Hierarchical, Reconfigurable Petri Nets

Jan-Uriel Lorbeer, Julia Padberg[1]

Abstract: Hierarchical Petri nets allow a more abstract view whereas reconfigurable Petri nets model dynamic structural adaptation. In this contribution we present the combination of reconfigurable Petri nets and hierarchical Petri nets yielding an hierarchical structure for reconfigurable Petri nets. Hierarchies are established by substituting transitions by subnets. These subnets are themselves reconfigurable, so they are supplied with their own set of rules, so-called local rules. Moreover, global rules, that can be applied in all of the net, are provided.

Keywords: reconfigurable Petri nets; Petri net transformations; hierarchical Petri nets

1 Introduction

Modelling modern systems presents a lot of challenges some of which can be eased by the use of appropriate models. A well known technique for system modelling are Petri nets. Petri nets provide a graphical language for constructing system models as well as a precise, mathematical semantics. These models can be used for simulations and to analyse the model's properties. This allows locating possible faults in the system at earlier stages which also can decrease overall development costs. The increasing sizes of modern systems results in models becoming rather large and possibly hard to comprehend. Additional abstraction layers as for example hierarchies present one solution for the increasing size. Hierarchical Petri nets use hierarchy to break down the complexity of a large model, by dividing it into a number of submodels. This helps to concentrate on a specific system part without the need to take the whole system into account. Also submodels can be reused with less effort at multiple locations in the same system or even in a different system where similar components are used.

Advanced systems that need dynamic structural adaptation can be modelled using reconfigurable Petri nets, an approach for dynamic changes in Petri nets. A reconfigurable Petri net consist of a Petri net and set of rules that modify the Petri net's structure at runtime. Hence they increase flexibility enabling the modification of a net while allowing the transitions to fire. The characteristic feature of reconfigurable Petri nets is the possibility to discriminate between different levels of change as they consist of a Petri net and a set of rules that can modify the Petri net. They provide powerful and intuitive formalisms to model dynamic software or hardware systems that are increasingly executed in dynamic infrastructures.

[1] Hamburg University of Applied Sciences, Hamburg, Germany, julia.padberg@haw-hamburg.de

Such infrastructures are dynamic as they are themselves subject to change and support various applications that may or may not share some of the resources. Reconfigurable Petri nets have been used in many different application areas, that require both the representation of their processes and of the system changes within one model. Examples are concurrent systems [LO04], mobile ad-hoc networks [Pa07], workflows in dynamic infrastructures [HEP08], communication spaces [MGH10, GE12], ubiquitous computing [GNH12, Bo06], flexible manufacturing systems [Tâ12], reconfigurable manufacturing systems [KBD16]). In [PK18] a comprehensive overview of reconfigurable Petri nets is given, including theoretical foundations and application areas. The term reconfigurable Petri nets is used since the underlying type of Petri net may vary (for example being place/transition nets, object nets, timed and/or stochastic nets, or high-level nets). Hence, this approach can be considered a family of formal modelling techniques (see [PK18] for details). Reconfigurable Petri nets are an instantiation of abstract transformation systems, called $\mathcal{M}$- adhesive transformations systems, that are formulated in category theory. The fundamental idea is to characterize those categories that allow double-pushout transformations: therefore only the diagrammatic descriptions are needed. This has the advantage of a thorough theory that yields a vast amount of results concerning the transformation part. Nevertheless, in this contribution we omit the categorical foundation.

Hierarchical, reconfigurable Petri nets combine the hierarchical Petri net types and reconfigurable Petri net types into one, allowing a focused design of submodels and their reusability and the ability for dynamic changes at runtime. The main idea is to use the hierarchies purely at the surface, so the semantics is defined in terms of the underlying flattened reconfigurable net.

Main Concept: Hierarchy as a syntactic extension
The hierarchy based on transitions being replaced by subnets is given as a syntactic abbreviation. The hierarchical reconfigurable net is defined purely by it's flattening into a reconfigurable net.

This fundamental design decision has the following advantages: First, only the consistency of the flattening construction can be guaranteed using well-known results, but no further semantic correctness needs to be proven. Second, this corresponds directly to the intended implementation of hierarchies in ReConNet [Re17, Pa12] (ReConfigurable Net). Moreover, the transformation systems needs not to be shown for another category of hierarchical nets. And, apart from the flattening construction, which needs to be done once, the analysis and verification of a hierarchical Petri net requires no more effort than it's counter part with no hierarchy, since the whole behaviour is defined in terms of the flattened net.

The software tool ReConNet has been developed so that the modelling and simulation capabilities of reconfigurable nets are supported adequately. The most important feature of the tool is the ability to create, modify and simulate reconfigurable nets in a single tool through an intuitive graphic-based user interface. It allows the design and verification of reconfigurable place/transition nets. In this paper we discuss the basis for the integration of hierarchical reconfigurable Petri nets in ReConNet.

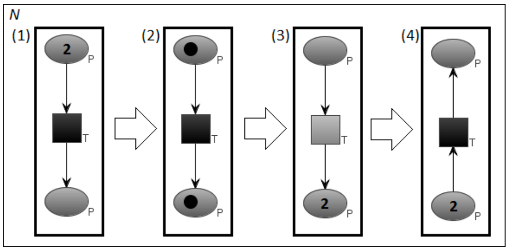
This paper is organised as follows: We start by given a very simple introductory example of hierarchical reconfigurable Petri nets in Sect. 2. In Sect. 3 related work, namely approaches to hierarchical Petri nets and hierarchical graph transformations are discussed. The subsequent section introduces reconfigurable place/transition nets with labels. Sect. 5 elaborates the formal definition of hierarchical reconfigurable Petri nets and their flattening, the definition of transformation rules and proves the correctness of the flattening construction. The integration of reconfigurable hierarchical Petri nets into the simulation tool for reconfigurable Petri nets ReConNetis discussed in Sect. 6. The conclusion in Sect. 7 completes this paper.

2 Introductory Example

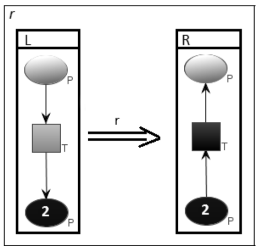
Reconfigurable Petri nets extend classical Petri nets to include the possibility of dynamic changes. This is achieved through the use of a rewriting system and allows the modification of the net's structure at run time. Such a system provides two kinds of changes:

- a change of state accomplished through the firing of the net's transitions
- a change of the process itself induced by the rules.

A reconfigurable Petri net N can either fire an activated transition or execute a transformation step $N \overset{r}{\Rightarrow} M$. An example for this process is displayed in Fig. 1. In Fig. 1a a reconfigurable Petri net N is shown and in Fig. 1b it's rule r with r's nets L, K and R is given. N consists of two places, two tokens and one transition which is black when activated. When rule r is executed the net's arcs are inverted. In it's initial state (1) the rule r is not executable because there is no match to the rule's left-hand net L. The net L specifies that at least two tokens are required at the place P at the figure's bottom where the edges lead to. So only the transition T can fire. Firing transition T twice state (3) is reached. In this state two tokens are located on the bottom place P and r can be executed inverting the arc directions resulting in state (4). If in state (3) there would have been an additional token on the upper place P either the transition T could have fired or r could have been executed. In state (4) r is no longer executable because the edges now lead to the upper place P, so only the transition T can fire. State (4) is very similar to state (1) and after T firing twice r would be executable once again.



