Fakultät für
Mathematik und Informatik

Robin Bergenthum and Ekkart Kindler (Eds.)

Algorithms and Tools for Petri Nets

Proceedings of the Workshop AWPN 2019

Hagen, Germany, October 10–11, 2019
Tagungsband

FernUniversität in Hagen
Fakultät für Mathematik und Informatik
IZ - Universitätsstraße 1
58097 Hagen, Germany

Contents

Preface

25 years ago, Jörg Desel was the driving force behind the first German AWPN workshop: "Algorithmen und Werkzeuge für Petrinetze", which is German for "Algorithms and Tools for Petri Nets". This event turned into a successful series of workshops, which from the beginning was organized by the Special Interest Group "Petri nets and related system models" of the German Gesellschaft für Informatik (GI). The main idea of the workshop is to focus on discussion! The workshop is informal and low-budget.

This year, the AWPN 2019 took place at the FernUniversität Hagen on October 10-11 – to the day 25 years after the first AWPN workshop. We are very happy that Jörg Desel gave an invited talk on the question whether a single transition can stop an entire Petri net. Furthermore, we had an invited session dedicated to Jörg Desel on the occasion of his 60th birthday where Ekkart Kindler, Andreas Oberweis, Gabriel Juhas, Friedrich Steimann, Jetty Kleijn, Laure Petrucci and Wil van der Aalst reflected on Jörg‘s extensive work for and contributions to the Petri net community and, more generally, the modelling community.

The topics of the workshop are analysis, simulation, visualization, and synthesis of Petri nets and related models. Theory, applications, and tools are welcome and were presented at AWPN 2019. Papers did not undergo a detailed reviewing process, but were inspected for relevance with respect to the topics of AWPN 2019. 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 2019

Analyzing and Improving the Efficiency of Current Synthesis Approaches Using Wrong Continuations

Johannes Metzger

University of Augsburg, Germany
johannes.metzger@informatik.uni-augsburg.de

Abstract. In this paper, we analyze and improve the so-called synthesis based modeling approach. Taking a look at the literature, compact regions together with wrong continuations define the state-of-the-art algorithm to synthesize a Petri net from a set of executions given by a partial language. Recently, we have lifted the notion of compact regions from Hasse diagrams to Prime event structures, making the algorithm even more viable. Still, in the context of structures with many events (e.g. in Process Mining), the synthesis approach is slow compared to existing algorithms delivering approximate results.
We aim to further improve the synthesis approach by investigating new ideas concerning the traversal of the tree of wrong continuations. A concept of so-called rich continuations is presented.

1 Introduction

Petri nets have an intuitive graphical representation, formal semantics, and are able to express concurrency among the occurrence of actions [1, 6, 12, 13]. However, modeling a Petri net form scratch is a costly and error-prone task [1, 11]. 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. In short, 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. It is often easier to come up with a set of Hasse diagrams and synthesize a Petri net than to produce the Petri net model from scratch.

However, in some areas of interest the synthesis algorithm exposes shortcomings when compared to other toolkits. For example, in the area of Process Mining, there are a lot of Process Discovery techniques which are fundamentally faster in model generation than the synthesis algorithm when used on big sets of data (e.g. [15], [8], [9]).
In recent work, research has been conducted on how to improve the efficiency of the synthesis approach. Important results of this research include the introduction of compact regions [3–5] and the application of labeled Prime event structures [2, 10], which both substantially increase the performance of the synthesis algorithm.

In this paper, our goal is to build a foundation for further improvements of the synthesis approaches using wrong continuations. We introduce the notion of *rich continuations* which are helpful in analyzing the efficient traversal of the tree of wrong continuations.

The paper is organized as follows: Section 2 introduces Petri nets, the synthesis problem, and compact regions for labeled Prime event structures. In Section 3, we recall the concept of wrong continuations. And finally, in Section 4, we discuss ways to order the set of wrong continuations and introduce the notion of rich continuations.

2 Preliminaries

Let $(V, <)$ be some acyclic and finite graph. We denote the transitive closure of an acyclic and finite relation $<$ by $<^*$, and the skeleton of $<$ by $<^\diamond$. The skeleton of $<$ is the smallest relation $\lhd$ such that $\lhd^* = <^*$ holds. $(V, <^\diamond)$ is called the Hasse diagram of $(V, <)$.
Furthermore, we model business processes by p/t-nets [7, 12, 13].

Definition 1 (Place/Transition Net). *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.*

Petri nets are able to express concurrency of the occurrences of transitions. However, firing sequences are not able to capture or describe such behavior. Instead, we use state-of-the-art compact tokenflows [4, 5] as characterization of the partial language of a Petri net. They are defined on the underlying Hasse diagrams, describing the potentially concurrent behavior.

Definition 2 (Hasse Diagram). *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 belongs to 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 [4, 5].

Definition 3 (Compact Tokenflow). *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 $L(N) = \{run^* \mid run \text{ is compact valid for } N\}$.

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

Definition 4 (Specification). *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 (The Synthesis Problem). *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 $N' : L(N) \backslash L(N') \neq \emptyset \Longrightarrow S \not\subseteq L(N')$.*

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 (Labeled Prime Event Structure). *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.

Such a labeled Prime event structure is shown in Figure 1. Each of the colors represents a different consistency set of the same labeled Prime event structure. For example, $C_{red} = \{1, 3\}$ is a consistency set of the given labeled Prime event structure.

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 (LPES-Specification). *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.*

Making use of the state-of-the-art theory of compact regions, we apply this theory to the notion of labeled Prime event structures. First, we need to introduce compact tokenflows on labeled Prime event structures.

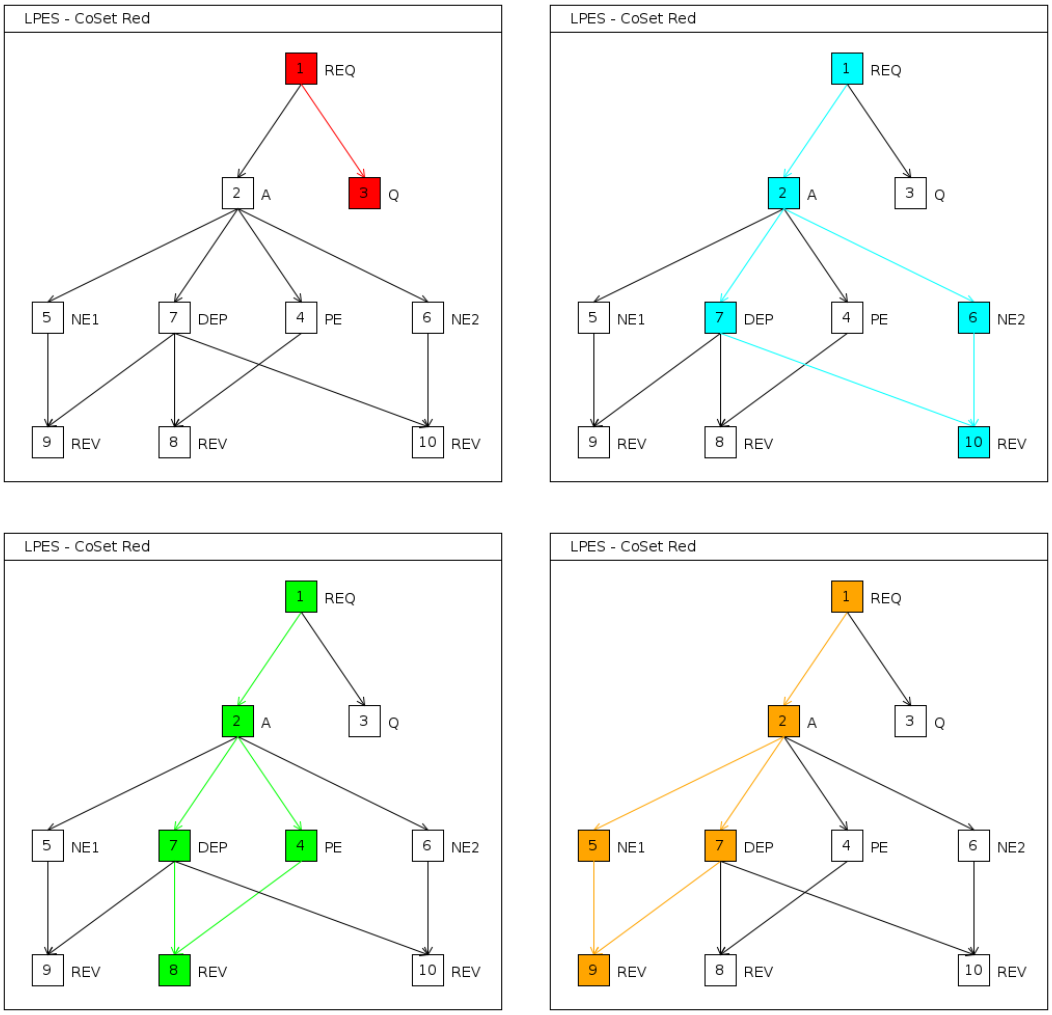

Fig. 1. A labeled Prime event structure with four different consistency sets.

Definition 8 (Compact Tokenflow for LPES). *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 $p \in P$.

Now we are able to define compact regions for labeled Prime event structures.

Definition 9 (Compact Region for LPES). *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_{min}. A function $r : (V_{min} \cup <) \cup (T \times \{p\}) \cup (\{p\} \times T) \cup \{p\}) \to \mathbb{N}$ is a compact region for pes iff $r|_{(V_{min} \cup <)}$ is compact valid for p in $(\{p\}, T, r|_{(T \times \{p\}) \cup (\{p\} \times T)}, r(p))$. p is called the place defined by v.*

3 Separation Representation and Wrong Continuations

In this section, we recall the notion of wrong continuations and separation representations with respect to LPES-based specifications [2].

In the following, we denote by $\mathbf{r}$ a compact region as introduced in Def. 9 and by $p_{\mathbf{r}}$ the place defined by $\mathbf{r}$. An idea to get a finite representation, that is a finite set of regions representing the infinite set of all regions (and thereby a finite set of places in the resulting Petri net), is to separate behavior specified by lpes from behavior not specified by lpes by such a finite set of regions (see [14] in the context of partial languages). The resulting representation is called *separation representation*. To derive a separation representation, an appropriate finite set $\{hd_1, \ldots, hd_n\}$ of Hasse Diagrams with the following properties is defined:

- The Hasse diagrams hd_i are no runs specified by lpes
- Each Hasse diagram hd_i extends a run specified by lpes by one event.

Then for each hd_i one tries to find a region $\mathbf{r}$ such that $p_{\mathbf{r}}$ prohibits hd_i (that means hd_i is not a run w.r.t. $p_{\mathbf{r}}$). If such a region exists, $p_{\mathbf{r}}$ is added to the separation representation. The Hasse diagrams hd_i are called *wrong continuations*. The aim is to define them in such a way that an exact solution of the synthesis problem exists if and only if each wrong continuation can be prohibited by a place. A solution (N, m_0) is an exact solution, if it does not have runs which are not specified by lpes.

Formally we split a wrong continuation into a prefix belonging to the partial language of lpes, a subsequently enabled step of transitions and an additional transition which should be prohibited.

Definition 10 (Wrong Continuation). *Let $\mathrm{lpes} = (E, \mathrm{Con}, \prec, l)$ be a finite LPES over a finite alphabet of transition names T.*

A wrong continuation *of* lpes *is a triple (C, τ, t), where*

- *C is a left-closed consistency set C of* lpes.
- *There is a maximal consistency set D of* lpes *with $C \subseteq D$, $\tau \leq l(S_D(C))$ and $\tau(t) = l(S_D(C))(t)$ (τ may be the empty multiset).*
- *There is no maximal consistency set D' of* lpes *with $C \subseteq D'$, $\tau \leq l(S_{D'}(C))$ and $\tau(t) < l(S_{D'}(C))(t)$.*

We call hd_C the prefix *and τ the* follower step *of the wrong continuation. The set $S_D(C) = \{v \in D \setminus C \mid w \prec v \implies w \in C\}$ is the set of the direct successors of C in D.*

Consider a wrong continuation (C, τ, t). Our aim is to compute a compact region $\mathbf{r}$ such that after the occurrence of the prefix hd_C the marking of $p_{\mathbf{r}}$ does not enable the transition step $\tau + t$. This can directly be expressed by a linear constraint using the variables of the linear inequation system defining compact regions. Then, a non-negative

integral solution can be computed which minimizes a given linear target function ϕ. More detailed steps on how to apply wrong continuations on labeled Prime event structures are shown in [2].

4 Analyzing and Improving the Synthesis-based Modeling Approach

4.1 Determining the Order of Wrong Continuations

In general, a place does not prohibit only one wrong continuation. If a wrong continuation w is prohibited by a place, which was already computed previously, it is not necessary to compute a separate solution for w. Consider two wrong continuations w_1, w_2. It is possible that a solution p_1 prohibiting w_1 also prohibits w_2, but that a solution p_2 prohibiting w_2 does not prohibit w_1. In such a case, it is better to consider w_1 first, since this order leads to a net with less places. That means, the order in which wrong continuations are considered, the so-called *wct-ordering*, has an influence on the synthesis result [14].

