Generation of Petri Nets Using Structural Property-Preserving Transformations

1,2 R.A. Nesterov, ORCID: 0000-0002-4162-9070 <rnmesterov@hse.ru>
1 S.Yu. Saveliev, ORCID: 0000-0003-1660-2615 <syusavelev@edu.hse.ru>
1 HSE University, 11, Pokrovsky boulevard, Moscow, 109028, Russia
2 Universita’ degli Studi di Milano-Bicocca, I Piazza dell’Ateneo Nuovo, 20126, Milan, Italy

Abstract. In this paper, we present an approach to the generation of Petri nets exhibiting desired structural and behavioral properties. Given a reference Petri net, we apply a collection of local refinement transformations, which extends the internal structure of the reference model. The correctness of applying these transformations is justified via Petri net morphisms and by the fact that transformations do not add new deadlocks to Petri nets. We have designed two Petri net refinement algorithms supporting the randomized and fixed generation of models. These algorithms have been implemented and evaluated within the environment of the Carassius Petri net editor. The proposed approach can be applied to evaluate and conduct experiments for algorithms operating with Petri nets.

Keywords: Petri nets; morphisms; property-preserving transformations; generation of models

For citation: Nesterov R.A., Saveliev S.Yu. Generation of Petri Nets Using Structural Property-Preserving Transformations. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 3, 2021, pp. 155-170. DOI: 10.15514/ISPRAS-2021-33(3)-12

Acknowledgments. This work is supported by the Basic Research Program at the National Research University Higher School of Economics (HSE University), Russia.

Генерация сетей Петри с помощью структурных трансформаций, сохраняющих поведенческие свойства

1,2 Р.А. Нестеров, ORCID: 0000-0002-4162-9070 <rnmesterov@hse.ru>
1 С.Ю. Савельев, ORCID: 0000-0003-1660-2615 <syusavelyev@edu.hse.ru>
1 Национальный исследовательский университет Высшая школа экономики, 109028, Россия, Москва, Покровский бульвар 11
2 Миланский университет-Бикокка, Италия, ул. Милан, пл. Атено Нуово, 1

Аннотация. В работе предложен подход к генерации сетей Петри, обладающих желаемыми структурными и поведенческими свойствами. На вход подаётся эталонная сеть Петри, к которой применяется набор локальных трансформаций, расширяющих её внутреннюю структуру. Корректность применения этих трансформаций обусловливается тем, что они порождают морфизмы на сетях Петри, а также не добавляют новых тупиков при расширении эталонных сетей Петри. Таким образом, сохраняются поведенческие и структурные свойства, которыми обладает эталонная сеть Петри. Были разработаны алгоритмы фиксированной и случайной генерации сетей Петри. Эти алгоритмы реализованы в виде расширения для редактора сетей Петри Carassius. Кроме того, проведена экспериментальная оценка разработанных алгоритмов. Предлагаемый подход к генерации сетей Петри и разработанные алгоритмы могут применяться для проведения комплексной экспериментальной оценки и нагруженного тестирования алгоритмов, на вход которых подаются модели поведения процессов в виде сетей Петри.

Ключевые слова: сети Петри; морфизмы; трансформации; сохранение свойств; генерация моделей

Для цитирования: Нестеров Р.А., Савельев С.Ю. Генерация сетей Петри с помощью структурных трансформаций, сохраняющих поведенческие свойства. Труды ИСП РАН, том 33, вып. 3, 2021 г., стр. 155-170 (на английском языке). DOI: 10.15514/ISPRAS–2021–33(3)-12.

Благодарности. Работа поддержана Программой фундаментальных исследований Национального исследовательского университета «Высшая школа экономики» (НИУ ВШЭ), Россия.

1. Introduction

Petri nets are widely used to formally represent the behavior of distributed systems for their precise semantics, which helps to prove many crucial behavioral properties, including boundedness, deadlock-freeness, covering by place invariants, and others [1]. The automated verification of these properties is supported by different algorithms. For instance, covering by place invariants can be decided using linear algebraic techniques [2].

The software implementation of algorithms operating with Petri nets naturally requires the preparation of model sets that exhibit the specific structural and behavioral properties. Such sets of models are then used for the thorough evaluation of algorithms under development. Firstly, the manual generation of Petri nets with specific properties is a time-consuming activity. Moreover, if one has to prepare a particularly large-scale model, then there arises an additional task to verify the necessary properties of this model. The computational cost of such a verification can grow too fast due to the well-known state explosion problem of distributed systems, when the number of reachable states grows exponentially compared to the size of a system model.

