Electronic Communications of the EASST
Volume 40 (2011)

Proceedings of the
4th International Workshop on Petri Nets and Graph Transformation
(PNGT 2010)

RONs Revisited: General Approach to Model Reconfigurable Object Nets based on Algebraic High-Level Nets

Claudia Ermel, Sarkaft Shareef and Winzent Fischer

19 pages

Guest Editors: Claudia Ermel, Kathrin Hoffmann
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

RONs Revisited: General Approach to Model Reconfigurable Object Nets based on Algebraic High-Level Nets

Claudia Ermel, Sarkaft Shareef and Winzent Fischer

Institut für Softwaretechnik und Theoretische Informatik
Technische Universität Berlin, Germany
winzent.fischer@berlin.de, sarkaft@cs.tu-berlin.de, claudia.ermel@tu-berlin.de

Abstract: Reconfigurable Object Nets (RONs) have been implemented in our group to support the visual specification of controlled rule-based transformations of marked place/transition (P/T) nets. RONs are high-level nets (system nets) with two types of tokens: object nets (P/T nets) and net transformation rules. System net transitions can be of different types to fire object net transitions, move object nets through the system net, or to apply a net transformation rule to an object net. The disadvantage of the RON approach and tool is the limitation of object nets to P/T nets and the limitation of the underlying semantics of RONs due to the fixed types for system net transitions. Often, a more general approach is preferred where the type of object nets and the behavior of reconfigurations may be defined in a more flexible way. In this paper, we propose to use Algebraic High-Level nets with individual tokens (AHLI nets) as system nets. In this more general approach, tokens may be any type of Petri nets, defined by the corresponding algebraic signature and algebra. To support this general approach, a development environment for AHLI nets is currently implemented which allows the user to edit and simulate AHLI nets. We present the formalization of RONs as special AHLI nets and describe the current state of the AHLI net tool environment.

Keywords: algebraic high-level net, reconfigurable Petri net, graph transformation, tool environment, RON

1 Introduction

Reconfigurable systems may be reconfigured at run-time to adapt to a changing environment. Application areas cover e.g. computer-supported cooperative work, multi-agent systems, dynamic process mining or mobile networks. One approach to combine formal modeling of dynamic systems and controlled model reconfiguration are reconfigurable Petri nets. The main idea is the stepwise development of place/transition nets by applying net transformation rules [EHP+08, PEHP08]. Think of these rules as replacement systems where the left-hand side is replaced by the right-hand side while preserving a context. This approach is more expressive than immutable Petri nets since it allows in addition to the well known token game a formal description and analysis of structural changes of nets.

Recently, the RON tool [RON11, BEHM07] (*Reconfigurable Object Nets*), has been developed to model, simulate and analyze reconfigurable Petri nets. Simulating a RON, not only P/T net transitions can be fired but also the net structure can be changed with respect to some new requirements of the environment by applying net transformation rules. In order to control the firing and reconfiguration of P/T nets, a high-level net (system net) is defined, where P/T nets (object nets) and P/T net transformation rules are used as tokens. A special type of system net transition applies a transformation rule to object nets on its pre-domain places and puts the resulting object net after the net transformation step to the system net places in its post-domain. Reconfiguration steps and object net firing steps may be interleaved. With the RON tool, it can be analyzed whether firing steps and rule applications are in conflict.

The disadvantage of the RON approach and tool is the limitation of object nets to P/T nets and the limitation of the underlying semantics of RONs due to four fixed types of system net transitions. Often, a more general approach is preferred where the type of object nets and the behavior of reconfigurations can be defined in a more flexible way.

In our approach, we propose to use Algebraic High-Level nets with individual tokens (AHLI nets) as system nets which in turn use tokens containing flexible data. In this way, different kinds of objects used as tokens in the system net can be defined by suitable algebraic signatures and algebras, e.g. for simple datatypes, different kinds of Petri nets, and net transformation rules. Even AHLI nets themselves might be used as object nets which leads to a hierarchical nesting of nets using nets as tokens, and allows us to model reconfigurable high-level nets [BEE+09]. In a previous paper [HME05], a signature for so-called algebraic higher-order nets was defined with the aim to define AHL nets with nets and rules as tokens. This approach was based on classical P/T nets, where rules could not be defined that were able to change markings since rules had to be strict on the marking. This was one motivation for us to use the definitions of AHL nets and P/T nets with individual tokens (AHLI nets and PTI nets), where marking-changing rules can be formulated as well. As main contribution of this paper, we formalize the behavior of RONs by defining a special kind of AHLI nets with a signature for P/T nets and P/T net transformation rules. Furthermore, we sketch a tool for AHLI nets that allows the user to define algebraic signatures and algebras, to model AHLI nets in a visual editor and to simulate their firing behavior.

The paper is structured as follows: Section 2 reviews the RON tool and sketches its limitations. In Section 3, we formalize RONs by defining datatypes for P/T nets and P/T net transformation rules which are used as tokens in our special kind of AHLI nets, called AHOI nets (Algebraic Higher-Order nets with Individual tokens). Section 3.2 presents the current state of the AHLI net tool. In Section 6, we conclude the paper and give directions for future work.

2 The RON Tool Environment for Reconfigurable Object Nets

The RON-tool [RON11, BEHM07] is a visual modeling environment supporting modelers to edit, simulate and analyze reconfigurable P/T nets. The RON-tool has been used to model case studies where subnets show a flexible behavior which cannot be foreseen from the beginning. Examples are a distributed producer-consumer system, where disjoint producer and consumer nets can be connected at run-time to model trading and can be separated again after their deals have been finished [BM08] (see Figure 1), and an emergency scenario, where a leaking gas pipeline is fixed by fire workers, and their workflows have to be adapted to changes in the environment (making e.g. evacuations necessary) [Tro09].

RONs are high-level nets (system nets) with two types of places, i.e. Object net places which carry object nets as tokens, and Rule places which are marked by net transformation rules. System net transitions can be of type FIRE, STANDARD, SPLIT or APPLYRULE. Firing of system net transitions thus either trigger the firing of an object net transition (type FIRE), or transport object net tokens through the system net (type STANDARD), or apply a net transformation rule to an object net (type APPLYRULE), or separate a single object net into its unconnected components (type SPLIT).

A visual editor and simulator for RONs has been developed as a plug-in for ECLIPSE using the ECLIPSE Modeling Framework (EMF) and Graphical Editor Framework (GEF) plug-ins [BEHM07]. A screenshot of the RON tool is shown in Figure 1.