(a) State sequence of net N



(b) Rule r

Fig. 1: Reconfigurable Petri net $RN = (N, \{r\})$

To give an intuition of hierarchical reconfigurable Petri nets we subsequently give an example that comprises the ideas presented in the following sections. Imagine some simple but adaptive process that can alternatively execute three different tasks `task1`, `task2`, and `task3`. An abstract view of this process is given in Fig. 2.

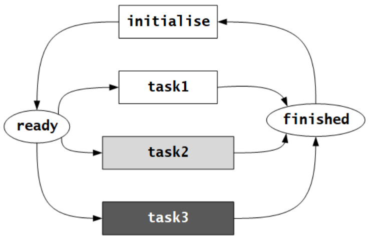


Fig. 2: Abstract view: Net AN

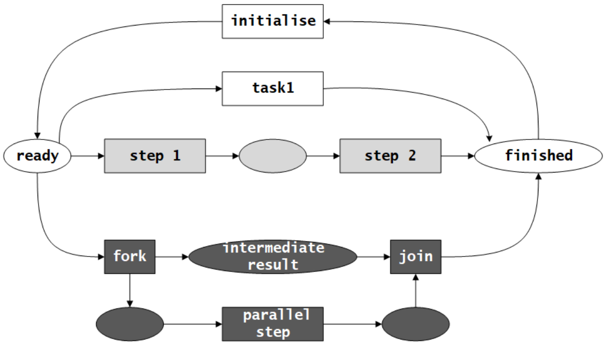


Fig. 3: Flattened net $FLAT(AN)$

The tasks `task2` and `task3` are more complex and are given by subnets, where `task2` is a sequence of steps and `task3` includes some forking. The hierarchy concept in Sect. 5 allows the substitution of the transitions with the subnets. The substitution of the transition `task2` replaces the transition and its adjacent places, that is Net(`task2`), by the subnet $SN1$ and Net(`task3`) is replaced by $SN2$, both in Fig. 4. Applying these substitutions to the abstract nets in Fig. 2 yields the flatted net in Fig. 3.

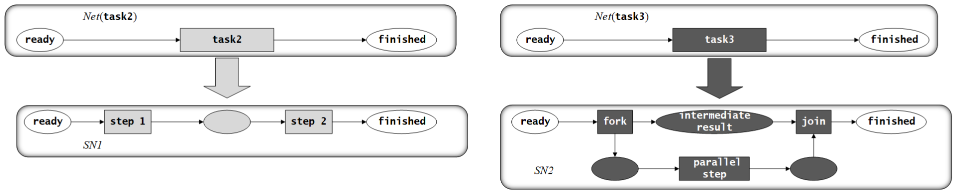


Fig. 4: Substitution of transitions

Now we add rules for the subnets for the adaptation of the tasks: `task1` is so simple, it requires no adaptation. In `task2` the sequence of steps can be changed (rules `SN1:r1` and `SN1:r2`) or an intermediate steps is introduced or removed (rules `SN1:r3` and `SN1:r4`). So we have the four rules that belong to subnet $SN1$ and are given in light grey in Fig. 5. In `task3` the intermediate step can be adapted by rule `SN2:r5` so that parallel step may require something from the intermediate result. And this adaptation can be reversed by rule `SN2:r6`. Both rules belong to subnet $SN2$ and are given in dark grey in Fig. 5. These six rules are local rules, that should be only applied in the corresponding subnet. The transitions have the name space A_T = {`initialise`, `task1`, `task2`, `task3`, `fork`, `join`, `step`, `step1`, `step2`, `intermediate step`, `parallel step`} that ensures the locality of the rules by the labels.

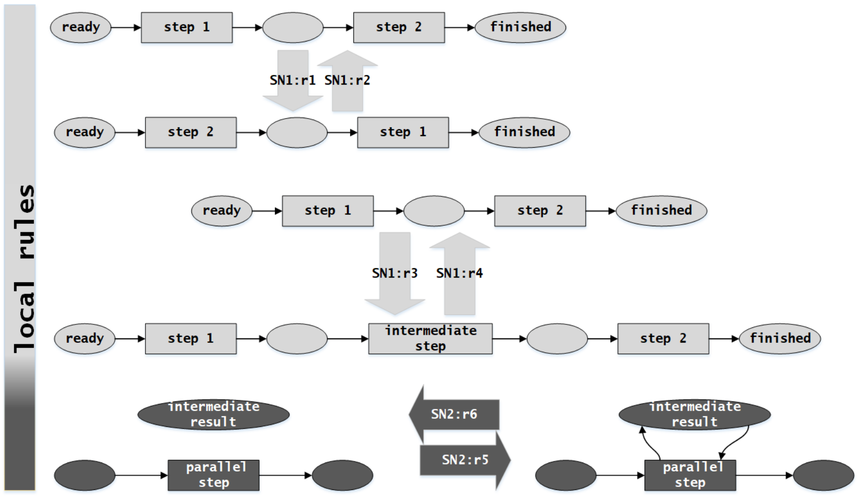


Fig. 5: Local rules for the subnets $SN1$ and $SN2$

Additionally, we want a global rule that adds to all steps a counting place. This rule is given below in Fig. 6. This rule can be applied at each transition with a lesser label. The name space A_T for the transition is ordered in the following way:
$\mathfrak{g}_T \geq l$ for all $l \in A_T$ and
`step` $\geq l$ for all $l \in$ {`step1`, `step2`, `intermediate step`, `parallel step`}
which are the other local steps.

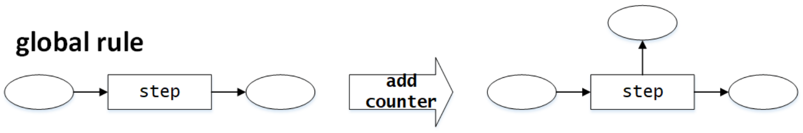


Fig. 6: Global rule for adding counter

3 Related Work

The main related research areas for hierarchical reconfigurable Petri nets are obviously hierarchical Petri nets and hierarchical graph transformations.

From 1991 on, hierarchies in Petri nets [HJS91] have been used in numerous papers and contributions and are still prominent, e.g. [HW18, Ak17]. In [HW18] the HiPS tool provides a common operating method, graphical user interface, the ability to describe hierarchical Petri nets and an on-the-fly linear temporal logic model checker. [Ak17] defines a hierarchy of places, and with various arcs as inhibitor arcs, reset arcs and transfer arcs that respect this hierarchy. A total ordering over the places encodes the hierarchy depending on the arc types various extensions are given. The decidability for almost all these extensions has been shown using different reductions and proof techniques. Besides hierarchical Petri nets based on transition substitution as e.g. in [JR91] nets based on place substitution and Object-Oriented Petri nets [La01, MK05] have been considered. CPN [CP17] also supports fusion places by defining sets of places that are functionally identical. The net-in-nets approach [Va98, Va04] as well as object nets [KH10, KH11] yield hierarchical composition as well.

There are a number of tools similar to ReConNet. *CPN tools* [Ra03, CP17] is the main tool for the modelling and simulation of coloured Petri nets. Using a graphic user interface CPN tools features syntax checking, code generation and state space analysis. *Snoopy* [He12] is a unifying Petri net framework with a graphical user interface. It allows the modelling and simulation of coloured and uncoloured Petri nets of different classes, supports analytic tools and the hierarchical structuring of models. The *HiPS* tool [Hi17] developed at the Department of Computer Science and Engineering, Shinshu University is a tool written in C# and also employs a graphical user interface. HiPS is a platform for design and simulation of hierarchical Petri nets. It also provides functions of static and dynamic net analysis. While all of these tools support the design of hierarchical Petri nets it is evident that they lack reconfiguration, ReConNet's core feature.