In previous experiments [2], we considered several different orderings $\leq_{wct}$ based on the following definitions for wrong continuations $w_1 = (C_1, \tau_1, t_1), w_2 = (C_2, \tau_2, t_2)$:

- $w_1 <_C w_2 :\Leftrightarrow |C_1| < |C_2|$ (can be used to consider small prefixes before big prefixes, or vice versa)
- $w_1 <_{dC} w_2 :\Leftrightarrow d(C_1) < d(C_2)$, where $d(C) := max\{d(e_0, e_i) \mid e_i \in C\}$ (can be used to consider short prefixes before long prefixes, or vice versa)
- $w_1 <_\tau w_2 :\Leftrightarrow |\tau_1| < |\tau_2|$ (can be used to consider small follower steps before big follower steps, or vice versa)

These components can be combined in different ways to derive orderings of wrong continuations for the synthesis algorithm. In our previous experiments [2], we identified the following wct-ordering as a good choice:

$$\leq_{wct} = <_{dC} \cup (=_{dC} \cap <_C) \cup (=_C \cap =_{dC} \cap <_\tau)$$

Note that this is a partial order. Unordered wrong continuations are ordered randomly by the algorithm.

4.2 A Theory of Rich Continuations

In order to significantly improve the efficiency of the underlying algorithm, we need to be able to traverse the tree of prefixes and wrong continuations as efficiently as possible. As stated in the last section, we aim to efficiently traverse these trees by intelligently selecting specific paths and continuations first.
Intuitively, a wrong continuation forbidding the addition of a single action t to a given prefix, also forbids adding more than one t. More sophisticated but still intuitive rules are [2]:

1. If (C, τ, t) is a wrong continuation and a place p forbids the step $\tau + t$ after the execution of hd_C, then p also prohibits each step $\tau' + t$ after the execution of hd_C for $\tau \leq \tau'$. In this case, the wrong continuations of the form (C, τ', t) need not be considered.

2. For two prefixes hd_C and $hd_{C'}$ with $l(C) = l(C')$, there holds: after the execution of hd_C, a step $\tau + t$ can be prohibited if and only if it can be prohibited after the execution of $hd_{C'}$. Considering several wrong continuations with such prefixes, their follower steps can be combined.

Building up on this intuition, we want to differentiate between three types of wrong continuations: *weak*, *strong*, and *rich continuations*. Beforehand, when studying wrong continuations, we have to make some assumptions about the chosen synthesis framework:

1. *Target Function*: Firstly, our definition of weak, strong, and rich continuations will be dependent on a fixed target function. When discovering patterns inherent to the structure and composition of wrong continuations, we take the chosen target function into account and determine the grade of dependency.
2. *WCT-Ordering*: Secondly, for the purpose of discovering relationships between wrong continuations, we distinguish between fixed and non-fixed wct-orderings when examining the set of wrong continuations. While the analysis of fixed orderings is relevant when determining a best possible wct-ordering (w.r.t to a given target function), the analysis of non-fixed orderings allows us (more generally) to examine the relationship between individual wrong continuations (again, w.r.t. to a given target function).

Definition 11 (Rich Continuation). *Let (C, τ, t) be a wrong continuation of a LPES $\mathrm{lpes} = (E, \mathrm{Con}, \prec, l)$ and W be the set of all possible wrong continuations of $lpes$. (C, τ, t) is called:*

- *a* rich continuation *of $lpes$ iff $\forall w \in W \backslash (C, \tau, t) : w$ does not forbid (C, τ, t) w.r.t to a given target function ϕ.*
- *a* strong continuation *of $lpes$ iff $\exists w, v \in W \backslash (C, \tau, t) : w$ forbids $(C, \tau, t) \wedge (C, \tau, t)$ forbids v w.r.t to a given target function ϕ.*
- *a* weak continuation *of $lpes$ iff $\forall w \in W \backslash (C, \tau, t) : (C, \tau, t)$ does not forbid $w \wedge \exists v \in W : v$ forbids (C, τ, t) w.r.t to a given target function ϕ.*

We say a wrong continuation w forbids *another wrong continuation v iff the place resulting from forbidding w also prohibits the execution of the wrong continuation v in the synthesized p/t-net.*

Obviously, when traversing the set of all possible wrong continuations, we want to prioritize on rich continuations and neglect weak continuations. In general, for labeled Prime event structures with many events, the set of strong continuations is the biggest set of the given three. This means, we have three goals in mind:

1. Find a strategy to easily determine the set of rich continuations. When calculating places for the final p/t-net, use these continuations first.
2. Exclude the set of weak continuations from our tree traversal.
3. Determine a best possible ordering for the set of strong continuations.

Assessing the best possible ordering is not easy because it depends on the given labeled Prime event structure.

Strictly speaking, rich continuations are rare due to their tight definition. In most scenarios, we want to relax the definition of rich continuations to better understand the relationship between wrong continuations. Among others, we are evaluating these ideas:

1. Analyze the binary relation emerging from the definition of ”forbids” in Def. 11. For example, we want to evaluate properties like symmetry and transitivity.
2. Relax the definition of rich continuations: define a continuation w *with degree* $i (i \geq 0)$ such that there is no wrong continuation v forbidding w with $|w - v| > i$ where the distance between two wrong continuations is defined using the ”forbids” relation. For example, assuming three wrong continuations w, v, u with w forbids v, v forbids u, and v forbids w, the distance between w and v is 1 whereas the distance between w and x is 2 if x does not forbid w. Presumably, rich continuations with a lower degree are of more importance than those with a higher degree.
3. Define place-equivalence on wrong continuations and examine the relation between place-equivalence and the ”forbids” relation.

Place-equivalence of wrong continuations can be defined on the basis of compact regions which directly relate to places.

Definition 12 (Place-Equivalence). *Two wrong continuations are called* place-equivalent *when they yield the same place in the resulting p/t-net w.r.t to a given target function* ϕ*.*

Finally, we have developed an auxiliary tool for visualizing the relationship between individual wrong continuations.
In Figure 2, each node represents a wrong continuation of the labeled Prime event structure in the running example of this paper (see Figure 1). Red nodes depict assumed weak continuations, the blue node, on the other hand, represents an assumed rich continuation, accordingly. For the rich continuation, a short textual labeling is given which describes the continuation (i.e. the affiliated prefix P, follower step F, and the accompanying wrong event). The depicted graphic only shows those relations between individual wrong continuations which occurred after a one fixed run of the underlying synthesis algorithm. Additional relations can easily be added when analyzing the whole set of wrong continuations independent of the wct-ordering.

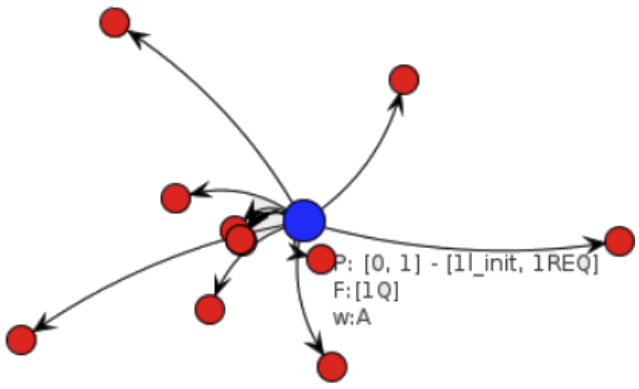

Fig. 2. Visualization of wrong continuations for the running example in Figure 1.

5 Conclusion

In this paper, we have analyzed bottlenecks of current synthesis approaches using wrong continuations and suggested a new foundation for improving their efficiency. In practice, this is especially needed for labeled Prime event structures with a huge number of events. More concretely, we have introduced a theory of rich continuations as a means of improving the traversal of the tree of wrong continuations.

In the future, we aim to provide traversal strategies suitable for all use cases, making use of the relationship between weak, strong, and rich continuations. We plan to further investigate the binary relation emerging from wrong continuations when considering the "forbids" definition and compare the properties of this relation to place-equivalence. Furthermore, we are looking for patterns obtained by investigating the relaxed definition of rich continuations.

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] Lorenz, R.; Metzger, J.; Sorokin, L.: *Synthesis of bounded Petri Nets from Prime Event Structures with Cutting Context using Wrong Continuations*. ATAED@ Petri Nets/ACSD. 2017.

[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] Weijters, A. J. M. M.; Ribeiro, J. T. S.: *Flexible heuristics miner (FHM)*. IEEE symposium on computational intelligence and data mining (CIDM). IEEE, 2011.

[9] Leemans, Sander J. J.; Poppe, E.; Wynn, M. T.: *Directly Follows-Based Process Mining: Exploration & a Case Study*. 2019 International Conference on Process Mining (ICPM), 2019.

[10] Bergenthum, R.; Metzger, J.; Sorokin, L.; Lorenz, R.: *Towards Compact Regions for Labeled Prime Event Structures*. Algorithms and Tools for Petri Nets, 2017.

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

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

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

[14] Lorenz, R.; Desel, J.; Juhas, G.: *Models from scenarios*. Transactions on Petri Nets and Other Models of Concurrency VII Springer Berlin Heidelberg, 2013, 314 – 371.

[15] Liesaputra, V.; Yongchareon, S:; Chaisiri, S.: *Efficient process model discovery using maximal pattern mining*. BPM, 2015, pp. 441456.

Cluster Inter-Process Communication in Petriflow Language

Milan Mladoniczky[1,2], Gabriel Juhás[1,2,3], and Juraj Mažári[1,2]

[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., Jána Stanislava 28/A, 841 05 Bratislava, Slovakia
[3] BIREGAL s. r. o., Klincova 37/B, 821 08 Bratislava, Slovakia

Every Petri net can be described as an event system. The simple act of firing a transition consists of several events like consuming tokens and producing tokens. In other words, firing a transition raises an event of a change of marking on the Petri net or on an instance of the net. Petriflow modelling language[1] takes advantage of this property of Petri nets and defines means to emit and react to events. Because Petriflow extends Petri nets by roles and data, the set of available events is much larger than in classical Petri nets.

In Petriflow, from modelled processes, instances are created to execute process business logic. Every process instance is defined by its Petri net marking, values in data variables, and enabled transitions. In a process instance for every enabled transition, an object called task is generated. A task express activity, that is performed by a user of the system or by another deployed process. A task can be assigned to a user, finished by the user to which was the task previously assigned to or cancelled to interrupt the execution of and revert all changes made in the state of task execution.

Petriflow language also defines a construct called Actions[2], which are small snippets of code attached to an event. A set of actions attached to an event is considered as the reaction on the event raised on the part of Petriflow process model. An event can be raised by user interaction with a process, called in an action or as a consequence of the behaviour of underlying Petri net.

So far processes are considered to run in a local environment only. When an action of an instance triggers event in another instance of the same process or another, it is assumed that the instance is run locally and so the event is executed immediately and in the same application as the instances. However, the concept of inter-process communication can be elevated to a clustered environment, where every process could be a separate application (service) in the cluster. The process event can be called across the whole cluster of application.

To every instance is assigned an identifier, which is unique across the cluster. It is derived from the instance's process identifier and the hosting application. A called event's target instance is realised by the defined identifier. The instance identifier helps to quickly determine if an event target instance is local to application or not. If the desired instance is found locally the event is invoked immediately. Otherwise, the event is broadcasted to the cluster.

When an application is added to the cluster, it has to register itself to the nearest available message broker. The message broker has the responsibility to

deliver events to the application hosting the requested process and its instances. The newly added application request message broker to create a channel (queue) for every hosted process. The created message queue is bounded to the process identifier. Message broker routes events according to its process register based on the process identifier. It can be said, that very deployed process in applications is a service. A process is event publisher and event consumer at the same time. By this definition, process identifier must be unique across the whole cluster. If there is a request to register new process (service) with a duplicate identifier and it is not from the same application, the request is rejected and the queue for the process will not be created.

A process has to have prior knowledge of other processes, with which it wants to communicate, but does not has to know their location in the cluster, because in a clustered environment the location and availability of an application hosted processes can often change. When calling an event, it is necessary to request for the desired instance first. If the instance is not found locally, the search request is broadcasted to the cluster. The search request is routed according to the requested process and the request is added to the appropriate event queue of the process service. When the requested process has the capacity to execute such a request, it consumes the request from the queue. The message broker does not have to log every message in the cluster when a message is consumed it is deleted from the message broker. Logging of incoming requests and events from the queue is the responsibility of the process.

In practice, if in a monolith architecture is desired to assign a process task, it can be implemented with an event call *assign(T)*, where T is transition which will be assigned. If the transition T is not enabled, the event call returns false. In the clustered environment, the event call is extended by a process identifier as the event attribute, *assign("Process A",T)*.

After the event execution, a response is returned to the caller with information about the success of the execution, and additional data if requested.

It is possible to connect more message broker into a more complex network. In this scenario, each broker is responsible for routing inside its domain. Each Broker knows the topology of the whole broker network so it can calculate a path to the target application.

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. Mažari, J., Juhás, G., Mladoniczky, M.: Petriflow in Actions:Events Call Actions Call Events. Proceedings of the Workshop Algorithms and Tools for Petri nets 2018, October 11-12, 2018, University of Augsburg, Germany, (2018)

Discussion of a Renew Implementation of a Modular Model Checking Framework for Reference Nets

Sven Willrodt and Daniel Moldt

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

Abstract A first prototype is discussed that approaches the model checking problem for Reference nets. As the tool environment Renew (Reference nets Workshop) is used. To serve as a foundation for further research and experimentation on this problem, the prototype is built with a modular architecture. This allows quick prototyping, since often only one module needs to be exchanged to implement a new algorithm.

Keywords: Reference Nets, Nets-In-Nets, Model Checking Framework, Renew, Software Architecture

1 Introduction

Verification of Petri nets has a long tradition. While low-level Petri nets have a large repertoire of methods and tools to cope with even large model sizes [3, 6, 7, 13, 15, 18, 19], for high-level Petri nets research is still needed. More tools and links to Petri net topics can be found at http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/ and http://www.petrinet.de.

This holds especially for Reference nets with their special features of synchronous channels and the nets-in-nets concept (Nin). Model checking as a sub discipline can be considered as a strong candidate for Reference nets.