In our paper, we propose an approach to the generation of Petri nets based on structural transformations. Firstly, a reference Petri net is constructed. This model has all the target structural and behavioral properties. Secondly, applying a collection of local transformations that extend the internal structure of a reference Petri net, we obtain a refinement exhibiting the same properties as an initial reference model. The general scheme of this approach is schematically shown in Fig. 1, where a refinement is a result of applying k local transformations to a reference Petri net. Note that a refinement has the more sophisticated structure than a reference model. Transformations, considered in our study, are called local, since they change only the specific part of a model, while the rest of the model remains untouched. The mathematical framework of these transformations is responsible for preserving the structural and behavioral properties of a reference Petri net. In addition, the application of transformations requires only the local checks of structural constraints.

We consider two generation schemes: fixed and randomized. Within the fixed generation of Petri nets, a specific sequence of transformations is applied to an initial reference model. Conversely, the randomized generation is a based on a non-deterministic choice of transformations.

Thus, the main results of our paper are as follows:

- the algorithms for the fixed and randomized generation of Petri nets from a given reference model;
- the software implementation and evaluation of these algorithms within the environment of the Carassius Petri net editor [3].

The remainder of this paper is structured as follows. The next section discusses the related research. In Section 3, we define a class of Petri nets considered in our paper. Section 4 describes the mathematical framework behind a collection of structural transformations that are used to refine
Carassius Petri editor to provide the generation of Petri nets with the desired structural and behavioral properties.

3. Elementary Net Systems

In our study, we consider the generation of elementary net systems (EN-systems). They form the fundamental class of Petri nets used to model the control-flow of distributed systems, while other aspects such as data and time are not considered. The structure of EN-systems is modeled using bipartite graphs with two kinds of nodes: places and transitions. Places in an EN-system can carry at most a single token. Thus, they can be interpreted as boolean conditions, truth values of those are exchanged by transition firings. Below we provide the formal definitions based on [15] concerning the structural and behavioral aspects of EN-systems.

Let $\mathcal{S}$ be a set. The set of all finite non-empty sequences over $\mathcal{S}$ is denoted by $\mathcal{S}^+$, and $\mathcal{S}^+ = \mathcal{S}^+ \cup \{\varepsilon\}$, where $\varepsilon$ is the empty sequence.

Definition 1 (Net): A net is a triple $N = (P, T, F)$, where $P$ and $T$ are two disjoint sets of places and transitions, and $F \subseteq (P \times T) \cup (T \times P)$ is flow relation. For any node $x \in P \cup T$:
1) $x = \{y \in X | (y, x) \in F\}$ is the preset of $x$.
2) $x^+ = \{y \in X | (x, y) \in F\}$ is the postset of $x$.
3) $x \cdot x^+ = \{x \cup \varepsilon\}$ is the neighborhood of $x$ in $X$.

The standard graphical notation is adopted: places are drawn as circles, and transitions are shown as boxes.

In our work, we consider nets without self-loops, i.e., $\forall x \in P \cup T: x \cdot x^+ = \emptyset$, and isolated markings $\varepsilon$. The set of all firing sequence in $\mathcal{N}$ generated by $\mathcal{N}$ is denoted $\mathcal{N}$. A marking in a net $\mathcal{N} = (P, T, F)$ is a subset of its places, i.e., $m \subseteq P$. Pictorially, markings are depicted by placing black dots inside corresponding places. A marking $m$ in a net $\mathcal{N} = (P, T, F)$ has a contact if $\exists t \in T$: $m$ and $m \cap t = \emptyset$.

Definition 2 (EN-system): An elementary net system (EN-system) is a couple $(N, m_0)$, where $N = (P, T, F)$ is a net, and $m_0 \subseteq P$ is the initial marking.

The behavior of an EN-system is defined by the firing rule. A marking $m$ in a net $\mathcal{N} = (P, T, F)$ enables transition $t \in T$ if $\exists t \subseteq m$ and $m \cap t \neq \emptyset$. Enabled transitions may fire.

A sequence $w \in T^*$ is a firing sequence in an EN-system $\mathcal{N} = (P, T, F, m_0)$ if $w = t_1t_2...t_n$ and $m_0(t_1)m_1(t_2)...m_{n-1}(t_n)$. Then we write $m(w)m_n$. The set of all firing sequence in $\mathcal{N}$ is denoted by $FS(\mathcal{N})$.

A marking $m$ in a net $\mathcal{N} = (P, T, F, m_0)$ is reachable exists $3w \in FS(\mathcal{N})$: $m_0[w]m$. The set of all reachable markings from $m$ will be denoted $[m]$.

