Based on experience each property of the language was abstracted from real life models of processes. In Petriflow it is possible to model control flow of processes via place/transition Petri nets enriched by reset arcs, inhibitor arcs and read arcs. Petriflow language also introduces layer of roles and data variables into nets. The roles define who can assign an enabled transition and who can execute a transition of the net. The relation between data variables can be defined with another property of the language named Actions. In contrast with other modelling languages, Petriflow allows to set visual aspect of a modelled process, such as behaviour and style of presentation for components.

Keywords: Petri net, Petriflow, modelling, roles, data, process

1 Introduction

Petri nets are a perfect tool for modelling processes. Everybody can understand the modelled process with minor knowledge of Petri nets rules. But the situation is different in the commercial sphere. A majority of people, who model processes as their everyday job, consider Petri nets too simple or incapable to grasp the complex nature of real business processes. Petriflow language was created for purpose of taking advantage of the simple nature of Petri nets and extend them to meet any requirements of the real life business modelling. Every feature is

added to the language based on an experience and on requirements from customers. Petriflow language is developed with the rapid development in mind. If a customer requirement to model a feature of a process is out of scope of the current version of the language, the property is analysed in order to understand the nature of the requirement. If the requirement can be generalised, the resulting requirement is abstracted and implemented as a new property of the language. Petriflow language is written in XML format for the most part. Data field actions are the only exception. They are written in a domain specific language based on Groovy programming language. There are different Petri net extensions and Petri net based tools for modelling workflow processes, such as CPN [1] based on Coloured Petri nets [6], Viptool [2], [3], Yasper [4] or ProM [5], to mention just some of them. The question arises why to create another extension of Petri nets. Most of the Petri net extensions are determined to create models and some of them to analyse the models. Models are only the first step in a life-cycle of a business process management (BPM). The main advantage of BPM is that the model can be used to control the workflow process according to the designed model using a workflow engine. The problem of the most existing Petri net extensions is that they were implemented with different aims, mostly to extend the expressiveness of the formalism, or to analyse models, but they do not provide all the information about implementation details needed for the generation of a deployable application, such as resource management, manipulation with data, or behavioural aspects of the graphical user interface. Another problem is with the case generation. Usually, a model obtained via a Petri net can be understood as a general definition of a model of a process, while the single cases can be understood as instances of the process. Using an analogy with object-oriented programming, a model can be understood as a class, while single cases can be understood as objects of that class. In Petri net based modelling tools, the realisation of cases is often done using coloured Petri nets [6]. But in such tools, colours are used both for distinguishing cases from each other as well as for modelling the data of the cases. For the above mentioned reasons, we develop a new language for creating deployable models of workflow processes based on Petri nets. From the very beginning, along with the definition of the Petriflow language, we develop the workflow engine called Netgrif Workflow Management System, where the Petriflow models can be deployed and executed.

2 From Petri nets to Petriflows

As mentioned before, Petriflow language extends Petri nets with other components. As the underlying model, we use place/transition nets enriched by reset arcs, inhibitor arcs and read arcs. The read arcs appear quite necessary in order to model unbounded number of concurrent reading of data in a case. To meet modern business modelling requirements other layers were brought to the language on top of Petri nets. Roles are the first layer to extend Petri nets. Roles layer defines who can fire transitions to which they are bound. Data variables were added as the second layer on top of modelled processes. Data variables rep-

resent all properties of an instance of a process during its life-cycle. To have more control over process instance data, data field actions were added to Petriflow. Actions can define relations or dependencies between data fields in the model of a process or generate values based on a process instance state. All extensions and layers create the right tool for modelling complex, yet simple to understand models of any process that comes to mind.

2.1 Transitions as tasks

In classical Petri nets, firing a transition is one atomic event. During this event a transition consumes tokens from connected input places and produce tokens to output places. The number of consumed and produced tokens is according to the firing rule in original Petri nets. In Petriflow language execution of a transition represent an activity executed by an actor (a user or a system). In the current version of Petriflow language transitions of the underlying net can either represent immediate events (event transitions) or transitions can represent tasks (task transitions) consisting of the assign event, an action which correspond to the state, during which the activity is executed, a cancel event, an optional delegate event and a finish event:

1. Assign event - consumes tokens from input places of the transition in the underlying net.
2. Action - is the state where an actor, who executes the activity, does his job defined by the activity.
3. Cancel event - cancels the task, i.e. produces the consumed tokens back to the input places of the transition in the underlying net.
4. Optional Delegate event - delegates the task to an another actor.
5. Finish event - produces tokens in the output places of the transition in the underlying net.

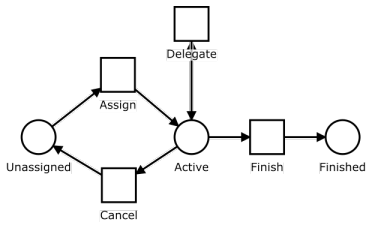


Fig. 1. A subnet modelling a transition representing a task

To illustrate the detailed behaviour of a task transition in the Petriflow language, consider that in the underlying net a task transition has an input place *Unassigned* connected with the task transition by an arc with weight one and an output place *Finished* connected with the task transition by an arc with weight

one. Detailed behaviour of the task transition occurrence representing the task execution can be expressed by a subnet of the underlying Petri net in Figure 1.

3 Role layer in Petriflow

Roles can be defined for Petriflow processes. If a role is associated with an event transition, then it specifies that an actor associated with the role can fire the event transition. In the case of a task transition Petriflow language enables to specify for the role associated with the task transition, whether the actor can fire the assign event, whether the actor can perform the action and fire the cancel event and the finish event or whether the actor (optionally) can delegate the event to be performed by other actor. Assigned or delegated actor has to have a role that authorise to perform the action and to finish the task.

