Generation of Petri Nets Using Structural Property-Preserving Transformations

Roman Nesterov*†, Semyon Savelyev*
*HSE University, 20 Myasnitskaya Ulitsa, 101000 Moscow, Russia
†Univeristà degli Studi di Milano-Bicocca, 1 Piazza dell'Ateneo Nuovo, 20126 Milan, Italy
E-mail: rnesterov@hse.ru, syusavelev@edu.hse.ru

***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.**

***Index Terms*—Petri nets, morphisms, property-preserving transformations, generation of models**

I. 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.

Fig. 1. Step-wise generation of a Petri net

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:

1) the algorithms for the fixed and randomized generation of Petri nets from a given reference model;
2) 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 III, we define a class of Petri nets considered in our paper. Section IV describes the mathematical framework behind a collection of structural transformations that are used to refine Petri nets. The algorithms for the fixed and randomized Petri net generation are presented in Section V. In Section VI, we describe a software implementation as well as evaluation of these algorithms, and Section VII concludes the paper.

II. Related Work

Process Log Generator PLG2 [4] is a well-known software used for the random generation of process models. It supports different notations, including Petri nets and Business Process Model and Notation (BPMN). As shown in [5], the specific classes of BPMN models correspond to Petri nets and vice versa. PLG2 generates process models based on randomly

generated context-free grammars and parameters such as the maximum model size, the frequencies of standard behavioral patterns, and others. Compared to our approach, PLG2 offers only the fully randomized model generation and guarantees the behavioral correctness of constructed models. However, within our approach, a reference model may have, for instance, deadlocks, which will be preserved in its refinement.

The generation of BPMN process models has also been considered in [6]. The authors of this approach allow specifying the parameters such as the size of models, the frequencies of behavioral patterns, the types of activities. Similar to our approach, they have also used a collection of initial BPMN models to generate a set of synthetic models.

PTandLogGenerator [7] is another tool supporting the randomized generation of process models. It produces so-called process trees, which specify relations among process activities, for example, sequential, alternative, or concurrent. Process trees can be converted to Petri nets. The prime objective of PTandLogGenerator and the previously mentioned PLG2 is to simulate the behavior of randomly generated process models.

An approach to the generation of *benchmarks*, using random step-wise Petri net refinements, has been presented in [8]. Within this approach, the authors have also defined a set of refinement transformations similar to those used in our study. Based on the proposed transformations, different Petri net classes have been identified and studied. It has been shown what transformations can be used to generate all Petri nets representing a given class.

Structural *transformations* of Petri nets have been first studied in the works [9], [10], [11], describing simple yet powerful reduction and extension transformations, s.t. liveness, boundedness, home states, and other behavioral properties are preserved.

Morphisms on Petri nets provide a formal and natural framework to express structural property-preserving relations between Petri nets [12], [13], [14]. Using morphisms, one can consider more sophisticated problems of property preservation, including, for instance, bisimulations between Petri nets, as discussed in [13]. For *elementary net systems* [15] – a fundamental class of Petri nets also considered in our paper – α*-morphisms* have been introduced in [16]. They help to formalize structural relations between abstract models and their *refinements*. Concerning our approach to the Petri net generation, a reference Petri net represents an abstract model. In addition, α-morphisms preserve the behavioral properties (reachable markings) as well as reflect them under the specific local requirements.

Since the direct application of α-morphisms is rather difficult for the sophisticated constraints to be checked, a collection of *local* transformations proposed in [17] can be used to define α-morphisms systematically in a step-by-step way. These transformations are used in our study to generate Petri nets, which preserve the properties of an initial reference model. Correspondingly, the mathematical framework behind transformations, which provide the property preservation, is based on α-morphisms.

The existing open-source Petri net editors, among the others, include *Platform Independent Petri Net Editor* [18], [19], *PNEditor* [20], *WoPeD* [21], [22], *Wolfgang* [23], *Carassius* [3]. They allow modeling, simulating and analyzing the behavior of Petri nets. The problem of the model generation has not been considered within these editors. In our study, we will extend the functionality of the *Carassius* Petri net editor to provide the generation of Petri nets with the desired structural and behavioral properties.