A reachable marking in $[m] \in [m_0]$ is reachable exists $3w \in FS(\mathcal{N})$: $m_0[m]w$. The set of all markings reachable from $m$ will be denoted $[m]$.
components: A (dotted line), B (dashed line), and C (dash-dotted line). Sequential components A, B, C have independent parts (concurrent behavior) and synchronous transitions, e.g., transition $t_4$, which will be executed by A and B simultaneously. Each token of a reachable marking in an SMD-EN system can be characterized by sequential components. For instance, a token in $p_2$, shown in fig. 2, belongs to two of three sequential components: A and B.

Further, we work with SMD-EN-systems unless otherwise stated explicitly. Thus, we omit the SMD abbreviation in their descriptions.

4. Refinement of EN-systems

In this section, we discuss the mathematical framework behind our approach to the generation of EN-systems using refinement transformations. Firstly, we consider $\alpha$-morphisms formalizing relations between abstract and refined EN-systems [16]. Secondly, we describe a set of local EN-system transformations that induce corresponding $\alpha$-morphisms and define them in a step-wise manner [17].

4.1 Morphisms

A class of $\alpha$-morphisms has been introduced in [16] to formalize relations between an abstract EN-system and its refinement, where subnets in a refined model can substitute places in an abstract model. Using the example shown in fig. 3, we briefly discuss the main intuition behind $\alpha$-morphisms.

An $\alpha$-morphism $\varphi: N_1 \to N_2$ is a total surjective map from the set of nodes of a refined EN-system $N_2$ on the set of nodes of an abstract EN-system $N_1$. Places in an abstract EN-system can be refined with 'emph[acyclic] subnets in its refinement. For example, subnet $N_1$ ($\varphi^{-1}(p_1)$) refines place $p_1$ in $N_2$ shown in fig. 3. The refinement of places can also result in a split of transitions, e.g., transition $t_5$ in $N_2$ is split into two transitions, $t_{14}$ and $t_{12}$, in $N_1$, as shown in fig. 3.

An $\alpha$-morphism $\varphi: N_1 \to N_2$ is defined in terms of how transitions in $N_1$ are mapped to nodes in $N_2$. If the image of transition in $N_1$ is also a transition in $N_2$, then their neighborhoods should correspond as well. For instance, since the image of transition $t_4$ in $N_1$ shown in fig. 3 is transition $t_1$ in $N_2$, the image of $\bullet t_4 \bullet$ is $\bullet t_1 \bullet$. If the image of transition in $N_1$ is a place in $N_2$, then the image of its neighborhood is this place as well. For instance, transitions in subnet $N_1$ ($\varphi^{-1}(p_2)$) are mapped to place $p_1$ in $N_2$ as well as their neighborhoods.

These constraints combined with several other structural restrictions imposed on subnets in a refined EN-system, discussed in detail in [16], assure the main motivation behind $\alpha$-morphisms: a refinement should behave «in a similar way» as an abstract model does. Whenever there is a token in a place in abstract EN-system, there exists the possibility to fire a transition that puts a token into a corresponding subnet in a refined EN-system, s.t. the other input transitions remain disabled afterwards (see Lemma 1 in [16]).

The direct application of $\alpha$-morphisms is rather difficult for their sophisticated structural constraints. An approach based on the subsequent application of local structural transformations [17] comes to the aid of this problem. It is discussed in the following section, where we redefine the refinement notion through these transformations.

4.2 Refinement Transformations

The main idea of structural transformations, defined in [17], lies in a step-by-step construction of a refined model from an abstract one. These transformations are called local because they change only a specific subnet in an initial model, while the rest of the model remains untouched. As shown in [17], every step of applying a transformation to an EN-system induces a corresponding $\alpha$-morphism from a transformed model to an initial one. Moreover, after a series of transformations is applied to an EN-system, there will be an $\alpha$-morphism from a result EN-system towards an initial EN-system. Fig. 4 shows the main idea of this approach, where $R$ is a refinement obtained from $A$ by a sequential application of $k$ transformations, s.t. there is an $\alpha$-morphism $\varphi: R \to A$, and $R$ preserves the behavioral properties of $A$, especially the presence or absence of deadlocks.

Structural transformations help us to reconsider the notion of a refinement without referring to the formal definition of $\alpha$-morphisms. In addition, within the framework of our approach to the Petri net generation, transformations play a crucial role. A reference model (see fig. 1) is an abstract EN-system, and its refinement is a result of applying transformations.

We next briefly consider the key aspects of refinement transformations, described in [17]. A transformation is a tuple $\rho = (L, R, c_L, c_R)$, where:

1) $L$ is the left part – a subnet to be transformed.
2) $R$ is the right part – a subnet replacing $L$.
3) $c_L$ – constraints imposed on $L$.
4) $c_R$ – constraints imposed on $R$.  