4 Data layer in Petriflow

In Petriflow language the data layer consists of data variables and their binding to transitions of the model of the process. In Petriflow language a data variable is primarily defined by its type, unique id and title. The type attribute of data variable determines the data structure of the variable. For example, if the data type of variable is set to number, the value is stored as a number of the type double. Petriflow language allows to use of primitive data types like number, boolean, as well as standard data types such as date, text as well as more complex types like enumeration, multi-choice, file and table. The language also introduces domain specific types user and case reference. The user type stores the reference to the specific user of the system, who has assigned a process role. The case reference data type is used when it is required to create connection between two different cases. Visual attributes can be also set in the data variable object to more precisely define the representation and behaviour of the data variable in graphical user interface. For example, the attribute placeholder sets a text value of a data field element when no user defined default value is present.

4.1 Binding data fields to task transitions

It is not a rule that every data variable has to be bounded to some task transition. A data variable should be bounded to a transition if it is desired to display its values to the user performing the task transition. A data variable associated to a task transition is called data field of the task transition. Obviously, a data variable can be bounded to several transitions. A data field, i.e. a connection between a data variable and a task transition is implemented as a reference inside of the task transition. The referenced data variable is identified in the reference task transition by its unique id set in the data variable object. In addition to data variable id, reference object has attributes to set data field behaviour and logic inside of the transition. Behaviour attribute defines relation of the data

field to the transition. The attribute values can be hidden, visible, editable, and required. When the data field in a transition is visible user can see the value of the data field but cannot modify it. When one or more data fields in a transition are set as required, this transition cannot be finished until every required data field is filled. Values in the logic attribute are functions. In Petriflow language they are named as Actions. They are executed when the value is changed inside of the referenced data field.

5 Petriflow Actions

Actions are functions executed every time when a value inside of a data field is changed. Actions can be placed into a data variable definition or into a data field logic attribute (i.e. into a data variable reference inside of a task transition). Each action contains two parts. Action variables are defined first. They reference to a data variable or a transition in the process model. Second part consist of keywords that define a desired expression. Values of data variables referenced by action variables are changed according to the evaluated expression. Actions are not written in XML format. Actions implementation is based on Groovy DSL meta language. Groovy allows to define own semantics for domain specific language and then compiles it to an executable code. All actions are compiled after successful import of a model to Netgrif WMS and then saved into a database. Actions are a big advantage of Petriflow language, because they define relations between data fields across whole process model. They modify information stored inside of each case of the model in real time as transitions are fired.

Algorithm 1 Usage of action defined in DSL

```
<data type="enumeration">
    <title>PERIODICITY</title>
    <id>108001</id>
    <values>yearly</values>
    <values>quarterly</values>
    <init>quarterly</init>
    <action trigger="set">
        field: f.this, amount: f.308004, payment: f.308006;
        change payment about {
            if (field.value == "yearly")
                return 0.95*amount as Double;
            if (field.value == "quarterly")
                return amount/4 as Double;
        }
    </action>
</data>
```

For better illustration, consider the data variable object with the action defined in Algorithm 1. The data variable object is an enumeration with two choices, namely *yearly* and *quarterly*, with the initial value *quarterly*. The action works with three action variables, namely *field* which refers to the data variable itself, action variable *amount*, which refers to the data variable with id 308004 and action variable *payment*, which refers to the data variable with id 308006. Consider, that in the data variable referred by action variable *amount* is stored the previously computed amount of an insurance. The action works as follows, if the value *yearly* is set, then 5% discount is given and the value of the data variable referred by action variable *payment* is set to the 95 % of the insurance. If the value *quarterly* is set, then no discount is given and the value of the data variable referred by action variable *payment* is set to the one quarter of the insurance.

6 Conclusion

We have briefly introduced Petriflow language, which is an extension of Petri nets using roles and data variables associated with transitions and functions over data variables called actions. The Petriflow language is suitable for creating deployable models of workflow processes. The further step is to enrich the Petriflow language by the possibility to define process scope (global) variables and by the communication with data passing between cases of different processes either via task transitions or via a constructor.

References

1. M. Beaudouin-Lafon, W. E. Mackay, M. Jensen, P. Andersen, P. Janecek, M. Lassen, K. Lund, K. Mortensen, S. Munck, A. Ratzer, K. Ravn, S. Christensen, K. Jensen: CPN/Tools: A Tool for Editing and Simulating Coloured Petri Nets In: Tools and Algorithms for the Construction and Analysis of Systems, LNCS 2031, pp. 574577, Springer-Verlag, 2001.
2. R. Bergenthum, J. Desel, G. Juhás, R. Lorenz: Can I Execute my Scenario in Your Net? VipTool tells you! In Application and Theory of Petri Nets and Other Models of Concurrency. LNCS 4024, pp. 381390, Springer-Verlag, 2006.
3. J. Desel, G. Juhás, R. Lorenz and C. Neumair: Modelling and Validation with VipTool. In: BPM 2003, LNCS 2678, pp. 380389, SpringerVerlag, 2003.
4. K.M. van Hee, J. Keiren, R. Post, N. Sidorova, J.M. van der Werf: Designing case handling systems. In Transactions on Petri Nets and Other Models of Concurrency I, LNCS 5100, pp. 119133, Springer, Berlin. 2008.
5. B. van Dongen, A.K. Alves de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM framework: A New Era in Process Mining Tool Support. In Application and Theory of Petri Nets 2005, LNCS 3536, pp. 444454. Springer-Verlag, Berlin, 2005.
6. C.W. Gunther, W.M.P. van der Aalst: Modeling the Case Handling Principles with Colored Petri Nets. Proceedings of the Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, 2005, Department of Computer Science, University of Aarhus, PB-576, 211230.