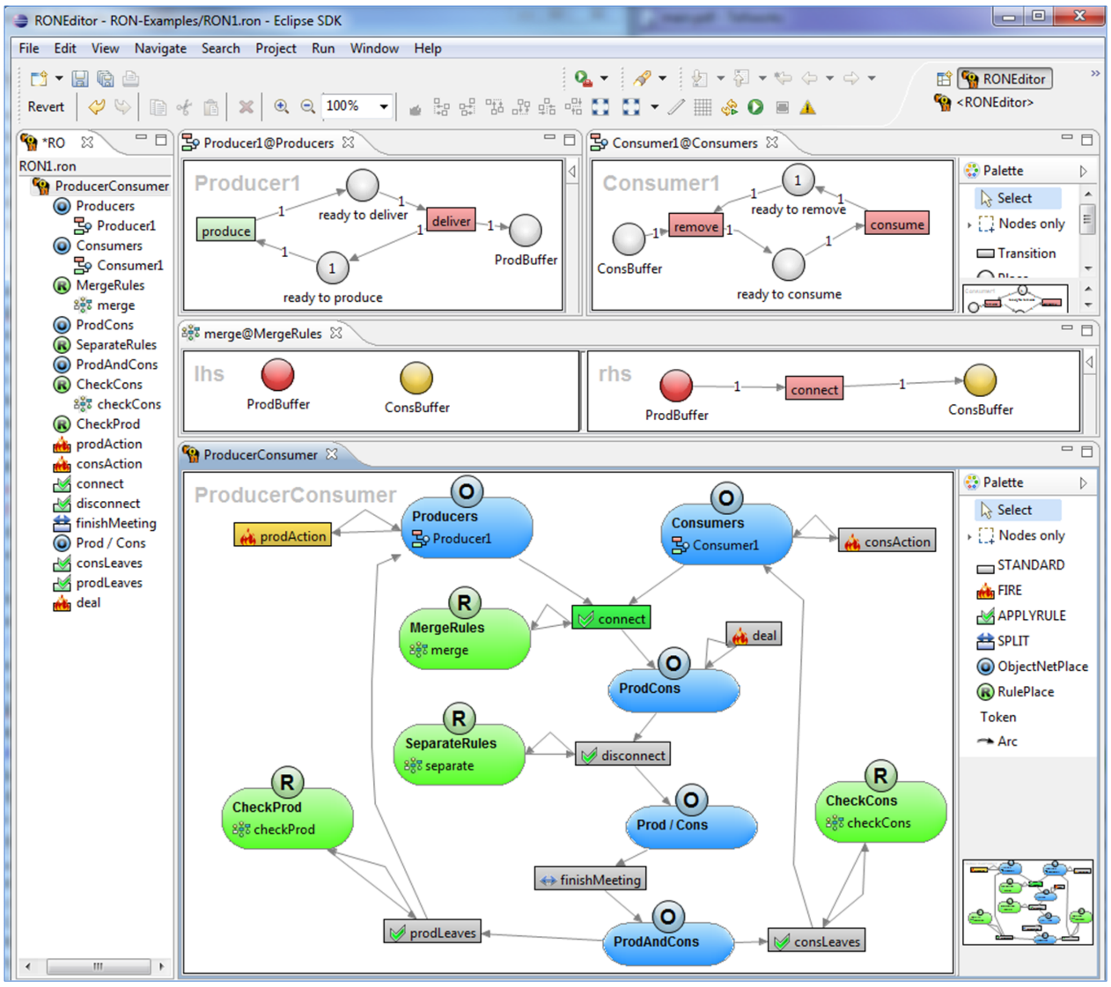


Figure 1: The RON environment for P/T Net Reconfiguration

Two object nets can be seen in the upper view, a net reconfiguration rule in the center, and the system net at the bottom, where the current object net names are written inside their corresponding object net places Producers and Consumers, and the reconfiguration rule merge is a token on the rule place MergeRules.

In our example, RONs are applied to model a distributed system of producers and consumers where several producers and consumers may interact with each other. In the initial state of the sample RON in Figure 1 potential producers and consumers are distributed on different Object net places as independent object nets without interaction. Producer object nets may fire, e.g. they can produce items and place them on the buffer place. Firing in object nets is triggered by firing a system net transition of type FIRE, which takes one object net with marking M from the Net place in its pre-domain and puts the same object net, now marked by one of the possible successor markings of M, into all of its post-domain places.

For producer-consumer interaction, a producer object net can be merged with a consumer object net by firing the APPLYRULE system transition *connect*. A transition of this type takes an object net from each of the pre-domain Net places, a rule from the pre-domain Rule place, applies this rule to the disjoint union of all the taken object nets and puts the resulting net to all post-domain Net places. The rule *merge*, depicted in the center of Figure 1, glues a producer object net and a consumer object net by inserting a *connect* transition between both buffers. Note that the system transition *connect* controls which producer interacts with which consumer.

By firing the FIRE transition *deal*, in the glued object net the consumer now can consume items produced by the producer as long as there are tokens on the place *Prod-Buffer*. Moreover, the producer may also produce more items and put them to the buffer. After the deal has been finished, the nets are separated again by firing the APPLYRULE system transition *disconnect*. This applies a rule *separate* (which is the inverse rule of rule *merge* in Figure 1 whith LHS and RHS exchanged) to the glued net, thus deleting the *connect* transition from the net. Note that the resulting net, which is put on place *Prod/Cons*, is still one single object net which consists of two unconnected components. In order to split a net with two components into two object nets, a system transition of type SPLIT is used. Firing the SPLIT system transition finishMeeting results in two separate object nets on place *ProdAndCons*. In a last step, the now separated producer and consumer nets are returned to their initial places by firing the STANDARD system transitions *ProdLeaves* and *ConsLeaves*. STANDARD transitions simply remove a net token from each pre-domain place and add the disjoint union of all removed object nets to each of the post-domain Net places. In our example, the STANDARD transitions may also put the object nets back to the “wrong” initial place. It is possible to avoid such behaviour by using APPLYRULE transitions instead where rules check the object nets for the occurrence of a *producer* or *consumer* place, respectively.

Up to now, the RON tool is limited to P/T nets as object nets and to the four fixed types of system net transitions FIRE, STANDARD, SPLIT or APPLYRULE. The aim of this paper is to come up with a more general approach and tool where the type of token objects and the behavior of reconfigurations can be defined in a more flexible way.

3 Algebraic High-Level Nets with Individual Tokens

As formal basis of our more general approach to define RONs, we rely on AHLI nets (Algebraic High-Level nets with Individual tokens), which we define in Section 3.1. The current state of our

AHLI net tool, an ECLIPSE plug-in, is sketched in Section 3.2.

3.1 Algebraic High-Level Nets with Individual Tokens (AHLI Nets)

AHLI nets [MGE+10] are a new formal definition of Algebraic High-Level (AHL) nets [PER95], where in addition to classical AHL nets [PER95], tokens are defined as individual objects. The overall motivation for this is a closer relationship to graph transformation systems, with the aim to translate Petri net firing behavior and net reconfigurations to graph transformation rules for simulation and analysis. In [MEE11], it has been shown already that for P/T nets with individual tokens, a category can be defined in a way that a suitable functor maps P/T net transformation systems to graph transformation systems with equivalent behavior. It remains to show similar properties also for the category of AHLI nets.

In this section, we review the basic definitions for AHLI nets from[MGE+10].

Definition 1 (Algebraic High-Level Nets with Individual Tokens) We define a marked AHL net with individual tokens, short AHLI net, as $AN = (\Sigma, P, T, pre, post, cond, type, A, I, m)$, where

- $AN = (\Sigma, P, T, pre, post, cond, type, A)$ is a classical AHL net with
  - signature $\Sigma = (S, OP, X)$ of sorts S, operation symbols OP and variables $X = (X_s)_{s\in S}$,
  - sets of places P and transitions T,
  - $pre, post : T \to (T_{OP}(X) \otimes P)^{\oplus}$[1], defining the transitions' pre- and postdomains,
  - $cond : T \to \mathscr{P}_{fin}(Eqns(S, OP, X))$ assigning a finite set of Σ-equations (L, R, X) as firing conditions to each transition,
  - $type : P \to S$ typing the places of the signature's sorts,
  - a Σ-algebra A,
- I is the (possibly infinite) set of individual tokens of ANI, and
- $m : I \to A \otimes P$ is the marking function, assigning the individual tokens to the data elements on the places. $m(I)$ defines the actual set of data elements on the places of ANI. m does not have to be injective.