Fig. 4. Refinement based on transformations and $\alpha$-morphisms
Constraints $c_1$ and $c_2$ are structural and marking restrictions. They are responsible for corresponding $\alpha$-morphisms.

The application of a transformation $\rho$ to an EN-system $N$ involves (1) finding a match for $L$ in $N$ according to $c_1$, i.e., subnet $N(X_L)$ with $X_L \subseteq P \cup T$ and (2) replacing $N(X_L)$ with $R$ according to $c_2$. The result of applying $\rho$ to $N$ is denoted by $\rho(N, X_L)$ and the specification of an affected subnet is not important.

The set of four refinement transformations $RT = \{ \rho_1, \rho_2, \rho_3, \rho_4 \}$ is described in Table 1, where we provide their constraints as well. Intuitively, $\rho_1$ adds concurrency, $\rho_2$ and $\rho_4$ introduce and extend choices, while $\rho_3$ adds a new transition into an initial model.

### Table 1. Refinement transformations

| Transformation | Constraints |
|---------------|-------------|
| $\rho_1$: Place duplication | $\{1, p_1 \neq p_2, p_2 \neq p_3, p_3 \neq p_4\}$ if $p \in m_0$. |
| $\rho_2$: Transition duplication | $\{1, \tau_1 = \tau_2, \tau_2 = \tau_3\}$ |
| $\rho_3$: Transition introduction | $\{1, \tau_1 = \tau_2, \tau_2 = \tau_3, \tau_3 = \tau_4\}$ |
| $\rho_4$: Place split | $\{1, \tau_1 = \tau_2, \tau_2 = \tau_3, \tau_3 = \tau_4\}$ |

Then we can define a refinement as an EN-system that is obtained by applying a sequence $\pi \in RT^*$ of refinement transformations to another EN-system, as formally given below.

**Definition 3 (Refinement):** Let $N_i = (P_i, T_i, F_i, m_0^i)$ be an EN-system with $i = 1, 2$, $N_i$ is a refinement of $N_2$ if there is a sequence of transformations $(\rho_1, \rho_2, ..., \rho_k) \in RT^*$, s.t. $N_2 \xrightarrow{\rho_1} N_2' \xrightarrow{\rho_2} ... \xrightarrow{\rho_k} N_1$.

Let us consider the example of applying transformation $\rho_3$ to place $h$ in the EN-system shown in fig. 2. According to Table 1, there are no specific restrictions imposed on a place in the left part of $\rho_3$. Then, we can replace place $h$ with a subnet, corresponding to the right part of $\rho_3$, as shown in fig. 5. Since $\rho_3$ is not marked, added places are also not marked.

As proven in [17], refinement transformations do not introduce new deadlocks, unless they are already present in an initial EN-system, i.e., the deadlocks in a transformed EN-system are the inverse images of the deadlocks, present in an initial EN-system (under the corresponding $\alpha$-morphism).

Thus, the following proposition holds.

**Proposition 1:** Let $N_i = (P_i, T_i, F_i, m_0^i)$ be an EN-system with $i = 1, 2$ s.t. $N_i$ is a refinement of $N_2$. If $N_2$ is deadlock-free, then $N_i$ is deadlock-free as well.

Now we can proceed to the design and implementation of algorithms, which use the set of refinement transformations to generate EN-systems.

### 5. Generation Algorithms

In this section, we discuss two algorithms that support the automated generation of EN-systems, using the structural transformations, described in Table 1, according to Definition 3. The first algorithm corresponds to the generation of an EN-system by applying a fixed sequence of the refinement transformations to an initial model. The second algorithm corresponds to the randomized EN-system generation, parameterized with the probability of applying each transformation.

#### 5.1 Fixed Generation of an EN-system

Algorithm 1 corresponds to a direct implementation of Definition 3. There is a fixed finite sequence, $\pi = (\rho_1, \rho_2, ..., \rho_n) \in RT^*$, of refinement transformation to be applied to an EN-system $N = (P, T, F, m_0)$.

**Algorithm 1: Fixed generation**

**Input:** EN-system $N = (P, T, F, m_0)$

**Transformations $RT = \{\rho_1, \rho_2, ..., \rho_n\} \in RT^*$

**Output:** EN-system $R = (P', T', F', m_0')$ - a refinement of $N$

```
R ← N
i ← 1
foreach $\rho_i \in \pi$ do
    if $\exists X_{L_i} \in P' \cup T'$ and $\rho_i$ is applicable to subnet
    R ← $\rho_i(R, X_{L_i})$
end
i ← i + 1
```

If a current transformation $\rho_i$ can be applied to some subnet generated by $X_{L_i}' \in P' \cup T'$, then we replace $R$ with a result of applying $\rho_i$ to $R$. If a current transformation $\rho_i$ cannot be applied to different subnets, the choice is made non-deterministically (it may be determined by the specific
implementation of Algorithm 1). Otherwise, if a current transformation \( \rho_i \) cannot be applied to a subnet in \( N \), we skip it and pass on to the next transformation in a sequence \( \pi \).