An idea that we work on for quite a while is the partitioning of the whole Reference net system into its sub nets. Based on the Nin concept partitioning of the net system shall ease to make model checking methods more efficient.

The reachability graph is a central means for the model checking and will be addressed first in our approach. Therefore a framework to support the Reference net formalism and its variants is discussed here. One purpose in the near future is to use the framework for our theory teaching lessons in the bachelor courses.

In the following we briefly present the basic notions in Section 2. Our requirements for and features of this prototype are covered in Section 3. The architecture and some basic behaviour is sketched in Section 4 before we conclude in Section 5.

2 Basics

Reference nets (see [10]) are a high-level Petri net formalism based on the modeling approach of nets-within-nets. They allow the use of net instance tokens and recognize Java expressions as inscriptions. This causes statements about places and transitions to be ambiguous during the simulation. Due to the dynamic instantiation of net instance tokens, markings may consist of arbitrary amounts of net instances of each net template. Templates can be considered as classes and instances as objects.

Renew is developed by our group since the end of the nineties (see [11]). However, verification has been addressed only for simple versions of the nets and only small studies have been performed. Tools like Maria [12], LoLa [20], GreatSPN [2], Maude [5] and others have been (partially) integrated to provide verification options for traditional net variants.

Model checking is a method to verify systems that are modeled as state-transition systems. With their capability to express concurrency and distribution, Petri nets are a popular modeling language for model checking. [4, 8] Every year, the Model Checking Contest [9] is held, where tools can compete in verifying Petri nets.

3 Requirements and Features

The framework should provide a general foundation for tasks concerning the model checking problem for Reference nets. There exists a multitude of optimisation techniques, primarily developed for P/T-nets, whose suitabilities for the Reference net formalism are only partially explored.

Therefore, the main focus for the framework is an extensible structure that allows easy modification and quick prototyping. This requirement is realized through a modular architecture, where each module can be exchanged independently to alter the behaviour of a model checking routine.

Since in the Reference net formalism net elements cannot uniquely be identified by name, the framework also needs to provide the means to extend propositional logic with custom expressions or operators. This makes more sophisticated specifications possible.

Another natural use case for the framework is teaching, because performance is less important for the usually smaller examples handed out to students. To be of use in this category, the user interface is central. Detailed and comprehensible feedback, important aspects of model checking, should be accessible when using this framework.

4 Architecture

The architecture consists mainly in three types of modules: *binding cores*, *storage managers* and *procedures*.

The *binding core* finds bindings for a given marking and can calculate the resulting markings. It can be set up to exclude certain types of transitions, e.g. transitions with inscriptions that have side-effects. Exchanging the binding core is expected to be less frequent, however it is possible to integrate new formalisms that way. The binding core is generally realized by a special usage of Renew's simulator, which is further described in [17].

Storage managers keep track of found markings. They store the reachability tree, insert new nodes into it and find duplicate markings. By exchanging the storage manager, it is possible to test new data-structures and storage heuristics. The sweepline method [3], symmetry [15] and bloomfiltering [20] are examples for techniques that can be implemented in storage managers.

Procedures carry the logic and steps to process a query. They define what the overall purpose of the query is and can be considered as the active components which utilize all other modules. The generation of a reachability graph, CTL model checking and model checking with partial order reduction are examples for procedures. Procedures can also use other procedures, to prevent basic algorithms from being implemented twice. Optimization techniques, that want to select which transitions are fired, like the stubborn-set method [18] or a depth-first search, can be realized in a procedure.

Figure 1 shows an example interaction of the just described modules. It displays the main interaction points of the binding core as well as the storage manager. Whenever the procedure processes bindings or nodes, it may remove some, which are then not carried on to the next step. Because the procedure has no obligations at all, the interaction may follow an entirely different structure.

Furthermore, the framework provides common functions like parsing and analysis of syntax trees, as well as the evaluation of atomic propositions on a marking. For the former, a parser generator is used that features inheritance of grammars, reducing the amount of duplicate code in grammar files.

Lastly, the framework consists of a UI that has multiple features. It dynamically presents the user compatible modules to a selected procedure. For this, procedures can define certain compatibility properties. Also, the included *result visualizer* takes on the task of presenting results to the user. For this, a data-structure that represents a result is introduced, that every procedure returns. Depending on which fields are filled out by the procedure, the result visualizer can choose an appropriate visualization method. One of the planned methods is a coloration of the nodes in the reachability graph, representing for each node whether it fulfills a (sub-)formula or not. This method is compatible with label-based approaches, e.g. CTL model checking.

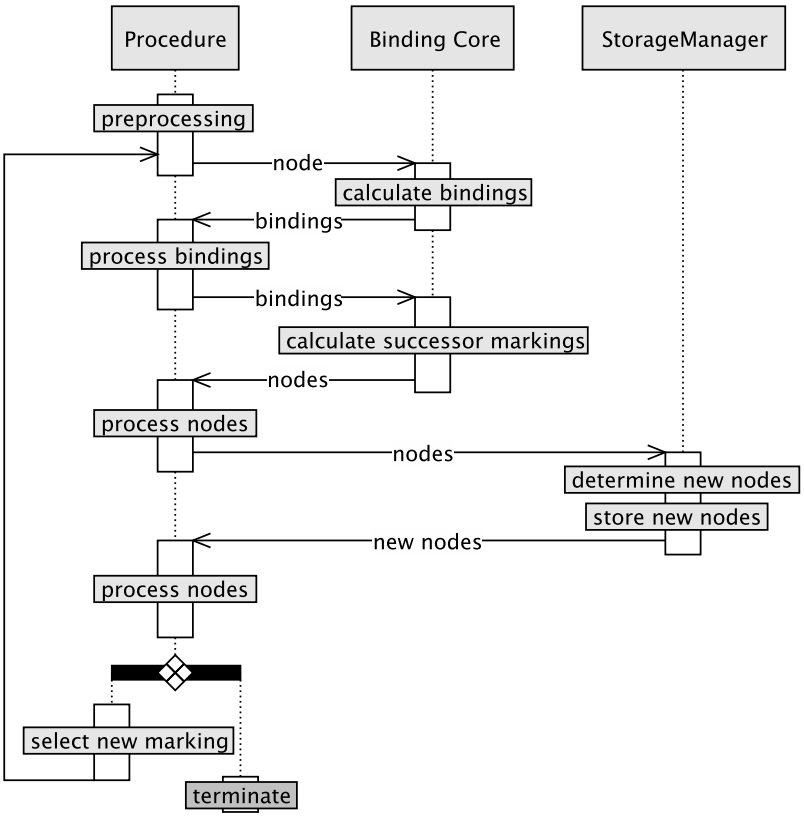

Figure 1. Example interaction between the procedure, binding core and storage manager

5 Conclusion

The framework provides a foundation for model checking in Renew. While it is right now not appropriate for industrial sized problems, its purpose is proof of concept. It also provides a basis to conduct further research on model checking algorithms designed for Reference nets. With the focus on an extensive and comprehensive visual presentation of results, it is well suited for teaching purposes.

Outlook

A promising approach is to use the framework for the *Curry Coloured Petri Net* formalism (see [16]), especially future extensions integrating hierarchical nets and/or Reference net concepts. Since the Curry inscriptions are side-effect free and can in some cases further be verified by code analysis [1], a complete verification of the state space potentially becomes possible.

As another strong point besides the transfer of algorithms and methods of traditional Petri net analysis, we address the effective analysis of net models in two further ways: A central idea is to use the pragmatic structure of the model, built in by modelers. The nets-within-nets concept supports a kind of object- or agent-oriented modeling which we will use to partition the models. Secondly, based on the partitioning we will use the concurrent and distributed execution of the analysis, for which we can utilize our research on Kubernetes in the context of Renew [14]. Each net template or the net instances of a net model can be analyzed in separate threads, processes, virtual machines or computers. Special treatment of the composition is required by algorithms and methods that still need to be developed in further research.

References

1. Antoy, S., Hanus, M., Libby, S.: Proving Non-Deterministic Computations in Agda. In: Proc. of the 24th International Workshop on Functional and (Constraint) Logic Programming (WFLP 2016). Vol. 234. Electronic Proceedings in Theoretical Computer Science. Open Publishing Association, 2017, pp. 180–195
2. Baarir, S., Beccuti, M., Cerotti, D., De Pierro, M., Donatelli, S., Franceschinis, G.: The GreatSPN tool: recent enhancements. ACM SIGMETRICS Performance Evaluation Review 36(4), 4–9 (2009)
3. Christensen, S., Kristensen, L. M., Mailund, T.: A Sweep-Line Method for State Space Exploration. In: Tools and Algorithms for the Construction and Analysis of Systems. Ed. by Margaria, T., Yi, W. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 450–464
4. Clarke Jr., E. M., Grumberg, O., Kroening, D., Peled, D. A., Veith, H.: *Model Checking*. third. Cambridge, MA, USA: MIT Press, 2018. ISBN: 0-262-03270-8
5. Clavel, M., Durán, F., Eker, S., Lincoln, P., Mart-Oliet, N., Meseguer, J., Talcott, C.: The maude 2.0 system. In: International Conference on Rewriting Techniques and Applications. Springer. 2003, pp. 76–87
6. Esparza, J., Heljanko, K.: *Unfoldings: a partial-order approach to model checking*. Springer Science & Business Media, 2008
7. Gerth, R., Kuiper, R., Peled, D., Penczek, W.: A partial order approach to branching time logic model checking. In: Proceedings Third Israel Symposium on the Theory of Computing and Systems. 1995, pp. 130–139
8. Girault, C., Valk, R.: *Petri nets for systems engineering: a guide to modeling, verification, and applications*. Springer Science & Business Media, 2013
9. Kordon, F., Garavel, H., Hillah, L. M., Hulin-Hubard, F., Amparore, E., Beccuti, M., Berthomieu, B., Ciardo, G., Dal Zilio, S., Liebke, T., Li, S., Meijer, J., Miner, A., Srba, J., Thierry-Mieg, Y., Pol, J. van de, Dirk, T. van, Wolf, K.: *Complete Results for the 2019 Edition of the Model Checking Contest*. http://mcc.lip6.fr/. Apr. 2019. (Visited on 2019)
10. Kummer, O.: *Referenznetze*. Berlin: Logos Verlag, 2002
11. Kummer, O., Wienberg, F., Duvigneau, M., Cabac, L., Haustermann, M., Mosteller, D.: *Renew – User Guide (Release 2.5)*. Release 2.5. University of Hamburg, Faculty of Informatics, Theoretical Foundations Group. Hamburg, June 2016. URL: http://www.renew.de/

12. Mäkelä, M.: Maria: Modular reachability analyser for algebraic system nets. In: International Conference on Application and Theory of Petri Nets. Springer. 2002, pp. 434–444
13. Pastor, E., Roig, O., Cortadella, J., Badia, R. M.: Petri net analysis using boolean manipulation. In: Application and Theory of Petri Nets 1994. Ed. by Valette, R. Springer Berlin Heidelberg, 1994, pp. 416–435. ISBN: 978-3-540-48462-2
14. Röwekamp, J. H., Moldt, D.: RenewKube: Reference Net Simulation Scaling with Renew and Kubernetes. In: Application and Theory of Petri Nets and Concurrency - 40th International Conference, PETRI NETS 2019, Aachen, Germany, June 23-28, 2019, Proceedings. Ed. by Donatelli, S., Haar, S. Vol. 11522. Lecture Notes in Computer Science. Springer, 2019, pp. 69–79. ISBN: 978-3-030-21570-5. DOI: 10.1007/978-3-030-21571-2. URL: https://doi.org/10.1007/978-3-030-21571-2
15. Schmidt, K.: *Symmetries of Petri Nets*. Citeseer, 1994
16. Simon, M.: 'Curry-Coloured Petri Nets: A Concurrent Simulator for Petri Nets with Purely Functional Logic Program Inscriptions'. Master Thesis. Vogt-Kölln Str. 30, D-22527 Hamburg, Germany: University of Hamburg, Department of Informatics, Apr. 2018
17. Simon, M., Moldt, D., Engelhardt, H., Willrodt, S.: A First Prototype for the Visualization of the Reachability Graph of Reference Nets. In: Petri Nets and Software Engineering. International Workshop, PNSE'19, Aachen, Germany, June 24, 2019. Proceedings. Ed. by Moldt, D., Kindler, E., Wimmer, M. Vol. 2424. CEUR Workshop Proceedings. CEUR-WS.org, 2019, pp. 165–166. URL: http://CEUR-WS.org/Vol-2424
18. Valmari, A.: A stubborn attack on state explosion. Formal Methods in System Design 1(4), 297–322 (Dec. 1992). ISSN: 1572-8102. URL: https://doi.org/10.1007/BF00709154
19. Wolf, K.: Generating Petri Net State Spaces. In: Petri Nets and Other Models of Concurrency – ICATPN 2007. Ed. by Kleijn, J., Yakovlev, A. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 29–42. ISBN: 978-3-540-73094-1
20. Wolf, K.: Petri Net Model Checking with LoLA 2. In: Application and Theory of Petri Nets and Concurrency. Ed. by Khomenko, V., Roux, O. H. Cham: Springer International Publishing, 2018, pp. 351–362. ISBN: 978-3-319-91268-4

SPIKE – as a Supporting Tool for a Model Parameters Optimization via Branched Simulations

Jacek Chodak*, Monika Heiner

Computer Science Institute, Brandenburg University of Technology
Postbox 10 13 44, 03013 Cottbus, Germany
jacek.chodak@b-tu.de, monika.heiner@b-tu.de
http://www-dssz.informatik.tu-cottbus.de

Abstract This paper presents the continuation of work on Spike - a command line tool for continuous, stochastic & hybrid simulation of (coloured) Petri nets (PN). It supports import from and export to various Petri net data formats and also imports SBML models. Spike's abilities includes: the configuration of models by changing arc weights, initial markings and transitions rates. It also unfolds coloured stochastic/continuous/hybrid Petri nets. To comply with the demand for reproducible simulation experiments, Spike builds on a scripting language in a human-readable format. Its core features permits the design of a set of simulation experiments by a single configuration file. These simulation experiments can be executed in parallel, on a multi-core machine; distributed execution is in preparation. By utilizing Spike's feature which allows scanning of parameters , Spike can serve as the supporting tool of model parameters optimization.