III. 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 changed by transition *firings*. Below we provide the formal definitions based on [15] concerning the structural and behavioral aspects of EN-systems.

Let S be a set. The set of all finite non-empty sequences over S is denoted by S^+, and $S^* = S^+ \cup \{\varepsilon\}$, where ε 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) ${}^\bullet x = \{y \in X \mid (y, x) \in F\}$ is the *preset* of x.
2) $x^\bullet = \{y \in X | (x, y) \in F\}$ is the *postset* of x.
3) ${}^\bullet x^\bullet = {}^\bullet x \cup x^\bullet$ is the *neighborhood* of $x \in X$

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

In our work, we consider nets without self-loops, i.e., $\forall x \in P \cup T\colon {}^\bullet x \cap x^\bullet = \varnothing$ and isolated transitions, i.e., $\forall t \in T\colon |{}^\bullet t| \geq 1$ and $|t^\bullet| \geq 1$.

The $\bullet$-notation can also be extended to subsets of nodes. Let $N = (P, T, F)$ be a net, and $Y \subseteq P \cup T$. Then ${}^\bullet Y = \bigcup_{y \in Y} {}^\bullet y$, $Y^\bullet = \bigcup_{y \in Y} y^\bullet$ and ${}^\bullet Y^\bullet = {}^\bullet Y \cup Y^\bullet$. $N(Y)$ denotes the subnet of N *generated* by Y, i.e., $N(Y) = (P \cap Y, T \cap Y, F \cap (Y \times Y))$.

A *marking* (state) m in a net $N = (P, T, F)$ 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 $N = (P, T, F)$ has a *contact* if $\exists t \in T\colon {}^\bullet m$ and $m \cap t^\bullet \neq \varnothing$.

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 EN-system is defined by the *firing rule*. A marking m in a net $N = (P, T, F)$ *enables* transition $t \in T$, denoted $m[t\rangle$, iff ${}^\bullet t \subseteq m$ and $m \cap t^\bullet = \varnothing$. Enabled transitions may *fire*. Firing t at m evolves N to a new marking $m' = (m \setminus {}^\bullet t) \cup t^\bullet$, denoted $m[t\rangle m'$.

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

A marking m in $N = (P, T, F, m_0)$ is *reachable* if $\exists w \in FS(N)\colon m_0[w\rangle m$. The set of all markings reachable from m will be denoted $[m\rangle$.

A reachable marking $m \in [m_0\rangle$ in $N = (P, T, F, m_0)$ is a *deadlock* iff it does not enable any transitions. An EN-system is deadlock-free iff there are no reachable deadlocks.

A *state machine* is a connected net $N = (P, T, F)$, where $\forall t \in T\colon |{}^\bullet t| = |t^\bullet| = 1$. A subnet of an EN-system $N = (P, T, F, m_0)$ generated by $Y \subseteq P$ and ${}^\bullet Y^\bullet$, i.e., $N(Y \cup {}^\bullet Y^\bullet)$ is a *sequential component* of N if it it is a state machine and has a single token in the initial marking. N is covered by sequential components if every place belongs to at least one sequential component. In this case, N is state machine decomposable (SMD). Reachable markings in SMD-EN systems are free from contacts.

State machine decomposability is a basic feature bridging the structural and behavioral properties of EN-systems [15]. The example shown in Fig. 2 provides an SMD-EN system with three sequential 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_7, shown in Fig. 2, belongs to two of three sequential components: A and B.

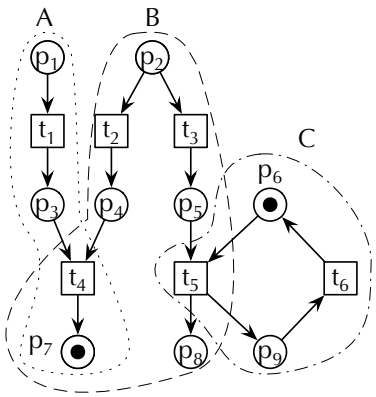


Fig. 2. SMD-EN system with three sequential components

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

IV. 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 α-morphisms formalizing relations between abstract and refined EN-systems [16]. Secondly, we describe a set of local EN-system transformations that induce corresponding α-morphisms and define them in a step-wise manner [17].