The correctness of the fixed generation algorithms follows from (a) the finiteness of \( \pi \) (the algorithm always terminates) and (b) Proposition 1, i.e., an obtained refinement \( R \) preserves the deadlock-freeness of \( N \).

5.2 Randomized Generation of an EN-system

Within the randomized generation algorithm, presented in this paragraph (see Algorithm 2), a sequence of refinement transformations is not known in advance, as opposed to the fixed generation. A specific sequence of refinement transformations is constructed with respect to the parameters defined by a user.

Algorithm 2: Randomized generation

Input: EN-system \( N = (P, T, m_0) \)

Transformations \( RT = \{ \rho_1, \rho_2, \rho_3, \rho_4 \} \)

Probabilities prob; \( RT \rightarrow [0, 1] \)

\( \forall \rho \in RT: \sum \text{freq}(\rho) = 1 \)

Maximum number of nodes maxNode

Maximum number of steps maxSteps

Output: EN-system \( R = (P', T', m'_0) \) — a refinement of \( N \)

\( R \leftarrow N \)

\( \text{totalSteps} \leftarrow 0 \)

while \( |P' \cup T'| < \text{maxSize} \) OR \( \text{totalSteps} \leq \text{maxSteps} \) do

\( \text{AT} \leftarrow \text{findApplicable}(R, RT) \)

\( \text{sumProb} \leftarrow \sum_{\rho \in \text{AT}} \text{prob}(\rho) \)

\( \text{for each } \rho \in \text{AT} \) do

\( \text{prob}(\rho) = \frac{\text{prob}(\rho)}{\text{sumProb}} \)

\( \text{order AT in the descending order of prob;} \)

\( r \leftarrow \text{RANDOMNUMBER}(0, 1) \)

\( \text{cumulProb} \leftarrow 0 \)

\( i \leftarrow 1 \)

while \( \text{cumulProb} < r \) do

\( \text{cumulProb} \leftarrow \text{cumulProb} + \text{prob}(\rho_i) \)

\( i \leftarrow i + 1 \)

end

\( R \leftarrow R \circ (\rho_i, X'_i) \)

\( \text{totalSteps} \leftarrow \text{totalSteps} + 1 \)

end

The randomized generation parameters include:

1) The maximum size of a refinement — the number of places and transitions;
2) The maximum number of steps — the number of applied transformations;
3) The probability of choosing a specific refinement transformation — the value in the interval \([0, 1]\).

Probabilities are set for the four refinement transformations, s.t. the sum of all four probabilities is equal to 1. Below we describe how the specific refinement transformation is chosen at each step of Algorithm 2.

Firstly, we find a set of refinement transformations \( AT \) that can be applied to a given EN-system (function findApplicable \((R, RT)\)) , according to constraints given in Table 1. Secondly, we normalize the probabilities of the applicable transformations in \( AT \) and obtain the values of prob’ function. Then, by generating a random number \( r \), we choose the specific refinement transformation \( \rho_i \). Intuitively, we divide the interval \([0, 1]\) into sub-intervals, according to the normalized probabilities of applicable refinement transformations, and check where the value of \( r \) is. We assume to use the uniform distribution for the random number generation.

The correctness of Algorithm 2 follows from the fact that (a) the total number of steps (the actual length of an applied transformation sequence) is bounded by the maximum size of a refinement and by the maximum number of steps that can be done; and from (b) Proposition 1, i.e., a constructed refinement preserves the deadlock-freeness of \( N \).

5.3 Example: the Fixed Refinement of an EN-System with a Deadlock

Here we consider an example of applying the fixed generation algorithm to the EN-system that has a deadlock (see fig. 6, where \( N \) has the deadlock \( \{p_2\} \) reachable from its initial marking \( \{p_1, p_3\} \)).

Let \( \pi = (\rho_4, \rho_3, \rho_1, \rho_2, \rho_1, \rho_2) \in RT^* \) be a sequence of refinement transformation to be applied to \( N \).

A possible result of applying \( \pi \) to \( N \) is provided in fig. 7, where we show transformations affecting disjoint subnets as a single step. It can be seen that none of the \( \rho_2 \)-elements in \( \pi \) have been applied to \( N \), since there are no places with two or more input transitions. That is why they have been skipped in this example.