Hierarchical graphs (and graph transformations) add some hierarchy to the nodes or to the edges. Various approaches to graphs with hierarchy have been proposed, e.g. [BH01, DHP02, BKK05, Br14]. The resulting techniques were used for modelling hierarchical hypermedia, distributed project management, mobile and ubiquitous systems among others. Hypergraphs in [DHP02] consist of two finite sets of nodes and hyperedges. The hierarchy is given in layers, in the sense that subsets in the same layer have the same nesting depth. In [BKK05] hierarchical graph is a system of graphs of various types that are grouped into packages. Hierarchical graphs in [Pa04] are obtained from hypergraphs by adding a parent assigning function to them. Nodes and edges can be assigned as a child of any other node or edge. But in contrast to our approach only the graph is structured, whereas the rules only are given at the topmost level.

4 Basics of Reconfigurable Petri Nets

In this section we give the basic notions. Note that in ReConNet the underlying type of nets are decorated place/transition nets. To concentrate on the essential notions, in this paper we use place/transition nets, but the corresponding technical report [Pa18] deals explicitly with decorated place/transition nets. Moreover, in [Pa18] the underlying categorical requirements are proven for place/transition nets, decorated place/transition nets, as well as for algebraic high-level nets.

We use the algebraic approach to Petri nets, where the pre- and post-domain functions $pre, post : T \to P^{\oplus}$ map the transitions T to a multiset of places $P^{\oplus}$ given by the set of all linear sums over the set P. A marking is given by $m \in P^{\oplus}$ with $m = \sum_{p \in P} k_p \cdot p$. The $\leq$ operator can be extended to linear sums: For $m_1, m_2 \in P^{\oplus}$ with $m_1 = \sum_{p \in P} k_p \cdot p$ and $m_2 = \sum_{p \in P} l_p \cdot p$ we have $m_1 \leq m_2$ if and only if $k_p \leq l_p$ for all $p \in P$. The operations "+ " and "– " can be extended accordingly.

Here, we introduce reconfigurable place/transition nets with labels and subtyping of labels for the rules. These labels need a name space that is given by a partial order $(A, \leq, \mathfrak{g}_A)$ with a greatest element, $a \leq \mathfrak{g}_A$ for all $a \in A$. $\mathfrak{g}_A$ is similar to the common supertype in a type hierarchy. The labelling function is provided with an order for subtyping, this allows more abstract rules that can be applied for occurrences with lesser labels, for an example see Sect.2.

Definition 4.1 (Labelled place/transition nets) *A (marked labelled place/transition) net is given by $N = (P, T, pre, post, pl, tl, M)$ over the name space $A = (A_P, A_T)$ with partial orders $(A_P, \leq_A, \mathfrak{g}_p)$ and $(A_T, \leq_T, \mathfrak{g}_T)$. P is a set of places and T is a set of transitions. $pre : T \to P^{\oplus}$ maps a transition to its pre-domain and $post : T \to P^{\oplus}$ maps it to its post-domain. Moreover, $pl : P \to (A_P, \leq_A, \mathfrak{g}_p)$ is a label function mapping places to a name space, $tl : T \to (A_T, \leq_T, \mathfrak{g}_T)$ is a label function mapping transitions to a name space and $M \in P^{\oplus}$ is the marking denoted by a multiset of places.*

A transition $t \in T$ is M-enabled for a marking $M \in P^{\oplus}$ if we have $pre(t) \leq M$. The successor marking m' is computed by $M' = M - pre(t) + post(t)$ and represents the result of a firing step $M[t > M'$.

Note, we do not require $P \cap T = \emptyset$ since this is not necessary. This condition would require constructing the transformation step explicitly to ensure that the target net M in Fig. 7 satisfies this construction as well.

A reconfigurable Petri net $RN = (N, \mathcal{R})$ consists of a Petri net N and a set of rules $\mathcal{R}$. This allows reconfigurable Petri nets to modify themselves. Rules and occurrences are defined by net morphisms. Net morphisms are given as a pair of mappings for the places and the transitions preserving the structure, the labels and the marking. Given two nets N_1 and N_2 as in Def. 4.1 a net morphism $f : N_1 \to N_2$ is given by $f = (f_P : P_1 \to P_2, f_T : T_1 \to T_2)$, so that $pre_2 \circ f_T = f_P^\oplus \circ pre_1$ and $post_2 \circ f_T = f_P^\oplus \circ post_1$ and $m_1(p) \le m_2(f_P(p))$ for all $p \in P_1$. The labels are mapped so that $tl_2 \circ f_T(t) \le tl_1(t)$ for all $t \in T_1$ and $pl_2 \circ f_p(p) \le pl_1(p)$ for all $p \in P_1$.

$$\begin{tikzcd}
 & T_1 \arrow[r, shift left, "pre_1"] \arrow[r, shift right, "post_1"'] \arrow[dl, "tl_1"] \arrow[dd, "f_T"] & P_1^\oplus \arrow[dr, "pl_1"] \arrow[dd, "f_P^\oplus"] & \\
(A_T, \le_T, g_T) & & & (A_P, \le_P, g_P) \\
 & T_2 \arrow[r, shift left, "pre_2"] \arrow[r, shift right, "post_2"'] \arrow[ul, "tl_2"] & P_2^\oplus \arrow[ur, "pl_2"'] &
\end{tikzcd}$$

Moreover, the morphism f is called strict if both f_P and f_T are injective, if $tl_2 \circ f_T = tl_1$ and $pl_2 \circ f_p = pl_1$, and if $m_1(p) = m_2(f_P(p))$ holds for all $p \in P_1$.

Rules $r = (L \leftarrow K \rightarrow R)$ consist of a span of strict net morphisms where L is the left-hand side and K is an interface between L and R the right-hand side. The basic idea is to find L in the net N and replace it by R. Therefore an occurrence morphism $o : L \rightarrow N$ is required to identify the relevant parts of the left-hand side L in N. Given a rule r and an occurrence o a transformation step $N \overset{(r,o)}{\Longrightarrow} M$ is constructed in two steps by the commutative squares via a gluing construction, that is a pushout. Fig. 7 illustrates the transformation of a net using two pushouts **(PO1)** and **(PO2)**. An intuitive explanation for a pushout is to take two nets N_1, N_2 that overlap in some I. Now we glue N_1 and N_2 together over this common interface I, obtaining a new net $N_1 +_I N_2$ that is the union of both nets factorized by an equivalence relation induced by the inclusion of I into N_1 and N_2.

$$\begin{tikzcd}
L \arrow[d, "o"'] \arrow[dr, phantom, "\textbf{(PO1)}"] & K \arrow[l] \arrow[r] \arrow[d] \arrow[dr, phantom, "\textbf{(PO2)}"] & R \arrow[d] \\
N & D \arrow[l] \arrow[r] & M
\end{tikzcd}$$

Fig. 7: Net transformation

Given a rule with an occurrence $o : L \rightarrow N$ the *gluing condition* has to be satisfied in order to apply a rule at a given occurrence. Its satisfaction requires that the deletion of a place implies the deletion of the adjacent transitions, and that the deleted place's marking does not contain more tokens than the corresponding place in L. After finding an occurrence of a left-hand side L in a net, the effect of applying a rule is to remove all obsolete elements and add all fresh elements. The elements of K are preserved, providing us with well-defined attachment points for R.

