PLACE_SUBST Transformation of P/T Petri Process Nets and Its Properties

Ivo Martiník

PLACE_SUBST Transformation of P/T Petri Process Nets and Its Properties

Ivo Martiník

VŠB-Technical University of Ostrava, Faculty of Economics,
Sokolská třída 33, 702 00 Ostrava 1, Czech Republic
ivo.martinik@vsb.cz

Abstract. Petri nets is one of mathematical modeling languages for the description of all kind of parallel systems. Property-preserving Petri net process algebras (PPPA) were originally designed for the specification and verification of Petri net processes representing the manufacturing systems. PPPA does not need to verify the composition of Petri net processes because all their algebraic operators preserve the specified set of the properties. These original PPPA are generalized for the newly introduced class of the P/T Petri process nets (PTPN) in this article. The only one PLACE_SUBST transformation is defined for the class of PTPN and its chosen properties are presented. The PTPN can be with the support of the PLACE_SUBST transformation easily applied into the area of design, simulation and verification of multithreading programming systems executed in parallel or distributed environment. This fact is demonstrated on the simple example of the client-server distributed programming system.

Keywords: P/T Petri process net, well-formed process net, property preservation, PLACE_SUBST transformation, parallel systems modeling.

1 Introduction

Petri nets [1, 2, 3, 4] is one of mathematical modeling languages for the description of all kind of parallel systems and they represent a popular formalism connecting advantages of the graphic representation of a modeled system with the possibilities of its simulation and the formal analyzability. Property-preserving Petri net process algebras (PPPA) [6] were originally designed for the specification and verification of manufacturing systems. PPPA also follows many ideas that can be originally seen in the class of workflow nets [5]. The elements of PPPA are the Petri net processes, an ordinary connected Petri nets with a unique entry place, a unique exit place and a set of places for handling resource sharing with their initial marking allowed. Among other features, PPPA does not need to verify composite components because all of their operators preserve many properties. PPPA have five types of operators: extensions, compositions, refinements, reductions and place-merging. All the operators can preserve about twenty properties (some under additional conditions), such as liveness, boundedness, reversibility, traps, siphons, proper termination, etc. Hence, if the primitive modules satisfy

adfa, p. 1, 2011.

the desirable properties, each of the composite components, including the system itself, also satisfies these properties.

In this article we will generalize PPPA for the special class of the P/T Petri process nets (PTPN), define the only one PTPN transformation called **PLACE_SUBST** and present its base characteristic of the well-formedness and purely-formedness properties preservation. This class of PTPN with the support of the **PLACE_SUBST** transformation can be then successfully used in the area of design, simulation and verification of programming systems executed in distributed or parallel environment. This fact is then demonstrated on the simple example of the distributed programming system based on the client-server architecture.

2 P/T Petri process nets and their properties

Let $\boldsymbol{N}$ denotes the set of all natural numbers, $\boldsymbol{N} := \{1, 2, \ldots\}$, $\boldsymbol{N_0}$ the set of all non-negative integer numbers, $\boldsymbol{N_0} := \{0, 1, 2, \ldots\}$, $\varnothing$ the empty set, $\neg$ the logical negation operator, $|A|$ the cardinality of the given set A, where $|\boldsymbol{N}| = \aleph_0$.

Let A be a non-empty set. By the (non-empty finite) **sequence** σ over the set A a function σ can be understood, σ: $\{1, 2, \ldots, n\} \to A$, where $n \in \boldsymbol{N}$. The function ε: $\varnothing \to A$ is called the **empty sequence** over the set A. The sequence σ: $\{1, 2, \ldots, n\} \to A$ is usually represented by the notation $\sigma = a_1\, a_2 \ldots a_n$ of the elements of the set A, where $a_i = \sigma(i)$ for $1 \le i \le n$.

Let $A = \{a_1, a_2, \ldots, a_n\}$, where $n \in \boldsymbol{N}$, is a finite non-empty set. **Vector** V is a function V: $A \to \boldsymbol{N_0}$. We will denote a vector V by the statement $V = (V(1), V(2), \ldots, V(n))$.

Let $\mathbf{V} = \{(V(1), V(2), \ldots, V(n)) \mid V(i) \in \boldsymbol{N_0}$ for $1 \le i \le n\}$ and $\mathbf{W} = \{(W(1), W(2), \ldots, W(m)) \mid W(j) \in \boldsymbol{N_0}$ for $1 \le j \le m\}$. Then

$\mathbf{V} \otimes \mathbf{W} = \{(V(1), \ldots, V(n), W(1), \ldots, W(m)) \mid V(i) \in \boldsymbol{N_0}$ for $1 \le i \le n$, $W(j) \in \boldsymbol{N_0}$ for $1 \le j \le m\}$, if $(\mathbf{V} \ne \varnothing) \wedge (\mathbf{W} \ne \varnothing)$,

$= \mathbf{V}$, if $(\mathbf{V} \ne \varnothing) \wedge (\mathbf{W} = \varnothing)$,

$= \mathbf{W}$, if $(\mathbf{V} = \varnothing) \wedge (\mathbf{W} \ne \varnothing)$,

$= \varnothing$, if $(\mathbf{V} = \varnothing) \wedge (\mathbf{W} = \varnothing)$.