A. Morphisms

A class of α-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 α-moprhisms.

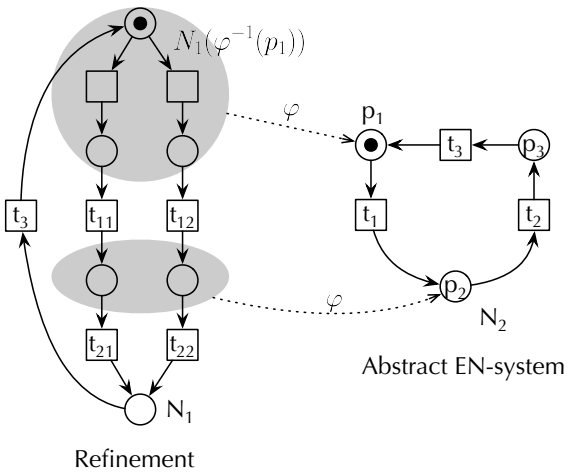


Fig. 3. The α-moprhism $\varphi\colon N_1 \to N_2$

An α-morphism $\varphi\colon N_1 \to N_2$ is a total surjective map from the set of nodes of a refined EN-system N_1 on the set of nodes of an abstract EN-system N_2. Places in an abstract EN-system can be refined with *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_1 in N_2 is split into two transitions, t_{11} and t_{12}, in N_1, as shown in Fig. 3.

An α-morphism $\varphi\colon N_1 \to N_2$ 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_{11} in N_1 shown in Fig. 3 is transition t_1 in N_1, the image of ${}^\bullet t_{11}{}^\bullet$ 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 $N_1(\varphi^{-1}(p_1))$ 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 α-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 α-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.

B. 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 α-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 α-morphism from a result EN-system towards an initial EN-system. Figure 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 α-morphism $\varphi\colon R \to A$, and R preserves the behavioral properties of A, especially the presence or absence of deadlocks.

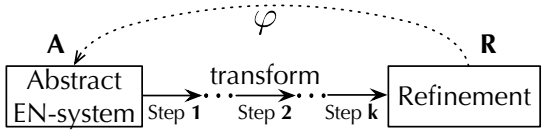


Fig. 4. Refinement based on transformations and α-morphisms

Structural transformations help us to reconsider the notion of a refinement without referring to the formal definition of α-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.

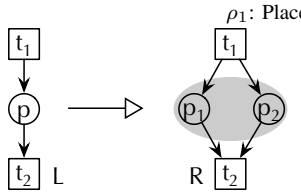
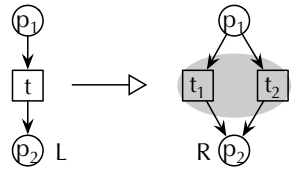
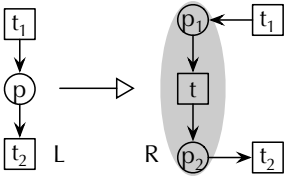
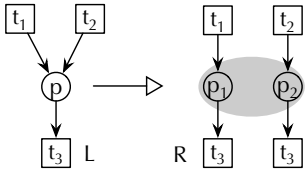
Constraints c_L and c_R are structural and marking restrictions. They are responsible for corresponding α-morphisms.

The application of a transformation ρ to an EN-system N involves (1) finding a match for L in N according to c_L, i.e., subnet $N(X_L)$ with $X_L \subseteq P \cup T$ and (2) replacing $N(X_L)$ with R according to c_R. The result of applying ρ to N is denoted by $\rho(N, X_L)$. We write $N \xrightarrow{\rho} N'$ if $N' = \rho(N, X_L)$ 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 I, where we provide their constraints as well. Intuitively, ρ_1 adds concurrency, ρ_2 and ρ_4 introduce and extend choices, while ρ_3 adds a new transition into an initial model. 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 $N_i = (P_1, T_i, F_i, m_0^i)$ be an EN-system with $i = 1, 2$. N_1 is a refinement of N_2 iff there is a sequence of transformations $\langle \rho_1, \rho_2, ..., \rho_k \rangle \in RT^*$, s.t. $N_2 \xrightarrow{\rho_1} N_2' \xrightarrow{\rho_2} ... \xrightarrow{\rho_k} N_1$.