A Simple Prototype of Distributed Execution of Reference Nets Based on Virtual Machines

Daniel Moldt, Jan Henrik Röwekamp, and Michael Simon

University of Hamburg, Faculty of Mathematics, Informatics and Natural Sciences,
Department of Informatics, http://www.informatik.uni-hamburg.de/TGI/

Abstract Fast execution of Petri nets is one goal of Petri net tool providers. For complex models with a large set of concurrent simulation tasks in each simulation engine there are multiple possibilities for a speed up. Beside parallel Petri net execution within one simulation engine distributed execution is another option. In our research group a proposal for the distributed simulation of Reference Nets has been made. The approach connects multiple simulations over a network, but is agnostic to the environment in which these are executed. In this contribution the distributed simulation's scalability by the usage of virtual machines is shown. We use an algorithm to calculate prime numbers to illustrate the performance changes. The costs of communication and synchronization for complex bindings are discussed to find a good trade-off for the decision whether to distribute an execution or not.

Keywords: Petri Nets, Reference Nets, Distributed Simulation, Virtual Machines, Synchronous Channels, Renew

1 Introduction

Distributed execution or distributed simulation of Petri nets is of high interest. Using a concurrent semantics (e.g. step or true concurrency semantics) the option for distributed execution is inherently given. Utilizing this option in the context of Petri nets brings up the major task of net partitioning. Several proposals have been made to implement a distributed simulation, see section 4. To allow for a flexible and expressible partitioning of the net models we follow the Paose-approach [13] to use modeling units that have a behavior with tight application-oriented internal *local* coupling and only loose coupling of larger units (e.g. message exchanging agents or organizations). Reference nets (see section 2) alongside with synchronous channels (see [5,10]) deliver the foundation for it. Together with our object- and agent-oriented modeling paradigm (see [14,8,2]) we have an intuitive separation according to the application oriented structuring of the models. While the default conceptual approach is to use synchronous channels for message exchange, asynchronous message passing without unification can be modeled as well.

In [16] and [17] algorithms and an implementation for distributed synchronous channels based on Java RMI are proposed. It allows a bi-directional synchroniza-

tion across physical machine boundaries using a network, however, without the full arbitrary nesting of variable binding calculations.

In [1] virtual machines[1] are used in general for a single execution run without distribution of the whole execution for a given model. Each net is simulated in isolation. Therefore the modeler has to manage the simulation of these nets.

In this paper we describe a simple prototype to apply technical solutions from the field of VMs to allow the execution of a single net model within several VMs. To the best of our knowledge our solution is the first to implement a synchronized firing of distributed simulated high-level Petri nets based on virtual machines supporting atomic firing of transitions with bi-directional parameter exchange. It is used to demonstrate the potential of distributed simulation for complex and highly concurrent nets where the number of processors is much smaller than the number of possible concurrent tasks.

Section 2 introduces our tool Renew, the basic formalism Reference Nets, their powerful synchronous channels and some relevant facts about virtualization as we consider it here. Our simple example of a primitive algorithm with deterministic runtime is described in Section 3 by discussing the Reference Nets and the technical side conditions as well as the implications when to use our approach. Comparison with other work is done in Section 4, before we conclude in Section 5.

2 Basics

Several modeling concepts in a wider sense are relevant for this contribution. We will briefly sketch the most relevant ones and refer to the respective literature.

Renew, Reference Nets and Synchronous Channels Based on concurrency theory and Petri nets we use Reference Nets [11]. Object-oriented structures (net models) that communicate via synchronous channels [5] are used to support the nets-within-nets paradigm [19]. Introducing new instances of Reference Nets is very similar to instantiating new Java objects. All concepts are supported by our tool Renew [3].

Synchronous channels provide a powerful bi-directional information exchange via unification which makes this concept turing complete in the general case. Transitions can have an inscription that is considered as a synchronous channel. Synchronous channels can be designed and implemented in different ways. A detailed description can be found on the website of Renew (`http://www.renew.de`) and in [11].

In Renew net instances can create new net instances and store them as tokens. These tokens have a reference semantic so a single net instance can be contained in and passed between multiple other net instances. Net instances can communicate with each other via synchronous channels. If a transition is

[1] For the rest of the paper we will use the term *VMs*.

inscribed with an uplink, it can be included in a synchronous firing. The synchronous firing is initiated by a transition that takes a net instance token and call an uplink. The inscription used to call an uplink is called a downlink. Each up- and downlink can provide values and unbound parameters. For a transition to be fired a complete binding of all transitions added by synchronous channels calls must be found. The unification algorithm of Renew allows an arbitrarily number of unifications passing information between all involved instances. This makes a full distribution of the algorithms behind the execution of synchronous channels in Renew very inefficient. In [16] a restricted version that supports distributed execution was presented.

Distributed Simulation and Virtualization Beside the parallel execution within one machine on several processors the distributed execution on multiple machines (each having several processors) is of high interest for better execution of Petri net tools. Relevant work was published e.g. in [6,18,9,4,7,15]. In general this area is under ongoing research in informatics. E.g. virtualization is known since the 60ties and had a boost on new tools and approaches in the last five to ten years. VMs can be used to execute several (virtual) machines on a single machine. The tool that is used for the implementation of this contribution is Virtualbox (see `https://www.virtualbox.org/` for details).

3 Example

In the following a simple proof-of-concept example of distributed execution of reference nets is presented.