It is clear that the $\otimes$ is an associative operation (i.e., $(\mathbf{V} \otimes \mathbf{W}) \otimes \mathbf{U} = \mathbf{V} \otimes (\mathbf{W} \otimes \mathbf{U})$).

Process net (PN) is an ordered 9-tuple $PN := (P, T, A, AF, TP, RP, IP, OP, RM_s)$, where

- P is a finite non-empty set of **places**,
- T is a finite set of **transitions**, $P \cap T = \varnothing$,
- A is a finite set of **arcs**, $A \subseteq (P \times T) \cup (T \times P)$,
- AF is the **arc function**, AF: $(P \times T) \cup (T \times P) \to \boldsymbol{N_0}$, $AF(x, y) \in \boldsymbol{N}$ iff $(x, y) \in A$, $AF(x, y) = 0$ iff $(x, y) \notin A$,
- TP is a **transition priority** function, TP: $T \to \boldsymbol{N}$,
- RP is a finite set of **resource places**, $RP \subset P$,
- IP is the **input place**, $IP \in (P \setminus RP)$, it is the only one place such that $\bullet IP = \varnothing$,
- OP is the **output place**, $OP \in (P \setminus RP)$, it is the only one place such that $OP\bullet = \varnothing$,

- RM_s is the set of all **allowed resource places static markings**, $RM_s \subseteq \mathbf{M_s}$, where $\mathbf{M_s}$ is the set of all the static markings M_s of the *PN* (see below),
- (*P*, *T*, *A*) is a **connected net**.

We will denote the class of all the PNs by the symbol ***PNET***. The given PN is then described with a bipartite graph containing a finite non-empty set *P* of places used for expressing of the conditions of a modeled process (we usually use circles for their representation), a finite set *T* of transitions describing the changes in the process (we usually draw them in the form of rectangles), a finite set *A* of arcs being principally oriented while connecting the place with the transition or the transition with the place and we usually draw them as lines with arrows, the arc function *AF* assigning each arc with a natural number (such number has the default value of 1, if not explicitly indicated in the PN diagram) expressing the number of removed or added tokens from or to the place associated with that arc when executing a particular transition. Transition priority function *TP* assigns with each transition the natural number value expressing its priority (with the default value of 1). The finite set *RP* of resource places is used for expressing conditions of a modeled process containing some initial resources and we use circles with the double line for their representation. The input place *IP* is the only one non-resource place of PN *PN* with no input arc(s) and the output place *OP* is the only one non-resource place of PN *PN* with no output arc(s). RM_s is the set of all the allowed resource places static markings that is subset of the set $\mathbf{M_s}$ of all the static markings M_s of the PN *PN* (the term static marking M_s of PN *PN* will be explained below). The net (*P*, *T*, *A*) must be connected.

Some commonly used notations for PNs are $\bullet y = \{x \mid (x, y) \in A\}$ for the **preset** and $y\bullet = \{x \mid (y, x) \in A\}$ for the **postset** of a PN element *y* (i.e., place or transition).

Marking *M* of the PN *PN* is a mapping $M: P \rightarrow \mathbf{N_0}$. Marking *M* then expresses the current status of a modeled process. Marking *M* can be written as a $|P|$-vector $M := (M(IP), M(P_1), M(P_2), \ldots, M(P_k), M(R_1), M(R_2), \ldots, M(R_m), M(OP))$, where $P := (IP, P_1, P_2, \ldots, P_n, R_1, R_2, \ldots, R_m, OP)$, $RP := \{R_1, R_2, \ldots, R_m\}$, $n \in \mathbf{N}_0$, $m \in \mathbf{N}_0$.

The transition $t \in T$ is **enabled** in the marking *M* of the PN *PN* if at each input place of the transition *t* are in the marking *M* at least as many tokens as required by the value of the arc function *AF* of the particular input arc of the transition *t*, i.e., if $\forall p \in \bullet t: M(p) \geq AF(p, t)$. If the transition *t* is enabled in the marking *M* of the PN *PN*, we denote that fact in the form of t ***en*** M. **Firing of the transition** $t \in T$ itself consists in the removal of as many tokens from each preset place of the transition *t* as required by the value of the arc function *AF* of the particular input arc of the transition *t*, and adding of as many tokens into each of the postset places of the transition *t* as required by the value of the arc function *AF* of the particular output arc of the transition *t*, i.e., it results in changing the marking *M* into the marking *M'*, where $\forall p \in P: M'(p) = M'(p) - AF(p, t) + AF(t, p)$, that is denoted by $M\,[t\rangle\,M'$.

We say that the marking *M''* is reachable from the marking *M* iff there exists a finite sequence $\sigma := t_1\, t_2\, \ldots\, t_n$, $n \in \mathbf{N}$, of the transitions $t_1, t_2, \ldots, t_n$, such that $M\,[t_1\, t_2 \ldots\, t_n\rangle\,M''$. The set of all the *PN*'s markings reachable from its given marking *M* will be denoted by the symbol $[M\rangle$, the set of all the transition sequences $\sigma := t_1\, t_2\, \ldots\, t_n$, $n \in \mathbf{N}$, of the transitions that are fireable from the marking *M* will be denoted by the symbol $[M\rangle\rangle$, i.e., $[M\rangle\rangle := \{t_1\, t_2 \ldots\, t_n \mid \exists M'' \in [M\rangle: M\,[t_1\, t_2 \ldots\, t_n\rangle\,M'', n \in \mathbf{N}\}$.