This transformation concepts is based on $\mathcal{M}$-adhesive transformation systems [Eh06] and is given in terms of category theory, so that is allows the application of the concepts and results to high-level structures such as graphs, typed attributed graph, hyper-graphs, various types of Petri nets, algebraic signatures and specifications. This is possible because place/transition nets can be proven to be an $\mathcal{M}$-adhesive transformation category [Eh06, Pa15] and nets with labels and subtyping in [Pa18]. Hence these results hold for the corresponding type of Petri net:

- Local Church-Rosser Theorem for pairwise analysis of sequential and parallel independence
  see Thm. 5.12 in [Eh06]
- Parallelism Theorem for applying independent rules and transformations in parallel
  see Thm. 5.18 in [Eh06]
- Concurrency Theorem for applying E-related dependent rules simultaneously
  see Thm. 5.23 in [Eh06]
- Embedding and Extension Theorem for transferring transformations and analysis results to more complex scenarios
  see Thms. 6.14 and 6.16 in [Eh06]
- Local Confluence Theorem and Completeness of critical pairs for analysing conflicts and for showing local Confluence
  see Thm. 6.28 and Lemma 6.22 in [Eh06]

5 Hierarchies of Nets and Rules

A hierarchical reconfigurable Petri net uses *substitution transitions* to implement the hierarchy. A substitution transition is a special kind of transition that itself does not fire, instead it contains a subnet that defines the behaviour that takes place in its stead. Following this basic definition of substitution transitions, different implementations suited for specific purposes are possible, this work focuses on the variant of the substitution transition based hierarchical Petri net that have been presented in [JK09].

Each substitution transition has its own subnet with its own local rules. All places that share an edge with a substitution transition are called the transition's *connecting places*. For each connecting place of the substitution transition there exists a corresponding connecting place in the transition's subnet with the same marking. Via these places tokens enter and leave the subnet. Any transition that may fire belongs either to the main net or to some subnet, but it cannot be a substitution transition. Any net may contain multiple substitution transitions, each instantiating exactly its own subnet. Although multiple substitution transitions may instantiate the same subnet layout, each substitution transition has it's own permanent instance. This leads to a behaviour of the main nets that relies solely on the firing of the subnets, i.e the firing of the flattened net.

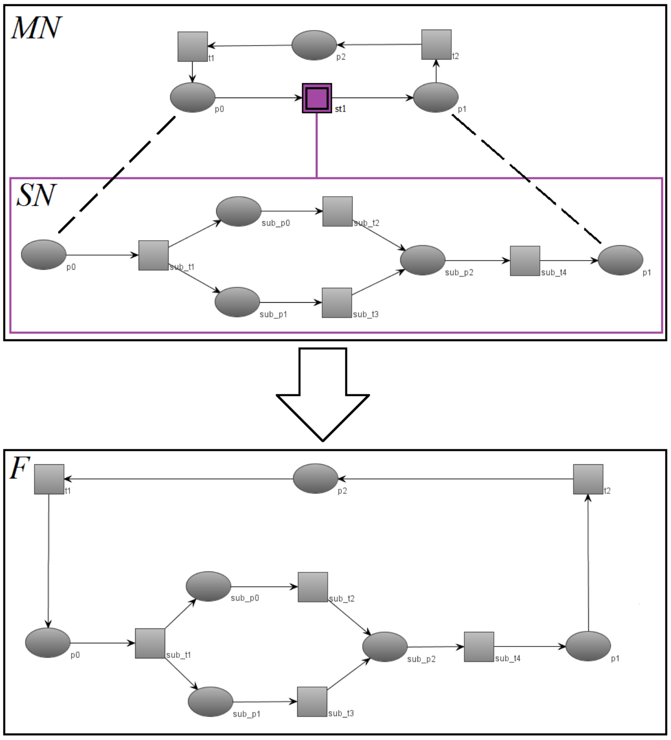


Fig. 8: Flattening of a substitution transition.

Figure 8 shows in the top half a hierarchical net with it's main net MN and a subnet SN. In the main net the substitution transition $st1$ has two connecting places: $p0$ has an edge connecting it to $st1$ and $st1$ has an edge connecting it to the place $p1$. These places can also be found in the subnet as connecting places with edges to and from different transitions. If tokens are added to the place $p0$ via the transition $t1$ these also appear in the subnet. There SNs transition sub_t1 can fire and remove tokens from $p0$ resulting in the removal of the same tokens from $p0$ in MN.

Subnets may contain substitution transitions containing further subnets resulting in a nested hierarchy. Reconfigurable Petri nets use transformation rules to reconfigure themselves. The chosen hierarchical model allows two different rule concepts: *global rules* and *local rules*.

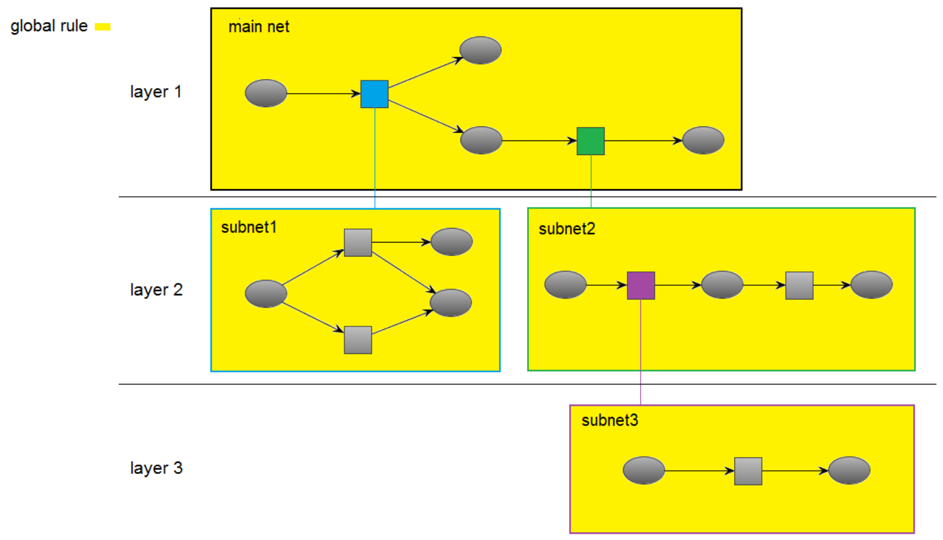


Fig. 9: Application of global rules to a hierarchical Petri net.

A global rule is a general rule that may be applied in any net on any level of the whole hierarchical net. So occurrences can be in the main net, its subnets, all their subnets and so forth.

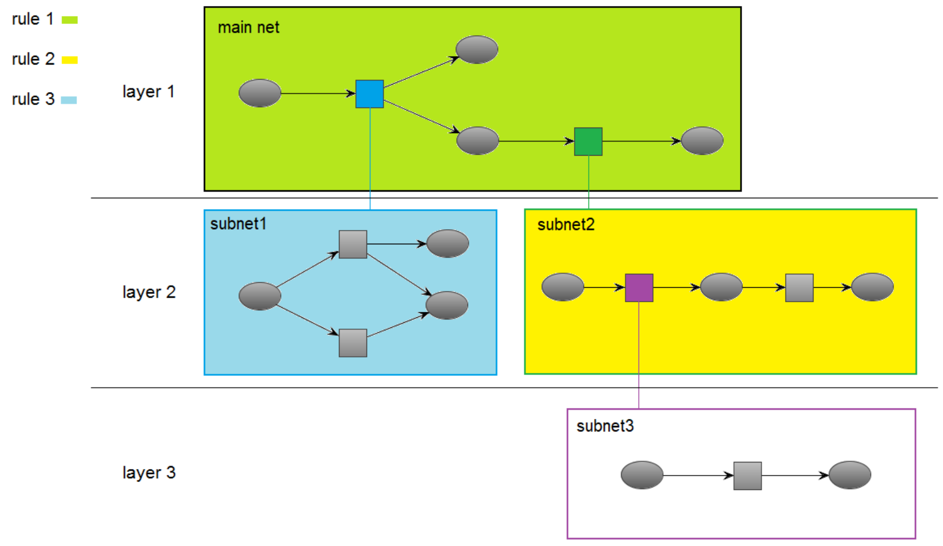


Fig. 10: Application of individual rules to a hierarchical Petri net.