Remark 1 (Individual tokens vs. classical algebraic data tokens) *Each AHLI net with finite individual token marking (I, m) can be interpreted as a classical AHL net with marking*

$$M = \sum_{(a,p)\in A\otimes P} |m^{-1}(a, p)|(a, p) = \sum_{i\in I} m(i)$$

where $|m^{-1}(a, p)|$ denotes the cardinality of individual tokens in I that m maps to (a, p).

The main difference to classical AHL nets is that in AHLI nets, we can distinguish tokens of the same algebraic value on the same place. Moreover, the individuals equip data tokens with identities. When firing a transition, we now can relate the input and output tokens so that a token's history can be traced along the firing steps.

[1] $T_{OP}(X) \otimes P = \{(t, p) \in T_{OP}(X) \times P | t \in T_{OP,type(p)}(X)\}$, i.e the pairs where term t is of sort $type(p)$.

Example 1 *Figure 2 shows the graphical representation of an AHLI net with*

- *signature* $\Sigma = (\{s_1, s_2, s_3\}, \{t_{11} :\to s_1, t_{12} :\to s_1, t_2 :\to s_2, t_3 :\to s_3\})$,
- *algebra carrier sets* $A_{s_1} = \{a_1, b_1, c_1\}, A_{s_2} = \{a_2, b_2\}, A_{s_3} = \{a_3, b_3, c_1\}$
- $pre(t) = (t_{11}, p_1) \oplus (t_{12}, p_1) \oplus (t_2, p_2), post(t) = (t_3, p_3)$,
- $type(p_1) = s_1, type(p_2) = s_2, type(p_3) = s_3$,
- $cond(t) = \emptyset$,
- $I = \{x_1, y_1, x_2, x_3\}$, $m(x_1) = m(y_1) = (a_1, p_1), m(x_2) = (a_2, p_2), m(x_3) = (a_3, p_3)$,

Note that the algebraic value of an individual token is given next to the dashed arc to its carrying place. In the following, if a transition has an empty set of conditions we just denote the transition name without an explicit $\emptyset$ *below.*

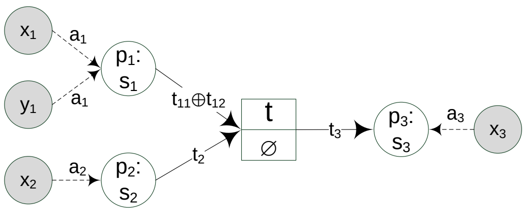


Figure 2: Example AHLI net

In Definition 2, we define firing steps in AHLI nets. We have to take into account assignments evaluating the variables at the arcs and in the transition conditions to check for enabled transitions. Therefore, we first introduce some additional notations:

- $Var(t) \subseteq X$ is the set of variables occurring in equations and on the environment arcs of t,
- $CP = (A \otimes P) = \{(a, p) \in A \times P | a \in A_{type(p)}\}$ as the set of consistent value/place pairs,
- $CT = \{(t, asg) \in T \times (Var(t) \to A) | \forall (L, R, X) \in cond(t) : \overline{asg}(L) = \overline{asg}(R)\}$ as the set of consistent transition assignments, i.e. all firing conditions of t are valid when evaluated with the variable assignment asg[2],
- $pre_A, post_A : CT \to CP^{\oplus}$, defined by

  $$pre_A(t, asg) = (\overline{asg} \otimes id_P)^{\oplus}(pre(t)),$$
  $$post_A(t, asg) = (\overline{asg} \otimes id_P)^{\oplus}(post(t))$$

  We can express e.g. the concrete required *number* of a token (a, p) for t to fire under assignment asg with $pre_A(t, asg)(a, p)$ by interpreting the monoid $pre_A(t, asg)$ as a function $CP \to \mathbb{N}$. Similarly, we get the produced *number* of (a, p) with $post_A(t, asg)(a, p)$.

Definition 2 (Firing Behavior of AHLI nets) A consistent transition assignment $(t, asg) \in CT$ for an AHLI net $ANI = (\Sigma, P, T, pre, post, cond, type, A, I, m)$ is *enabled* under a *token selection* (M, m, N, n), where $M \subseteq I$, m is the token mapping of *ANI*, N is a set with $(I \setminus M) \cap N = \emptyset$, and $n : N \to A \otimes P$ is a function, if it meets the following *token selection condition*:

$$\sum_{i \in M} m(i) = pre_A(t, asg) \wedge \sum_{i \in N} n(i) = post_A(t, asg)$$

[2] where $\overline{asg} : T_{OP}(X) \to A$ is the evaluation of Σ-terms over variables in X to values in A. Technically, $\overline{asg} = xeval(asg)_A$ results from a free construction over asg.

If such an *asg*-enabled t fires, the successor marking (I', m') is given by

$$I' = (I \setminus M) \cup N, \quad m' : I' \to A \otimes P \text{ with } m'(x) = \begin{cases} m(x), & x \in I \setminus M \\ n(x), & x \in N \end{cases}$$

leading to $ANI' = (AN, I', m')$ as the new AHLI net in the *firing step* $ANI \xrightarrow{t,asg} ANI'$ via (M, m, N, n).

Remark 2 (Token Selection) *The purpose of the token selection is to specify exactly which tokens should be consumed and produced in the firing step. Thus, $M \subseteq I$ selects the individual tokens to be consumed, and N contains the set of individual tokens to be produced*[3]*. Functions m and n relate the tokens to their place/value pairs. It would be sufficient to consider only the restriction $m_{|M}$ here or to infer it from the net but as a compromise on symmetry and readability we denote m in the token selection.*

In [MGE$^+$10], also AHLI net morphisms are defined which extend classical AHL net morphisms to individual tokens. It is shown that AHLI nets and AHLI net morphisms form a category **AHLINets**. Moreover, **AHLINets**, together with a set of AHLI net transformation rules according to the double pushout approach are shown in [MGE$^+$10] to be $\mathscr{M}$-adhesive transformation systems [EGH10], i.e. important results like the Local Church-Rosser and the Parallelism Theorem hold for AHLI net transformations.

3.2 The AHLI Net Tool

Tool support for AHLI nets has been implemented in our group[4] which allows the user to model and simulate AHLI nets in a visual editor.

Based on the Eclipse Modeling Framework [EMF11], the AHLI net environment comprises components to define algebraic signatures Σ and Σ-algebras, as well as a visual editor for AHLI nets which also serves as simulation viewer.

Creating Algebraic Signatures and Algebras

The main challenge while implementing an AHLI net editor except for the graphical editor itself is to provide a user-friendly way of creating an algebraic signature along with its algebras. This includes the evaluation of terms over the signature in a given algebra.