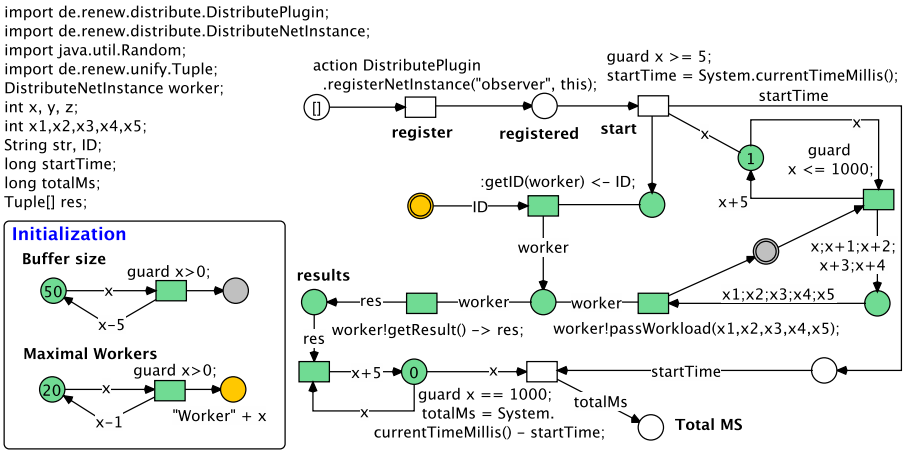
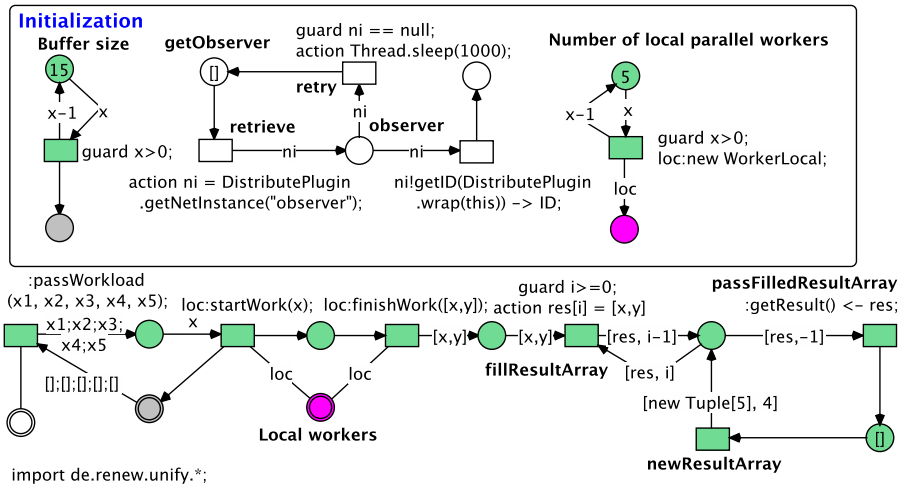


Figure 1: WorkDistributor

Task The sample task pursued here is to perform a naive prime factorization of the natural numbers from 1 to 1000. The algorithm tests for each number, whether it is integer divisible by an iterated counter. Obviously this is not the best possible algorithm for the task, but one that generates some deterministic workload to show the possible speed increase by using distribution. Note, that in this approach the computation for each value is completely independent from all the other values and thus the task is highly parallelizable.

Possible Architectures Technically the distribution itself is realized in two steps. First an instance of the net `WorkDistributor` (see figure 1) will pass work-




```
import de.renew.unify.*;
import de.renew.distribute.DistributePlugin;
import de.renew.distribute.DistributeNetInstance;
import de.renew.net.*;
import java.util.*;
DistributeNetInstance ni;
int x, x1, x2, x3, x4, x5, i;
int y, y1, y2, y3, y4, y5;
String ID;
NetInstance loc;
Tuple[] res;
```

Figure 2: WorkLocalParallelizer

load packages to instances of the net `WorkLocalParallelizer` (see figure 2) running in a different simulation somewhere on the (local) network. Then, the net `WorkLocalParallelizer` will unpack these packages and deliver it to multiple local instances of `WorkerLocal` (see figure 3), which again realize the computation itself. Upon completion of a certain number of items, `WorkLocalParallelizer` will pack a result array and return it to the `WorkDistributor`. For our example, we just count the returned items.

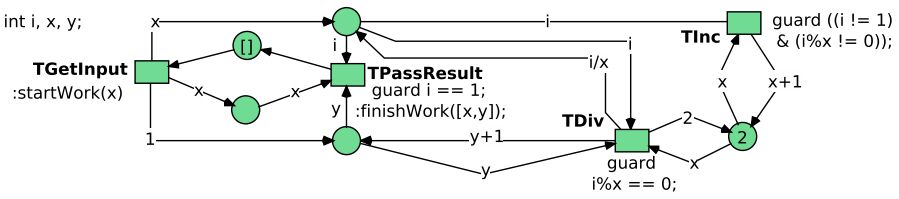


Figure 3: WorkerLocal

Figure 4 shows another possible setup of five VMs running on top of two different physical machines with two distributors for a single task each and three parallelizers with a different amount of workers.

Implemented Architecture For our very basic prototype implementation we chose a setup of a total of five VMs on one bare metal machine. To simulate limitations imposed by insufficient local CPU resources each VM is limited to 1 logical CPU core. The physical host provides 6 physical cores in form of an Intel Core i7-6800K CPU. Each `WorkLocalParallelizer` net instance launches five `WorkerLocal` net instances.