Let $k \in \boldsymbol{N}$. The following special markings of the PN $PN := (P, T, A, AF, TP, RP, IP, OP, RM_s)$ are defined:

- **static marking** M_s: $\forall p \in RP: M_s(p) \geq 0$; $\forall p \notin RP: M_s(p) = 0$; $\forall t \in T: \neg(t \; \boldsymbol{en} \; M_s)$,
- **entry marking** M_e: $M_e(IP) = k$; $\forall p \in (P \setminus \{IP\}): M_e(p) = M_s(p)$,
- **exit marking** M_x: $M_x(IP) = 0$; $M_x(OP) = M_e(IP) = k$; $\forall p \notin (RP \cup \{OP\}): M_x(p) = 0$; $\forall t \in T: \neg(t \; \boldsymbol{en} \; M_x)$.

For the given PN *PN* we will denote the set of all its static markings M_s by the symbol $\mathbf{M_s}$, the set of all its entry markings M_e by the symbol $\mathbf{M_e}$ and the set of all its exit markings M_x by the symbol $\mathbf{M_x}$.

Fig. 1, illustrates the PN $PROC := (P, T, A, AF, TP, RP, IP, OP, RM_s)$, where $P = \{\text{IP, P1, R1, OP}\}$, $T = \{\text{T1, T2, T3}\}$, $A = \{(\text{IP, T1}), (\text{IP, T2}), (\text{T1, P1}), (\text{T2, P1}), (\text{R1, T1}), (\text{P1, T3}), (\text{T3, R1}), (\text{T3, OP})\}$, $AF = \{((\text{IP, T1}), 1), ((\text{IP, T2}), 1), ((\text{T1, P1}), 1), ((\text{T2, P1}), 1), ((\text{R1, T1}), 1), ((\text{P1, T3}), 1), ((\text{T3, R1}), 2), ((\text{T3, OP}), 1)\}$, $TP = \{(\text{T1}, 1), (\text{T2}, 1), (\text{T3}, 1)\}$, $IP = \text{IP}$, $OP = \text{OP}$, $RM_s = \{(RM_s(\text{R1}))\} = \{(n) \mid n \in \boldsymbol{N}\}$, in its static M_s, entry M_e and exit M_x markings where $k = 3$ (note the resource place R1 with the initial token in the static marking M_s and the fact that no transition must be enabled in (any) static M_s or exit M_x markings) where the notation $RM_s = \{(RM_s(\text{R1}))\} = \{(n) \mid n \in \boldsymbol{N}\}$ means that in any allowed resource static marking M_s the resource place R1 must contain at least one token.

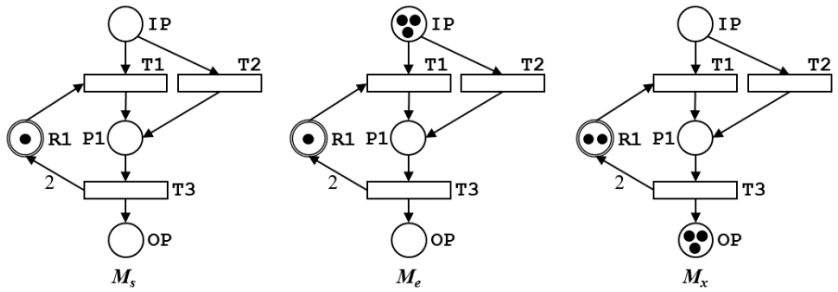

Fig. 1. Static, input and output markings of the PN *PROC*

A **P/T Petri process net** (PTPN) is an ordered couple $PTPN = (PN, M_e)$, where *PN* is a PN and M_e is an entry marking of the PN *PN*. We will then denote a PTPNs by an ordered 10-tuple $PTPN := (P, T, A, AF, TP, RP, IP, OP, RM_s, M_e)$ and the class of all the PTPNs by the symbol ***PTPNET***.

When enabling individual transitions of a given PTPN so called **conflicts** can originate in its certain markings (or **conflict transitions**). At the enabling of the transitions t_1 and t_2 of the given PTPN in its marking M the conflict occurs, if both transitions t_1 and t_2 have at least one input place, each of the transitions t_1 and t_2 is individually enabled in the marking M, but the transitions t_1 and t_2 are not in the marking M enabled in parallel and enabling of one of them will prevent enabling the other (i.e., $(\bullet t_1 \cap \bullet t_2 \neq \varnothing) \wedge (t_1 \; \boldsymbol{en} \; M) \wedge (t_2 \; \boldsymbol{en} \; M) \wedge \neg(\{t_1, t_2\} \; \boldsymbol{en} \; M)$). The term of conflict transitions can be obviously easily generalized for the case of a finite set $t_1, t_2, \ldots, t_n$ ($n \in \boldsymbol{N}$) of the transitions of a given PTPN.