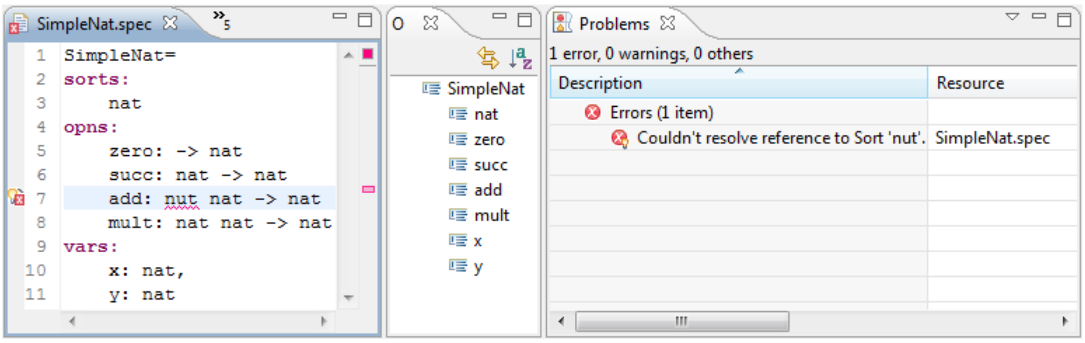
To provide a user-friendly editor for algebraic signatures, the XTEXT framework [Xte10] was used: from a textual grammar of algebraic signatures defined in EBNF, XTEXT generates a textual editor along with an EMF model for algebraic signatures[5]. The editor generated by XTEXT has been extended by additional validation and formatting features. The validation feature enhances the checks performed by the editor for syntactical correctness of the algebraic signature,

[3] We require that $(I \setminus M) \cap N = \emptyset$ must hold because it is impossible to add an individual token to a net that already contains this token.

[4] Available at the TFS update site http://tfs.cs.tu-berlin.de/software, packages agg, ahli, alspec and muvitor

[5] The grammar even defines algebraic specifications, i.e. signatures together with equations. Since AHLI nets are signature-based, the equation part may be used by the language designers for documentation purposes, but satisfaction of equations by terms is not checked by the tool.

whereas the formatting feature provides the layout of the notation as it is used in the literature (e.g. [EM85]). Figure 3 shows the editor for algebraic signatures where a simple signature SimpleNat for natural numbers has been edited. The figure also shows an example for an error message which has been issued due to a syntax error of the specified signature.



```
SimpleNat=
sorts:
    nat
opns:
    zero: -> nat
    succ: nat -> nat
    add: nut nat -> nat
    mult: nat nat -> nat
vars:
    x: nat,
    y: nat
```

SimpleNat
- nat
- zero
- succ
- add
- mult
- x
- y

1 error, 0 warnings, 0 others

Description	Resource
Errors (1 item)	
Couldn't resolve reference to Sort 'nut'.	SimpleNat.spec

Figure 3: Editor for algebraic signatures of the AHLI net tool.

Using the editor, on the abstract syntax level, an algebraic signature is defined as model instance of the EMF model for algebraic signatures. Thus, other EMF-based tools may use the EMF model for signatures and model instances without caring for our textual representation of an algebraic signature. A Σ-algebra over a given algebraic signature Σ can be defined by the user of the AHLI net tool by implementing a suitable Java class. This allows us to use Java data types such as *Integer* to implement sorts from the signature. As example, Listing 1 shows the Java class SimpleN implementing an algebra for the signature SimpleNat in Figure 3. The evaluation of terms of the signature to elements of the given algebra is provided by an evaluator class written especially for this purpose. This evaluator provides evaluation of terms with and without variables.

Listing 1: Java class SimpleN implementing an algebra of signature SimpleNat

```
public class SimpleN  {
        public static final Integer ZERO = Integer.valueOf(0);

        public static Integer add(Integer i1, Integer i2) {
                return (i1 + i2);        }
        public static Integer mult(Integer i1, Integer i2) {
                return (i1 * i2);        }
        public static Integer succ(Integer i) {
                return (i + 1);          }
}
```

The Visual AHLI Net Editor and Simulator

The heart of the AHLI net tool is the EMF model for AHLI nets shown in Figure 4. The main class AHLINet defines two attributes specId and algebraId which are references (IDs) of the corresponding extensions used as signature and algebra for the AHLI net. Moreover, an AHLINet contains Places, Transitions and Tokens. A Transition contains Arcs and Conditions. A Token holds a Java object as content (its value)[6].

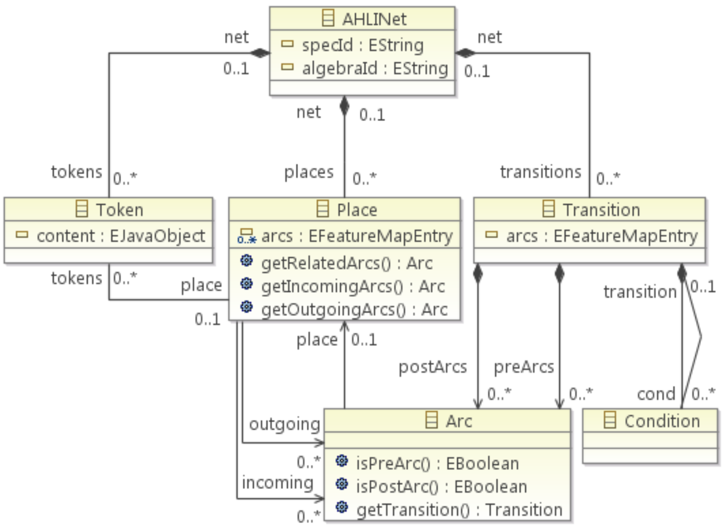


Figure 4: EMF model for AHLI nets

The main component of the visual editor is the tree editor, which contains all model elements of an AHLI net. The model elements can be altered by the user via context menus, which show the appropriate selection of actions that can be performed on the currently selected model element. Additionally there exists a graphical view of the AHLI net. The graphical view provides also context menus and a tool palette with a set of tools to alter the model graphically. Figure 5 shows the editor with the tree editor, the graphical view and the tool palette. In the tree editor, the elements of the SimpleNat signature and algebra can be seen. The visual editor panel contains an AHLI net with one transition modeling the addition of two natural numbers.

The key benefit of modeling an AHLI net using a tool is to be able to actually simulate the firing behavior of the net. This means to calculate the activation state of a transition (the consistent transition assignments) and the successor marking of a firing step. In the visual AHLI net tool, a transition firing step is triggered by the transition's context menu. One of the possible consistent transition assignments is selected randomly, and the successor marking is shown after completion of the firing step.

Note that in the current state of the AHLI net editor, tokens on AHLI net places are string representations of the data contained in them. One of the main tasks is to provide an adequate visualization for complex data representing e.g. P/T nets or P/T net transformation rules. The extension mechanism of ECLIPSE is well suited to achieve this goal and it we plan to implement an extension point supporting the flexible visualization of data structures.

[6] The classes AHLINet, Place, Transition and Arc also inherit from class NamedElement, i.e. they have name attributes which are not shown in Figure 4.

