Electronic Communications of the EASST
Volume 54 (2012)

Proceedings of the
7th International Workshop on Graph Based Tools
(GraBaTs 2012)

ReConNet: A Tool for Modeling and Simulating with Reconfigurable Place/Transition Nets

Marvin Ede, Kathrin Hoffmann, Gerhard Oelker, Julia Padberg

11 pages

Guest Editors: Christian Krause, Bernhard Westfechtel
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

ReConNet: A Tool for Modeling and Simulating with Reconfigurable Place/Transition Nets

Marvin Ede, Kathrin Hoffmann, Gerhard Oelker, Julia Padberg

Hochschule für Angewandte Wissenschaften Hamburg, Gemany

Abstract:

In this contribution we present a tool for modeling and simulation with reconfigurable Petri nets. Taking the idea of algebraic graph transformations to marked Petri nets we obtain Petri nets whose net structure can be changed dynamically. The rule-based change of the net structure enables the adequate modeling of complex, dynamic structures as for example of the scenarios of the Living Place Hamburg. The tool ReConNet uses decorated place/transition nets that are extended by various annotations. Especially, they have transition labels that may change when the transition fires. The transformation approach is based on the well-known algebraic transformation approach, but here we use a variant, namely the cospan approach, that inverts the relation between left- and right-hand sides and interface in the rules.

Keywords: net transformation tool, reconfigurable Petri nets, cospan DPO approach

1 Motivation

Reconfigurable Petri nets [EP03, LO04, EHP$^+$07, PEHP08] are a powerful and intuitive formalism for describing complex systems with dynamic structures that has been available since 2003. The characteristic feature is the possibility to discriminate between different levels of change. In this paper we present a tool that directly allows modeling and simulating reconfigurable place/ transition nets. The main motivation for the tool ReConNet is the formal description of scenarios from the Living Place Hamburg, that is a smart home under constant development since 2009. It is a loft apartment with dynamic mapping of functions to spaces according to the respective situation of the resident (e.g. bedroom, kitchen, living room). The Living Place Hamburg is a laboratory for applied research in different areas of ambient intelligence and covers different areas of IT-based urban living. Scenarios of the Living Place Hamburg describe the way the resident interacts with the smart home. One scenario illustrates for example an everyday procedure of the resident at the Living Place and how the system can accomplish its aim to support the residents of the Living Place in his everyday procedure. In order to achieve a better understanding, we want to abstract from sensor data using nondeterminism instead of complicated control structures. Reconfigurable place/transition nets are suitable for this purpose, as they offer the possibility to differentiate between the resident's actions and the smart home's reaction. The smart home's reaction is modeled by the change of the infrastructure that is given by the underlying place/transition net structure. In a comprehensive case study [Rei12] the "morning scenarios" of the Living Place are modeled using reconfigurable place/transition nets. For illustration in Fig. 2 in Section 3 a screenshot is given with a rule similar to those in the case study. It shows the rule `ringAlarm` that belongs to a "morning scenario". The rule models the extension of the scenario in case an alarm clock should be used. The transition `wakeUp` is replaced dynamically by the net `R`, describing a simple alarm clock with a snooze function. Unfortunately for the case study [Rei12] the tool ReConNet could not be employed as both have been developed simultaneously.

In this contribution we first summarize the underlying formal technique that is based on decorated place/transition nets and net transformations in the cospan approach. Then we present the tool ReConNet for modeling and simulation reconfigurable place/transition nets The tool is the result of two courses at the HAW Hamburg. Concluding remarks concern related and future work.