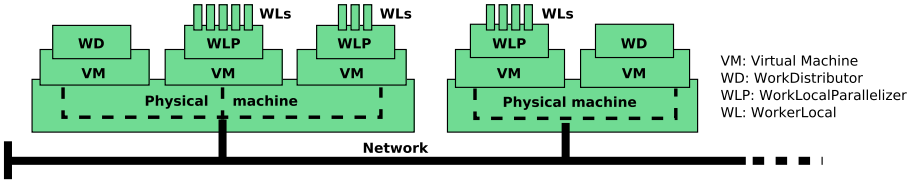


Figure 4: Possible architecture illustration

Results and Discussion Our experimental results have shown, that a distribution to one VM hosting a `WorkLocalParallelizer` net instance took about 129 seconds to complete the task on the sample machine, while a simulation with three VMs only took 84 seconds and one with five 55 seconds. Running the simulation with only one parallelizer directly on the host system (5 cores) provides a running time of about 28 seconds. Therefore the networking and VM overhead only slows down the simulation by roughly factor two in our example, while gaining the possibility to acquire far more processor resources by distribution. The possibility to combine several VMs to speed up the process of simulating reference nets looks promising. On one hand it might be a very useful approach to dynamically distribute workload to other net instances on different VMs, when a local computation slows down. On the other hand the binding cost (time) of

a transition increases with binding complexity and token counts in the related places. Therefore, for a fast distribution easy bindings and low bounded input places are desirable.

4 Related Work

In the past different approaches have been made to allow for a distributed simulation. Several problems arise due to technical restrictions. Best results are often reached on large bare metal machines with many resources (memory, processors, speed of execution of a single statement) and not on distributed execution environments. The main problem is the partitioning, where we use an application/model alignment. Large computing parts of an application usually require more execution power for the corresponding models. At the same time larger units usually have looser coupling. A looser coupling increases the communication cost over the network. In our case this cost is lower due to the partitioning.

Other valuable works (e.g. [6,18,9,4,7,15]) in Petri net simulation / execution have followed more Petri net specific properties which also imply some kind of efficient partitioning. However, this does not scale when a model is extended by new modeling units (net structure). In [12] a distribution of software components was made for Design/CPN and Java applications, but it also did not scale well with the application.

5 Conclusion

The extension of distributed execution of Reference Nets from bare metal machines towards virtual machines has been demonstrated here. Most of the original results from [16] can be used directly. By setting up virtual machines Renew can be tested on a single machine by emulating the distribution of the simulation. Due to the underlying mechanism of RMI calls the costs are comparable to those of other means of network communication. Efficient concurrent and distributed execution is still a challenge. Therefore, modelers can now experiment on virtual machines with distributed simulations to test their design.

The use of new container technology like e.g. Docker (see `https://www.docker.dom`) will give even further possibilities. Especially the transparent distribution of a simulation and load balancing will become much easier. Applications are planned in the field of image processing. We plan to distribute cost intensive calculations according to the application and the used algorithms. This requires an appropriate design as many parametrized experimental executions are necessary. For this we now have a first simple prototype to test the applicability of agent-oriented structuring mechanisms to image processing applications.

References

1. Bendoukha, S.: Multi-Agent Approach for Managing Workflows in an Inter-Cloud Environment. Dissertation, University of Hamburg, Department of Informatics (2016)

2. Cabac, L.: Modeling Petri Net-Based Multi-Agent Applications. Dissertation, University of Hamburg, Department of Informatics (Apr 2010)
3. Cabac, L., Haustermann, M., Mosteller, D.: Renew 2.5 - towards a comprehensive integrated development environment for petri net-based applications. In: Kordon, F., Moldt, D. (eds.) PETRI NETS 2016, Toruń, Poland, June 19-24, 2016. Proceedings. LNCS, vol. 9698, pp. 101–112. Springer (2016)
4. Chiola, G., Ferscha, A.: Distributed simulation of Petri nets. IEEE Parallel Distrib. Technol. 1(3), 33–50 (Aug 1993)
5. Christensen, S., Hansen, N.: Coloured Petri nets extended with channels for synchronous communication. In: Valette, R. (ed.) ICATPN. LNCS, vol. 815, pp. 159–178. Springer (1994)
6. Hauschildt, D.: A Petri net implementation. Fachbereichsmitteilung FBI-HH-M-145/87, University of Hamburg, Department of Computer Science (1987)
7. Kaim, W.E., Kordon, F.: An integrated framework for rapid system prototyping and automatic code distribution. In: Proceedings of RSP, Grenoble, France. pp. 52–61. IEEE (1994)
8. Köhler, M., Moldt, D., Rölke, H.: Modelling mobility and mobile agents using nets within nets. In: van der Aalst, W., Best, E. (eds.) ICATPN. LNCS, vol. 2679, pp. 121–139. Springer (2003)
9. Kordon, F.: Prototypage de systèmes parallèles à partir de réseaux de Petri colorés, application au langage Ada dans un environment centralisé ou réparti. Dissertation, Université P & M Curie (May 1992)
10. Kummer, O.: Simulating synchronous channels and net instances. In: Desel, J., Kemper, P., Kindler, E., Oberweis, A. (eds.) 5. Workshop AWPN. pp. 73–78. No. Forschungsbericht Nr. 694, Fachbereich Informatik, Universität Dortmund (1998)
11. Kummer, O.: Referenznetze. Logos Verlag, Berlin (2002)
12. Kummer, O., Moldt, D., Wienberg, F.: Symmetric communication between coloured Petri net simulations and Java-processes. In: Donatelli, S., Kleijn, J. (eds.) ICATPN. LNCS, vol. 1639, pp. 86–105. Springer (Jun 1999)
13. Moldt, D.: PAOSE: A way to develop distributed software systems based on Petri nets and agents. In: Barjis, J., Ultes-Nitsche, U., Augusto, J.C. (eds.) Proceedings of MSVVEIS'06, Paphos, Cyprus 2006. pp. 1–2 (2006)
14. Moldt, D., Wienberg, F.: Multi-agent-systems based on coloured Petri nets. In: Azéma, P., Balbo, G. (eds.) ICATPN. pp. 82–101. No. 1248 in LNCS, Springer, Berlin Heidelberg New York (1997)
15. Pommereau, F., de la Houssaye, J.: Faster simulation of (coloured) petri nets using parallel computing. In: van der Aalst, W.M.P., Best, E. (eds.) PETRI NETS. LNCS, vol. 10258, pp. 37–56. Springer (2017)
16. Simon, M.: Concept and Implementation of Distributed Simulations in Renew. Bsc thesis, University of Hamburg, Department of Informatics (2014)
17. Simon, M., Moldt, D.: Extending Renew's algorithms for distributed simulation. In: Cabac, L., Kristensen, L.M., Rölke, H. (eds.) PNSE'16. CEUR Workshop Proceedings, vol. 1591, pp. 173–192. CEUR-WS.org (2016)
18. Taubner, D.: On the implementation of Petri nets. In: Rozenberg, G. (ed.) Advances in Petri Nets 1988. LNCS, vol. 340, pp. 418–439. Springer (1988)
19. Valk, R.: Petri nets as token objects - an introduction to elementary object nets. In: Desel, J., Silva, M. (eds.) 19th International Conference on Application and Theory of Petri nets, Lisbon, Portugal. pp. 1–25. No. 1420 in LNCS, Springer, Berlin Heidelberg New York (1998)