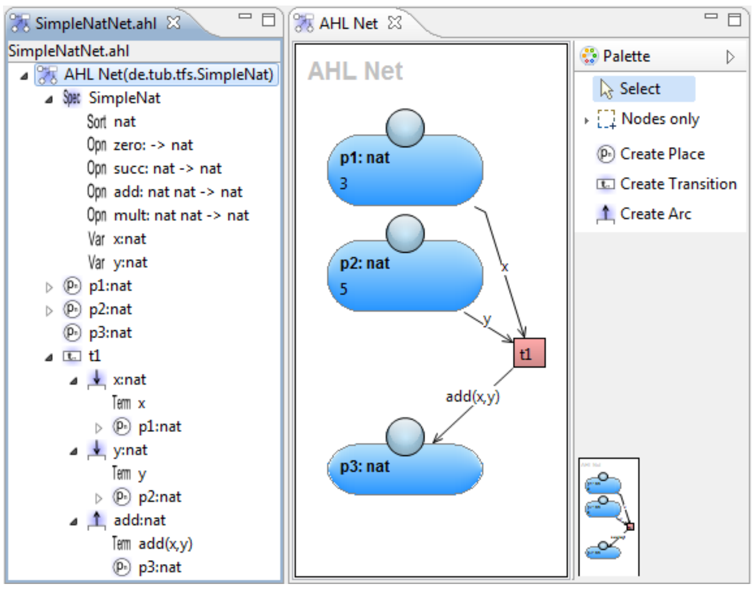


Figure 5: Visual AHLI net editor with the simple Nat example

4 Formalizing RONs as Special AHLI Nets

In this section we formalize RONs as a special type of AHLI nets: the data structure for tokens formalizes P/T nets and P/T net transformation rules, and operations to fire P/T net transitions, to compose and decompose P/T nets, and to apply net transformation rules to P/T nets. Section 4.1 defines this data type, and in Section 4.2 we show how the four standard transition types of RONs are specified.

4.1 Definition of $AHOI_{PTI}$ Nets

We now formalize RONs as special AHLI nets. That is, we define data types for AHLI net tokens which correspond to P/T nets and P/T net transformation rules, respectively, to model the token types in RONs. More precisely, we here use the formal definition of P/T nets with individual tokens (PTI nets [MGE+10, MGH11]) where classical P/T nets are extended by individual tokens in a similar way as we generalized AHL nets to AHLI nets. To distinguish this special kind of AHLI nets from general AHLI nets introduced in Section 3.1, we call AHLI nets with PTI net tokens $AHOI_{PTI}$ nets (Algebraic Higher-Order nets with Individual Tokens). $AHOI_{PTI}$ nets are AHLI nets with a well-defined signature Σ_{PTI} and a fixed Σ_{PTI}-Algebra A_{PTI}.

Definition 3 (Algebraic Higher-Order Nets with Individual Tokens ($AHOI_{PTI}$ Nets)) An $AHOI_{PTI}$ net $AHOI_{PTI} = (\Sigma, P, T, pre, post, cond, type, A, I, m)$, is an AHLI net according to Definition 1 with $\Sigma = (\Sigma_{PTI})$ (see Definition 4), and a Σ_{PTI}-Algebra A_{PTI} (see Definition 5).

With the signature Σ_{PTI} for $AHOI_{PTI}$ nets in Definition 4, we can use PTI nets as object nets in $AHOI_{PTI}$ nets, and PTI net transformation rules as rule tokens.

Definition 4 (Signature Σ_{PTI} for $AHOI_{PTI}$ Nets)

$\Sigma_{PTI} =$	$sorts:$	$Transition, Bool, Mor, Rule,$ $ObjectNet, ONMSet, Selection,$
	$opns:$	$tt, ff :\to Bool$
		$enabled : ObjectNet\ Transition\ Selection \to Bool$
		$fire : ObjectNet\ Transition\ Selection \to ObjectNet$
		$applicable : Rule\ Mor \to Bool$
		$transform : Rule\ Mor \to ObjectNet$
		$empty : ONMSet \to Bool$
		$coproduct : ObjectNet\ ObjectNet \to ObjectNet$
		$isomorphic : ObjectNet\ ObjectNet \to Bool$
		$left : Rule \to ObjectNet$
		$insert : ObjectNet\ ONMSet \to ONMSet$
		$make : ObjectNet \to ONMSet$
		$splittable : ObjectNet \to Bool$
		$dom, cod : Mor \to ObjectNet$
		$split : ObjectNet \to ONMSet$
	$vars:$	$X := (X_s), s \in \{Transition, Bool, Mor, Rule, ObjectNet, ONMSet, Selection\}$

Sort *ObjectNet* denotes object nets (PTI Nets) over sets P_0, T_0 and I_0, *Rule* denotes **PTINet** transformation rules, *ONMSet* denotes the monoid over the set of object nets, *Mor* denotes PTI net morphisms, *Selection* represents the set of the token selections, and *Transition* the set of object net transitions.

The semantics of the operation symbols in Σ_{PTI} is formally defined by the Σ_{PTI}-Algebra A_{PTI} in Definition 5.

Constant $enabled_A$ takes an object net (PTI net) from set $A_{ObjectNet}$, one of its transitions from set $A_{Transition}$ and a token selection from set $A_{Selection}$ and returns tt_A (true) if the transition is enabled to be fired under this token selection, otherwise ff_A (*false*). If a given transition of an object net is enabled to be fired, operation $fire_A$ returns the object net with its successor marking after firing this transition.

To model rule application, operation $applicable_A$ checks whether a rule is applicable to an object net at a given match morphism. Operation $transform_A$ applies the rule at this match if applicable (since the match is a morphism, the object net the rule is applied to can be inferred as the morphism's codomain), realizing the rule-based transformation of an object net. Operations *dom* (resp. *cod*) yield the domain (resp. codomain) of a given morphism.

For moving object nets and rules through the system net, composing them by disjoint union and splitting a disjoint union into component nets, we need some auxiliary operations: $coproduct_A$ builds the disjoint union of two given object nets, operation $empty_A$ checks whether a given set of object nets is empty, the $isomorphic_A$ operation checks whether two given object nets are isomorphic to each other, operation $insert_A$ adds an object net to a sum of object nets, operation $make_A$ generates a sum of object nets containing exactly one given object net, $splittable_A$ checks whether an object net is the disjoint union of more than one component of object nets, and finally, the $split_A$ operation generates from an object net with n components a sum of n object nets.

Definition 5 (Σ_{PTI}-Algebra A_{PTI} of AHOI Nets)

$$
\begin{array}{ll}
A_{PTI} = & A_{Transition} := T_0,\ A_{Bool} := \{true, false\}, \\
& A_{ObjectNet} := PTIobject \cup \{undef\}, \\
& \qquad \text{with } PTIobject := \{N_I | N_I = (P,T,pre,post,I,m), P \subseteq P_0, T \subseteq T_0, I \subseteq I_0\} \\
& A_{Mor} := \{f |\ f : NI_1 \to NI_2\ PTI\ Morphism\ with\ NI_1,\ NI_2 \in PTIobject\} \\
& A_{Rule} := \{r | r = ((L \xleftarrow{i_1} I \xrightarrow{i_2} R), NAC)\ rule\ with\ injections\ i_1, i_2 \\
& \qquad \text{and } NAC \subseteq \{c | (c : L \to Nac)\}\}, \\
& A_{ONMSet} := A^{\oplus}_{ObjectNet}, \\
& A_{Selection} := \{(M,m,N,n) | M,N \subseteq I_0, m : M \to P_0, n : N \to P_0\}
\end{array}
$$

$$tt_A := true \in A_{Bool}, \qquad ff_A := false \in A_{Bool}$$

$$enabled_A : A_{ObjectNet} \times A_{Transition} \times A_{Selection} \to A_{Bool}$$

$$(on,t,s) \mapsto \begin{cases} tt_A & \text{, for } on = (P,T,pre,post,I,m),\ t \in T,\ s = (M,m,N,n) \\ & \quad \text{and } t \text{ is enabled under } s \\ ff_A & \text{, else.} \end{cases}$$