Keywords: continuous, stochastic, hybrid, coloured (hierarchical) Petri nets · simulation · configuration · reproducibility · parameters optimization

1 Objectives

Parameters optimization of biochemical reaction networks, for which we use Petri nets as an umbrella modelling paradigm, is a problem. During development of a model, frequently not all parameters are known and only a data from wet-lab experiments are available. To estimate not know values of a model parameters it is necessary to run a set of simulation experiments and compare acquired results against a wet-lab data. The size of the simulation set can be very large and depends on parameters set size e.g. 5 parameters with a value range of size 10, the size of simulations set is equal to 10^5. Doing this manually, by preparing a new simulation run for each new model configuration, is time consuming and potentially error-prone.

* Corresponding author

One of the ways to address these issues is utilizing Spike, which is builds on a human-readable configuration script, supporting the efficient specification of multiple model configurations as well as multiple simulator configurations in a single file. Each specific model and simulator configuration determines a specific simulation experiment, for which Spike creates a separate branch, ready to be executed on a server, with all branches treated as parallel processes.

2 Functionality

As presented in [3,2], Spike is a slim, but powerful brother of Snoopy [6] - it is the latest addition to the PetriNuts family of tools for modelling, analysis and simulation with Petri nets, specifically tailored to the investigation of biochemical reaction networks.

Depending on the configuration, Spike is capable to run three basic types of simulations: stochastic, continuous and hybrid, each comes with several algorithms. Simulation of coloured stochastic, continuous and hybrid PN models is supported by unfolding them automatically to uncoloured models.

A given model is simulated according to the specified simulation type, despite place and transition types in the model. That means all places and transitions are converted to the appropriate type. For example, if a user wants to run a stochastic simulation on a continuous model, all places and transitions are converted to the stochastic type. Likewise, for stochastic models to be simulated continuously, all stochastic transitions are converted to continuous type.

The main focus of Spike lays on efficient and reproducible simulation of PN models. New features allow for configuring simulation over a set of parameters (parameters scanning) and runs simulation task parallel.

Branching During configuration evaluation, it can be split into separate branches. Branching process is triggered by defining in the configuration, a set of parameters to scan. The set of values is assigned to the configuration parameter. For each value in the set, a new configuration branch is created. Such a feature allows for mutating configuration script, what results in multiple simulation configuration.

Let's consider the following use case, the diffusion model presented in Fig. 1. Over this model, constant D has been defined. With the help of this constant, it is possible to set the size of the diffusion grid. With the help of the parameter scanning introduced in Spike, it is possible to reuse the same model and through the configuration script, set the range of values to scan for the constant D, e.g.:

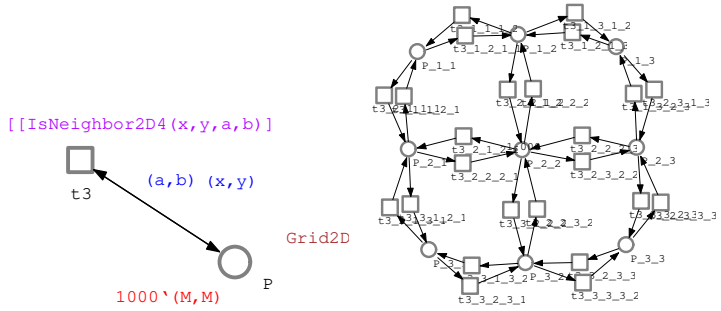

Figure 1: Coloured (a) and unfolded (b) model of diffusion 2D4.

```
constants: {
  all: {
    D: [3, 5, 7];
  }
}
```

By using the array operator [ ], the set of 3 values is assigned to the constant D. The number of branches depends on the size of the set. For each value in the set, Spike creates a new branch of configuration script. In this case, Spike will split the configuration and create 3 branches

Simulation The set of configuration branches can be executed sequentially or parallel. Each branch is executed as a separated process of Spike. During the simulation, Spike crate 2 types of processes. One so-called master process and one or more slaves processes.

The master process act as a broker and owner of the simulation experiment. It takes care of creating slave processes on a local machine. The slave process is responsible for executing exactly one branch of the simulation configuration. The number of slave processes running parallel is depended on an option passed to Spike. If only one slave process is allowed then each simulation branch, will be executed sequentially. In this case, the master process will wait for the end of execution of one of the branches before starting execution of the next one. Otherwise, the master process will start slave processes at most in the number specified by Spike's option. If the number of the branches exceeds the number of the slave processes, the master process will postpone starting new one until one of currently running processes will finish its task. Starting the slave process by Spike does not mean that the number of the running threads is equal to the number of processes. The number of threads may depend on the applied simulation algorithm. For example, a stochastic simulation may involve multi-threading to execute in parallel the independent individual runs, which are later averaged.

3 Optimization through Simulation

Optimization through simulation can be used as a search method [7] of best candidates of input variables among all valid alternatives at any system state. By adopting heuristic evaluation it is possible to reduce search space without explicitly evaluating each possibility. Newly introduced features to Spike, such as parameters scanning and parallel execution of configuration branches, make it more suitable for this task. In the following two scenarios of parameters optimization, Spike fits well as a universal simulator.

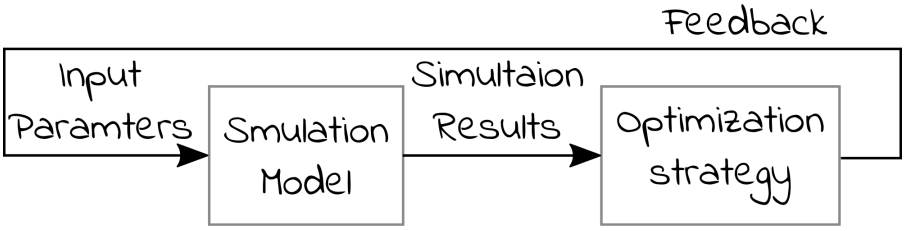

Figure 2: Optimization through simulation

Brute force This straight forward approach, where for each combination of parameters a new simulation is executed. Thanks to introducing it into the Spike, branching of simulation (parameters scanning) can be easily done, by use of one experiment configuration. After the execution of all possible simulations, the best matching results should be selected by the use of fitness function, which tells how good results are. This approach is presented by Algorithm 1 .

Algorithm 1: Use case: Brute force multiple parameter optimisation.

```
1  Load model;
2  Determine simulator configuration;
3  for each unique combination of parameter values do
4  |   Determine model configuration;
5  |   Create new configuration branch;
6  |   Run simulation;
7  |   Save results of the simulation;
8  end
9  for each stored results do
10 |   if results not fitted then
11 |   |   Remove results;
12 |   end
13 end
```

Heuristic Brute force approach does not reduce the search space of parameters values. For each combination of parameters a new simulation is executed, what is exhausting time and resources. A heuristic method is much better suitable, where a number of parameters and/or their values ranges are too large, what prevents them to be optimized in a finite time. In [5] genetic algorithm is used to drive the optimization strategy of models parameters. The DIRECT method and its derivatives [1], as presented in [4], also suite well as optimization strategy in the use case presented by Algorithm 2. As shown in Fig. 2, an optimization strategy use results from simulation to reduce the space of parameters values by checking the fitness of set of parameters and provide a set of best fitted parameters values as feedback to the model.

Algorithm 2: Use case: Space reduction multiple parameter optimisation.

```
1  Load model;
2  Determine simulator configuration;
3  repeat
4  |  for each unique combination of parameter values do
5  |  |  Determine model configuration;
6  |  |  Create new configuration branch;
7  |  |  Run simulation;
8  |  end
9  |  Run optimization strategy;
10 until space reduced;
```

4 Remarks

This paper presents the idea of using Spike as a tool supporting parameter optimization. Currently, there is undergoing work on Spike's API (Application Programming Interface), which allows Spike act as a service for simulation execution. In the future, the presented idea will make full use of it.

5 Installation and future work

Spike is written in C++ and available for Linux, Mac/OSX and Windows. Binaries are statically linked and can be downloaded from Spike's website `https://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Spike`, where one also finds documentation, installation instruction and a set of examples.

Spike is still under development. Future work will incorporate sophisticated model reduction, model decomposition, and distributed simulation, either for a set of simulations or a decomposed model. We are open for further suggestions.

Acknowledgement

Spike uses software libraries (data format conversions, simulation algorithms) which have been previously developed by former staff members and numerous student projects at Brandenburg Technical University (BTU), chair Data Structures and Software Dependability.

References

1. D. R. Jones, C. D. Perttunen, B. E. Stuckman : Lipschitzian optimization without the Lipschitz constant . Journal of optimization Theory and Applications **79**(1), 157–181 (1993)
2. J. Chodak, M. Heiner : Spike - a command line tool for continuous, stochastic & hybrid simulation of (coloured) Petri nets. In: Proc. 21th German Workshop on Algorithms and Tools for Petri Nets (AWPN 2018), pp. 1–6. University of Augsburg (October 2018), `https://opus.bibliothek.uni-augsburg.de/opus4/frontdoor/deliver/index/docId/41861/file/awpn18-lorenz-metzger-OPUS.pdf#page=9`
3. J. Chodak, M. Heiner : Spike Reproducible Simulation Experiments with Configuration File Branching . In: LNBI 11773, Springer (2019). https://doi.org/10.1007/978-3-030-31304-3_19
4. J.D. Griffin, T.G. Kolda : Asynchronous parallel hybrid optimization combining DIRECT and GSS . Optimization Methods and Software **25**(5), 797–817 (2010). https://doi.org/10.1080/10556780903039893
5. R. Donaldson, D. Gilbert : A Model Checking Approach to the Parameter Estimation of Biochemical Pathways . In: M. Heiner . A.M. Uhrmacher (ed.) Computational Methods in Systems Biology . pp. 269–287. Springer Berlin Heidelberg , Berlin, Heidelberg (2008)
6. M. Heiner, M. Herajy, F. Liu, C. Rohr, M. Schwarick: Snoopy – A Unifying Petri Net Tool. In: ATPN 2012. pp. 398–407. Springer, LNCS 7347 (2012)
7. Y. Carson, A. Maria : Simulation optimization: methods and applications. In: Proceedings of the 29th conference on Winter simulation. pp. 118–126. IEEE Computer Society (1997)

Enhanced Discovery of Uniwired Petri Nets Using eST-Miner

Lisa L. Mannel(✉) and Wil M. P. van der Aalst

Process and Data Science (PADS), RWTH Aachen University, Aachen, Germany,
mannel@pads.rwth-aachen.de; wvdaalst@pads.rwth-aachen.de

More and more processes executed in companies are supported by information systems which record events. Extracting events related to a process results in an *event log*. Each event in such a log has a name identifying the executed activity (activity name), a case id specifying the respective instance of the process, a time when the event was observed (timestamp), and possibly other data related to the activity and/or process instance. In *process discovery*, a process model is constructed aiming to reflect the behavior defined by the given event log: the observed events are put into relation to each other, preconditions, choices, concurrency, etc. are discovered, and brought together in a process model.

Process discovery is non-trivial for a variety of reasons. The behavior recorded in an event log cannot be assumed to be complete, since behavior allowed by the process specification might simply not have happened yet. Additionally, real-life event logs often contain noise, and finding a balance between filtering this out and at the same time keeping all desired information is often a non-trivial task. Ideally, a discovered model should be able to produce the behavior contained within the event log, not allow for unobserved behavior, represent all dependencies between the events, and at the same time be simple enough to be understood by a human interpreter. It is rarely possible to fulfill all these requirements simultaneously. Based on the capabilities and focus of the used algorithm, the discovered models can vary greatly, and different trade-offs are possible.

Our discovery algorithm eST-Miner [1] aims to combine the capability of finding complex control-flow structures like longterm-dependencies with an inherent ability to handle low-frequent behavior while exploiting the token-game to increase efficiency. Similar to region-based algorithms, the basic idea is to evaluate all possible places to discover a set of fitting ones. Efficiency is significantly increased by skipping uninteresting sections of the search space based on previous results [2]. This may decrease computation time immensely compared to evaluating every single candidate place, while still providing guarantees with regard to fitness and precision. Implicit places are removed in a post-processing step to simplify the model.

In [3] we introduce the subclass of *uniwired Petri nets* as well as a variant of eST-Miner discovering such nets. In uniwired Petri nets all pairs of transitions are connected by at most one place, i.e. there is no pair of transitions (a_1, a_2) such that there is more than one place with an incoming arc from a_1 and an outgoing arc to a_2. Still being able to model long-term dependencies, these Petri nets provide a well-balanced trade-off between simplicity and expressiveness, and thus introduce a very interesting representational bias to process

discovery. *Constraining ourselves to uniwired Petri nets allows for a massive decrease in computation time compared to the basic algorithm by utilizing the uniwiredness-requirement to skip an astonishingly large part of the search space.* Additionally, the number of returned implicit places, and thus the complexity of post-processing, is greatly reduced.