TABLE I
REFINEMENT TRANSFORMATIONS

Transformation	Constraints
ρ_1: Place duplication 	**1**. ${}^\bullet p_1 = {}^\bullet p = {}^\bullet p_2$; **2**. $p_1{}^\bullet = p^\bullet = p_2{}^\bullet$; **3**. ($p_1 \in m_0'$ and $p_2 \in m_0'$) iff $p \in m_0$.
ρ_2: Transition duplication 	**1**. ${}^\bullet t_1 = {}^\bullet t = {}^\bullet t_2$; **2**. $t_1{}^\bullet = t^\bullet = t_2{}^\bullet$.
ρ_3: Transition introduction 	**1**. ${}^\bullet t = \{p_1\}$, $t^\bullet = \{p_2\}$; **2**. $p_1{}^\bullet = {}^\bullet p_2 = \{t\}$; **3**. ${}^\bullet p_1 = {}^\bullet p$, $p_2{}^\bullet = p^\bullet$; **4**. $p_1 \in m_0' \Leftrightarrow p \in m_0$;
ρ_4: Place split 	**1**. ${}^\bullet p_1 \subset {}^\bullet p$, ${}^\bullet p_2 \subset p$; **2**. ${}^\bullet p_1 \cup {}^\bullet p_2 = {}^\bullet p$; **3**. ${}^\bullet p_1 \cap {}^\bullet p_2 = \varnothing$; **3**. $p_1{}^\bullet$, $p_2{}^\bullet$ are two complete copies of $p^\bullet$; **4**. ${}^\bullet(p_i{}^\bullet) \setminus \{p_i\} = {}^\bullet(p^\bullet)$; **5**. if $p \in m_0$, then $p_1 \in m_0'$ iff $p \in m_0$;

Let us consider the example of applying transformation ρ_3 to place p_9 in the EN-system shown in Fig. 2. According to Table I, there are no specific restrictions imposed on a place in the left part of ρ_3. Then, we can replace place p_9 with a subnet, corresponding to the right part of ρ_3, as shown in Fig. 5. Since p_9 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 α-morphism). Thus, the following proposition holds.

Proposition 1: Let $N_i = (P_1, T_i, F_i, m_0^i)$ be an EN-system with $i = 1, 2$ s.t. N_1 is a refinement of N_2. If N_2 is deadlock-free, then N_1 is deadlock-free as well.

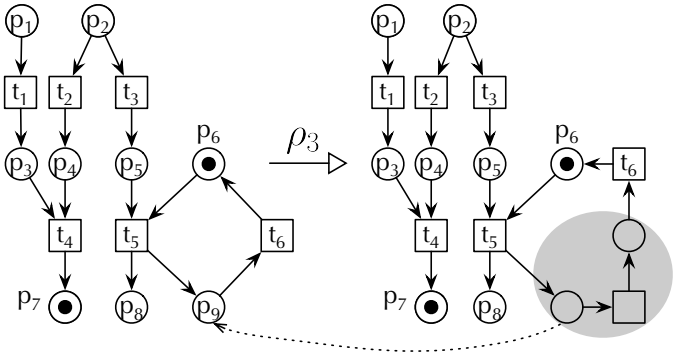


Fig. 5. Application of ρ_3 to the EN-system from Fig. 2

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

V. Generation Algorithms

In this section, we discuss two algorithms that support the automated generation of EN-systems, using the structural transformations, described in Table I, 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.

A. Fixed Generation of an EN-system

Algorithm 1 corresponds to a direct implementation of Definition 3. There is a fixed finite sequence, $\pi = \langle \rho_1, \rho_2, ..., \rho_n \rangle \in RT^*$, 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 $RT = \{\rho_1, \rho_2, \rho_3, \rho_4\}$
Sequence $\pi = \langle \rho_1, \rho_2, ..., \rho_n \rangle \in RT^*$
Output: EN-system $R = (P', T', F', m'_0)$ – a refinement of N

$R \leftarrow N$
$i \leftarrow 1$
foreach $\rho_i \in \pi$ **do**
  if $\exists X'_L \in P' \cup T'$ *and* ρ_i *is applicable to subnet* $R(X'_L)$ *in* R **then**
    $R \leftarrow \rho_i(R, X'_L)$
  end
  $i \leftarrow i + 1$
end

If a current transformation ρ_i can be applied to some subnet generated by $X'_L \in P' \cup T'$, then we replace R with a result of applying ρ to R. If a current transformation ρ_i can 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 ρ_i cannot be applied to a subnet in R, we skip it and pass on to the next transformation in a sequence π.