A local rule that is applied to a (sub-)net allows occurrences to be found in that specific net only. Local rules enable the dynamic modification of a specific part of a hierarchical net. The advantage is that rules can be created without knowledge of other parts of the hierarchical net. Local rules are given for one subnet only, whereas global rules belong to the hierarchical net. They can be applied in all subnets since their labels are greater than the labels in the subnet. For details see Subsection 2.2 in [Pa18]. The name space is given by the disjoint union of all local name spaces, so that local rules can be applied only with in the given subnet. Local rules respect the hierarchical net borders so that no transformation may effect more than one (sub-)net. Hence, one restriction is imposed on the rules: Substitution transitions may not be part of a rule. As a consequence connecting places may not be deleted or added by a rule, but they can be part of one. This is due to the gluing condition which requires that places may only be deleted if the adjacent transition is deleted as well and no tokens are left over. Since connection places are neighbours of substitution transitions that cannot occur in a rule, they remain unchanged.

Subsequently, we define substitution os transition by subnets formally. Figure 11 shows an example for a very basic substitution rule. Substitution transition are mapped to such rules given by the mapping $subst : sT \longrightarrow SR^N$ where SR^N is a set of substitution rules. The definition of the reconfigurable hierarchical Petri net requires the substitution transition

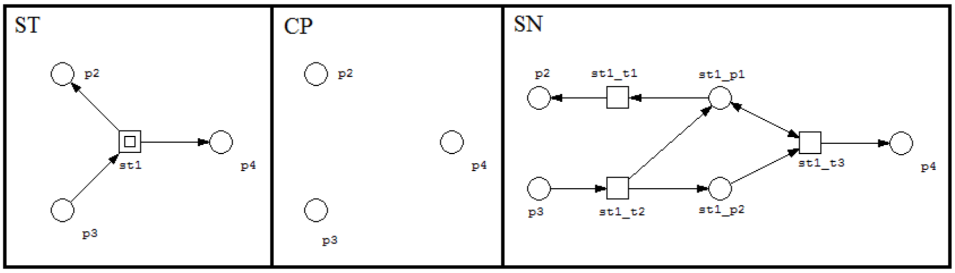


Fig. 11: An exemplary basic substitution rule.

together with its adjacent places, called net $Net(t)$ of a transition t.

Definition 5.1 **($Net(t)$)** *Given $N = (P, T, pre, post, pl, tl)$ then for a transition $t \in T$ the net of t is the net $Net(t) = ({}^\bullet t \cup t^\bullet, \{t\}, pre_{|t}, post_{|t}, pl_{|{}^\bullet t \cup t^\bullet}, tl_{|\{t\}})$.*

With this, reconfigurable hierarchical Petri nets can be formally defined. A reconfigurable hierarchical Petri net consists of a reconfigurable net with substitutions and a set of global rules. They both are defined over a common name space$A = (A_P, A_T)$The reconfigurable net with substitutions consist of a place/transition net N, a set of local rules $\mathcal{R}^N$ and a set of substitution rules SR^N together with the mapping of substitution transitions to substitution rules $subst : sT \longrightarrow SR^N$.

Definition 5.2 (Hierarchical reconfigurable Petri net) *A hierarchical reconfigurable Petri net $HN = (RN, A, GR)$ is given by its name space $A = (A_P, A_T)$, a set of global rules GR over $A = (A_P, A_T)$, and a reconfigurable net with substitutions $RN = (N, \mathcal{R}^N, SR^N)$:*

1. *$N = (P, T, pre, post, pl, tl, M)$ is a place/transition net over (A_P^N, A_T^N) so that,*
   - *P is a set of places.*
   - *T is a set of transitions.*
   - *T contains substitution transitions $sT \subseteq T$.*
   - *$pre : T \rightarrow P^{\oplus}$ is a function used for the pre-domain of each transition.*
   - *$post : T \rightarrow P^{\oplus}$ is a function used for the post-domains of each transition.*
   - *$tl : T \rightarrow A_T^N$ is a naming function for transitions, where substitution transitions have their own name space $A_{sT} \subseteq A_T^N$ so that $tl(sT) \subseteq A_{sT}$ and injective $tl_{|sT}$. Moreover, regular transitions are not allowed to use that name space. $tl(T \backslash sT) \subseteq A_T^N \backslash A_{sT}$.*
   - *$pl : P \rightarrow A_P^N$ is a naming function for places, where the set of connecting places $cP = \{\bullet t \cup t \bullet \mid t \in sT\} \subseteq P$is given by the neighbourhood of the substitution transitions and the name space of the connecting places $A_{cP} \subseteq A_P^N$ satisfies $pl(cP) \subseteq A_P^N$.*
     Moreover, regular places are not allowed to use that name space. $tl(P \backslash cP) \subseteq A_P^N \backslash A_{cP}$.
   - *M is a set of tokens with$M \in P^{\oplus}$.*
2. *$\mathcal{R}^N$ is a set of local rules over $(A_P^N, A_T^N \setminus A_{sT})$.*
3. *SR^N is a set of substitution rules together with a mapping of substitution transitions to substitution rules $subst : sT \rightarrow SR^N$ so that $subst(t) = (Net(t) \leftarrow CP(t) \rightarrow SN^t)$ with*
   - *the interface $CP(t) = ({}^{\bullet}t \cup t^{\bullet}, \emptyset, \emptyset, \emptyset, pl_{|{}^{\bullet}t \cup t^{\bullet}}, \emptyset)$ consisting of connecting places only.*
   - *a reconfigurable net with substitutions $SN^t = (RN^t, \mathcal{R}^t, SR^t)$ over $A^t = (A_P^t, A_T^t)$ with $A_{cP} \subseteq A_P^t$.*

Chapter 5 in [JK09] states that the flattening of a hierarchical net that uses substitution transitions must remove each substitution transition and insert its subnet into the supernet by fusing the connecting places. This process corresponds to applying the substitution rules from Def. 5.2. Only one substitution for each substitution transition is applicable to RN.

Corollary 5.3 (Set of substitutions S^N) *Given a reconfigurable net with substitutions $RN = (N, \mathcal{R}^N, SR^N)$. For every substitution transition $t \in sT$ and its substitution rule $sr = subst(t)$ there exists exactly one injective occurrence o of sr. These substitutions are* *collected in a set of substitutions* $S^N = \{(sr, o) \mid sr = subst(t)$ *and injective* $o : Net(t) \hookrightarrow RN\}$.

Due to the global and local rules, flattening construction is more complex than for a normal hierarchical Petri net. Flattening of a normal hierarchical Petri net looses all information of the hierarchical borders, but this information is needed for the correct application of local and global rules in the flattened net. Hence, we use the disjoint union of the subnet's name spaces.