2 Reconfigurable Place/Transition Nets

Using the algebraic approach to Petri nets facilitates the combination of the algebraic approach to transformation. So, a marked place/transition net is given by $N = (P, T, pre, post, M)$ with pre- and post-domain functions $pre, post : T \to P^{\oplus}$ and a marking $M \in P^{\oplus}$, where $P^{\oplus}$ is the free commutative monoid over the set P of places. For $M_1, M_2 \in P^{\oplus}$ we have $M_1 \leq M_2$ if $M_1(p) \leq M_2(p)$ for all $p \in P$. A transition $t \in T$ is M-enabled for a marking $M \in P^{\oplus}$ if we have $pre(t) \leq M$, and in this case the follower marking M' is given by $M' = M \ominus pre(t) \oplus post(t)$ and $M[t\rangle M'$ is called firing step. To gain an adequate modeling technique a few new features needed to be added. Obvious is the extension to capacities and names. More interesting are the transition labels that may change, when the transition is fired. This allows a better coordination of transition firing and rule application, for example can be ensured that a transition has fired (repeatedly) before a transformation may take place.

2.1 Decorated Place/Transition Nets

In order to provide the technical basis for the modeling of scenarios we need place/transition nets that have the following additional decorations: capacities, names for places as well as transitions and additional transition labels that can be changed by firing that transition. This last extension is conservative with respect to Petri nets as it does not change the net behavior. But it is crucial for the application of the rules and provides the possibility to control the application of rules.

A decorated place/transition net is a marked place/transition net $N = (P, T, pre, post, M)$ together with names and labels. Based on A_P, A_T a name space with $pname : P \to A_P$ and $tname : T \to A_T$ we have explicit names for places and transitions. Moreover, transitions need to be equipped with labels that may change when the transition fires. For the correct application of a rule it may be important that a transition has already fired. This cannot be modeled with usual place/transition nets. Considering the tokens in the transition's post places, does not work as these tokens may be consumed as well. So we have to change the label of the transition. For example, the label changes from `false` (indicating that the transition has not yet fired) to `true` (indicating that it has already fired). In Fig. 2 in Section 3 the transition `stopAlarm` in the net K (or R) of rule `ringAlarm` has the following node attributes (given in the upper middle panel): `Id:28, Name: stopAlarm, Label: false, Renew: toggle`, where toggle is a function mapping `false` to `true` and the other way round. More formally, `toggle` can be expressed using propositional logic $toggle(x) := \neg x$.

Formalizing this notion we introduce a set W of changing labels for each transition and a function $tlb : T \to W$ mapping transitions to transition labels W. Then each transition is decorated with a endomorphism on W, i.e. a function that maps labels to labels. Th is decoration is given by the function $rnw : T \to END$ where END is a set containing some endomorphisms on W, so that $rnw(t) : W \to W$ is the function that renews the transition label. Note that a partition of the changing labels $(W_t)_{t\in T}$ with respect to the transitions can easily be obtained (see [Pad12]). These labels are important for the control of the rules, but not for the net's behavior.

The firing of the extended nets is the same as in place/transition net except for the changing transition labels. Moreover, this extension works for parallel firing as well. Given a transition vector $v = \sum_{t\in T} k_t \cdot t$ then the label is renewed by firing $tlb[v\rangle tlb'$ and tlb' is computed by:

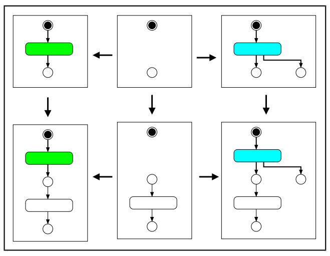
$$tlb'(t) = rnw(t)^{k_t} \circ tlb(t)$$

Analogously to the marking the transition labels also evolve during the token game.

2.2 Net Transformations in the Cospan-Approach

The cospan approach is a variant of algebraic transformations - namely the double pushout (DPO) approach - where the left-hand side and right-hand side of the rule are embedded in the interface $(L \to K \leftarrow R)$, hence cospan. In contrast to the classical DPO approach (where to morphisms starting from the interface $L \leftarrow K \to R$) we add new items first and then delete (some of) the old items in the second step.