For details we refer the reader to the original papers [1,3]. The basic eST-Miner, as well as the uniwired variant, take an event log and user-definable parameter τ as input. Inspired by language-based regions, the basic strategy of the approach is to begin with a Petri net, whose transitions correspond exactly to the activities used in the given log. From the finite set of unmarked, intermediate places a subset of fitting places is inserted. A place is considered fitting, if at least a fraction of τ traces in the event log is fitting, thus allowing for local noise-filtering. To increase efficiency, the candidate space is organized as a set of trees, where uninteresting subtrees can be cut off during traversal, significantly increasing time and space efficiency.

While the basic algorithm maximizes precision by guaranteeing to traverse and discover all possible fitting places, the uniwired variant chooses the most interesting places out of a selection of fitting candidates wiring the same pair of transitions. Subtrees containing only places that wire the same pair of transitions can be cut off. The output Petri net is no longer unique but highly dependent on the traversal and selection strategy. The approach presented in [3] prioritizes places with few arcs. Between places with the same number of arcs, places with high token-throughput are preferred. This strategy often allows us to discover adequate models, but fails in the presence of long loops which require places with more arcs. To overcome this restriction, we propose to use a reversed strategy, prioritizing places with high token throughput and using the number of arcs as a second criteria. This might slightly decrease the fraction of cut-off candidates but is expected to greatly increase model quality.

The running time of the eST-Miner variants strongly depends on the number of candidate places skipable during the search for fitting places. For the basic approach ([1]) our experiments show that 40-90 % of candidate places are skipped, depending on the log. The uniwired variant ([3]) has proven to find usable models while evaluating less than 1 % of the candidate space in all test-cases.

References

1. Mannel, L., van der Aalst, W.: Finding complex process-structures by exploiting the token-game. In: Application and Theory of Petri Nets and Concurrency. Springer Nature Switzerland AG (2019)
2. van der Aalst, W.: Discovering the "glue" connecting activities - exploiting monotonicity to learn places faster. In: It's All About Coordination - Essays to Celebrate the Lifelong Scientific Achievements of Farhad Arbab (2018)
3. Mannel, L., van der Aalst, W.: Finding uniwired Petri nets using eST-miner. In: Business Process Intelligence Workshop 2019. Springer (to be published)

The PNK, the PNML and the ePNK: What became of our dreams? Extended Abstract

Ekkart Kindler

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

1 Introduction

In Wolfgang Reisig's group at the Technische Universität München and later at Humboldt Universität zu Berlin, we were strictly forbidden "to waste our time on developing tools for Petri nets" for a long time. Then, in 1997, Wolfgang Reisig was on a sabbatical at the International Computer Science Institute (ICSI), Berkeley, California, where he, after his presentations and talks, was repeatedly asked a simple question: "do you have a tool for this?" After he returned from this sabbatical, Wolfgang changed his tune and requested: "Ekkart! We need a tool!"

Luckily, some of his group had been disobedient before and had started thinking of and developing a tool for Petri nets already. In particular, Jörg Desel and I had started dreaming of a *universal tool for Petri nets* already and had come up with a name for it: the *Petri Net Kernel*[1] (*PNK*) [1]. Based on this dream, different versions and extensions of the PNK have been implemented over the years [2–5]. And, as a sideline, the concepts of the PNK strongly influenced the Petri Net Markup Language (PNML) [6] – even though one of the main motivations behind our original dream was to get away from the never ending discussions on file formats for exchanging Petri nets.

Now, many years have gone by and several recent anniversaries and retrospectives concerning people and events within the Petri net community inspired me to have a look back at the original dream of a *universal Petri net tool* [1] and to see to which extend today's ePNK [5] actually fulfills the original dream. This paper and the related talk are a first attempt of discussing and structuring some of these insights, and of relating our original motivations and expectations to the actual result in a very loose and informal way – starting with only a few of the original ideas [1].

[1] Originally, it was in German *Petrinetz-Kern*, which we later translated to *Petri Net Kernel*.

2 What became of the ideas?

When we started dreaming of a universal Petri net tool, there was actually a discussion on exchange formats for Petri nets going on. The most promising proposal at the time was the *Abstract Petri Net Notation* (*APNN*) [7]; still, the discussion did not seem to converge. Therefore, we explicitly did not aim for a Petri net exchange format, but for an API[2] for creating, querying and updating Petri nets. We were well aware that a unified API for Petri nets might be even more ambitious than a unified exchange format. But, we believed that an API for Petri nets might contribute more to the understanding of how to unify Petri nets from an algorithms and tools perspective. Cautiously, we even added that we would consider the *Petri Net Kernel* (*PNK*) a success even in case its reference implementation did not survive. We claimed that the PNK would be a success already if it contributed to a better understanding of the needed API.

The main idea for making the PNK universal was to identify the common structure of all Petri nets, which included places, transition and arcs, and even structuring mechanisms like pages. All the features that where specific to certain types of Petri nets where supposed to be captured in additional inscriptions or labels attached to the common Petri net elements. Later, we called the set of allowed inscriptions of a certain version of Petri net a *Petri net type definition* (*PNTD*).

This way, the PNK could be understood as a universal data model for Petri nets. And the PNTDs allowed to define the specific (abstract) syntax of all kinds of Petri nets. In the beginnings of the PNK, we also meant to capture the semantics of the different types of Petri nets, and there have been different academic attempts to implement the semantics of a Petri net type and plug them into the PNK. But, this idea never really took off and is not part of today's ePNK.

The API or, if you will, the data model of the PNK and its concepts of PNTDs came in handy when – as part of standardizing high-level Petri nets [8] – a generic interchange format for all kinds of Petri nets was standardized: the *PNML* [6]. The *PNML core model* is the essence distilled from the PNK API turned into a UML class diagram. And it is maybe a bit ironic that a move to avoid the discussion on standard exchange formats for Petri nets, eventually, contributed to them anyway.

Making the *PNML core model* into a UML-model might eventually have contributed to the second generation ePNK [4], since the ePNK was developed in a model-based way using the Eclipse Modeling Framework (EMF) [9]. Actually, the PNML Framework [10] had adopted this approach before. But, since the PNML Framework did not come with a graphical user interface and did not have a mechanism for dynamically plugging in new PNTDs and applications, I decided to implement the ePNK as new version of the PNK based on EMF. And maybe this move guaranteed the survival of the ePNK not only in its concept but

[2] In the paper [1], we did not use the word API, though. We used the German word "Schnittstelle".

as software; actually it brought us closer to the original dream: Whereas most of the programming effort in the first generation PNK went into infrastructure like graphical user interfaces, editors, parsers and our own plug-in architecture, in the second generation ePNK, all these things could be taken from Eclipse and EMF. So, we could focus on the core functionality of Petri nets when implementing, extending and maintaining the ePNK. And this is very much what the initial dream of the PNK was about.

When looking back, I realized that there was one idea in our original paper [1] that the PNK, the PNML and, as a consequence, also the ePNK had lost track of: Originally, the PNK had the idea that everything should be a graph, and graphs were supposed to be at the core of the PNK. In the different implementations, however, and, in particular, in the graphical editor and the generic infrastructure of the ePNK, this was not properly taken care of: This editor and the underlying infrastructure assume that there are two[3] types of nodes: places and transitions. And this choice makes it a bit artificial defining Petri net types with more than two "dominant" types of nodes in the ePNK. Here, I actually regret not having read our original paper again before reimplementing the ePNK. I could have made the ePNK graph-based, basically, without any extra effort, but starting from the PNML core model, which did not have graphs at its basis, I had just forgotten about that.

3 Conclusion

After now almost 25 years, we can say that at least some parts of the dream of a universal Petri net tool have became true, and its ideas and concepts have contributed to a better understanding of how to unify Petri net tools. Even more, they have contributed to standardizing an exchange format for Petri nets, which expressly was not part of the dream at all.

Reading the paper again even reminded me of other ideas and lessons, which would be worth a more detailed discussion. I hope that, at some point, I will find the time to discuss and share them, too.

References

1. Kindler, E., Desel, J.: Der Traum von einem universellen Petrinetz-Werkzeug — Der Petrinetz-Kern. In Desel, J., Oberweis, A., Kindler, E., eds.: 3. Workshop Algorithmen und Werkzeuge für Petrinetze. Number 341 in Forschungsberichte, Institut AIFB, Universität Karlsruhe (1996)
2. Kindler, E., Weber, M.: The Petri Net Kernel: An infrastructure for building Petri net tools. In: Petri Nets ´99, 20^{th} International Conference on Application and Theory of Petri Nets: Petri Net Tool Presentations. (1999) 10–19
3. Weber, M., Kindler, E.: The Petri Net Kernel. In Ehrig, H., Reisig, W., Rozenberg, G., Weber, H., eds.: Petri Net Technologies for Modeling Communication Based Systems. Volume 2472 of LNCS. Springer (2003) 109–123

[3] Ignoring for a moment that pages, technically, are also represented as nodes in the ePNK.

4. 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
5. Kindler, E.: ePNK applications and annotations: A simulator for YAWL nets. In Khomenko, V., Roux, O.H., eds.: Application and Theory of Petri Nets and Concurrency - 39th International Conference, PETRI NETS 2018, Bratislava, Slovakia, June 24-29, 2018, Proceedings. Volume 10877 of LNCS., Springer (2018) 339–350
6. ISO/IEC: Systems and software engineering – High-level Petri nets – Part 2: Transfer format, International Standard ISO/IEC 15909-2:2011 (2011)
7. Bause, F., Kemper, P., Kritzinger, P.: Abstract Petri net notation. Petri Net Newsletter **49** (1995) 9–27
8. ISO/IEC: Software and Systems Engineering – High-level Petri Nets, Part 1: Concepts, Definitions and Graphical Notation, International Standard ISO/IEC 15909 (2004)
9. Budinsky, F., Steinberg, D., Merks, E., Ellersick, R., Grose, T.J.: Eclipse Modeling Framework. 2nd edn. The Eclipse Series. Addison-Wesley (2006)
10. Hillah, L.M., Kordon, F., Petrucci, L., Trèves, N.: PNML Framework: An extendable reference implementation of the Petri Net Markup Language. In Lilius, J., Penczek, W., eds.: Petri Nets. Volume 6128 of Lecture Notes in Computer Science., Springer (2010) 318–327

Execution of Event Chains in a Petriflow Model

Juraj Mažári[1,2], Gabriel Juhás[1,2,3], and Milan Mladoniczky[1,2]

[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., Jána Stanislava 28/A, 841 05 Bratislava, Slovakia
[3] BIREGAL s. r. o., Klincova 37/B, 821 08 Bratislava, Slovakia

In [1] and [2] we introduced synchronisation between instances of complex business processes using Petriflow language. This technique brings questions about parallelism and deadlocks in such models and implementations of Petriflow syntax. In the following, we explain the mechanics of our implementation in simple use cases.

In Petriflow we have defined actions as a small pieces of code. They can be used to change data variable values, call web-services, create new cases or call events on other cases. Actions call also be triggered by those events. Actions triggered before the task event are called pre-actions. Post-actions are triggered after the task event is executed. Together with the event they are executed atomically and form an event chain. Execution of an event chain works as follows: first, we check if the event can occur, then we execute all pre-actions, then the event itself and finally we execute all post-actions. If an exception occurs during the execution of any event chain step the whole execution is stopped. No steps executed prior to the failure are reverted so there is no automatic transaction mechanism. Whether the execution should be reverted or there will be any reaction to the failure is up to the modeller and his decision.

In general, there can by any finite number of threads executing events on a single case. Using multiple thread executors brings problem with conflicting executions. If one thread executor is assigning a task no other thread can execute event chain that would disable the task.

For the sake of simplicity our implementation uses one single thread executor for each process instance. This way all atomic events are executed sequentially. Requesting multiple events in the same case puts them into a queue of that case event executor. The executor then pulls events one by one from the queue to be executed in the same order they were added in. During the execution of one event, the event executor is blocked by that execution and no other events can be executed. This brings two concerns: deadlocks and parallelism.

For example deadlock will occur when pre-assign action of a task *t1* in case *A* requests a task *t2* in case *B* to be assigned and pre-assign action of task *t2* requests assign of task *t3* in case *A*. First the system puts the request *assign(t1)* in the queue of *executor A*. *Executor A* pulls the request from its queue and start the execution of its pre-actions. One of the pre-actions requests *assign(t2)* in case *B*. This request is put in the queue of *executor B* and *executor A* waits for the response from *executor B*. *Executor A* is now blocked by the request *assign(t2)*. Executor B pulls the request from its queue and start the execution of its pre-actions. Again one of the pre-actions requests *assign(t3)* in case *A*.

The request is put in the queue if *executor A* and *executor B* is now blocked and waiting for this request to be executed. Since *executor A* is still blocked and waiting for the *assign(t2)* to be finished by *executor B* it will not execute *assign(t3)* and cause a deadlock.

To solve this problem we allow the action to create sub-chain and all related events and actions will be executed by this sub-chains executor without being put in the executor's queue. In our case request *assign(t3)* will be executed in such sub-chain. Task *t3* will be assigned. *Executor B* will receive the response of calling *assign(t3)* and continue with the execution of *assign(t2)*. After finishing the execution *executor B* will send the response to the *executor A* which will then continue with the execution of *assign(t1)*. In this execution of event chain using sub-chains, no deadlock will occur.

In our implementation task parallelism is preserved. Each task takes some time to execute. This is the time between the *assign()* event and the *finish()* event of a task. For two parallel tasks *t1* and *t2* events can occur in order *assign(t1)*, *assign(t2)*, *finish(t1)* and *finish(t2)*. From the timeline of these events, it is clear that they were executed in parallel. If we use actions to synchronise these two tasks we can ensure that the same state is reached as if all events were executed at the same time. When the *assign(t1)-assign(t2)-finish(t1)-finish(t2)* chain is executed the executor is blocked and no other event chain will be executed until its finished. Since no other events could be executed between events *assign(t1)* and *finish(t2)* no intermediate state is actually reached and persisted in the database.