First we investigate the parallel independence [Eh06] of the substitution rules. If any two $s_1, s_2 \in S$ with $s_1 \neq s_2$ are pairwise parallel independent the Parallelism Theorem states that they are also sequentially independent [Eh06]. All substitution rules sr together with their occurrences are independent from each other if any two sr_1, sr_2 with $sr_1 \neq sr_2$ are pairwise independent. So the proof of parallel independence of two arbitrary substitutions $s_1, s_2 \in S$ is sufficient to prove Lemma 5.4.

Lemma 5.4 (Pairwise Independence of Substitutions) *Given a reconfigurable net with substitutions* $RN = (N, \mathcal{R}^N, SR^N)$. *Any two substitutions* $s_1, s_2 \in S^N$ *with* $s_1 \neq s_2$ *are pair-wise independent.*

Proof:
We show for two arbitrary substitutions $s_1 \neq s_2$ the set theoretic representation of parallel independence $o_1(ST_1) \cap o_2(ST_2) \subseteq o_1(l_1(CP_1)) \cap o_2(l_2(CP_2))$ holds.

$$\begin{tikzcd}
SN^{t_1} \arrow[d, "n_1"] & CP(t_1) \arrow[l, "r_1"'] \arrow[r, "l_1"] \arrow[d, "k_1"] & Net(t_1) \arrow[dr, "o_1"] \arrow[drrr, dashed, "g_1"] & & Net(t_2) \arrow[dl, "o_2"'] \arrow[dlll, dashed, "g_2"'] & CP(t_2) \arrow[l, "r_2"'] \arrow[r, "l_2"] \arrow[d, "k_2"] & SN^{t_2} \arrow[d, "n_2"] \\
H_1 & D_1 \arrow[l, "r'_1"'] \arrow[rr, "l'_1"] & & G & & D_2 \arrow[ll, "r'_2"'] \arrow[r, "l'_2"] & H_2
\end{tikzcd}$$

The left-hand side of any rule rs of $(rs, o) \in S^N$ contains by Def. 5.2 only a net $Net(t)$. As specified in Def. 5.1 $Net(t)$ contains only a substitution transition t and t's pre- and post-domains. The interface $CP(t)$ contains only t's pre- and post-domains. Considering two substitutions $s_1, s_2 \in S^N$ with $s_1 \neq s_2$, the intersection between their occurrences only considering transitions must be empty because otherwise $t_1 = t_2$ and thus $s1 = s2$. Since $CP(t_1)$ only contains places and since $Net(t_1)$ contains one distinct transition t_1 and $Net(t_2)$ the another one t_2, it follows:

$$o_{1T}(Net(t_1)) \cap o_{2T}(Net(t_2)) = \emptyset \subseteq \emptyset = o_{1T}(l_1(CP(t_1))) \cap o_{2T}(l_2(CP(t_2)))$$

Now we consider the places. Let $p \in o_{1P}(Net(t_1)) \cap o_{2P}(Net(t_2))$.
Hence $p \in ({}^\bullet t_1 \cup t_1^\bullet) \cap ({}^\bullet t_2 \cup t_2^\bullet)$ that is $p \in CP(t_1) \cap CP(t_2)$ by definition of CP.
Since l_1, l_2, o_{1P} and o_{2P} are functions we conclude that $p \in (l_1(CP(t_1))) \cap (l_2(CP(t_2)))$ and $p \in o_{1P}(l_1(CP(t_1))) \cap o_{2P}(l_2(CP(t_2)))$. Thus:
$o_{1P}(Net(t_1)) \cap o_{2P}(Net(t_2)) \subseteq o_{1P}(l_1(CP(t_1))) \cap o_{2P}(l_2(CP(t_2)))$ which proves any two $s_1, s_2 \in S$ with $s_1 \neq s_2$ are pairwise parallel independent. $\square$

Theorem 5.5 is proven using Lemma 5.4.

Theorem 5.5 ($FLAT(N, SR^N)$ **Flattening of reconfigurable net with substitutions)** *Given a reconfigurable net with substitutions (N, SR^N) any possible transformation sequence of rules S^N yields the same (up to isomorphism) well-defined net $N \overset{S^N}{\Longrightarrow} FLAT(N, SR^N)$.*

Proof:
With all $s \in S^N$ being mutually independent, [Eh06] states all the transformation sequences $HN \overset{*}{\Rightarrow} F$ are equivalent and there exists a parallel transformation sequence $HN \xRightarrow{\sum_{s \in S^N} s} F$. Let $FLAT(N, SR^N) := F$. Such a parallel transformation sequence can always be constructed and is unique up to isomorphism. □

The flattening of a hierarchical reconfigurable Petri net to a reconfigurable net needs to include global as well as local rules and is given recursively based on flattening of nets with substitution.

Definition 5.6 (Flattening) *The flattening is defined for a hierarchical net $HN = (RN, A, GR)$ given by a reconfigurable net $RN = (N, \mathcal{R}^N, SR^N)$, a name space $A = (A_P, A_T)$ and a set of global rules GR as given in Def. 5.2 recursively by:*

1. *Given $RN = (N, \mathcal{R}^N, SR^N)$ over $A = (A_P^N, A_T^N)$ with substitution transitions $sT = \emptyset$ we have:*
$$flat(RN) = (N, \mathcal{R}^N) \text{ over } A$$
2. *Given $RN = (N, \mathcal{R}^N, SR^N)$ over $A = (A_P^N, A_T^N)$ with substitution transitions $sT \neq \emptyset$ we have:*
$$flat(RN) = flat(FLAT(N, SR^N), \overline{\mathcal{R}}, \overline{SR}) \text{ over } \overline{A} \text{ with}$$
   - $\overline{A} = \biguplus_{t \in sT}(A_P^t \setminus A_{cp}) \uplus A_{cP}$
   - $\overline{\mathcal{R}} = (\biguplus_{t \in sT} \mathcal{R}_t) \uplus \mathcal{R}^N$
   - $\overline{SR} = (\biguplus_{t \in sT} SR^t$
3. *$flat(HN) = (N_{Flat}, GR \cup \mathcal{R}_{Flat})$ over $\mathfrak{A}$ with $flat(RN) = (N_{Flat}, \mathcal{R}_{Flat})$ where the name space $\mathfrak{A}$ is the union of the name spaces, so that the global labels are greater than the corresponding local labels (see [Pa18]).*

Definition 5.7 (Well-defined hierarchical reconfigurable Petri net) *A hierarchical reconfigurable Petri net $HN = (RN, A, GR)$ is well-defined if and only if the $flat(HN)$ is well-defined.*

6 Hierarchies in ReConNet

During the simulation ReConNet's simulation engine uses the flat representation of a hierarchical reconfigurable Petri net for transition firing and rule application, because this allows using ReConNet's simulation engine to handle the hierarchical net, i.e its flattened net. However, for the user this remains transparent and the visual interface continues presenting the hierarchical view. While transitions are fired and transformations are made on the flat net the hierarchical view visualizes the changes accordingly. While developing the nets and transformation rules, the hierarchy is established and the simulation requires the flattened net that is the result of the flattening construction.