A typical example of the conflict transitions in the particular marking of the PTPN is shown in Fig. 2, where transitions T1 and T2 have a common input place IP, both are enabled, but they are not enabled in parallel. When solving such transitions conflict we will therefore follow the rule which determines, informally said, that from the set of conflict transitions the one will be enabled, whose value of the transition priority function *TP* is the highest. If such transition from the set of conflict transitions does not exist, the given conflict would have to be solved by other means. The transition T2 is then enabled on the basis of that rule in our studied example (because *TP*(T1) = 1 and *TP*(T2) = 2).

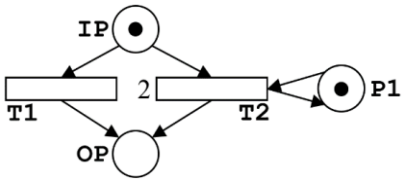

Fig. 2. Conflict transitions in P/T Petri process net

Definition 1. Let $PN := (P, T, A, AF, TP, RP, IP, OP, RM_s)$ be a PN, M_e be its entry marking. Then:

- *PN* is said to be **safe** iff $\forall p \in P\ \forall M \in [M_e\rangle$: $M(p) \leq 1$,
- *PN* is said to be *k*-**bounded** iff $\exists k \in \boldsymbol{N}\ \forall p \in P\ \forall M \in [M_e\rangle$: $M(p) \leq k$,
- *PN* is said to **terminate properly** iff $(\forall M \in [M_e\rangle\ \exists M_x \in \mathbf{M}_x$: $M_x \in [M\rangle) \wedge (|[M_e\rangle\rangle| \neq \aleph_0)$,
- *PN* is said to be **well-formed** PN iff $\forall M_e \in \mathbf{M}_e$: *PN* terminates properly,
- well-formed *PN* is said to be **purely-formed** PN iff $\forall M_e \in \mathbf{M}_e\ \forall M_x \in \mathbf{M}_x$: $((M_e(IP) = M_x(OP)) \Rightarrow (\forall r \in RP$: $M_e(r) = M_x(r))$.

Example 1. Fig. 3, shows the PNs *NET1*, *NET2* and *NET3* in their entry markings M_e, where:

- PN *NET1* does not terminate properly, because there exists the infinite sequence of reachable markings M_e [T1⟩ M_1 [T1⟩ M_2 [T1⟩ …, and so the PN *NET1* is not a well-formed PN,
- PN *NET2* terminates properly for every of its entry markings M_e (and so it is a well-formed PN), but it is not purely-formed PN, because $\exists M_e \in \mathbf{M}_e\ \exists M_x \in \mathbf{M}_x$: $(M_e(\text{IP}) = M_x(\text{OP}) \wedge (\exists \text{R1} \in RP$: $M_e(\text{R1}) \neq M_x(\text{R1}))$, where $M_e = (1, 0, 1, 0)$ and $M_x = (0, 0, 2, 1)$,
- PN *NET3* is a purely-formed PN.

Lemma 1. Let $PN := (P, T, A, AF, TP, RP, IP, OP, RM_s)$ be a PN. Then

(*PN* is well-formed PN) ⇒ ($\forall M_e \in \mathbf{M}_e\ \exists k \in \boldsymbol{N}$: *PN* is *k*-bounded).

Proof. Clear. (*PN* is well-formed PN) ⇒ ($\forall M_e \in \mathbf{M}_e$: *PN* terminates properly) ⇒ ($\forall M_e \in \mathbf{M}_e$: $((\forall M \in [M_e\rangle\ \exists M_x \in \mathbf{M}_x$: $M_x \in [M\rangle) \wedge (|[M_e\rangle\rangle| \neq \aleph_0))) \Rightarrow (\forall M_e \in \mathbf{M}_e$: ($(|[M_e\rangle\rangle|$ is finite) ∧ ($\forall M \in [M_e\rangle$: *M* is finite marking of *PN*))) ⇒ ($\forall M_e \in \mathbf{M}_e\ \exists k \in \boldsymbol{N}$: *PN* is *k*-bounded). □

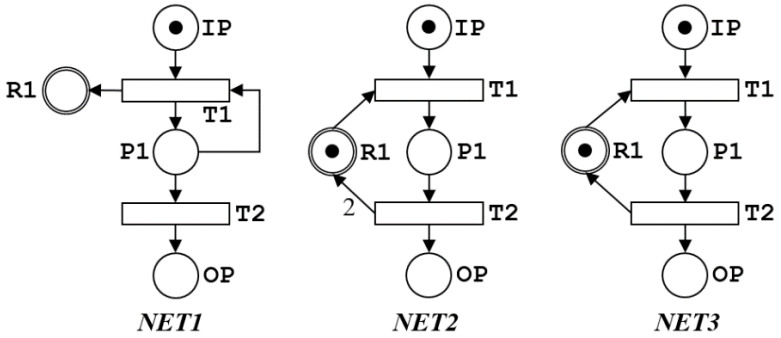

Fig. 3. PNs *NET1*, *NET2* and *NET3* in their entry markings M_e

3 Base P/T Petri process nets and their properties

As it was mentioned in the introduction of this article we want to generalize the PPPA for the class of the PNs. From many properties of PNs we are especially interesting in their well-formedness, resp. purely-formedness, property (in the terminology of programming systems well-formedness property informally means that the programming system modeled by a given PN will not cause deadlock and memory or device overflow and purely-formedness property informally means that given well-formed programming system will not have any side effects). It is of course possible to inspect the properties of traps, siphons, P- and T-invariants etc., of a given PN but it is outside the scope of this article. We will then represent given multithreading programming system by the PN, i.e., programming systems that will be executed in parallel environment by k programming threads, where $k \in \boldsymbol{N}$. These programming threads will be represented by k tokens in the input place *IP* of given PN in its entry marking M_e.