When synchronising multiple tasks in different cases we have no guarantee that all will be executed successfully. Even if we in one action check if all tasks can be executed we have to make sure that nothing will be executed in those cases that will cause the task to be inexecutable. With sub-chains, we can do just that. Creating sub-chain in cases will block their executors. Now it is impossible for anyone else to execute any event chains in those cases. In other words, we make sure that between the check and the execution of the task no other event will be executed in the given case. If all checks will return true all tasks will be successfully synchronised.

Reactive programming could improve the performance of our implementation in the future. Creating thread executor for each sub-chain can potentially lead to exceeding the maximum number of processes allowed by the operating system. Since reactive programming allows to use non-blocking operations new thread executors would not be necessary and sub-chain events could be executed by the cases executor.

References

1. Mažári, J., Juhás, G., Mladoniczky, M.: Petriflow in actions: Events call actions call events. Algorithms and Tools for Petri Nets pp. 21–26 (2018)
2. Mladoniczky, M., Juhás, G., Mažári, J.: Process communication in petriflow: A case study. Algorithms and Tools for Petri Nets pp. 27–32 (2018)

Bericht zur Konsolidierung der Workflow-Modellierung in Renew

Marcel Hansson und Daniel Moldt

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

Zusammenfassung. Prozesse und deren Modellierung lassen sich sehr gut mit (höheren) Petrinetzen und anderen Formalismen darstellen. Die graphische Darstellung der Zusammenhänge von einzelnen Aktionen, Ereignissen und den teilweise eingesetzten Artefakten fällt oft leicht. Seit dem Beginn der Entwicklung des Werkzeugs Renew wurden daher Prozesse und Modellierungstechniken immer wieder in ihren vielen Varianten unterstützt. Nach mehr als zwei Jahrzehnten Entwicklung stellt sich die Frage nach einer geeigneten Konsolidierung der vorhandenen Software. Dazu liefert dieser Beitrag einen Zwischenbericht, bei dem Fragen zum Anwendungsbereich und zur unterstützenden Software behandelt werden.

Schlüsselwörter: Workflow-Modellierung, Renew, höhere Petrinetze, Referenznetze, Softwarearchitektur

Einleitung

Prozessmodellierung ist ein wichtiger Bereich der Modellierung in der Informatik. Dafür werden zahlreiche Modellierungstechniken für Geschäftsprozesse und Arbeitsabläufe eingesetzt, wie z.B. Workflownetze [25], erweiterte ereignisgesteuerte Prozessketten (eEPKs) [19], Aktivitätsdiagramme der UML [14] oder Business Process Model and Notation (BPMN) [11]. Mit dem *Reference net workshop* Werkzeug (Renew) [16] lassen sich Petrinetze modellieren und simulieren. Zudem wurden für Renew zahlreiche Plugins entwickelt, die es ermöglichen zum einen Modelle in anderen Modellierungstechniken zu erstellen und zum anderen auch mit Hilfe von Referenznetzen auszuführen. Weiterhin existieren Analyseverfahren und -werkzeuge, die entweder innerhalb von Renew für Lehrzwecke integriert wurden oder als externes Tool mittels einer Exportfunktion verwendet werden können.

Einzelne Lösungen/Implementierungen und Plugins sind bereits bis zu zwei Jahrzehnte alt und bedürfen teilweise einer Überholung sowohl bzgl. der Anwendung als auch der Softwareimplementation. Einige sind aufgrund der kontinuierlichen Weiterentwicklung von Java und/oder Renew nicht mehr voll funktionsfähig oder lassen sich nicht (einfach) miteinander kombinieren.

Mit diesem Beitrag wird berichtet, wie die bisher entwickelten Plugins, die in Hinblick auf Workflow-Modellierung, -Simulation und -Analyse entwickelt wurden, dokumentiert und mittels einer homogenisierten Umgebung verbessert und zueinander kompatibel gemacht werden können. Dabei dient die Plugin-Architektur von Renew als Grundlage für eine Konsolidierung. Das Schichten- und Modulkonzept von Java als zentrale Implementierungssprache zur Unterstützung dieser Bearbeitungen wird kurz diskutiert. Als langfristiges Ziel soll für Renew die Modellierung, Simulation und Analyse von Workflows und Prozessen mittels interner und externer Werkzeuge bestmöglich unterstützt werden. Dieser Beitrag konzentriert sich dabei vorwiegend auf Workflownetze (siehe [25]) in Verbindung mit dem Werkzeug Renew, welches sich zur Modellierung, Analyse und Simulation von Workflownetzen eignet, da die meisten relevanten Fragestellungen auch für dieses Thema adressiert werden müssen.

Umfeld

Für Workflows, Geschäftsprozesse, Arbeitsabläufe etc. existieren zahlreiche Modellierungsmöglichkeiten und zu den einzelnen Modellierungstechniken (-sprachen) (siehe oben) existiert eine Vielzahl von Werkzeugen für Modellierung, Verifikation, Validierung und Simulation der Modelle (siehe z.B. [26,16]). Eine allgemeine Übersicht über Petrinetzwerkzeuge lieferten [8,21]. Eine (unvollständige) Liste von Petrinetzwerkzeugen wird auf der Webseite der Petrinetz-Community geführt: `http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/`

Renew

Renew ist ein Java-basiertes Werkzeug [16], welches mehrere Petrinetzvarianten abdeckt, insbesondere Referenznetze. Referenznetze sind gefärbte Petrinetze mit Java-Anschriften, synchronen Kanälen (siehe [13,4]) und Referenzen auf Netze als Marken, wodurch Netze-in-Netzen darstellbar werden [15]. Es ermöglicht die Modellierung und Simulation von Referenznetzen mittels einer Plugin-Architektur, wodurch die Funktionalität von Renew durch die Plugins dynamisch und modular ergänzt werden kann. Im Laufe der letzten beiden Jahrzehnte sind zahlreiche Plugins mit verschiedensten Funktionen entwickelt, untersucht und erprobt worden.

Ein Plugin kann Renew zum Beispiel um einen neuen Formalismus oder andere Funktionalitäten (z.B. für Analyse) erweitern. Auf dieser Basis wurden auch Multiagentensystemanwendungen und zahlreiche Workflow-Plugin-Varianten entwickelt. Der Kern von Renew wurde um die Jahrtausendwende entwickelt und wird seitdem regelmäßig aktualisiert. So wurde eine neue Benutzungsoberfläche entwickelt ([30]) und eine prototypische Umstellung auf Java 11 einschließlich einer Umstellung auf das Modulkonzept von Java durchgeführt ([5]). Zum Teil führte dies dazu, dass einzelne Plugins nicht mehr (vollständig) einsetzbar sind, da die Interaktion mittels der neuen Benutzersoberfläche eine diesbezügliche Anpassung erfordert. Die Wartung und Konsolidierung der Workflow-bezogenen

Plugins werden aktuell in der BSc-Arbeit von Hansson [9] bearbeitet. Aufgrund des Umfangs und der Heterogenität der Plugins ergeben sich zahlreiche Schwierigkeiten, die hier beschrieben und diskutiert werden.

Insgesamt ist also eine grundsätzliche Überarbeitung der vorhandenen Plugins erforderlich. Unser Ziel ist es, für den Bereich der Workflows alle in Renew verfügbaren Funktionalitäten (Plugins) und die Erweiterungsmöglichkeiten in einem neuen Rahmenwerk als spezielle Renew-Variante zusammenzufassen.

Workflow-Plugins

Wie oben bereits erwähnt, gibt es zahlreiche Plugins für Renew, die einen Bezug zu Workflows haben. Dabei existieren Plugins, die Renew um neue Modellierungsmöglichkeiten erweitern, neue Formalismen einführen oder neue Analyseverfahren bereitstellen (für mehr Details siehe `https://paose.informatik.uni-hamburg.de/paose/wiki/WorkflowConsolidation`). Da auch die meisten Plugins für Workflows länger nicht gewartet wurden und ihre Dokumentationen sehr lückenhaft sind, ist es lohnenswert, einen Blick auf jedes Plugin zu werfen um diese zu dokumentieren und nach Möglichkeit zu aktualisieren. Im Nachfolgenden wird eine knappe Übersicht über die vorhandenen Plugins und Arbeiten geboten.

Modellierung und neue Formalismen

Da Workflow-Modellierung ein zentraler Anwendungsbereich für Petrinetze ist, wurde von Jacob eine Workflow-Transition mittels eines WF-Formalismus eingeführt (siehe [12]). Durch die Bereitstellung eines einfachen Referenznetz-basierten Formalismus kann somit für die Simulation eine Aktivität, beschrieben durch ein Referenznetz, als Workflow-Transition ausgeführt werden. Das Besondere an dieser Transition ist, dass sie mit dem Schalten anfängt (sie ist dann *markiert*), wenn ein Nutzer die zugehörige Aktivität, die dem Task zugewiesen ist, annimmt. Anschließend kann sie normal beendet werden oder aber die Aktivität kann abgebrochen werden, so dass die Anfangsmarkierung vor dem Beginn des Schaltens wieder hergestellt wird. (siehe [12]). Zugehörige Konzepte der Rollen und Managementoptionen von WfMS (Workflow Managementsystemen) wurden ebenfalls implementiert. Die Netzstruktur wurde nicht weiter beschränkt, was eine generelle Nutzung der Workflow-Transition erlaubt. Intensive Verwendung hat dieses Konstrukt z.B. in [23] für die Bearbeitung von Aufgabenblättern gefunden.
Mit den hierarchischen Workflownetzen wurde von Nikolas Lohmann ein auf hierarchischen Netzen (siehe [7]) basierender Formalismus geschaffen, der es erlaubt komplexe geschachtelte Workflowmodelle zu erstellen (siehe [17]). Dank eines hierarchischen Baums ermöglicht das Plugin den schnellen Wechsel zwischen verschiedenen Sichten.
Ein BPMN-Plugin ermöglicht es mit Renew BPMN-Diagramme zu erzeugen und diese in Referenznetze umzuwandeln (siehe [11]).

Weitere Beiträge wurden in der Verknüpfung von Workflows und Agentensysteme geleistet. So wurden von Reese, Duvigneau, Cabac, Wester, Markwardt, Bendoukha, Wagner und anderen im Rahmen des petrinetzbasierten, agenten- und organisationsorientierten Softwareentwicklungs(Paose)-Ansatzes das Paradigma der Agenten mit dem Konzept der Workflows / WfMS kombiniert und integriert (siehe [20,6,3,28,18,2,27]). Dies ermöglicht die Entwicklung komplexer, verteilter, adaptiver Systeme, die sowohl komplexe Strukturen als auch komplexe Prozesse beinhalten.

Analyse-Werkzeuge

Zum Nachweis von Eigenschaften ist neben der Ausführung / Simulation, die in den oben genannten Themenbereichen einen Schwerpunkt bildet, eine formale Analyse notwendig. Im Bereich der Analyseverfahren hat Renew mehrere Plugins um Netze zu analysieren. Das NetAnalysis-Plugin ermöglicht es einen Erreichbarkeitsgraphen zu erzeugen, verschiedene strukturelle Eigenschaften, wie z.B Free-Choice, zu prüfen und eine Reduktionsanalyse durchzuführen (siehe [10]). Andere interne Werkzeuge für einfache P/T-Netze bieten z.B. einen einfachen ModelChecker für CTL und LTL-Formeln (siehe [24]) oder die Analyse von Workflownetzen auf Korrektheit an (siehe [1]). Durch Plugins wird ebenfalls die Nutzung externer Werkzeuge wie Maria, Woflan, Prom, Lola, GreatSPN etc. ermöglicht (siehe die Petrinetzwerkzeuglisten auf der Webseite der Petrinetz-Community). Dies gelingt entweder über die Einbettung in ein Plugin oder über den PNML-Export.

Konsolidierung

Die Bandbreite der oben genannten Beispiele verdeutlicht die Herausforderungen für die aktuelle Arbeit (siehe [9]) an der Konsolidierung unserer zahlreichen Plugins. Die Konsolidierung findet im Kontext mehrerer anderer Arbeiten statt, die architekturelle und implementatorische Alternativen/Ergänzungen (siehe [5]), neue Oberflächen (siehe [30]), sprachliche Erweiterungen (z.B. Curry als Beschriftungssprache (siehe [22]) oder Entwicklung eines Analyserahmenwerks (siehe [29])) etc. adressieren.

Softwaretechnisch besteht die Arbeit daher im Refactoring, allgemeiner Wartung und zielgerichtetem Umbau der jeweiligen Plugins und Softwarekomponenten. Dabei lassen sich Plugins aufgrund ihrer separaten Struktur besser pflegen als die in Renew integrierten Softwarekomponenten, die (in)direkter Bestandteil von Renews Kernfunktionalität sind. Vorrangig wird die Software auf ihre inhaltliche Notwendigkeit geprüft. Nur bei hinreichendem Bedarf werden diese dann überarbeitet. In Hinblick auf die Bereitstellung einer Workflow-Suite werden zudem einzelne Softwarebestandteile zerlegt, so dass sie in Form neuer Varianten besser kombinierbar werden und die alte Software vollständig ersetzen. Zudem ist das Zusammenspiel mit der neuen Oberfläche, die von Martin Wincierz erstellt wurde (siehe [30]), aufgrund neuer Möglichkeiten und einiger Einschränkungen eine weitere Herausforderung bei der Softwarewartung.