The application of local rules in the flat net needs one single name space for places and transitions $A = (A_P^N, A_T^N)$. This name space needs to include all of the disjoint name spaces of the (sub-)nets. This single name space is created during the flattening. Whenever a subnet is inserted into its supernet all places and transitions that are not connecting places get a prefix to their names that is unique to the substitution transition that was replaced. This way the naming preserves hierarchy borders and (sub-)net identities and so the names of places and transitions are specific enough that a rule meant for only a specific (sub-)net can be limited to the correct part of the flat net. For persistence of a hierarchical reconfigurable Petri net from ReConNet the hierarchical reconfigurable Petri net's flat net and the substitution rules. The hierarchical reconfigurable Petri net is saved as a tuple of the main net, as a reconfigurable Petri net, its substitution rules and the flat net, so that $HN = \langle RN, SR, Flat(RN, SR)\rangle$. So, the flat net can be loaded directly and needs not to be computed each time again.

In a transformation unit $N_1 \overset{R!}{\Longrightarrow} N_2$ [KKR08] the *as-long-as-possible* operator ! forces the rule set R to be applied as long as possible. Since ReConNet is provided with transformation units [Ho16] the flattening construction can be realized using this *as-long-as-possible* operator !. In the transformation unit $RN \overset{S^N!}{\Longrightarrow} F$ an applicable substitution rule s with an injective occurrence is randomly picked and applied. This step is repeated until all $s \in S^N$ have been applied, since each s has by construction only one occurrence.

Lemma 6.1 (Flattening based on the *as-long-as-possible* operator !) *Given a reconfigurable net with substitutions (N, SR^N) the result F of $(N, SR^N) \overset{sr!}{\Longrightarrow} F$ is well-defined up to isomorphism and isomorphic to $FLAT(N, SR^N)$.*

Proof:
Given the set of substitutions S^n based on (N, SR^N) as in Cor. 5.3, any two substitutions $s_1, s_2 \in S^N$ are pairwise independent, then the fact that F is well-defined up to isomorphism is proven indirectly:

If F is not well-defined the transformation sequences $(N, SR^N) \overset{S^N!}{\Longrightarrow} F$ and $(N, SR^N) \overset{S^N!}{\Longrightarrow}$

$\widehat{F}$ exists so that $F \not\cong \widehat{F}$. So, there has to exist some M in diagram **(1)** with $RN \overset{*}{\Rightarrow} M \overset{s_j}{\Longrightarrow} M_j$ and $RN \overset{*}{\Rightarrow} M \overset{s_i}{\Longrightarrow} M_i$ so that there is neither $M_i \overset{*}{\Rightarrow} F$ nor $M_j \overset{*}{\Rightarrow} F$:

$$\begin{tikzcd}
 & & M_j \arrow[dr, Rightarrow, "|" marking, "*"] & \\
RN \arrow[r, Rightarrow, "*"] & M \arrow[ur, Rightarrow, "s_j"] \arrow[dr, Rightarrow, "s_i"'] & \textbf{(1)} & F \\
 & & M_i \arrow[ur, Rightarrow, "|" marking, "*"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
 & & M_j \arrow[dr, Rightarrow, "|" marking, "s_i"] & & \\
RN \arrow[r, Rightarrow, "*"] & M \arrow[ur, Rightarrow, "s_j"] \arrow[dr, Rightarrow, "s_i"'] & \textbf{(2)} & M_{ij} \arrow[r, Rightarrow, "*"] & F \\
 & & M_i \arrow[ur, Rightarrow, "|" marking, "s_j"'] & &
\end{tikzcd}$$

For $RN \overset{\text{sr!}}{\Longrightarrow} F$ and $RN \overset{\text{sr!}}{\Longrightarrow} \widehat{F}$ both substitutions s_i and s_j have to be applied. All $s \in S$ are pairwise sequential independent, so any sequence can be applied in arbitrary order yielding the same well-defined resulting net [Eh06]. Hence, we obtain diagram **(2)**. Any two $s \in S$ are pairwise parallel independent, so are s_i and s_j, thus their sequence is interchangeable s_i is always applicable to M_j and s_j is always applicable to M_i both always leading to the same net M_{ij}. So $F \cong \widehat{F}$ for $HN \overset{\text{sr!}}{\Longrightarrow} F$ and $HN \overset{\text{sr!}}{\Longrightarrow} \widehat{F}$ and thus F is well-defined up to isomorphism. □

Since all $s \in S$ are independent from another, each $s \in S^N$ of the transformation $RN \overset{sr!}{\Longrightarrow} \widehat{F}$ can be applied exactly once for each of its occurrences. $RN \xRightarrow{\sum_{s \in S} s} F$ can equivalently be applied as a transformation sequence $RN \overset{*}{\Rightarrow} F = RN \overset{s_1}{\Longrightarrow} \ldots \overset{s_n}{\Longrightarrow} F$. So to prove that $F \cong \widehat{F}$ requires to show that no s can be applied more than once.

Theorem 6.2 (Equivalence of Flattening Constructions) *Given a reconfigurable net with substitutions (N, SR^N). The transformation using the as-long-as-possible operator ! is equivalent to the flattening construction in Thm. 5.5:*
$RN \xRightarrow{\sum_{s \in S^N} s} FLAT(N, SR^N)$ *and* $RN \overset{S^N!}{\Longrightarrow} F$ *implies* $FLAT(N, SR^N) \cong F$.

Proof:
For any $s \in S$ to be able to be applied more than once it would have to be independent from itself. $s_1 = s_2$ implies they contain the same substitution transition $t_1 = t_2$, it follows: $(o_{1T}(net(t_1)) \cap o_{2T}(net(t_2)) \neq \emptyset) \not\subseteq \emptyset = o_{1T}(l_1(CP_1)) \cap o_{2T}(l_2(CP_2))$. Thus s_1 and s_2 are not independent.
Hence any s is not independent from itself and thus can only be applied once. □

7 Conclusion

This paper introduces substitution transitions for hierarchical reconfigurable Petri nets. The main contribution is a formal definition of hierarchical reconfigurable Petri nets and its flattening construction.

This work presents a first step to the integration of reconfigurable hierarchical Petri nets into the ReConNet tool [Pa12, Re17]. Ongoing work will accomplish support of hierarchical Petri nets in ReConNet. First a hierarchy needs to be introduced into ReConNet to allow transformation simulation, including an appropriate update to ReConNet's persistence module to allow proper storing and restoring of hierarchical nets. Then individual rules will be added to allow the functionality of a reconfigurable net. For net verification and validation purposes the flat representation of the hierarchical reconfigurable Petri net will be used.

References

[Ak17] Akshay, S.; Chakraborty, Supratik; Das, Ankush; Jagannath, Vishal; Sandeep, Sai: On Petri Nets with Hierarchical Special Arcs. In: 28th International Conference on Concurrency Theory, CONCUR 2017, September 5-8, 2017, Berlin, Germany. pp. 40:1–40:17, 2017.

[BH01] Busatto, Giorgio; Hoffmann, Berthold: Comparing Notions of Hierarchical Graph Transformation. Electr. Notes Theor. Comput. Sci., 50(3):310–317, 2001.

[BKK05] Busatto, Giorgio; Kreowski, Hans-Jörg; Kuske, Sabine: Abstract hierarchical graph transformation. Mathematical Structures in Computer Science, 15(4):773–819, 2005.

[Bo06] Bottoni, Paolo; Rosa, Fabio De; Hoffmann, Kathrin; Mecella, Massimo: Applying Algebraic Approaches for Modeling Workflows and their Transformations in Mobile Networks. Mobile Information Systems, 2(1):51–76, 2006.

[Br14] Bruni, Roberto; Montanari, Ugo; Plotkin, Gordon D.; Terreni, Daniele: On Hierarchical Graphs: Reconciling Bigraphs, Gs-monoidal Theories and Gs-graphs. Fundam. Inform., 134(3-4):287–317, 2014.