What is more important is that the reachable deadlock \( \{p_2\} \) have not been lost in the transformed EN-system. The inverse image of \( \{p_2\} \) (under the corresponding \( \alpha \)-morphism, refer to fig. 4) in the transformed EN-system is also the reachable deadlock \( \{s_2\} \), as formally proven in [17]. New deadlocks have not been introduced into the transformed EN-system.
5.4 Example: a Step in the Randomized Refinement of a Deadlock-Free EN-System

In this paragraph, we consider a step of Algorithm 2 in more detail. Given the EN-system \( N \) shown in fig. 8 and the following probabilities: \( \text{prob}(\rho_1) = 0.15 \), \( \text{prob}(\rho_2) = 0.10 \), \( \text{prob}(\rho_3) = 0.05 \), \( \text{prob}(\rho_4) = 0.7 \), we will show how the choice of a refinement transformation is performed. We start with finding the applicable transformations. Here we have that only \( \rho_1 \), \( \rho_2 \), and \( \rho_3 \) can be applied to the EN-system from fig. 8. Their normalized probabilities are: \( \text{prob}(\rho_1) = 0.50 \), \( \text{prob}(\rho_3) = 0.33 \), and \( \text{prob}(\rho_3) = 0.17 \).

Then we generate a random number \( r \). Let \( r = 0.73 \). We check where the value of \( r \) is in the interval \([0, 1]\) concerning the cumulative normalized probabilities (see fig. 9).

The value of \( r \) is in the interval \([0.5, 0.83]\), corresponding to the refinement transformation \( \rho_2 \). Thus, we apply this transformation to the EN-system from fig. 8, and a possible result is shown in fig. 10, if we choose transition \( t_2 \) to be transformed.

Then, according to Algorithm 2, we continue choosing refinement transformations, according to their probabilities, until we reach either the limit of the size or the limit of the total number of applied transformations.

6. Software Implementation and Evaluation

In this section, we describe details concerning the implementation of the two generation algorithms discussed in the previous section. We have evaluated the randomized generation algorithm using Petri net models for interaction patterns described in [24] as reference models. They provide a highly abstract view of typical asynchronous agent interactions, whereas a refinement of an interface pattern can be seen as the model of a specific system implementing this pattern.

6.1 Carassius Petri Net Editor

The Carassius software tool has been presented in [3]. It supports various modeling notations, including (communicating) finite state machines and Petri nets. The Carassius allows one to simulate Petri nets according to the transition firing rule, import and export files in different formats, visualize process behavior. Apart from that, the Carassius has a modular architecture, and it can be easily extended with new features. For example, in [25], the authors have described an extension to the Carassius that supports the simulation of Petri nets with two special types of arcs: reset and inhibitor. The main window of the editor is shown in fig. 11.

We have introduced the following features into the Carassius Petri net editor:

1. the internal storage of refinement transformations;
2. the choice and application of a single transformation to a given EN-system;
3. the generation of an EN-system by applying a fixed transformation sequence (Algorithm 1);
4. the generation of an EN-system by applying a randomly constructed transformation sequence (Algorithm 2).

The implementation of the generation algorithms has also been enriched with the possibility to «roll back» following a transformation sequence to check intermediate results.

The parameters necessary for the fixed and randomized generation are configured in the top panel. A fixed transformation sequence (Algorithm 1 is constructed using a drop-down menu, where one may choose a transformation and assign the corresponding number of occurrences to it (see fig. 12). The configuration of probabilities and other parameters of Algorithm 2 is shown in fig. 13.

As described in the following paragraph, we have considered the application of Algorithm 2 to construct refinements of so-called interface patterns.

6.2 Evaluation: Randomized Refinement of Interaction Patterns

Modeling complex information systems is a rather difficult task due to the coordination of several interacting components. Service interaction patterns, introduced in the Business Process...
Management (BPM) community [26], provide generic solutions for designing composite systems with several interacting entities. The patterns give a highly abstract view of component interactions. The identification of the typical interface patterns and their modeling using Petri nets have been considered in [24], where the seven asynchronous interaction patterns have been discussed. The models of these patterns are shown in fig. 14. For instance, IP-4 describes the simple message exchange, when the first component sends a message to the second one, and the latter sends back an acknowledgment.

We have used these interaction patterns to evaluate the randomized generation algorithm. Given an interface pattern, we apply Algorithm 2 and obtain a possible refinement of this pattern. A refinement of an interface pattern inherits its structural and behavioral properties. Intuitively, such a refinement represents a possible system model implementing an interaction pattern.