We start our generalization of PPPA for the class of PNs by introduction of so called **base PNs** (BPN). BPNs represent the subclass of PNs that are elementary purely-formed, (i.e., purely-formedness of these PNs can be trivially proved).

Fig. 4, shows six simple BPNs in their static markings M_s (i.e., the letter m in every of their resource places denotes m tokens, $m \in \boldsymbol{N}$). The BPN *BASE1* is the simplest purely-formed BPN at all and it contains only one place IOP that is simultaneously its input and output place (definition of PN allows this case). Purely-formed BPN *BASE3* can be used for the modeling of the programming statement IF … THEN … ELSE … (note that the value of the transition priority function TP(T2) = 2 and it helps to solve potential conflict in the firing of the transitions T1 and T2), purely-formed BPN *BASE4* represents the model of the programming critical section, where typically $m = 1$ in the resource place R1. Purely-formed BPN *BASE5* then represents the synchronous programming method calling mechanism, where the called method is represented by the place P2. Purely-formed BPN *BASE6* can be then used for the modeling of the programming barrier.

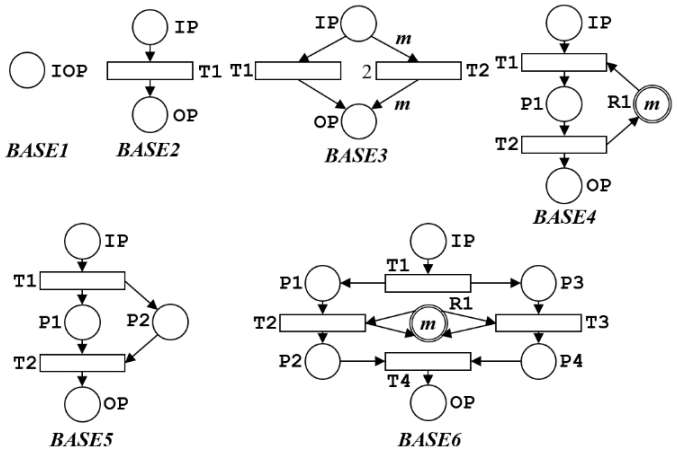

Fig. 4. Examples of BPNs

Fig. 5, shows purely-formed BPN *BASE7* that represents the programming statement PARALLEL FOR i:=1 TO *m* DO …, where $m \in \boldsymbol{N}$. Very interesting purely-formed BPN *BASE8* will be used in the next paragraph for the purpose of the multiple-readers/single-writer lock modeling in the multithreading programming environment. Its purely-formed property can be easily proved for instance with using of the PN marking graph construction (see [1], [3], [7]).

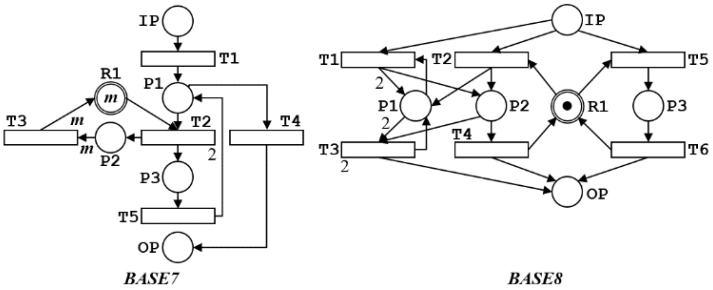

Fig. 5. Examples of BPNs

4 PLACE_SUBST transformation and its properties

The only one transformation **PLACE_SUBST** (i.e., place substitution transformation) is defined for the class of all PNs and it is shown to be preserving the well-formedness and purely-formedness properties. The well-formed, resp. purely-formed, PNs form a closed set (i.e., informally said, the result of an application of the **PLACE_SUBST** transformation onto any two well-formed, resp. purely-formed, PNs will result into another well-formed, resp. purely-formed, PN). The design of a given programming system will typically starts with the BPN *BASE1* and it then follows with the several **PLACE_SUBST** transformations (i.e., substitutions of the selected BPN over the non-

resource place of the actual PN) that result into the complex well-formed, resp. purely-formed, PN that models the whole programming system.

We will denote the set of all ordered pairs (PN, p) where $PN \in \boldsymbol{PNET}$, $PN := (P, T, A, AF, TP, RP, IP, OP, RM_s)$, $p \in (P \setminus RP)$, by the symbol ***PNETPL***. We will then denote the members of the set ***PNETPL*** (i.e., ordered pairs (PN, p)) by the shortened notation $PN.p$.