$$fire_A : A_{ObjectNet} \times A_{Transition} \to A_{ObjectNet}$$

$$(on,t,s) \mapsto \begin{cases} (P,T,pre,post,I',m') & \text{, if } enabled_A(on,t,s) = tt_A \\ undef & \text{, else.} \end{cases}$$

for: $on = (P,T,pre,post,I,m), I' = (I \backslash M) \cup N,$

$$m' = I' \to P\ with:\ m'(x) = \begin{cases} m(x) & \text{, if } x \in I \backslash M \\ n(x) & \text{, else.} \end{cases}$$

$$applicable_A : A_{Rule} \times A_{Mor} \to A_{Bool}$$

$$(r,f) \mapsto \begin{cases} tt_A & \text{, if } r \text{ is applicable at match } f \\ ff_A & \text{, else.} \end{cases}$$

$$transform_A : A_{Rule} \times A_{Mor} \to A_{ObjectNet}$$

$$(r,f) \mapsto \begin{cases} NI_2 & \text{, if } applicable_A(r,f) = tt_A \\ undef & \text{, else.} \end{cases}$$

with the direct transformation is written as: $NI_1 \xRightarrow{r,f} NI_2$

$$empty_A : A_{ONMSet} \to A_{Bool}$$

$$(m) \mapsto \begin{cases} tt_A & \text{, if } m = 0 \\ ff_A & \text{, else.} \end{cases}$$

$$coproduct_A : A_{ObjectNet} \times A_{ObjectNet} \to A_{ObjectNet}$$

$$(NI_1, NI_2) \mapsto \begin{cases} undef & \text{, if } NI_1 = undef \text{ or } NI_2 = undef \\ NI_3 & \text{, else.} \end{cases}$$

with:

$$NI_3 = (P_3, T_3, pre_3, post_3, I_3, m_3),\ P_3 = P_1 \uplus P_2,\ T_3 = T_1 \uplus T_2$$

$$pre_3 : T_3 \to P_3^{\oplus}, \qquad pre_3(t) = \begin{cases} pre_1(t) & \text{, if } t \in T_1 \\ pre_2(t) & \text{, else.} \end{cases}$$

$$post_3 : T_3 \to P_3^{\oplus}, \qquad post_3(t) = \begin{cases} post_1(t) & \text{, if } t \in T_1 \\ post_2(t) & \text{, else.} \end{cases}$$

$$
\begin{array}{l}
isomorphic_A : A_{ObjectNet} \times A_{ObjectNet} \to A_{Bool} \\
\quad (on_1, on_2) \mapsto \begin{cases} tt_A & \text{, if } on_1 \cong on_2 \\ ff_A & \text{, else.} \end{cases} \\
\quad on_1 \cong on_2 : \ there\ is\ a\ PTI-Morphism\ f\ : on_1 \to on_2 \\
\quad \text{with: } f = (f_P, f_T, f_I) \text{, and } f_P, f_T, f_I \text{ are bijection functions.} \\
insert_A : A_{ObjectNet} \times A_{ONMSet} \to A_{ONMSet} \\
\quad insert(n,m) = m \oplus n \\
make_A : A_{ObjectNet} \to A_{ONMSet} \\
\quad (n) \mapsto n \\
splittable_A : A_{ObjectNet} \to A_{Bool} \\
\quad (n) \mapsto \begin{cases} tt_A & \text{, if } \exists\, n_1, n_2 \in A_{ObjectNet}, \\ \quad with: & \\ \qquad coproduct(n_1, n_2) = n & \\ \qquad and\ n_1 \neq \emptyset & \\ \qquad and\ n_2 \neq \emptyset & \\ ff_A & \text{, else.} \end{cases} \\
dom_A : A_{Mor} \to A_{ObjectNet} \\
\quad dom_A(f : on_1 \to on_2) = on_1 \\
cod_A : A_{Mor} \to A_{ObjectNet} \\
\quad cod_A(f : on_1 \to on_2) = on_2 \\
\\
split_A : A_{ObjectNet} \to A_{ONMSet} \\
\quad (n) \mapsto \begin{cases} make_A(n) & \text{, if } splittable(n) = ff_A \text{ or } n = undef \\ insert_A(n_1, split_A(n_2)) & \text{, else.} \\ \quad with: & \\ \qquad splittable(n_1) = ff_A & \\ \qquad and\ coproduct(n_1, n_2) = n & \\ \qquad and\ n_1 \neq \emptyset & \\ \qquad and\ n_2 \neq \emptyset & \end{cases}
\end{array}
$$

4.2 Expressing RON Transition Types in AHOI Nets

Using the signature Σ_{PTI} and the Σ_{PTI}-algebra A_{PTI} introduced in Section 4, we now can express the firing behavior of RONs in AHOI nets. The four transition types available for RONs correspond to constructions of AHOI net parts as follows:

Transitions of kind STANDARD

These transitions are used in RONs to move (or copy) object net or rule tokens in the system net without actually changing object nets. We can realize the behavior of STANDARD transitions using an $AHOI_{PTI}$ net transition with the firing condition $\{on' = on_1\ coproduct\ (on_2\ coproduct\ (on_3 ... coproduct\ on_n)) ..)\}$ where $on_i, i \in \{1, .., n\}$ are object nets (data elements from $A_{ObjectNet}$)

from the pre domain of the transition to which the coproduct operation is applied iteratively. The result on' of the operation is the object net constructed as disjoint union of all on_i. The net on' is put onto all object net places in the STANDARD transition's post domain. In addition, the STANDARD transition may take a rule from a Rule place in its pre-domain and puts copies of this rule to all post domain Rule places. Figure 6 shows how RON STANDARD transitions are modelled as AHOI net transitions. Note that we here depict a scheme for all possible STANDARD transitions with an arbitrary number of object net places in the pre- and post-domains, and an arbitrary number of Rule places in the post-domain.

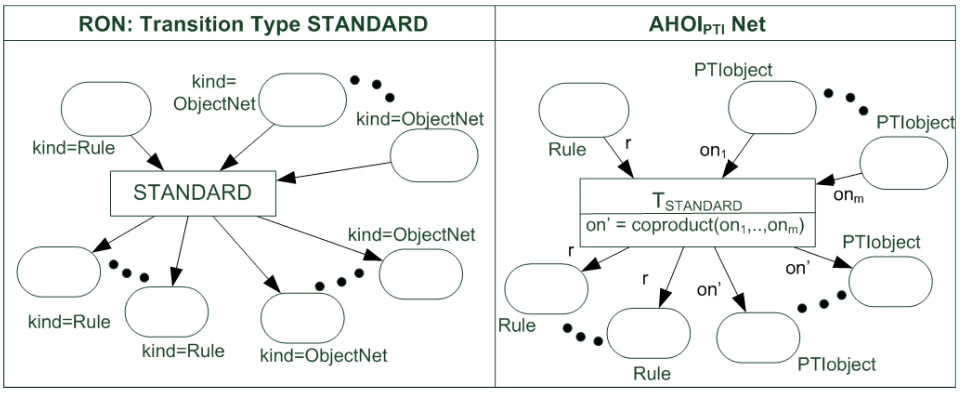


Figure 6: Modelling RON STANDARD transitions in $AHOI_{PTI}$ nets

Transitions of kind FIRE

A FIRE transition in a RON takes an object net from its pre domain place (there must be exactly one), fires a selected enabled transition t of this object net, and puts copies of the object net with the successor marking to all post-domain places. The condition set for an AHOI net transition modeling a FIRE transition is defined by $\{enabled(on,t) = tt,\ on' = fire(on,t)\}$, where on is the PTI net from the pre-domain, t is the transition of on that has to be fired, and on' is object net with the successor marking. Figure 7 shows how FIRE transitions are modelled in AHOI nets.