The results of applying the randomized refinement to the interaction patterns with different parameters are provided in Table 2, where we show the number of places and transitions in the reference model and the obtained refinements. Correspondingly, we have considered five different cases:

1. the randomized refinement with equal probabilities for each transformation ($\rho_1 = 0.25$);
2. the four cases when the probability of one transformation (0.67) outweighs the equal probabilities of the other three transformations (0.11).

As it can be seen from Table 2, the number of places and transitions in the constructed refinements is consistent with transformation application probabilities. Within all transformations being equally probable, we do not observe notable differences in the number of places and transitions in the obtained refinements. However, when the place (transition) duplication has the highest probability, we have that the number of places (transitions) significantly outweighs the number of transitions (places) in the refinement. The predominance of the transition introduction ($\rho_3$) and place split ($\rho_4$) also does not lead to substantial differences in the number of places and transitions. The application of $\rho_3$ requires places with two more input transitions, which may not be present in the reference model.

In addition, fig. 15 provides a possible result of applying ten refinement transformations to the interface pattern IP-1, where the transformations have equal probabilities.

6. Conclusions

In this paper, we have presented an approach to the generation of Petri nets using structural property-preserving transformations. We have considered the generation of elementary net systems, which form the basic class of Petri nets. Elementary net systems reflect the control-flow of a process, while data and time aspects are ignored. Given a reference model, we apply a sequence of refinement transformations to obtain a Petri net with similar structural and behavioral properties valid for the reference Petri net. Refinement transformations extend a reference model by adding new places and transitions, i.e., make the structure of a reference model more sophisticated. The proposed approach
can be applied for a complex evaluation of algorithms operating with Petri nets requiring the preparation of model sets containing Petri nets with the specific structural and behavioral properties. The correctness of applying these transformations is based on two observations. Firstly, the transformations induce morphisms between reference and transformed Petri nets. Secondly, the transformations do not introduce new deadlocks, unless they are already present in reference models. We have designed two algorithms supporting the automated generation of Petri nets with the help of structural property-preserving transformations. The fixed generation corresponds to the direct application of a fixed sequence of refinement transformations. Within the randomized generation, a user chooses the maximum size of a target model and sets the probability of applying each transformation. We have conducted a series of experiments to evaluate the developed algorithms using Petri net models of service interaction patterns. The experimental results confirm the consistency of the randomized generation algorithm, according to changes in the number of places and transitions with respect to probability values. These algorithms have also been implemented in the existing Carassius Petri net editor. The main limitation to the proposed approach, based on transformations, is that it is impossible to generate a cyclic Petri net from a reference model without cycles. In the future, we plan to relax these constraints and to extend the collection of property-preserving transformations correspondingly. In this light, we also plan to develop a «designer» of structural Petri net transformations that will allow us to construct new transformations. Another direction for the future research is the development of property-preserving transformations for different extensions of Petri nets, including, e.g., colored Petri nets, where tokens can carry data, or timed Petri nets, where transitions are assigned firing time intervals. Note that certain extensions of Petri nets can also be «unfolded» to elementary net systems.