In [EHP09] the cospan DPO approach has been shown to be equivalent to the classical DPO approach. In this way we are able to switch between these two approaches and, finally, omit the interface in the graphical description. In this paper we use the cospan DPO approach, because from implementation point of view, it is often more convenient to add the new items first and delete some of the old items in a second step. This idea is adopted in the cospan DPO approach where a rule is given by a cospan of morphisms, while a transformation step via a cospan rule is still defined by two pushouts. Roughly spoken, in the classical DPO approach the intermediate net obtained by rule application is often full of holes like Swiss cheese, while in the cospan DPO approach this net includes the source net and the target net.


(a) DPO span

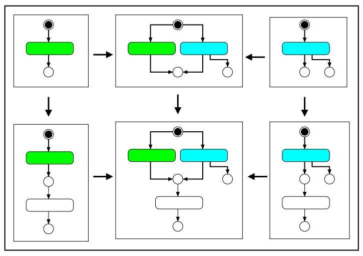

(b) DPO cospan

Figure 1: Comparing and cospan (DPO) approach

For example in Fig. 1(a) the rule is in the DPO approach and the rule in Fig. 1(b) is in the cospan DPO approach. But both of these two rules replace the transition on the left-hand side by one on the right-hand side, that has a further place in its post-domain. In contrast to the classical approach in the cospan DPO approach we are able to relate the old and new items in an easy way. The interface of this cospan rule (see Fig. 1(b)) states for example that the environment of both transitions have to have at least the same places in the pre- and post-domain.

Thus, on the one hand several properties could be formulated in a more intuitive way and on the other hand some aspects could be investigated that have escaped our attention in the classical DPO approach. The main results in [HEH10] are not only sufficient (as in [EHP+07] using the classical approach) but also necessary conditions, so that a transformation step and a firing step can be executed in arbitrary order. Moreover in future work we will consider further property preserving net transformations in the cospan DPO approach. Of special interest are strongly connectivity and liveness and we expect that especially these properties could be formulated and proven in a more intuitive way in the cospan DPO approach.

For decorated place/transition nets as given above, we obtain with the following notion of morphisms an $\mathcal{M}$-adhesive HLR-category (see [Pad12]). $\mathcal{M}$-adhesive HLR systems can be considered as a unifying framework for graph and Petri net transformations providing enough structure that most notions and results from algebraic graph transformation systems are available (e.g. in [EEPT06, EGH+12]).

Morphisms are given as a pair of mappings for the places and the transitions, so that the structure and the decoration is preserved and the marking may be mapped strictly. Given two nets $N_i = (P_i, T_i, pre_i, post_i, M_i, pname_i, tname_i, tlb_i, rnw_i)$ for $i = 1,2$ then the morphism $f : N_1 \to N_2$ is given by $f = (f_P, f_T)$ the mapping of places to places and the mapping of transitions to transitions. Additionally, the usual equations have to hold that ensure the preservation of markings and of labels [Pad12].

Given a decorated place/transition net and a rule together with an occurrence morphism o, the a transformation step $(N_1, M_1) \overset{(rule,o)}{\Longrightarrow} (N_2, M_2)$ consists of the following pushout diagrams (1) and (2).

$$\begin{tikzcd}
(L, M_L) \arrow[r, "l"] \arrow[d, "o"'] \arrow[dr, phantom, "(1)"] & (K, M_K) \arrow[d] \arrow[dr, phantom, "(2)"] & (R, M_R) \arrow[l, "r"'] \arrow[d, "n"] \\
(N_1, M_1) \arrow[r] & (N_0, M_0) & (N_2, M_2) \arrow[l]
\end{tikzcd}$$

A rule in the cospan approach is given by the left-hand side, interface and right-hand side net, respectively, and a cospan of two net morphisms l and r. An occurrence morphism o identifies the relevant parts of the left-hand side in the given net (N_1, M_1). Then a transformation step $(N_1, M_1) \overset{(rule,o)}{\Longrightarrow} (N_2, M_2)$ can be constructed in two steps, provided that gluing conditions hold. The characterization of specific points is a sufficient condition for the existence and uniqueness of the so-called pushout complement which is needed for the second step in a transformation.