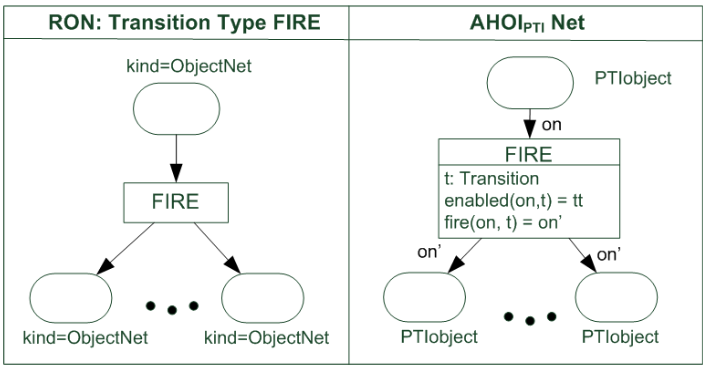


Figure 7: Modelling RON FIRE transition in $AHOI_{PTI}$ nets

Transitions of kind APPLYRULE

In RONs, APPLYRULE transitions take a rule from the only *Rule* place in the pre-domain, apply this rule to the disjoint union of all object nets taken from *ObjectNet* places in the pre-domain, and put the transformed object net to all *ObjectNet* places in the post-domain. Moreover, the rule is moved to the post-domain Rule place.

The condition set for an $AHOI_{PTI}$ net transition modeling an APPLYRULE transition is defined by $\{dom(m) = left(r),\ cod(m) = on,\ applicable(r,m) = tt,\ on' = transform(r,m)\}$, where r is the rule to be applied to a net n in the codomain of a morphism m, n' is the resulting net after applying the rule r if it is applicable at match morphism m, X_{Rule} is the set of the variables of the sort *Rule*, and X_{Mor} is the set of the variables of the sort *Mor*. Figure 8 shows how APPLYRULE transitions are modeled in $AHOI_{PTI}$ nets.

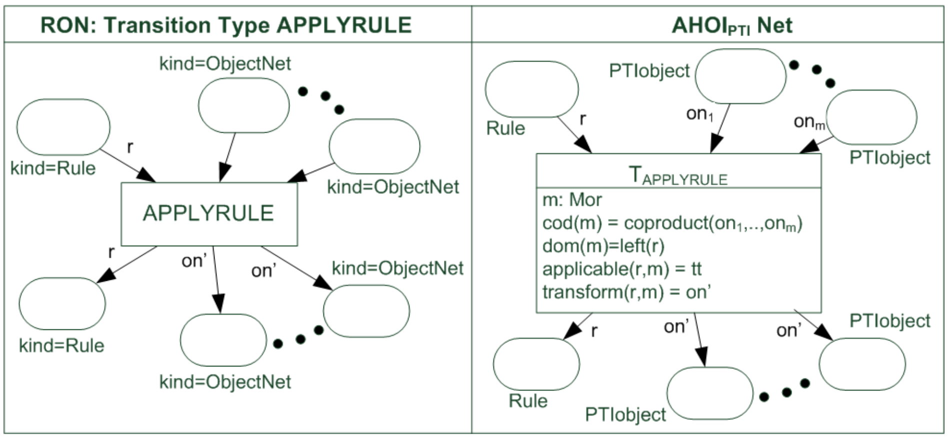


Figure 8: Modelling RON APPLYRULE transitions in $AHOI_{PTI}$ nets

Transitions of kind SPLIT

In RONs, a SPLIT transition has exactly one pre-domain place and one post-domain place, both of kind *ObjectNet*. The transition takes an object net from the pre-domain place and puts all unconnected components of this object net as new object nets to the post-domain place. This kind of transition is the most complicated transition to be modelled in $AHOI_{PTI}$ nets. The main problem is how to split one object net (PTI net) consisting of n components into n independent single object nets. We use the recursive operation $split_A$ that uses the operations $insert_A$, $make_A$ and $splittable_A$. The operation $make_A$ makes a set containing a net that cannot be split, otherwise the operation $insert_A$ puts the nets stepwise recursively in a set. The split operation in AHOI nets is well-defined, because it puts only those nets that consist of exactly one component into the sum of the split net, and the termination is also ensured.

In order to simulate the firing of a transition of kind SPLIT in AHOI nets we need two more transitions and one more place then in RONs. Note that for one firing step of a SPLIT transition in a RON, we get a firing sequence in the $AHOI_{PTI}$ net. In this sequence, transition SPLIT is fired once, making a sum of component nets from the original object net. Afterwards, transition

SELECT1 fires as many times as there are unconnected components in this sum. Each time, one unconnected component is put to the PTIobject place in the bottom, while the sum containing the remaining unconnected components is put back to the ONMSet place. When there is only one component left in this sum, transition SELECT2 is fired, which puts this last remaining component as object net to the PTIobject place. Figure 9 shows how a SPLIT transition from RONs is modeled as AHOI net.

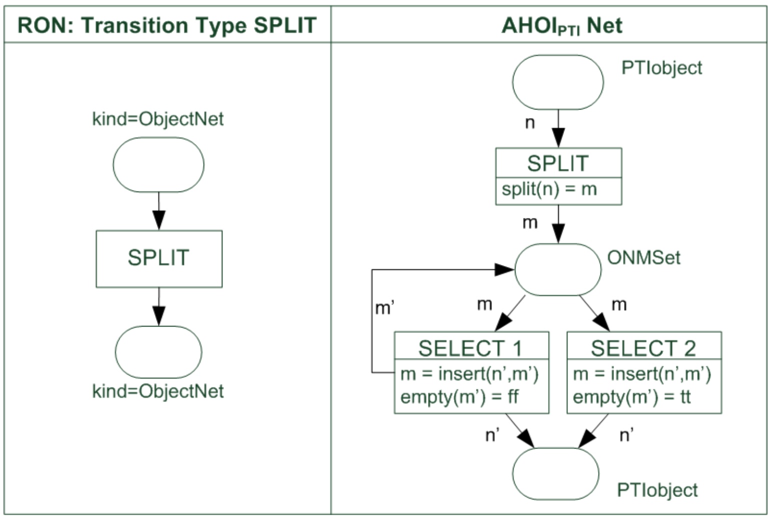


Figure 9: Modelling RON SPLIT transitions in $AHOI_{PTI}$ nets

5 Related Work

There exist related approaches and tools that follow the "nets as tokens"-paradigm, based on elementary object nets introduced in [Val98]. Mobile object net systems [FK04] are an algebraic formalization of the elementary object nets that are closely related to our approach. Mobile object net systems are implemented in the Renew tool [KMR03]. The data types, respectively the colors represent the nets that are the token nets. Our approach goes beyond those approaches as we may additionally have rules as tokens, and transformations of nets as operations. In [KR03] concurrency aspects between token nets have been investigated, but naturally not concerning net transformations. In [LO04], rewriting of Petri nets in terms of graph grammars are used for the reconfiguration of nets as well, but this approach lacks the "nets as tokens"-paradigm.