References

[1] W. Reisig. Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies. Springer, 2013, 257 p.
[2] J. Desel. Basic linear algebraic techniques for place/transition nets. Lecture Notes in Computer Science, vol. 1491. Springer, 1998, pp. 257-308.
[3] N. Nikitina, A. Mitsyuk. Carassius: A Simple Process Model Editor. Trudy ISP RAN/Proc. ISP RAS, vol. 27, issue 3, 2015, pp.219-236. DOI: 10.15514/ISPRAS-2015-27(3)-15.
[4] A. Burattin. Multiperspective process randomization with online and offline simulations. In Proc. of the BPM Demo Track 2016, CEUR Workshop Proceedings, vol. 1789, 2016, pp. 1-6.
[5] A. Kalenkova, W. van der Aalst, I. Lomazova, and V. Rubin. Process mining using BPMN: relating event logs and process models. Software & Systems Modeling, vol. 16, 2017, pp.1019-1048.
[6] Z. Yan, E. Dijkman, and P. Grefen. Generating process model collections. Software & System Modeling, vol. 16, 2017, pp. 979-995.
[7] T. Jouck and B. Depaire. PTrainLogGenerator: A generator for artificial event data. In Proc. of the BPM Demo Track 2016, CEUR Workshop Proceedings, vol. 1789, 2016, pp. 23-27.
[8] K. van Hee and Z. Liu. Generating benchmarks by random stepwise refinement of petri nets. In Proc. of the Workshops of the 31st International Conference on Application and Theory of Petri Nets and Other Models of Concurrency (PETRI NETS 2010) and of the 10th International Conference on Application of Concurrency to System Design (ACSD 2010), CEUR Workshop Proceedings, vol. 827, 2010, pp. 403-417.
[9] G. Berthelot. Checking properties of nets using transformations. Theoretical Computer Science, vol. 222, 1999, pp. 19-40.
[10] T. Murata. Petri nets: Properties, analysis and applications. Proceedings of the IEEE, vol. 77, no. 4, 1989, pp. 541-580.
[11] T. Murata and I. Suzuki. A method for stepwise refinement and abstraction of Petri nets. Journal of Computer and System Sciences, vol. 27, issue 1, 1983, pp. 51-76.
[12] G. Winskel. Petri nets, algebras, morphisms, and compositionality. Information and Computation, vol. 72, no. 3, 1987, pp. 197-238.
[13] G. Winskel and M. Nielsen. Petri nets and bisimulations. Theoretical Computer Science, vol. 153, pp. 211-244, 1996.
[14] J. Desel and A. Merceron. Vicinity respecting homomorphisms for abstracting system requirements. Lecture Notes in Computer, vol. 6550, 2010, pp. 1-20.
[15] L. Bernardinello and F. De Cindio. A survey of basic net models and modular net classes. Lecture Notes in Computer Science, vol. 669, 1992, pp. 304-351.
[16] L. Bernardinello, E. Mangioni, and L. Pomello. Local state refinement and composition of elementary net systems: An approach based on morphisms, Lecture Notes in Computer Science, vol. 8100, 2013, pp. 48-70.
[17] L. Bernardinello, I. Lomazova, R. Nesterov, and L. Pomello. Property preserving transformations of elementary net systems based on morphisms. In Proc. of the International Workshop on Petri Nets and Software Engineering, CEUR Workshop Proceedings, vol. 2651, 2020, pp. 49-67.
[18] N. Dingle, W. Knottenbelt, and T. Suto. PIPE: A tool for the performance evaluation of generalised stochastic petri nets. ACM SIGMETRICS Performance Evaluation Review, vol. 36, no. 4, 2009, pp. 34-39.
[19] Platform Independent Petri Net Editor. Available at https://github.com/sarahatattersall/PIPE, accessed: 2021-03-20.
[20] PNEEditor (Petri Net Editor). Available at https://github.com/matmas/pneditor, accessed: 2021-03-20.
[21] T. Freytag and M. Sänger. WoPeD - An Educational Tool for Workflow Nets. in Proc. of the BPM Demo Track 2016, CEUR Workshop Proceedings, vol. 1789, CEUR-WS.org, 2014, pp. 31-36.
[22] WoPDE (Workflow Petri Net Designer). Available at https://woped.dibwkarlsruhe.de/, accessed: 2021-03-20.
[23] WOLFGANG - Petri Net Editor. Available at https://github.com/ig-unifreiburg/WOLFGANG, accessed: 2021-03-20.
[24] R. Nesterov and I. lomazova. Asynchronous interaction patterns for mining multi-agent system models. Lecture Notes in Computer Science, vol. 3649, 2005, pp. 302-318.

Информация об авторах / Information about authors

Роман Александрович НЕСТЕРОВ – аспирант факультета компьютерных наук НИУ ВШЭ и департамента информатики, системной инженерии и коммуникаций Миланского университета Бикокка (UNIMIB), младший научный сотрудник лаборатории ПОИС НИУ ВШЭ. Область его научных интересов составляет теория параллелизма, сети Петри, формальные методы для моделирования многолабиринтных информационных систем.

Roman NESTEROV is a postgraduate student at the Faculty of computer science, HSE University and at the Department of Informatics, System Engineering and Communications, University of Milano-Bicocca (UNIMIB), a junior research fellow at the Laboratory of Process-Aware Information Systems (PAIS Lab), HSE University. His research interests are the theory of concurrency, Petri nets and formal methods for modeling multi-agent information systems.

Семен Юрьевич САВЕЛЬЕВ – студент четвертого курса бакалавриата факультета компьютерных наук НИУ ВШЭ, стажер-исследователь лаборатории ПОИС НИУ ВШЭ. Область научных интересов составляют: объектно-ориентированное программирование, архитектура информационных систем, моделирование и анализ поведения процессов с помощью сетей Петри и других формализмов.

Semyon SAVELEYEV is a fourth-year undergraduate student at the Faculty of computer science, HSE University, a research assistant at the PAIS Lab, HSE University. His research interests include object-oriented programming, architecture of information systems, modeling and analyzing process behavior using Petri nets and other formalisms.