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

B. 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 $N = (P, T, F, m_0)$
Transformations $RT = \{\rho_1, \rho_2, \rho_3, \rho_4\}$
Probabilities $prob$: $RT \to [0, 1]$, s.t.
$\forall \rho \in RT$: $\sum freq(\rho) = 1$
Maximum number of nodes $maxSize$
Maximum number of steps $maxSteps$
Output: EN-system $R = (P', T', F', m'_0)$ – a refinement of N

$R \leftarrow N$
$totalSteps \leftarrow 0$
while $|P' \cup T'| < maxSize$ *OR* $totalSteps \leq maxSteps$ **do**
  $AT \leftarrow$ FINDAPPLICABLE(R, RT)
  $sumProb \leftarrow \sum_{\forall \rho \in AT} prob(\rho)$
  foreach $\rho \in AT$ **do**
    $prob'(\rho) = \dfrac{prob(\rho)}{sumProb}$
  end
  order AT in the descending order of $prob'$;
  $r \leftarrow$ RANDOMNUMBER(0, 1)
  $cumulProb \leftarrow 0$
  $i \leftarrow 1$
  while $cumulProb < r$ **do**
    $cumulProb \leftarrow cumulProb + prob'(\rho_i)$
    $i \leftarrow i + 1$
  end
  $R \leftarrow \rho_i(R, X'_L)$
  $totalSteps \leftarrow totalSteps + 1$
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 I. 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 ρ_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.

C. 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 $\{p_1, p_2\}$). Let $\pi = \langle \rho_4 \rho_3 \rho_1 \rho_3 \rho_4 \rho_4 \rho_2 \rangle \in RT^*$ be a sequence of refinement transformation to be applied to N.

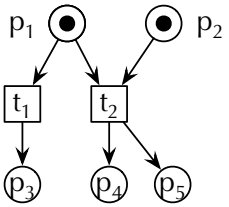


Fig. 6. EN-system with a deadlock

A possible result of applying π 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 ρ_4-elements in π 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 α-morphism, refer to Fig. 4) in the transformed EN-system is also the reachable deadlock $\{s_6\}$, as formally proven in [17]. New deadlocks have not been introduced into the transformed EN-system.

D. 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: $prob(\rho_1) = 0.15$, $prob(\rho_2) = 0.10$, $prob(\rho_3) = 0.05$, $prob(\rho_4) = 0.7$, we will show how the choice of a refinement transformation is performed.

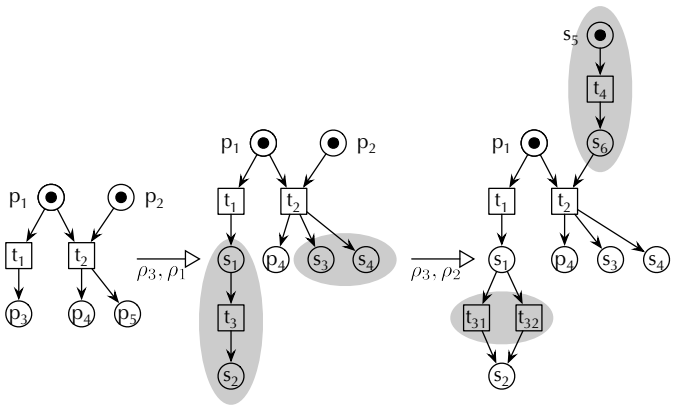


Fig. 7. A result of applying π to the EN-system from Fig. 6

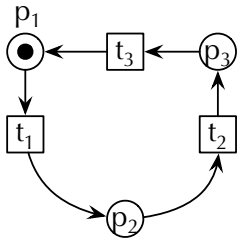


Fig. 8. Deadlock-free EN-system

We start with finding the applicable transformations. Here we have that only ρ_1, ρ_2, and ρ_3 can be applied to the EN-system from Fig. 8. Their normalized probabilities are: $prob'(\rho_1) = 0.50$, $prob'(\rho_2) = 0.33$, and $prob'(\rho_3) = 0.17$.