Schluss

Schwachstellen von Renew werden systematisch beschrieben und zielgerichtet behoben. Neueste Verfahren der Softwareerstellung werden eingesetzt und veraltete Sprachelemente ersetzt. Damit steht für die Modellierung, Simulation und Analyse ein homogener Satz an Plugins in Renew zur Verfügung, der verschiedene Modelle im Bereich der Workflow-Modellierung oder allgemeiner der Prozessmodellierung abdeckt.

Neben der bisherigen Architektur ist mittelfristig eine Variante von Renew als Micro Service Architektur vorgesehen. Hierbei helfen die funktionale Zerlegung und die bisher prototypische Umstellung auf das neue Modulkonzept von Java. Die Architektur ist darauf ausgelegt, dass wir leicht neue Kombinationen von Konzepten und Konstrukten vornehmen können. In Hinblick auf domänenspezifische Sprachen (DSLs), die in unserer Gruppe mittels Metamodellierung konzipiert werden, stehen uns damit auch wesentlich ausdrucksstärkere Mittel zur Verfügung.

Literatur

1. L. Azzalini. Sichtung, Diskussion und Nutzung allgemeiner Verifikationsverfahren für Petrinetze und prototypische Umsetzungen ausgewählter Verfahren im Werkzeug Renew. Diploma thesis, University of Hamburg, Dept. of Informatics, 2017.
2. S. Bendoukha. *Multi-Agent Approach for Managing Workflows in an Inter-Cloud Environment*. Dissertation, University of Hamburg, Dept. of Informatics, 2017.
3. Lawrence Cabac. *Modeling Petri Net-Based Multi-Agent Applications*. Dissertation, University of Hamburg, Dept. of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg, April 2010.
4. S. Christensen and N. D. Hansen. Coloured petri nets extended with channels for synchronous communication. In R. Valette, editor, *Application and Theory of Petri Nets 1994, 15th International Conference, Zaragoza, Spain, June 20-24, 1994, Proceedings*, volume 815 of *Lecture Notes in Computer Science*, pages 159–178. Springer, 1994.
5. A. Daschkewitz. Modularisierung des Renew Plugin Systems (geplant für Oktober 2019). Master thesis, University of Hamburg, Dept. of Informatics, 2019.
6. Michael Duvigneau. *Konzeptionelle Modellierung von Plugin-Systemen mit Petrinetzen*. Logos Verlag, Berlin, 2010.
7. Rainer Fehling. *Hierarchische Petrinetze: Beiträge zur Theorie und formale Basis für zugehörige Werkzeuge*. Verlag Dr. Kovač, Hamburg, 1992.
8. F. Feldbrugge. Petri net tools. In G. Rozenberg, editor, *Advances in Petri Nets 1985*, volume 222 of *LNCS*, pages 203–223. Springer, 1985.
9. M. Hansson. Überblick über Workflownetze und Konsolidierung der workflowbezogenen Renew-Plugins (geplante Fertigstellung Dezember 2019). Bachelor thesis, University of Hamburg, Dept. of Informatics, 2019.
10. M. Haustermann. Analyse von Workflows auf Basis von Petrinetzen. Bachelor thesis, University of Hamburg, Dept. of Informatics, March 2010.
11. M. Haustermann. BPMN-Modelle für petrinetzbasierte agentenorientierte Softwaresysteme auf Basis von Mulan/Capa. Master thesis, University of Hamburg, Dept. of Informatics, September 2014.

12. T. Jacob. Implementierung einer sicheren und rollenbasierten Workflowmanagement-Komponente für ein Petrinetzwerkzeug. Diploma thesis, University of Hamburg, Dept. of Computer Science, 2002.
13. E. Jessen and R. Valk. *Rechensysteme: Grundlagen der Modellbildung*. Studienreihe Informatik. Springer-Verlag, Berlin Heidelberg New York, 1987.
14. C. Knieke. *Modellierung und Validierung von ausführbaren Anforderungsspezifikationen mit erweiterten UML Aktivitätsdiagrammen*. PhD thesis, University of Braunschweig - Institute of Technology, 2011.
15. O. Kummer. *Referenznetze*. Logos Verlag, Berlin, 2002.
16. O. Kummer, F. Wienberg, M. Duvigneau, L. Cabac, M. Haustermann, and D. Mosteller. Renew – the Reference Net Workshop, June 2016. Release 2.5.
17. N Lohmann. Werkzeugunterstützung für hierarchische Workflow-Netze. Diploma thesis, University of Hamburg, Dept. of Informatics, 2012.
18. K. Markwardt. *Strukturierung petrinetzbasierter Multiagentenanwendungen am Beispiel verteilter Softwareentwicklungsprozesse*. PhD thesis, University of Hamburg, Dept. of Informatics, 2013.
19. M. Nüttgens and F. J. Rump. Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In J. Desel and M. Weske, editors, *Promise 2002, 9.-11. Oktober 2002, Potsdam*, volume 21 of *LNI*, pages 64–77. GI, 2002.
20. C. Reese. *Prozess-Infrastruktur für Agentenanwendungen*, volume 3 of *Agent Technology – Theory and Applications*. Logos Verlag, Berlin, 2010.
21. Harald S. An evaluation of high-end tools for Petri-nets. Bericht 9802, Ludwig Maximilians Universität München, Institut für Informatik, München, June 1998.
22. Michael S., D. Moldt, D. Schmitz, and M. Haustermann. Tools for Curry-Coloured Petri Nets. In Susanna Donatelli and Stefan Haar, editors, *PETRI NETS 2019, Aachen, Germany, June, 2019*, volume 11522 of *Lecture Notes in Computer Science*, pages 101–110. Springer, 2019.
23. D. Schmitz, D. Moldt, L. Cabac, D. Mosteller, and M. Haustermann. Utilizing Petri Nets for Teaching in Practical Courses on Collaborative Software Engineering. In *ACSD 2016, Toruń, Poland, 2016*, pages 74–83. IEEE Computer Society, 2016.
24. A. Stiefelmann. Entwicklung und Untersuchung von Model-Checker-Prototypen für ausgewählte beschränkte Referenznetze als Plugin für den Referenznetzsimulator Renew. BSc-Arbeit, University of Hamburg, Dept. of Informatics, 2016.
25. Wil M.P. van der Aalst. Verification of Workflow Nets. In *ICATPN '97: Proceedings of the 18th International Conference on Application and Theory of Petri Nets*, pages 407–426, Berlin Heidelberg New York, 1997. Springer-Verlag.
26. H. M. W. Verbeek, T. Basten, and Wil M. P. van der Aalst. Diagnosing workflow processes using woflan. *Comput. J.*, 44(4):246–279, 2001.
27. T. Wagner. *Petri Net-based Combination and Integration of Agents and Workflows*. PhD thesis, University of Hamburg, Dept. of Informatics, 2018.
28. Matthias Wester-Ebbinghaus. *Von Multiagentensystemen zu Multiorganisationssystemen – Modellierung auf Basis von Petrinetzen*. Dissertation, University of Hamburg, Dept. of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg, 2010.
29. S. Willrodt. Towards Model Checking for Reference Nets (geplant für Dezember 2019). Bachelor thesis, University of Hamburg, Dept. of Informatics, 2019.
30. M. Wincierz. Verbesserung der Erweiterbarkeit und Benutzbarkeit der grafischen Oberfläche des Petrinetz Simulators Renew. Master thesis, University of Hamburg, Dept. of Informatics, 2018.

Spatial Encoding of Systems Using Coloured Petri Nets

George Assaf *, Monika Heiner

Computer Science Institute, Brandenburg Technical University
Postbox 10 13 44, 03013 Cottbus
George.Assaf@b-tu.de, monika.heiner@b-tu.de
https://www-dssz.informatik.tu-cottbus.de

Abstract Modelling spatial information of multiscale systems is crucial for understanding the underlying events happening at different temporal and spatial levels. Unfortunately, the limitation of current modelling approaches including Petri nets is to efficiently model such systems. Colored Petri nets are an excellent formalism for modeling and analysing complex systems. We present two spatial encoding schemes using coloured Petri nets in *Snoopy* tools. We demonstrate each scheme by an example and finally discuss the advantages and disadvantages of each scheme.

Keywords: spatial modelling · simulation · unfolding · coloured continuous, stochastic and hybrid Petri nets

1 Objectives

Complexity and multiscaleness are main characteristics of many systems in which numerous events happen at different temporal and spatial levels. Reaction diffusion systems [4] are a basic example of biochemical processes which develop over time and space. However, Modelling such processes requires taking location information into consideration, for example, diffusion can only be happened between two neighbouring grid positions.

Coloured Petri nets ($\mathcal{PN}^{\mathcal{C}}$) are a powerful modelling mechanism. They combine the expressive power of Petri nets with programming languages [3]. While Petri nets offer a graphical notation for modelling concurrent and synchronized systems, programming languages provide numerous data types which allow to enrich Petri nets with user-defined functions, coloured expressions and other annotations, and thus they allow to model complex systems in a compact fashion which is the most important feature of coloured Petri nets. As standard Petri nets, coloured Petri nets consist of places, transitions and arcs. Moreover, $\mathcal{PN}^{\mathcal{C}}$ is enriched by a set of discrete data types (called colour sets), and a set of expressions that are used to define the initial marking, arc inscriptions, and guards. Each place gets assigned a colour set and may contain distinguishable tokens, represented as a multiset expression over the assigned colour set. A multiset is a set, where one element can occur several times. Each transition gets

*corresponding author

a guard, which is a Boolean expression over variables, constants or functions of the defined colour sets. The guard of a transition has to be evaluated to true for enabling the transition.

Coloured Petri nets are a powerful modelling tool and thus they are excellent choice for modelling processes which develop over time and space where all spatial information is encoded using colours and other annotations. We are going to demonstrate how this can be done using two encoding schemes.

2 First encoding scheme

We are going to translate the idea of this scheme using a well-known biochemical process called Continuous Diffusion [1]. Diffusion is a basic biochemical reaction process which evolves over time and space. It is regarded as the simplest form of passive mobility; diffusion starts with a higher concentration of species at the middle of space, with all other space positions initially set to 0, and then it goes from regions of higher concentration to regions of lower concentration.

By using coloured Petri nets, space is discretised into a grid of one, two or three dimensions. Each dimension is represented by a colour set of type integer; sets of colours are thus discrete and finite. A position at the 2D grid is represented by Cartesian Product of simple colour sets e.g, D1 x D2, where D1 and D2 are colour sets of the first and second dimensions respectively. Diffusion between two neighbouring grid positions is modeled by using a coloured transition. The coloured transition is associated with a guard working as a neighbourhood function (Boolean function); checking the neighburhood relation between two grid positions. It is worth mentioning that the neighbourhood function can be easily defined to check the two, four, eight or diagonal neighbourhood relation which means the inner grid positions have two, four, eight or diagonal neighbours respectively. When a guard is evaluated to true (two grid positions are neighbours), one instance of a coloured transition will fire and re-colouring tokens of the coloured place will take place.

2.1 Example

Let us assume 2D Biochemical Diffusion defined on a grid of 15 x 15, we assume also that diffusion starts with a concentration of 100 from the middle of the grid. To model each dimension, one colour set has to be defined per each dimension e.g, 15 colours/dimension in our example meaning we have 15 colours (positions) per each dimension. To control the diffusion through a grid, a product colour set has to be defined based on the previously defined simple colour sets and then the product colour set has to be assigned to a coloured place. To initialise the marking of the coloured place with 100 at the center of our grid, the marking of coloured tokens has to be assigned by using a marking function which is an expression e.g, 100‘(x=MIDDLE, y=MIDDLE) where x and y are two variables declared on the first and second dimension (colour set) respectively, MIDDLE

is a constant gets assigned an integer value representing the middle of each dimension. The former colour function will assign 100 tokens of colour ‘8‘ from the two colour sets. Since diffusion can only be taken place between two neighbouring grid positions, a Boolean neighbourhood function has to be defined. The Neighbourhood function takes two grid positions (one colour of each dimension) as an input and checks whether the two positions are neighbours or not. The neighbourhood function has to be assigned to the coloured transition as a guard; compare Figure 1 for the whole model.

The following lines explain the definitions of required colour sets, neighbourhood function (each grid position has four neighbours) and the declarations of required variables and constants:

```
const D1 = int with 15; // grid size first dimension
const D2 = D1; // grid size second dimension
const MIDDLE = int with D1/2+1;

colorset CD1 = int with 1−D1; // row index
colorset CD2 = int with 1−D2; // column index
colorset Grid2D = product with CD1 x CD2; // 2D grid

var x , a : CD1;
var y , b : CD2;

fun bool neighbour2D4 (CD1 x , CD2 y , CD1 xn , CD2 yn ) {
// ( xn , yn ) is one of the up to four neighbours of ( x , y )
( xn=x & yn=y− 1) | ( xn=x & yn=y+1)
| ( yn=y & xn=x− 1) | ( yn=y & xn=x+1)
& (1<=xn & xn<=D1) & (1<=yn & yn<=D2) } ;
```

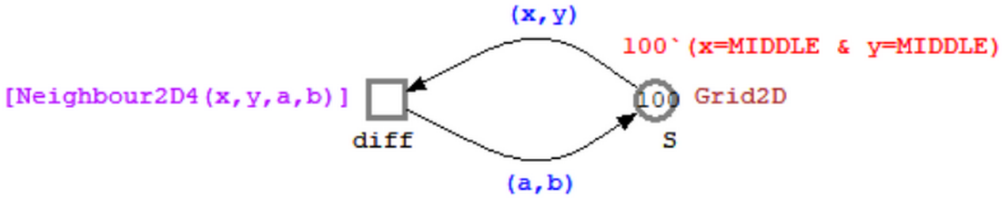

Figure 1: Continuous coloured Petri net of 2D Diffusion and its annotations using the first spatial encoding scheme.