[CP17] CPN-Tools. `http://http://cpntools.org/`, 2017. Last accessed: 2017-11-01.

[DHP02] Drewes, Frank; Hoffmann, Berthold; Plump, Detlef: Hierarchical Graph Transformation. J. Comput. Syst. Sci., 64(2):249–283, 2002.

[Eh06] Ehrig, H.artmut; Ehrig, Karsten; Prange, Ulrike; Taentzer, Gabriele: Fundamentals of Algebraic Graph Transformation. EATCS Monographs in TCS. Springer, 2006.

[GE12] Gabriel, Karsten; Ehrig, Hartmut: Modelling of Communication Platforms Using Algebraic High-Level Nets and Their Processes. In (Heisel, Maritta, ed.): Software Service and Application Engineering: Essays Dedicated to Bernd Krämer on the Occasion of His 65th Birthday. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 10–25, 2012.

[GNH12] Gottmann, Susann; Nachtigall, Nico; Hoffmann, Kathrin: On Modelling Communication in Ubiquitous Computing Systems using Algebraic Higher Order Nets. ECEASST, 51, 2012.

[He12] Heiner, Monika; Herajy, Mostafa; Liu, Fei; Rohr, Christian; Schwarick, Martin: Snoopy–a unifying Petri net tool. In: International Conference on Application and Theory of Petri Nets and Concurrency. Springer, pp. 398–407, 2012.

[HEP08] Hoffmann, Kathrin; Ehrig, Hartmut; Padberg, Julia: Flexible Modeling of Emergency Scenarios using Reconfigurable Systems. ECEASST, 12, 2008.

[Hi17] HiPS : Hierarchical Petri net Simulator. `https://sourceforge.net/projects/hips-tools/`, 2017. Last accessed:2017.03.22.

[HJS91] Huber, Peter; Jensen, Kurt; Shapiro, Robert M.: Hierarchies in coloured Petri nets. In (Rozenberg, Grzegorz, ed.): Advances in Petri Nets 1990. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 313–341, 1991.

[Ho16] Hoff, Christian: Transformationseinheiten als Kontrollstruktur für rekonfigurierbare Petrinetze in ReConNet. Master's thesis, University of Applied Sciences Hamburg, 2016.

[HW18] Harie, Yojiro; Wasaki, Katsumi: A Petri Net Design and Verification Platform Based on The Scalable and Parallel Architecture: HiPS. In: Information Technology-New Generations, pp. 265–273. Springer, 2018.

[JK09] Jensen, Kurt; Kristensen, Lars M: Coloured Petri nets: modelling and validation of concurrent systems. Springer, 2009.

[JR91] Jensen, Kurt; Rosenberg, Gregorz: High-Level Petri nets: Theory and Application. 1991. Springer, 1991.

[KBD16] Kahloul, Laid; Bourekkache, Samir; Djouani, Karim: Designing reconfigurable manufacturing systems using reconfigurable object Petri nets. Int. Journal of Computer Integrated Manufacturing, pp. 1–18, 2016.

[KH10] Köhler-Bußmeier, Michael; Heitmann, Frank: Safeness for Object Nets. Fundam. Inform., 101(1-2):29–43, 2010.

[KH11] Köhler-Bußmeier, Michael; Heitmann, Frank: Liveness of Safe Object Nets. Fundam. Inform., 112(1):73–87, 2011.

[KKR08] Kreowski, Hans-Jorg; Kuske, Sabine; Rozenberg, Grzegorz: Graph Transformation Units– An Overview. Lecture Notes in Computer Science, 5065:57–75, 2008.

[La01] Lakos, Charles: Object Oriented Modeling with Object Petri Nets. In (Agha, Gul; de Cindio, Fiorella; Rozenberg, Grzegorz, eds): Concurrent Object-Oriented Programming and Petri Nets, Advances in Petri Nets. Lecture Notes in Computer Science. Springer, pp. 1–37, 2001.

[LO04] Llorens, Marisa; Oliver, Javier: Structural and Dynamic Changes in Concurrent Systems: Reconfigurable Petri Nets. IEEE Trans. Computers, 53(9):1147–1158, 2004.

[MGH10] Modica, Tony; Gabriel, Karsten; Hoffmann, Kathrin: Formalization of Petri Nets with Individual Tokens as Basis for DPO Net Transformations. ECEASST, 40, 2010.

[MK05] Miyamoto, Toshiyuki; Kumagai, Sadatoshi: A survey of object-oriented Petri nets and analysis methods. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, 88(11):2964–2971, 2005.

[Pa04] Palacz, Wojciech: Algebraic hierarchical graph transformation. J. Comput. Syst. Sci., 68(3):497–520, 2004.

[Pa07] Padberg, Julia; Hoffmann, Kathrin; Ehrig, Hartmut; Modica, Tony; Biermann, Enrico; Ermel, Claudia: Maintaining Consistency in Layered Architectures of Mobile Ad-Hoc Networks. In (Dwyer, Matthew B.; Lopes, Antónia, eds): Fundamental Approaches to Software Engineering 2007. volume 4422 of Lecture Notes in Computer Science. Springer, pp. 383–397, 2007.

[Pa12] Padberg, Julia; Ede, Marvin; Oelker, Gerhard; Hoffmann, Kathrin: ReConNet: A Tool for Modeling and Simulating with Reconfigurable Place/Transition Nets. ECEASST, 54, 2012.

[Pa15] Padberg, Julia: Reconfigurable Petri Nets with Transition Priorities and Inhibitor Arcs. In (Parisi-Presicce, Francesco; Westfechtel, Bernhard, eds): Graph Transformation - 8th Int. Conf. volume 9151 of Lecture Notes in Computer Science. Springer, pp. 104–120, 2015.

[Pa18] Padberg, Julia: Subtyping for Hierarchical, Reconfigurable Petri Nets. ArXiv e-prints, 2018.

[PK18] Padberg, Julia; Kahloul, Laid: Overview of Reconfigurable Petri Nets. In: Festschrift in Memory of Hartmut Ehrig. Springer, 2018. accepted.

[Ra03] Ratzer, Anne Vinter; Wells, Lisa; Lassen, Henry Michael; Laursen, Mads; Qvortrup, Jacob Frank; Stissing, Martin Stig; Westergaard, Michael; Christensen, Søren; Jensen, Kurt: CPN tools for editing, simulating, and analysing coloured Petri nets. In: International Conference on Application and Theory of Petri Nets. Springer, pp. 450–462, 2003.

[Re17] ReConNet. `https://reconnetblog.wordpress.com/`, 2017. Last accessed: 2017.05.16.

[Tâ12] Târnauca, Bogdan; Puiu, Dan; Comnac, Vasile; Suciu, Constantin: Modelling a flexible manufacturing system using reconfigurable finite capacity Petri nets. In: 13th Int. Conf. on Optimization of Electrical and Electronic Equipment. pp. 1079–1084, May 2012.

[Va98] Valk, Rüdiger: Petri Nets as Token Objects: An Introduction to Elementary Object Nets. In (Desel, Jörg; Suárez, Manuel Silva, eds): Application and Theory of Petri Nets 1998, 19th International Conference, ICATPN ’98, Lisbon, Portugal, June 22-26, 1998, Proceedings. volume 1420 of Lecture Notes in Computer Science. Springer, pp. 1–25, 1998.

[Va04] Valk, Rüdiger: Object Petri nets. In: Lectures on Concurrency and Petri Nets, pp. 819–848. Springer, 2004.