Prototypical Graphical Simulation Feedback in Reference Net-Based Domain-Specific Languages within a Meta-Modeling Environment

David Mosteller, Michael Haustermann, and Daniel Moldt

University of Hamburg, Faculty of Mathematics, Informatics and Natural Sciences,
Department of Informatics, http://www.informatik.uni-hamburg.de/TGI/

Abstract The development of domain specific models requires apropriate tool support for modeling and execution. Meta-modeling facilitates solutions for the generation of modeling tools from abstract language specifications. The RMT-approach applies transformational semantics using Petri net formalisms as target languages in order to produce quick results for the agile development of modeling techniques. The problem with transformational approaches is that the inspection of the system during execution is not possible in the original representation.
We present a concept for providing simulation feedback for DSML that are developed with the RMT-approach on the basis of meta-models and translational semantics using Petri nets. Details of the usage of this new approach are illustrated by some well known constructs of BPMN models.

Keywords: Meta-Modeling, BPMN, Petri Nets, Reference Nets, Simulation, Graphical Feedback

1 Introduction

The construction of abstract models is an essential part of software and systems engineering. Meta-modeling provides a conceptual basis to develop modeling languages that are tailored to satisfy the demands of specific application domains. Tools may be generated from the language specifications to support the modeling process.

We present a concept for the rapid prototyping and direct simulation (and simulation feedback) of domain specific modeling languages (DSML) within the RENEW simulation environment. The focus of this contribution is on the integrated simulation and graphical feedback of the executed language during simulation. With our contribution we combine and advance two branches of our current research: First, the development of domain specific modeling languages using the RENEW Meta-Modeling and Transformation Framework (RMT) [9] and second, the provision and coupling of multiple modeling perspectives during execution within RENEW [8].

The approach provided by the RMT framework supports the rapid prototypical development of domain specific modeling languages with Petri net-based

semantics. The RMT-approach is based on the idea of providing translational semantics for the modeling language in development (source language) through a mapping of its constructs to a target language. The latter is implemented using net components, which are reusable and parametrizable code templates – quite comparable to code macros – modeled with a Petri net formalism.

We choose the Reference Net formalism as a target language, but we are not restricted to this formalism and we intend to elaborate on different variations and (Petri net related) formalisms in the future. Reference Nets combine concepts of object-oriented programming and Petri net theory. They are well-suited as a target language, because of their concise syntax and broad applicability. With the presented solution Reference Nets provide the operational semantics for the target language and the simulation events are reflected in the source language during execution.

Tool support for our approach comes from RENEW, which provides a flexible modeling editor and simulation environment for Petri net formalisms with development support for the construction of Reference Net-based systems [2].

2 DSML Tools with Graphical Simulation Feedback

The RENEW Meta-Modeling and Transformation Framework (RMT) [9] is a conceptual model-driven framework for the agile development of DSML. It follows concepts from software language engineering [6]. The specification of a language and a corresponding modeling tool may be derived from a set of models, which are defined by the developer of a modeling technique. A meta-model defines the structure (abstract syntax) of the language, the concepts of its application domain and their relations. The visual instances (concrete syntax) of the defined concepts and relations are provided using graphical templates from the repertoire of RENEW's modeling constructs. They are configurable by stylesheets and complemented with icon images to facilitate the generation of a modeling tool that nicely integrates into RENEW's development environment. A generic compiler is implemented in RENEW to provide translational semantics for the modeling technique using Reference Nets. On this basis the generated technique may be executed within RENEW's simulation engine.

Up to now there was no (sufficient) user feedback when executing models of the generated modeling technique. In this paper we address the extension of our tool framework to allow feedback from the simulation of the underlying Petri net. The main idea is that within the domain models the internal state (resp. marking) of the Petri net is reflected directly in the domain of the generated modeling technique. This allows for an adaptive feedback depending on the translational semantics for each generated modeling technique individually.