Recently, research has been done to translate Petri net transformation systems of different categories to graph transformation systems in a semantically equivalent way [MEE11]. On this basis, we will now be able to analyse confluence of reconfiguration and firing steps by using the AGG graph transformation analyser [AGG10]. The implementation of the AHLI net tool will serve as basis for future extensions towards analysing the behavior of reconfigurable systems.

6 Conclusion and Future Work

In this paper we formalized the approach and tool for Reconfigurable Object Nets (RONs) to a special kind of AHLI nets, called $AHOI_{PTI}$ nets. We defined an algebraic signature and algebra that formally specify RONs. This fixes a formal semantics for RONs, using a framework that can easily be extended towards other kinds of Petri nets as token objects (e.g. using AHLI nets and AHLI net transformation rules as tokens).

Moreover, in future work, this formal basis will be used to analyse the behavior of $AHOI_{PTI}$ nets and object net reconfiguration based on the interleaving semantics of the corresponding graph transformation system, including rules for firing and for reconfiguration. This semantics specifies the equivalent re-orderings of a given transformation sequence. Formal equivalence results (e.g. on permutation equivalence [HCEK10]) allow all equivalent transformation sequences to be constructed in an efficient way. In [Sha10], the permutation equivalence analysis has been performed exemplarily for the producer/consumer system $AHOI_{PTI}$ net on the level of object nets.

We have described in this paper an ongoing implementation of a flexible AHLI net tool environment that supports editing and simulation of AHLI nets in general. In particular, up to now AHLI nets with primitive data types as tokens can be edited and simulated in the visual modeling environment. The visual appearance of different token types in AHLI nets shall be implemented using the extension points of the AHLI net tool. With respect to the tool, the next steps will be to provide pre-defined visual layouts for $AHOI_{PTI}$ nets (P/T nets and P/T net transformation rules as tokens) as well as algebraic specifications and algebras for nets with AHLI nets and AHLI net transformation rules as tokens, and for other kinds of reconfigurable dynamic modeling techniques.

References

[AGG10] TFS-Group, TU Berlin. AGG. 2010.
http://tfs.cs.tu-berlin.de/agg

[BEE+09] E. Biermann, H. Ehrig, C. Ermel, K. Hoffmann, T. Modica. Modeling Multicasting in Dynamic Communication-based Systems by Reconfigurable High-level Petri Nets. In *Proc. Int. Symposium on Visual Languages and Human-Centric Computing (VL-HCC'09)*. Pp. 47–50. IEEE, 2009.

[BEHM07] E. Biermann, C. Ermel, F. Hermann, T. Modica. A Visual Editor for Reconfigurable Object Nets based on the ECLIPSE Graphical Editor Framework. In *Proc. 14th Workshop on Algorithms and Tools for Petri Nets (AWPN'07)*. GI, 2007.

[BM08] E. Biermann, T. Modica. Independence Analysis of Firing and Rule-based Net Transformations in Reconfigurable Object Nets. In *Proc. Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT'08)*. Volume 10. ECEASST, 2008.

[EGH10] H. Ehrig, U. Golas, F. Hermann. Categorical Frameworks for Graph Transformation and HLR Systems based on the DPO Approach. *Bulletin of the EATCS* 102:111–121, 2010.
http://tfs.cs.tu-berlin.de/publikationen/Papers10/EGH10.pdf

[EHP$^+$08] H. Ehrig, K. Hoffmann, J. Padberg, C. Ermel, U. Prange, E. Biermann, T. Modica. Petri Net Transformations. In *Petri Net Theory and Applications*. Pp. 1–16. I-Tech Education and Publication, 2008.

[EM85] H. Ehrig, B. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*. EATCS Monographs on Theoretical Computer Science 6. Springer, Berlin, 1985.

[EMF11] Eclipse Consortium. Eclipse Modeling Framework (EMF) – Version 2.5. 2011.
http://www.eclipse.org/emf

[FK04] B. Farwer, M. Köhler. Mobile Object-Net Systems and their Processes. *Fundamenta Informaticae* 60(1–4):113–129, 2004.

[HCEK10] F. Hermann, A. Corradini, H. Ehrig, B. König. Efficient Analysis of Permutation Equivalence of Graph Derivations Based on Petri Nets . *ECEASST* 29:1–15, 2010.

[HME05] K. Hoffmann, T. Mossakowski, H. Ehrig. High-Level Nets with Nets and Rules as Tokens. In *Proc. of 26th Intern. Conf. on Application and Theory of Petri Nets and other Models of Concurrency*. LNCS 3536, pp. 268–288. Springer, 2005.

[KMR03] M. Köhler, D. Moldt, H. Rölke. Modelling Mobility and Mobile Agents Using Nets within Nets. In *Proc. Int. Conf. on Applications and Theory of Petri Nets (ICATPN'03)*. LNCS 2679, pp. 121–139. Springer, June 2003.

[KR03] M. Köhler, H. Rölke. Concurrency for mobile object net systems. *Fundamenta Informaticae, 54(2-3)*, 2003.

[LO04] M. Llorens, J. Oliver. Structural and Dynamic Changes in Concurrent Systems: Reconfigurable Petri Nets. *IEEE Transactions on Computers* 53(9):1147–1158, 2004.

[MEE11] M. Maximova, H. Ehrig, C. Ermel. Formal Relationship between Petri Net and Graph Transformation Systems based on Functors between M-adhesive Categories. In Ehrig et al. (eds.), *Proc. of 4th Workshop on Petri Nets and Graph Transformation (PNGT)*. Volume 40. ECEASST, 2011.

[MGE$^+$10] T. Modica, K. Gabriel, H. Ehrig, K. Hoffmann, S. Shareef, C. Ermel, U. Golas, F. Hermann, E. Biermann. Low- and High-Level Petri Nets with Individual Tokens. Technical report 2009/13, Technische Universität Berlin, 2010.
http://www.eecs.tu-berlin.de/menue/forschung/forschungsberichte/2009

[MGH11] T. Modica, K. Gabriel, K. Hoffmann. Transformation of Petri Nets with Individual Tokens. In Ehrig et al. (eds.), *Proc. of 4th Workshop on Petri Nets and Graph Transformation (PNGT)*. Volume xx. ECEASST, 2011. Submitted.

[PEHP08] U. Prange, H. Ehrig, K. Hoffman, J. Padberg. Transformations in Reconfigurable Place/Transition Systems. In *Concurrency, Graphs and Models*. LNCS 5065, pp. 96–113. Springer, 2008.

[PER95] J. Padberg, H. Ehrig, L. Ribeiro. Algebraic High-Level Net Transformation Systems. *Mathematical Structures in Computer Science* 5:217–256, 1995.

[RON11] TFS, TU Berlin. Reconfigurable Object Nets. 2011.
http://www.tfs.cs.tu-berlin.de/roneditor

[Sha10] S. Shareef. Formal Modelling and Analysis of Reconfigurable Object Nets Based on the RON Editor. Master's thesis, TU Berlin, Fak. IV, 2010.

[Tro09] F. Trollmann. Modeling Emergency Scenarios using Algebraic Higher Order Nets. Master's thesis, Technische Universität Berlin, 2009.

[Val98] R. Valk. Petri Nets as Token Objects: An Introduction to Elementary Object Nets. In *Proc. Int. Conf. on Application and Theory of Petri Nets (ICATPN '98)*. LNCS 2987, pp. 1–25. Springer, 1998.

[Xte10] Eclipse Consortium. Xtext - Language Development Framework – Version 1.0. 2010.
http://www.eclipse.org/Xtext/