Definition 2. Let $PN1 := (P_1, T_1, A_1, AF_1, TP_1, RP_1, IP_1, OP_1, RM_{s1})$ and $PN2 := (P_2, T_2, A_2, AF_2, TP_2, RP_2, IP_2, OP_2, RM_{s2})$ are the PNs, let $p \in (P_1 \setminus RP_1)$ is a non-resource place of the PN *PN1*. The transformation **PLACE_SUBST**: $\boldsymbol{PNETPL} \times \boldsymbol{PNET} \rightarrow \boldsymbol{PNET}$ of the PN *PN2* substitution over the place p of the PN *PN1* resulting into the new PN $PN := (P, T, A, AF, TP, RP, IP, OP, RM_s)$ will be denoted by the statement

$$PN := PN1.p <] PN2,$$ where:

- $P := (P_1 \setminus \{p\}) \cup P_2$,
- $T := T_1 \cup T_2$,
- $A := A_2 \cup \{(x, y) \in A_1 \mid (x \neq p) \vee (y \neq p)\} \cup \{(x, IP_2) \mid (x, p) \in A_1\} \cup \{(OP_2, y) \mid (p, y) \in A_1\}$,
- $AF := AF_2 \cup \{((x, y), v) \in AF_1 \mid (x \neq p) \vee (y \neq p)\} \cup \{((x, IP_2), v) \mid ((x, p), v) \in AF_1\} \cup \{((OP_2, y), v) \mid ((p, y), v) \in AF_1\}$,
- $TP := TP_1 \cup TP_2$,
- $RP := RP_1 \cup RP_2$,
- $IP := IP_1$, if $p \neq IP_1$; $IP = IP_2$, if $p = IP_1$,
- $OP := OP_1$, if $p \neq OP_1$; $OP = OP_2$, if $p = OP_1$,
- $RM_s := RM_{s1} \otimes RM_{s2}$.

Example 2. The result of the **PLACE_SUBST** transformation *BASE5*.P2 <] *BASE2* where the place P2 of the BPN *BASE5* (see Fig. 4) was substituted by the BPN *BASE2* can be shown in Fig. 6.

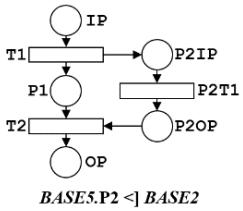

Fig. 6. Result of **PLACE_SUBST** transformation *BASE5*.P2 <] *BASE2*

Lemma 2. Let $PN1 := (P_1, T_1, A_1, AF_1, TP_1, RP_1, IP_1, OP_1, RM_{s1})$, $PN2 := (P_2, T_2, A_2, AF_2, TP_2, RP_2, IP_2, OP_2, RM_{s2})$ and $PN3 := (P_3, T_3, A_3, AF_3, TP_3, RP_3, IP_3, OP_3, RM_{s3})$ are the PNs. Let $p1 \in (P_1 \setminus RP_1)$ is a non-resource place of the PN *PN1* and $p2 \in (P_2 \setminus RP_2)$ is a non-resource place of the PN *PN2*. Then:

$$PN1.p1 <] (PN2.p2 <] PN3) = (PN1.p1 <] PN2).p2 <] PN3.$$

Proof. Follows directly from the *Definition 2*. □

Lemma 3. Let *PN1* := (P_1, T_1, A_1, AF_1, TP_1, RP_1, IP_1, OP_1, RM_{s1}) is a well-formed, resp. purely-formed, PN and *PN2* := (P_2, T_2, A_2, AF_2, TP_2, RP_2, IP_2, OP_2, RM_{s2}) is a well-formed, resp. purely-formed, PN. Let $p \in (P_1 \setminus RP_1)$ is a non-resource place of the PN *PN1*. Let $(\forall t \in p\bullet: (p, t) \in A_1 \Rightarrow \bullet t \times p\bullet \subseteq A_1) \wedge (\forall t \in p\bullet: AF_1(p, t) = 1))$.
Then *PN1*.*p* <] *PN2* is a well-formed, resp. purely-formed, PN.

Proof. Clear. Because both PN *PN1* and *PN2* are well-formed, i.e., ($\forall M_e \in \mathbf{M}_e$: *PN1* terminates properly) $\wedge$ ($\forall M_e \in \mathbf{M}_e$: *PN2* terminates properly), then for the PN *PN1* holds true that $\forall M \in [M_e\rangle$, $M(p) > 0$, $\exists M_x$: $M_x \in [M\rangle$, and for the PN *PN2* holds true that $\forall M \in [M_e\rangle$, $M(IP_2) > 0$, $\exists M_x$: $M_x \in [M\rangle$), then the whole PN *PN1*.*p* <] *PN2* must also terminate properly for any of its input markings M_e. The property of purely-formedness of the resulting PN *PN1*.*p* <] *PN2* can be proved similarly.
The necessity of the condition $(\forall t \in p\bullet: (p, t) \in A_1 \Rightarrow \bullet t \times p\bullet \subseteq A_1)$ fulfilment is presented in Fig. 7., where both *PN1* and *PN2* are purely-formed PN but the resulting PN *PN1*.P2 <] *PN2* is not any more purely-formed PN. This fact can be seen for instance from the transition sequence (1, 0, 0, 0, 0) [T1⟩ (0, 1, 1, 0, 0) [T2⟩ (0, 0, 0, 1, 1) that causes deadlock. This problem follows from the fact that for the PN *PN1* is not satisfied the condition (P2, T3) $\in A_1 \Rightarrow$ (P2, T2) $\in A_1$. Satisfaction of this condition that is well-known from the class of free choice Petri nets (see [8]) implies of the purely-formedness of the resulting PN *PN1*.P2 <] *PN2*.