This construction as well as a huge amount of notion and results are available since decorated place/transition nets can be proven to be an $\mathcal{M}$-adhesive HLR category (see [Pad12]). Hence we can combine one net together with a set of rules leading to reconfigurable place/transition nets.

A reconfigurable place/transition net $RN = ((N,M),\mathscr{R})$ is given by

- a decorated place/transition net $N = (P,T,pre,post,pname,tname,cap,tlb,rnw)$ and its marking M and
- a set of rules $\mathscr{R}$, where rules $rule$ are given in the cospan approach $rule = (L,M_L) \to (K,M_K) \leftarrow (R,M_R)$.

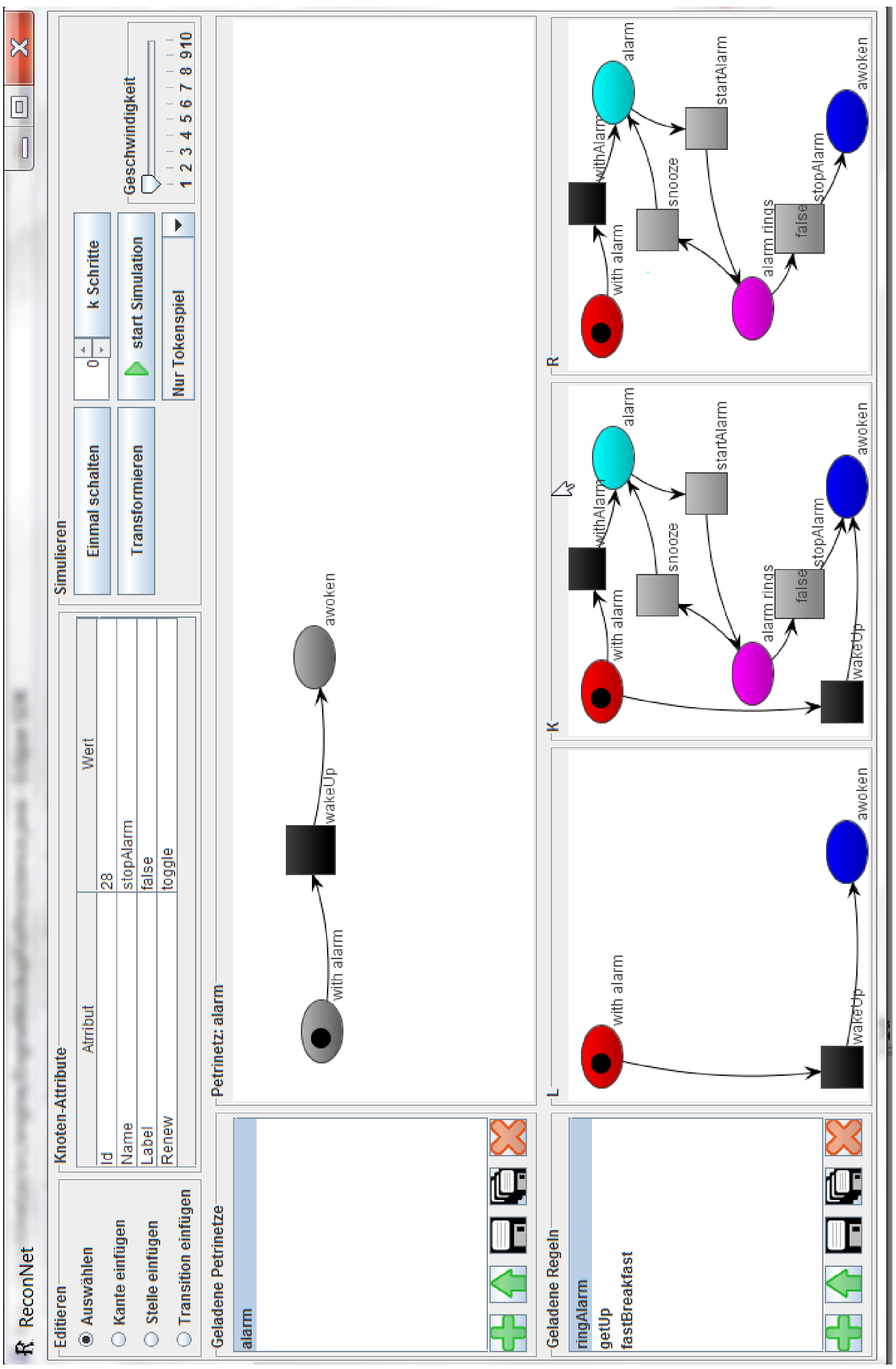
3 ReConNet: Editor and Simulator

The software tool ReConNet (ReConfigurable Net) has been developed so that the modeling 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. ReConNet is completely implemented in Java 6, therefore being fully portable. For the purposes of the user interface, SWING [SWI] was found to be suitable. In combination with JUNG [JUN], the JAVA universal network/graph framework, a first net rendering and editing component can be forged readily. For more customized net editing capabilities, many JUNG behaviors need to be overwritten or to be used in custom classes.

3.1 Editor and Simulator

The most important aspects of ReConNet are reflected in its graphical user interface. There are two different areas for nets and rules, underlining the importance of both those constructs. The user can basically choose among four modes that can be operated in. Note in Fig. 2 the check-box with the options: `pick` (Auswählen), `insert arc` (Kante einfügen), `insert place` (Stelle einfügen), and `insert transition` (Transition einfügen). In the `pick` mode, the user can click on nodes (i.e. a place or a transition) to select them to highlight them or edit its attributes like marking. Also transition labels and the renew function can be set by typing into the attribute table in the top middle. Nodes can be easily moved by drag-and-drop. When the user drag-and-drops onto the white space, the whole net is moved. By scrolling the mouse wheel the zoom function can be used. When right-clicking a node, it can be deleted (also its incident arcs) or a color can be assigned. Colors do not interfere with the graph matching or any logical function at all. All of this applies to the rule editing, also. The renew function is