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).

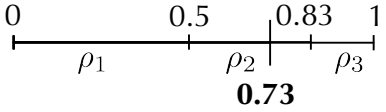


Fig. 9. Checking the placement of the random number r

The value of r is in the interval $[0.5, 0.83]$, corresponding to the refinement transformation ρ_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.

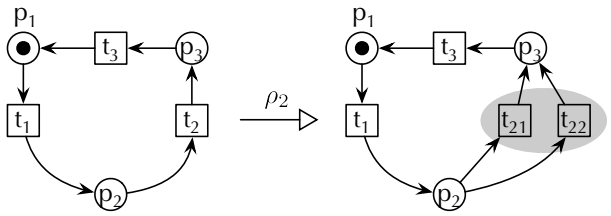


Fig. 10. Applying the chosen transformation to the EN-system from Fig. 8

VI. 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.

A. 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.

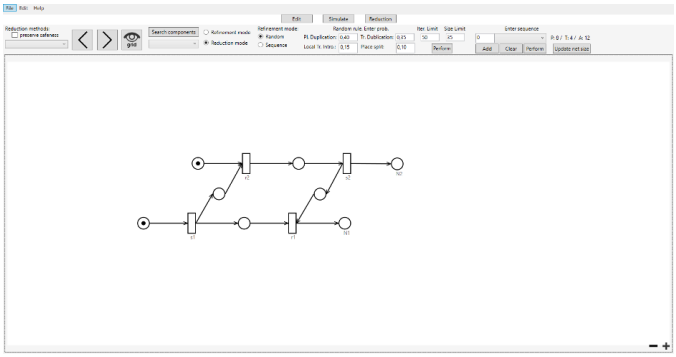


Fig. 11. Carassius process model editor

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.

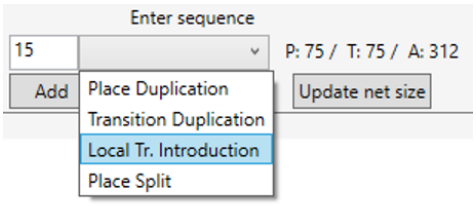


Fig. 12. Constructing a sequence of transformations

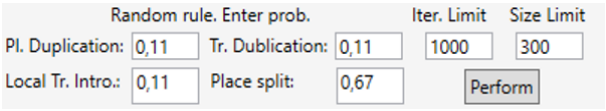


Fig. 13. Parameters of the randomized generation

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

B. 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.

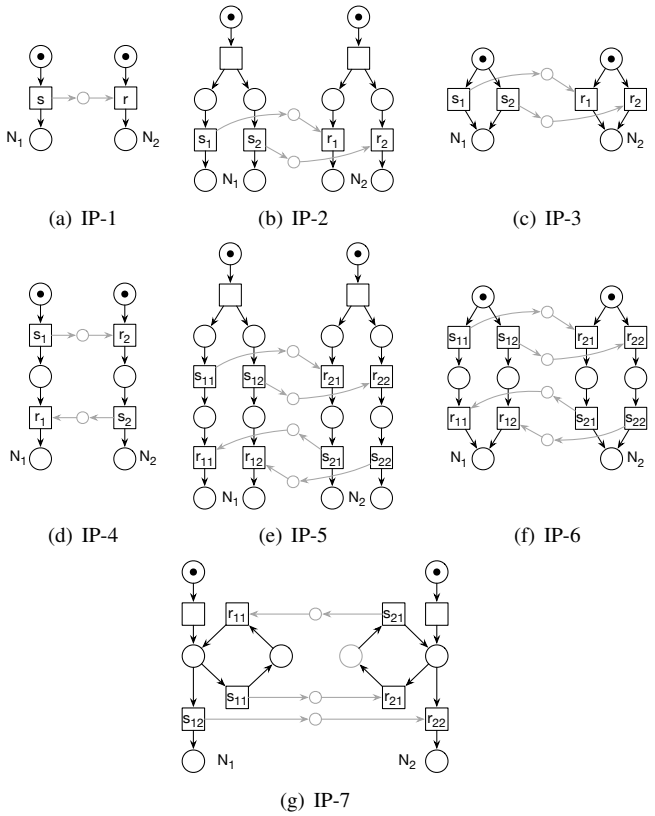
The results of applying the randomized refinement to the interaction patterns with different parameters are provided in Table II, where we show the number of places and transitions