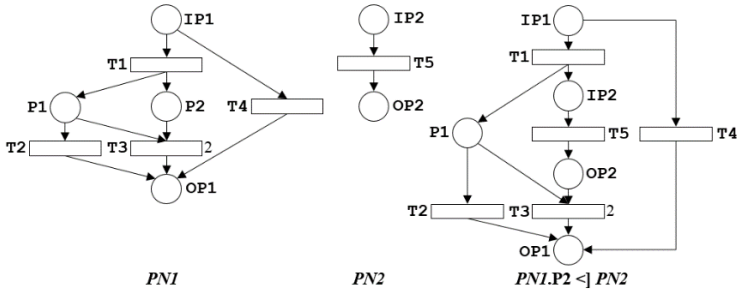

Fig. 7. Place substitution operation *PN1*.P2 <] *PN2*

The necessity of the condition $(\forall t \in p\bullet: AF_1(p, t) = 1)$ fulfilment is presented in Fig. 8., where both *PN3* and *PN4* are purely-formed PN but the resulting PN *PN3*.P1 <] *PN4* is not any more purely-formed PN. This fact can be seen for instance from the transition sequence (1, 0, 0, 0, 1, 0) [T1⟩ (0, 3, 0, 0, 1, 0) [T4⟩ (0, 2, 1, 0, 0, 0) [T5⟩ (0, 2, 0, 1, 1, 0) [T3⟩ (0, 2, 0, 1, 1, 0) that causes impossibility of reaching of the exit marking M_x of the PN. This problem follows from the fact that for the PN *PN1* is not satisfied the condition *AF*(P1, T2) = 1. Satisfaction of this condition implies of the purely-formedness of the resulting PN *PN3*.P1 <] *PN4*. □

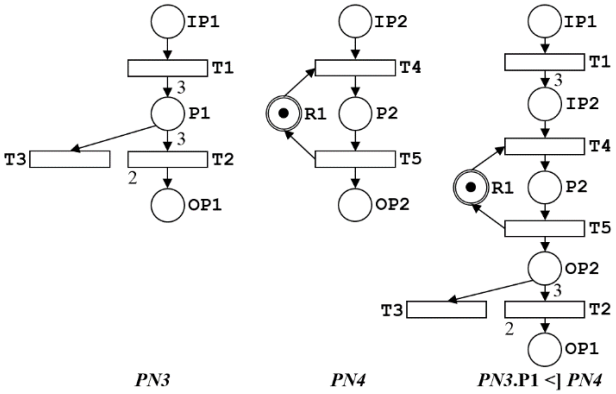

Fig. 8. Place substitution operation *PN3*.P1 <] *PN4*

5 An example of PLACE_SUBST transformation application at the modeling of distributed programming system

In the following simple example we will develop a design of a PN modeling the distributed programming system operating on the client and application server side and realizing reading and writing procedures from the given database system on the basis of a client requests. Individual client requests are executed via a finite set of programming threads on the server side (in our model of that programming system we assume availability of totally three server side programming threads) stored in the created pool of threads. At the moment of accepting the user request is for the needs of its responding a free thread randomly selected in pool which realizes such request, i.e., it will ensure execution of reading or writing data into the database system. In so doing it is required that in the case of realization of the request for the writing of the database no other thread can access the database environment and in the case of request for the reading any finite number of threads realizing reading of data can access the database.

Fig. 9a, shows the PN *BASE7*.P3 <] *BASE5* in its entry marking M_e (where M_e(R1) = 4) that represents the client side of the whole distributed programming system. Client side will perform PARALLEL FOR cycles with two programming threads (the tokens in the input place IP) requesting the read and write server side database operations (firing of the transition T6). Server side database service is represented by the place P4 and will be requested by synchronous manner.

Fig. 9b, shows the PN *BASE4*.P1 <] *BASE8* determined for the execution of user requirements on reading or writing into the database while using three common accessible programming threads located in the common programming pool (see the resource place R10). The respective threads realize either reading (firing of the transition T20, resp. T21) or writing (firing of the transition T22) of the data from or to the database via the mechanisms of BPN *BASE8* (see Fig. 5) that models the access into the database environment with the support of multiple-readers/single-writer lock. The token in the resource place R20 represents single lock determined for obtaining the multiple-read-

ers/single-writer access of the database environment. Following realization of the relevant event the programming thread will be returned back into the common pool of threads represented by the resource place R10 and it will be ready for the execution of the next user request.