Model simulation Simulation of the model is undertaken in *Snoopy* tools by unfolding the coloured continuous Petri net ($\mathcal{CPN}^{\mathcal{C}}$) model into a standard continuous Petri net. Unfolding of the model is automatically performed in the background which allows to make use of all the existing powerful Petri net analysis and simulation techniques. Figure 2 shows the 2D plot of simulation results at different three time points.

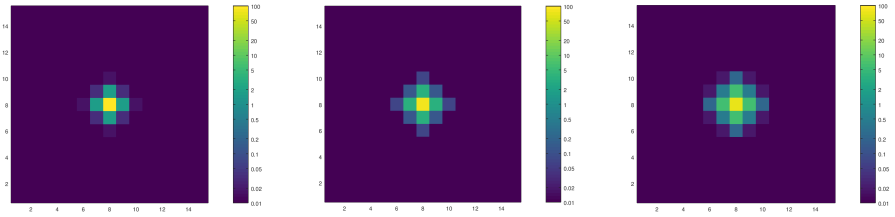

Figure 2: 2D plot of simulation trace at simulation time 20s (left), 40s (middle) and 80s (right).

3 Second spatial encoding scheme

In this scheme, space is encoded by introducing spatial places; places hold tokens representing coordinates of a moving object (location information). Increasing a spatial place means adding tokens to that place which can be achieved by connecting a pre-spatial transition to it using a standard arc and thus increasing the movement on space. On the other hand, decreasing a spatial place means removing tokens from that place which can be achieved by connecting the spatial place to a post-spatial transition using a standard arc and thus decreasing the movement on space. Please note that the spatial places and transitions are standard Petri net elements e.g, discrete places and stochastic transitions; we use the term "Spatial" just to recognise their spatial functionalities in the studied model. To avoid trespassing of the defined finite spatial space, Read and Inhibitory arcs have to be used. The inhibitory arcs have to be weighted by the maximum size of space, whereas the read arcs have to be weighted by the minimum size of space. Figure 3 demonstrates movement of an object over 2D space using this approach. To permit the movement ON/OFF, a circular Petri net component has to be used; compare Figure 4.

3.1 Example

We are going to demonstrate this encoding scheme using a test case called Moving Robot. The Robot has a size of up to 2x2 meters and it moves on a straight line in the center of the room covering a distance of 7 meters. Since it arrives the

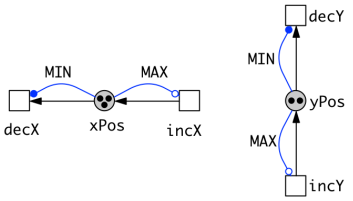

Figure 3: Modelling spatial information of a 2D space.

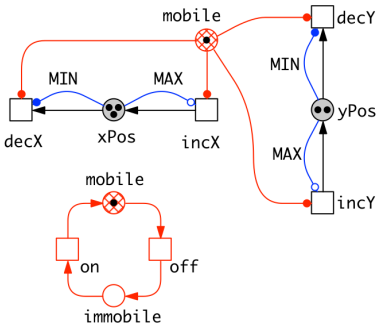

Figure 4: Embedding a circular Petri net component to permit the mobility.

first end, it moves back in the opposite direction and so on. The Robot initially occupies the positions 2 and 3 on the X-Axis and 3 and 4 on the Y-Axis.

Since the Robot has a size of 2x2m, two colour sets have to be defined. The first colour set defines its width and the second one defines its height. Each colour set contains a number of colours equals to the size of Robot on each dimension. We assume that the Robot moves horizontally (on the X-Axis) which means that location information of the Y-Axis needs not to be changed during the movement. For each movement, two positions of space need to be changed on the x-Axis, for example, if the positions 1 and 2 are occupied by the Robot on the x-Axis, the next occupied positions will be 2 and 3. This requires to decorate the read and inhibitory arcs with the minimum and maximum of the two positions of space that the Robot may reach; this can be done using coloured expressions. To model the movement way that the Robot follows, a circular Petri net component with its own variable has to be used in our model. For the whole model, compare Figure 5. The following lines show definitions of the coloured Petri net :

const SIZE = int with 2;//size of Robot on each dimension
//constants declaration of movement borders
const X1MAX = int with 6;
const X2MAX = int with 7;
const X1MIN = int with 2;
const X2MIN = int with 3;

colorset Width = int with 1 − SIZE;// number of occupied positions on x-axis
colorset Height = int with 1 − SIZE;// number of occupied positions on y-axis

var x : Width;//row index
var y : Height;//column index
var s : Width;// switching movement from right to left and vice versa.

Model simulation Simulation of the Moving Robot model is undertaken by automatic unfolding the coloured stochastic Petri net ($\mathcal{SPN}^{\mathcal{C}}$) into a standard stochastic Petri net ($\mathcal{SPN}$). Figure 6 depicts the 2D plot of simulation results at different three time points. Please note that the coloured positions at the 2D plot mean that those positions have been visited more than one time by the Robot.

For reasons of space, all references to Figures of this example relates to the Appendix.

4 Discussion

We presented two spatial encoding schemes by exploiting the expressive power of coloured Petri nets. Simulation is undertaken at the uncoloured level by unfolding in *Snoopy* tools [2]. Both encoding schemes can be applied to continuous, stochastic and hybrid Petri nets. The first encoding scheme is characterized by descretising space using finite colour sets and thus finite universe. Moreover, all space-related information is encoded in colour, and thus chaining the notion of space means adaption of colour-related definitions; coloured Petri net structure remains the same. A drawback of this scheme is that the size of the unfolded model increases extremely as the size of space increases. This has an impact on the analysis and simulation efficiency. On the other hand, space is represented by introducing coordinate places using the second scheme, and thus local states of each moving object can be tracked which is the main feature that the second encoding scheme outperforms the first scheme. Additionally, each coordinate place represents one dimension of the encoded space; the positioning information of an object is characterized by the number of tokens on these places. Similar to the first encoding scheme, changing the notion of space using the second spatial encoding scheme means adaption colour-related information. The next step will be modelling more complicated spatial models using the second encoding scheme; this may require further development on *Snoopy* tools which is our powerful modelling, animation and simulation tool of various types of Petri net classes. Interested users can reproduce the results of the two presented models in this paper by retrieving the required source materials from `https://www-dssz.informatik.tu-cottbus.de/DSSZ/Software/Examples`.

References

1. Gilbert, D., Heiner, M.: Petri nets for multiscale Systems Biology. `http://multiscalepn.brunel.ac.uk/`, Brunel University, Uxbridge/London (2011)
2. Heiner, M., Herajy, M., Liu, F., Rohr, C., Schwarick, M.: Snoopy – A Unifying Petri Net Tool. In: Proc. ATPN. pp. 398–407. Springer, LNCS 7347 (2012)
3. Liu, F.: Colored Petri Nets for Systems Biology. Ph.D. thesis, BTU Cottbus, Computer Science Institute (January 2012)
4. Liu, F., Blaetke, M., Heiner, M., Yang, M.: Modelling and simulating reaction diffusion systems using coloured Petri nets. p. 297308. Elsevier (2014)

5 Appendix

Here we provide two screenshots outlining the workflow of modelling and simulation the Moving Robot test case using the second spatial encoding scheme.

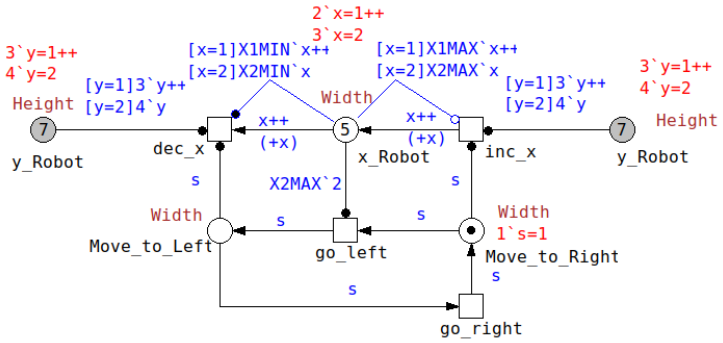

Figure 5: Stochastic coloured Petri net of the moving Robot model using the second spatial scheme.

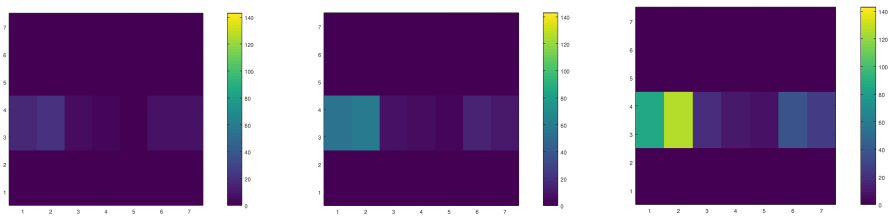

Figure 6: 2D plot of simulation trace at simulation time 20s (left), 40s (middle) and 80s (right).

Implementation Semantics of Petriflow Models

Gabriel Juhás[1,2,3], Juraj Mažári[1,2], Milan Mladoniczky[1,2], and Ana Juhásová[3]

[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., Jána Stanislava 28/A, 841 05 Bratislava, Slovakia
[3] BIREGAL s. r. o., Klincova 37/B, 821 08 Bratislava, Slovakia

Petriflow is a Petri net based language for developing process driven client server applications, see e.g. [1] and [2] for more details. Briefly, Petriflow uses Petri nets (with read arcs, inhibitor arcs and reset arcs) to describe workflow processes where transitions represent tasks. Data variables can be attached to a process, creating its data set, and they can be associated with tasks creating so called data references in tasks. Similarly, roles can be attached to a process, and they can be associated with tasks creating so called role references in tasks. Petriflow provides also events for constructors and destructors, that create and destroy instances of workflow processes in runtime. In fact, the semantics of a Petriflow instance is given using more detailed Petri net derived from the original Petri net modelling the workflow. By calling constructor of a process a copy of such Petri net is created.

In this more detailed Petri net, also called Petriflow event net, transitions of the original net representing tasks are refined into several transitions representing events of the task (such as assign, finish, cancel, reassign). Similarly, places representing that the task is being executed and the task is not being executed are added. In Petriflow task autoconcurrency is not supposed, i.e. there cannot be two copies of a task running in parallel in the same instance. Implicit places representing for each variable the states, namely whether it is locked or unlocked are added. Implicit transitions representing the events on data variables, such as setData are added.

For data references, we distinguish whether they lock or not their data variables. Namely, if a data reference lock its data variable for its task by assigning the task, then no other task that has a lock data reference to the same data variable can be assigned. This is realized in the event net by an ingoing arc from the unclocked place of the data variable to the locked places. For each data variable, two values are stored, one is called stable value and another is called actual value. Once a constructor is creating an instance, both values are equal. If a data variable is locked by firing assign event transition of a task with lock data reference to this data variable, its actual value can be changed by firing setData event transition. By firing cancel event transition of the task, the stable value is assigned to the actual value, while by firing finish event transition, the actual value is assigned to the stable value. Thus, when a data variable is unlocked then stable and actual values are equal, i.e. when no task of an instance is executed, then stable and actual values are equal.

The semantics of the Petriflow event net representing an instance is given by firing event transitions. For the purpose of this paper, the state of the instance

itself is given by the Petriflow event net marking and the two values of each data variable. The state is changed by the event transition firing, where cancel, finish and setData change the state of values of data variables as described.

As it was stated, the purpose of the Petriflow language is to enable development of client-server based applications, in which, among others, client request firing of event transition of process instances. Obviously, clients also request some responses that do not require firing of event transitions in process instances on server. Some of such requests include search queries, that as a response waits for instances or list of instances, or tasks fulfilling the query. Another client requests just do not require a server response at all, such as front end requests on a client machine. In this contribution, we are concerning only on the processing of those parts of client requests that require response of a server in form of a firing of an event transition in a process instance.

Thus, we only consider two kinds of requests, one asking whether an event transition is enabled, and other asking an event transition to fire. The response is an object obtaining the status of the response. It also may obtain some specific attributes, such as actual value in case of the request for event transition setData.

If a transition is enabled to fire, then the response for the request for enabledness is the object obtaining the status true, otherwise the response is the status false. If a transition is enabled to fire, then the response for the request for firing is the object containing the status true, otherwise the response is status false.

Petriflow also provide actions. Namely, each event transition can have a pre-action and a post-action, which are pieces of codes, that can contain requests for event transition firing either in the same instance or in other instance of the same process or another process. The requests of an instance on server can come from many clients independently and in any frequency. The requests can be in conflict. It is not necessary to execute the requests of an instance sequentially in one thread, but it is necessary to decide for any request before it is processed whether it is in conflict with the currently processed requests of the instance. However, it is supposed that there exists a unique boundary such that the number of executed threads for each instance is bounded by this boundary. Thus, it is supposed that implementation of a Petriflow engine has a priority queue and a unique scheduler for each instance. Once a request has the highest priority, it creates a sub-queue with the highest priority containing requests from its pre-action, followed by the request itself and the requests from its post-action. In order to enable priority response to a sequence of requests called from an action in another instance, one can use a special request called begin, which will create a sub-queue with the highest priority (once the begin request has the highest priority) and will finish this sub-queue by a special request called end.

References

1. Mažári, J., Juhás, G., Mladoniczky, M.: Petriflow in actions: Events call actions call events. Algorithms and Tools for Petri Nets pp. 21–26 (2018)
2. Mladoniczky, M., Juhás, G., Mažári, J.: Process communication in petriflow: A case study. Algorithms and Tools for Petri Nets pp. 27–32 (2018)