TABLE II
RANDOMIZED REFINEMENT OF INTERACTION PATTERNS

	Reference		**Randomized generation** ($maxSize$=300, $maxSteps = 1000$)									
			$\rho_i = 0,25$		$\rho_1 = 0,67$		$\rho_2 = 0,67$		$\rho_3 = 0,67$		$\rho_4 = 0,67$	
	$\|P\|$	$\|T\|$	$\|P\|$	$\|T\|$	$\|P\|$	$\|T\|$	$\|P\|$	$\|T\|$	$\|P\|$	$\|T\|$	$\|P\|$	$\|T\|$
IP-1	5	2	134	166	234	66	76	224	156	144	141	166
IP-2	12	6	147	153	216	84	66	256	155	145	146	154
IP-3	6	4	154	149	212	88	85	215	154	147	156	144
IP-4	8	4	132	168	217	83	71	229	152	148	144	156
IP-5	18	10	139	163	207	94	78	222	156	145	157	143
IP-6	12	8	107	193	218	83	72	232	158	142	158	143
IP-7	11	8	140	161	190	110	59	256	143	158	85	215

in the reference model and the obtained refinements. Correspondingly, we have considered five different cases:

- the randomized refinement with equal probabilities for each transformation ($\rho_i = 0,25$);
- the four cases when the probability of one transformation $(0,67)$ outweighs the equal probabilities of the other three transformations $(0,11)$.



(a) IP-1 (b) IP-2 (c) IP-3

(d) IP-4 (e) IP-5 (f) IP-6

(g) IP-7

Fig. 14. Interaction patterns: reference models

As it can be seen from Table II, 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 (ρ_3) and place split (ρ_4) also does not lead to substantial differences in the number of places and transitions. The application of ρ_4 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.

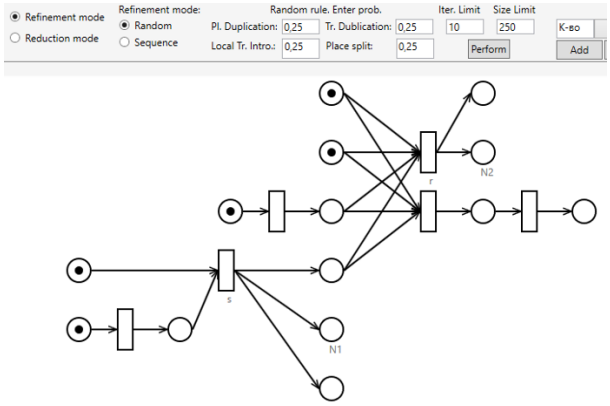


Fig. 15. Refinement of IP-1: 10 steps, equal probabilities

VII. CONCLUSION

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.

VIII. Acknowledgments

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

References

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

[2] J. Desel, "Basic linear algebraic techniques for place/transition nets," in *Lectures on Petri Nets I: Basic Models: Advances in Petri Nets*, ser. Lecture Notes in Computer Science, W. Reisig and G. Rozenberg, Eds., vol. 1491. Springer, Heidelberg, 1998, pp. 257–308.

[3] A. Mitsyuk and N. Nikitina, "Carassius: A simple process model editor," *Proceedings of the Institute for System Programming of the RAS*, vol. 27, no. 3, 2015.

[4] A. Burattin, "Multiperspective process randomization with online and offline simulations," in *BPMD 2016*, ser. CEUR Workshop Proceedings, vol. 1789. CEUR-WS.org, 2016, pp. 1–6.