restricted to one of the three modes: `id`, that does not change the labels, being the identity function, `toggle`, that swaps the Boolean values true and false, and `count`, that increases an integer given as the label. In the arc mode, arcs can be added by drag-and-dropping among nodes of different types. In the place and transition mode, places or transitions can be added to the net by clicking onto the white space.

Adding a node (i.e. a place or a transition) into a rule needs to differ from the procedure for nets. For each node that is created in one of the rule's panels, one or two corresponding node are inserted to one or two other panels of the rule. This ensures that there are only injective morphisms from the left -and right side to the interface. Moreover, a unique color is auto-generated and assigned to the new places illustrating the underlying morphism.



Atribut	Wert
Id	28
Name	stopAlarm
Label	false
Renew	toggle

Figure 2: Screenshot of ReConNet's GUI

This procedure also enables the user to quickly model the basic functions of a rule: By adding a node to the left part of the rule **L**, it automatically is added into the interface **K**, this leads to the deletion of the node when applying the rule. If a node is added into the interface **K**, it is added in both left and right parts of the rule to ensure being part of the matching subgraphs. Adding a node into the right-hand side **R** of the rule also inserts the node into **K**, meaning this node is added when the rule is applied. The same treatment is used for arcs as well except the coloring.

Another important feature of RECONNET is the simulation of nets, which can be done in different fashions. First of all there is a button to fire one transition. The transition is chosen randomly among those that are active. A transition is represented as a dark gray rectangle if it is activated and as a light gray rectangle if not. The follower marking is computed and the tokens are moved accordingly. The application of rules leads to the transformation of the net. The rule that is applied to the displayed net is chosen randomly among those who are selected in the left hand overview. Also the morphism is found in a non-deterministic fashion. If there is an injective match morphism, then the gluing conditions will be checked for the intermediate net. Then the resulting net is computed and displayed.