A conceptual image from the simulation of two modeling techniques that interact with each other is displayed in Figure 1. The image originates from [8, p. 8] where we presented a concept for multi-formalism simulation with the synchronization of multiple modeling techniques on the basis of Reference Nets. The presented solution sketched the idea of providing feedback into a DSML

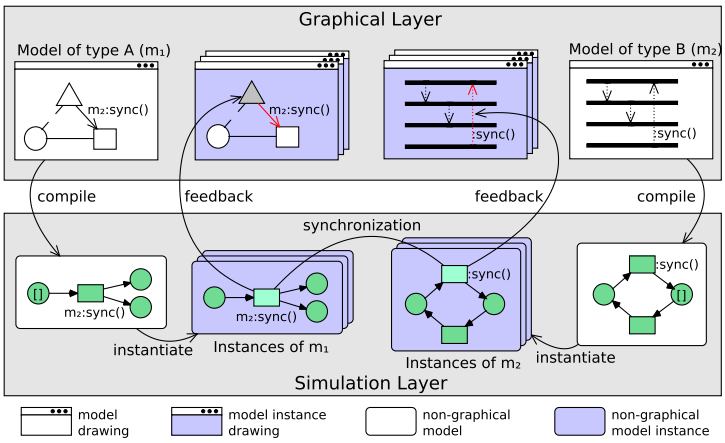


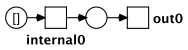
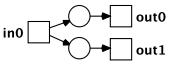
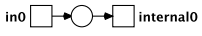
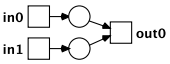
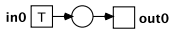
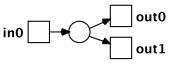
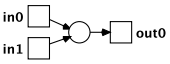
Figure 1: Conceptual model of the model synchronization from [8, p. 8].

but the realization was specific for a finite automata modeling tool. With our current work this idea is generalized to facilitate feedback into principally any DSML that is developed with the RMT-approach using model-driven development. This opens up the possibility to develop and research different simulation semantics or modes of simulation for these DSML. However, there exist numerous ways to realize an integrated simulation. Some of them are covered by previous results: Sedrakyan et al. present a model-driven graphical feedback in a model-to-code transformation with the goal of validating semantic conformance especially for novice modelers [10]. They focus on feedback for errors that occur in the compiled system rather than a complete interactive inspection of the execution. Other approaches aim at providing interactive visual behavior for domain specific modeling tools by using model-driven techniques [1,3].

3 BPMN Example

In the following we present a solution for the simulation of a selected subset of BPMN. As required by the RMT-approach, the constructs of a DSML receive abstract and concrete syntax by model-based specifications, which are both omitted here for the reason of space (the complete example can be found in [9, p. 12]). The semantic mapping of the BPMN constructs is implemented as a slight variation of the mapping to Petri nets from Dijkman et al. [4] and provided using net components as displayed in Table 1. In contrast to Dijkman et al., instead of using place fusion on the connected components, we have chosen to define the BPMN constructs as transition bordered components, which are connected by place bordered components for relations. The BPMN constructs have *ports* that identify connectors, where relations may be connected and the components are correspondingly tagged by labels (`in0`, `in1`, `out0`, `out1`, ...). On the one hand this technically facilitates a straight forward implementation by connecting adja-

Table 1: Adjusted mapping of BPMN and Reference Net constructs [9]

BPMN	Reference Net	BPMN	Reference Net
			
			
			
			

cent components with plain Petri net arcs on the other hand it has the advantage of being able to attach Petri net semantics to the relations as well. The proposed transformation of modeling constructs of a source language into net components is related to the static hierarchy concepts, such as the ones for Coloured Petri Nets [5] and other formalisms. The research results in this area can be transferred to our approach. The capabilities of Reference Nets regarding dynamic hierarchies can be helpful to the development of more complex semantics.

Figure 2 shows a snapshot from the simulation of a BPMN model. The topmost part shows Renew's main window with context menus, editor tool bars and the status bar. In the middle are two overlapping windows, which are arranged to complement each other in order to show a view of the modeled BPMN from two perspectives. Visible to the right is a part of the template drawing that was modeled using the BPMN constructs from the BPMN toolbar. The task reading *finish* is selected and shows the blue circled ports of the figure, where sequence flow relations are connected.

The left window contains an instance of this model and was created from that template. The simulation is paused right after the *assign* task was completed and the highlighted `xor-decision-gateway` is currently activated. This is reflected by the black token on the place in the corresponding Petri net in the lowermost part of Figure 2. The subsequent simulation step may be invoked by right-clicking on the activated `xor-decision-gateway` figure in the BPMN model. In this case, one of the possible alternatives is chosen nondeterministically. All actions and executions are performed by the underlying Petri nets, which therefore determine the semantics of the domain specific language model, while the interaction of the user is performed through the BPMN model. The behavior may be customized by providing alternative net components that may contain colored tokens, variables, inscriptions, synchronous channels, etc. The GUI interaction is provided with the Rmt integration.

For the demonstration of the underlying model the green area marks an excerpt of the BPMN model that corresponds to a part of the Petri net model, which was generated using the semantic mapping from Table 1. For the presen-