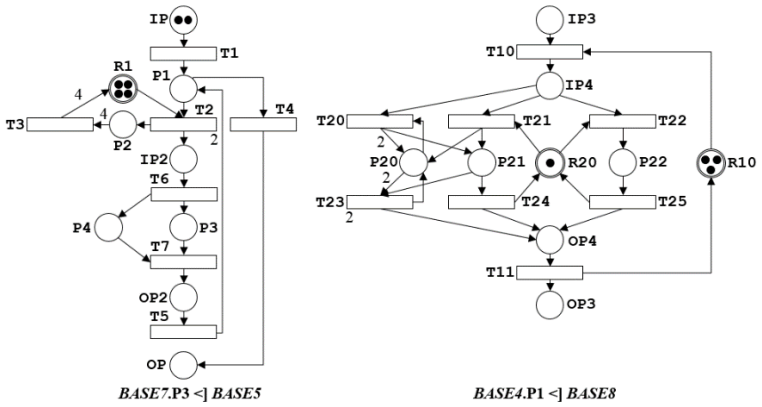

Fig. 9a, b. PNs modeling client and server side of distributed programming system

We will obtain the resulting PN that represents the whole distributed programming systems as shown in Fig. 10, by applying the following **PLACE_SUBST** transformation: (*BASE7*.P3 <] *BASE5*).P2 <] (*BASE4*.P1 <] *BASE8*). This resulting PN is according to *Lemma 3* (because the BPNs *BASE4*, *BASE5*, *BASE7* and *BASE8* are purely-formed PNs) purely-formed PN.

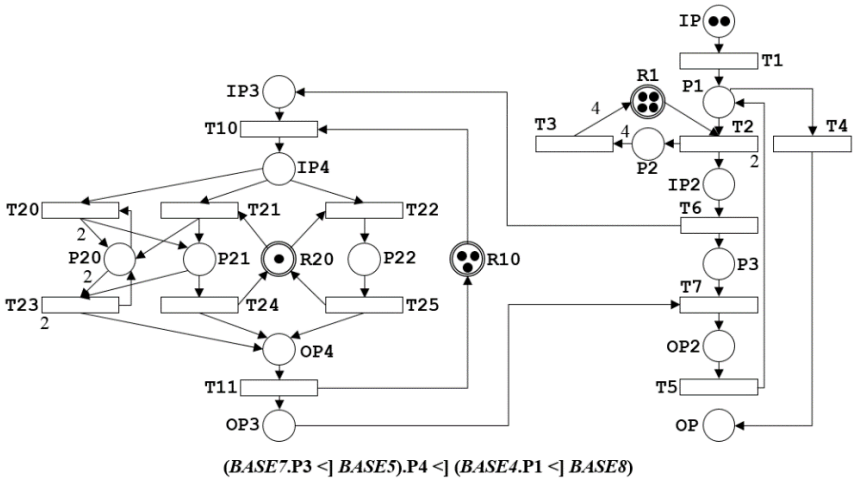

Fig. 10. Resulting PN modeling distributed programming system

6 Conclusions

There is an opportunity to define additional transformations and operators for the class of PNs and to generalize it (for instance transformations for connecting two or more PNs). Selected PN then represents typically a method in the model of the programming system realized with the support of the Petri nets formalism. PNs can be thus successfully applied also in the area of the bi-relational P/T Petri nets [9] that represent an interesting modification of conventional P/T Petri nets.

The principles introduced in the class of PNs can be further generalized and implemented in the definition of the class of high-level Petri nets called the sequential object Petri nets [10, 11] which differ from PNs mainly by mutually differentiable tokens. The tokens alone are formed by non-empty sequences over the set of all integer numbers (this is also where the name of this Petri nets class came from). Sequential object Petri nets are then specially determined for the design, modeling and verification of distributed multithreading programming systems based on the object oriented and functional paradigmas.

This paper has been elaborated in the framework of the project CZ.1.07/2.3.00/20.0296 supported by the European Social Fund.

7 References

1. Reisig, W., Rozenberg, G.: Lectures on Petri Nets I: Basic Models. Springer, Heidelberg (1998)
2. Diaz, M.: Petri Nets: Fundamental Models, Verification and Applications. John Willey & Sons, ISTE Ltd. (2009)
3. Reisig, W.: Elements of Distributed Algorithms. Springer, Heidelberg (1998)
4. David, R., Alla, H.: Discrete, Continous and Hybrid Petri Nets. Springer (2010)
5. van der Alst, W., van Hee, K.: Workflow Management: Models, Methods and Systems. The MIT Press, Massachusetts (2002)
6. Huang, H., Jiao, L., Cheung, T., Mak, W. M.: Property-Preserving Petri Net Process Algebra In Software Engineering. World Scientific Publishing Co. Pte. Ltd., Singapore (2012)
7. Martiník, I.: Modeling of Distributed Programming Systems with Using of Property-preserving Petri Net Process Algebras and P/T Petri Net Processes. In: ICIA 2013 proceedings: the Second International Conference on Informatics & Applications (ICIA2013). pp. 258-263. Lodz University of Technology, IEEE (2013)
8. Desel, J., Esparza, J.: Free Choice Petri Nets. Cambridge University Press, New York (1995)
9. Martiník, I.: Bi-relational P/T Petri Nets and the Modeling of Multithreading Object-oriented Programming Systems. Communications in Computer and Information Science. Volume 188 CCIS (Part 1), pp. 222-236 (2011)
10. Martiník, I.: Sequential Object Petri Nets and the Modeling of Multithreading Object-Oriented Programming Systems. In: Petri Nets - Manufacturing and Computer Science. pp. 195-224. InTech, Rijeka, Croatia (2012)
11. Martiník, I.: Modeling of Object-Oriented Programming Systems with Using of Petri Nets. SAEI, vol. 5. Ostrava, VŠB-TU Ostrava (2015)