[5] A. Kalenkova, W. van der Aalst, I. Lomaozva, and V. Rubin, "Process mining using BPMN: relating event logs and process models," *Software & Systems Modeling*, vol. 16, pp. 1019–1048, 2017.

[6] Z. Yan, R. Dijkman, and P. Grefen, "Generating process model collections," *Software & System Modeling*, vol. 16, pp. 979–995, 2017.

[7] T. Jouck and B. Depaire, "PTandLogGenerator: A generator for artificial event data," in *BPMD 2016*, ser. CEUR Workshop Proceedings, vol. 1789. CEUR-WS.org, 2016, pp. 23–27.

[8] K. van Hee and Z. Liu, "Generating benchmarks by random stepwise refinement of petri nets," in *ACSD 2010*, ser. CEUR Workshop Proceedings, vol. 827. CEUR-WS.org, 2010, pp. 403–417.

[9] G. Berthelor, "Checking properties of nets using transfomations," in *Advances in Petri Nets 1985*, ser. Lecture Notes in Computer Science, G. Rozenberg, Ed., vol. 222. Springer, Heidelberg, 1986, pp. 19–40.

[10] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[11] T. Murata and I. Suzuki, "A method for stepwise refinement and abstraction of petri nets," *Journal of Computer and System Sciences*, vol. 27, pp. 51–76, 1983.

[12] G. Winskel, "Petri nets, algebras, morphisms, and compositionality," *Information and Computation*, vol. 72, no. 3, pp. 197–238, 1987.

[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 systemn requirements," in *Transactions on Petri Nets and Other Models of Concurrency IV*, ser. Lecture Notes in Computer Science, K. Jensen, S. Donatelli, and M. Koutny, Eds., vol. 6550. Springer, Heidelberg, 2010, pp. 1–20.

[15] L. Bernardinello and F. De Cindio, "A survey of basic net models and modular net classes," in *Advances in Petri Nets 1992*, ser. Lecture Notes in Computer Science, G. Rozenberg, Ed., vol. 609. Springer, Heidelberg, 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," in *Transactions on Petri Nets and Other Models of Concurrency VIII*, ser. Lecture Notes in Computer Science, M. Koutny, W. van der Aalst, and A. Yakovlev, Eds., vol. 8100. Springer Heidelberg, 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 *Proceedings of PNSE-2020*, ser. CEUR Workshop Proceedings, vol. 2651. CEUR-WS.org, 2020, pp. 49–67.

[18] N. Dingle, W. Knottenbelt, and T. Suto, "PIPE2: A tool for the performance evaluation of generalised stochastic petri nets," *ACM SIGMETRICS Performance Evaluation Review*, vol. 36, no. 4, p. 34–39, 2009.

[19] "Platform Independent Petri Net Editor," https://github.com/sarahtattersall/PIPE, accessed: 2021-03-20.

[20] "PNEditor (Petri Net Editor," https://github.com/matmas/pneditor, accessed: 2021-03-20.

[21] T. Freytag and M. Sänger, "WoPeD - An Educational Tool for Workflow Nets," in *Proceedings of the BPM Demo Sessions*, ser. CEUR Workshop Proceedings, vol. 1295. CEUR-WS.org, 2014, pp. 31–36.

[22] "WoPeD (Workflow Petri Net Designer," https://woped.dhbw-karlsruhe.de/, accessed: 2021-03-20.

[23] "WOLFGANG - Petri Net Editor," https://github.com/iig-uni-freiburg/WOLFGANG, accessed: 2021-03-20.

[24] R. Nesterov and I. lomazova, "Asynchronous interaction patterns for mining multi-agent system models from event logs," in *Proceedings of MACSPro-2019*, ser. CEUR Workshop Proceedings, vol. 2478. CEUR-WS.org, 2019, pp. 62–73.

[25] P. Pertsukhov and A. Mitsyuk, "Simulating petri nets with inhibitor and reset arcs," *Proceedings of the Institute for System Programming of the RAS*, vol. 31, no. 4, pp. 151–162, 2019.

[26] A. Barros, M. Dumas, and A. ter Hofstede, "Service interaction patterns," in *Business Process Management*, ser. Lecture Notes in Computer Science, W. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, Eds., vol. 3649. Springer Heidelberg, 2005, pp. 302–318.