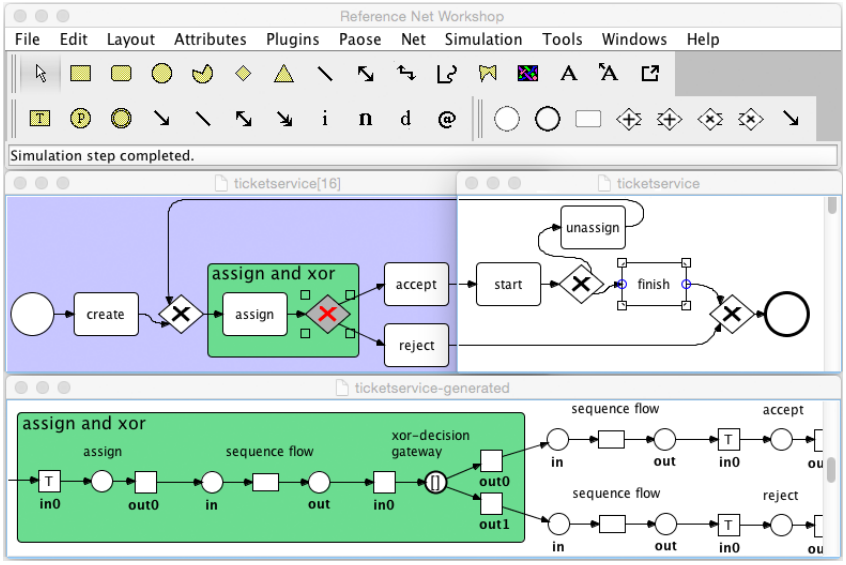


Figure 2: The lifecycle of tickets in a issue tracking system, as BPMN [9, p. 12].

tation in this paper, the Petri net model was created by hand, the generated Petri net that actually performs the simulation has no visual representation at all. This is a design decision to maintain the ability to execute these models without graphical feedback in server mode, which is essential to building large scale applications from net models.

In all, this allows to provide graphical feedback in the BPMN model by reflecting simulation events from the simulated (invisible) Petri net to the above layer. With the current solution, a construct from the source model is highlighted if the corresponding semantic component from the Petri net contains a token. Of course, there are many ways to interpret the semantics of such simulation feedback. We are working on means to conceptualize the visual behavior of simulations based on translational semantics and Petri nets.

4 Conclusion

In this contribution we present a concept for providing simulation feedback for DSML that are developed with the RMT-approach on the basis of meta-models and translational semantics using Petri nets. As a target formalism Reference Nets are applied, which benefit from powerful modeling capabilities, Java integration, the underlying concurrency theory and the RENEW integrated development and simulation environment. The proposed transformation to a powerful (turing complete) formalism is attractive on the one hand, because the mentioned advantages of this formalism may be exploited. On the other hand, the possibilities to perform formal analysis are restricted due to the complexity of the

formalism. In the future we may benefit from the presented conceptual approach by conceptualizing the transformation and restrictions of the target language, e.g. to Place / Transition nets, to perform analysis. The flexibility with respect to the formalisms opens up the possibility to apply a whole array of methods from low-level analysis – e.g. using Renew's integration of LoLA [7] – to normal software engineering validation like unit testing [11]. Based on our new feature for visual feedback directly in the simulated domain specific model we provide an improved experimentation environment to have interactive experiences with the behavior of newly designed domain specific languages, without extra work for animating the models.

References

1. Biermann, E., Ehrig, K., Ermel, C., Hurrelmann, J.: Generation of simulation views for domain specific modeling languages based on the Eclipse modeling framework. In: 2009 IEEE/ACM ASE. pp. 625–629 (2009)
2. Cabac, L., Haustermann, M., Mosteller, D.: Renew 2.5 - towards a comprehensive integrated development environment for Petri net-based applications. In: Kordon, F., Moldt, D. (eds.) PETRI NETS 2016, Toruń, Poland, June 19-24, 2016. Proceedings. LNCS, vol. 9698, pp. 101–112. Springer-Verlag (2016)
3. Combemale, B., Crégut, X., Giacometti, J.P., Michel, P., Pantel, M.: Introducing Simulation and Model Animation in the MDE Topcased Toolkit. In: 4th European Congress Embedded Real Time Software (ERTS). p. http://www.erts2008.org/. Toulouse, France, France (Jan 2008)
4. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Information and Software Technology 50(12), 1281–1294 (2008)
5. Huber, P., Jensen, K., Shapiro, R.M.: Hierarchies in coloured Petri nets. In: Rozenberg, G. (ed.) Advances in Petri Nets 1990 [10th International Conference on Applications and Theory of Petri Nets, Bonn, Germany, June 1989, Proceedings]. LNCS, vol. 483, pp. 313–341. Springer (1989)
6. Kleppe, A.: Software Language Engineering: Creating Domain-Specific Languages Using Metamodels. Pearson Education (Dec 2008)
7. LoLA: A Low Level Petri Net Analyzer. `http://service-technology.org/lola/`
8. Möller, P., Haustermann, M., Mosteller, D., Schmitz, D.: Simulating multiple formalisms concurrently based on reference nets. In: Moldt, D., Cabac, L., Rölke, H. (eds.) PNSE'17, Zaragoza, Spain, June 25-26, 2017. Proceedings. CEUR Workshop Proceedings, vol. 1846, pp. 137–156. CEUR-WS.org (2017)
9. Mosteller, D., Cabac, L., Haustermann, M.: Integrating Petri net semantics in a model-driven approach: The Renew meta-modeling and transformation framework. Transactions on Petri Nets and Other Models of Concurrency 11, 92–113 (2016)
10. Sedrakyan, G., Snoeck, M.: Enriching model execution with feedback to support testing of semantic conformance between models and requirements - design and evaluation of feedback automation architecture. In: Calabrò, A., Lonetti, F., Marchetti, E. (eds.) AMARETTO@MODELSWARD 2016, Rome, Italy, February 19-21, 2016. pp. 14–22. SciTePress (2016)
11. Wincierz, M.: A tool chain for test-driven development of reference net software components in the context of CAPA agents. In: Moldt, D., Cabac, L., Rölke, H. (eds.) PNSE'17, Zaragoza, Spain, June 25-26, 2017. Proceedings. CEUR Workshop Proceedings, vol. 1846, pp. 197–214. CEUR-WS.org (2017)