Additionally there are two more advanced simulation options - `k steps` (k Schritte) and running simulation - which can operate in three different modes: `tokens` (Nur Tokenspiel), `transformation` (Transformation) and `both` (Beides). The `k steps` options executes a definable amount of steps on the net. In the token mode only transitions are fired, in the transformation mode only rules are applied. In the `both mode` it is randomly chosen whether a transition is fired or a rule is applied for each step. Those three operations can be applied by k steps instantly, but also in a simulation running mode, which can be set to run fast or slow in 10 speed levels from 1 step every 2 seconds to basically as fast as the running system can offer.

3.2 Architecture and Persistency

The software is designed with component based architecture to allow refinement throughout the process of development. Thus its functionalities are divided into the five broad components **Petrinet**, **Transformation**, **Engine**, **Persistence** and **GUI**. While **Petrinet** and **Transformation** offer the more abstract functionalities of nets and rules, the **Engine** combines them to a more tool-specific interface, transforming data, checking correctness of user input, performing simulations and managing session data. On top is the **GUI** component that defines the displays and controls of nets and rules. Aside from that, the **Persistence** component is adjoined by the engine to load and store Petri nets, rules and simulation results to the local file system.

Since interoperability is a desirable feature, we have used for storing place/transition nets XML schemata as given by the PNML standard (e.g. in [HKPT10]). Parsing XML files is implemented via JAXB [JAX], the JAVA architecture for XML binding. It enables to access XML files as an object tree with special classes for each type of XML element. This method requires less high level custom code for parsing the XML, rather a set of classes similar to structs that need to be written or generated. One of its downfalls is the disability to partly load an XML file into the memory that might get problematic for large nets on computers with little memory. Saving nets and rules in XML files rather than a serialized form, enables the interaction with other tools via standardized formats, later on. For rules there are no XML schemata available in the PNML standard. Therefore we have developed the following approach for saving rules.

Each part of a rule is a net element, just as a Petri net. In addition it has a net type attribute which can have one of three values L, K and R. Also each net element in K has its own id which is shared with their respective counterpart in L and R. A node that occurs in all parts of a rule will appear in each three net elements of the rule with always the same attributes and always the same id. This id is used to encode the corresponding morphism form $L \subseteq K$ and $K \supseteq R$. In Fig. 3 in the appendix an excerpt of the XML-code of the rule `ringAlarm` from Fig. 2 is given that represents the rule's left-hand side L.

4 Conclusion

To conclude this paper we discuss related and future work.
Up to now there have been two tools that also implement some kind of reconfigurable Petri nets. MCReNet [LO05] is a tool for the specification, modeling, simulation, and verification of concurrent systems that are subject to dynamic changes by using Marked-Controlled Reconfigurable Nets. In addition to the Petri nets it is equipped with a configuration graph, a labeled directed graph whose nodes are the configurations.

The Reconfigurable Object Nets (RON) Editor [BEHM07, BM08, BEMS08] integrates transition firing and rule-based net structure transformation of place/transition nets during system simulation. In contrast to our approach they are high-level nets with two types of token: object nets (place/transition nets) and net transformation rules (a dedicated type of graph transformation rules). Firing of high-level transitions may involve firing of object net transitions, transporting object net token through the high-level net, and applying net transformation rules to object nets. Net transformations include net modifications such as merging or splitting of object nets, and net refinement. Nevertheless the RON Editor is capable of simulating reconfigurable place/transition nets by putting all place/transitions nets as high-level tokens on high-level object-net places, putting all rules on high-level net reconfiguration rule places, and connecting a rule place with a net place by a transition that triggers the application of a net transformation rule to an object net. Moreover the RON Editor supports negative application conditions [BM08] and an independence analysis of net transitions [BEMS08].

In [GN11] the Living Place Hamburg is modeled using Algebraic High-Level Nets with Individual Token, short AHLI nets, as well as rule-based transformation of such nets following the double pushout approach. In that thesis the focus was on the detailed modeling of the data types.

Since the scenario modeling in [Rei12] makes use of negative application conditions as well as capacities these are the obvious extensions that need to be implemented next. In [Pad12] the construction of a reachability graph for reconfigurable place/transition nets is given. Moreover, the modeled scenarios - due to negative application conditions and capacities – are bounded leading to bounded reachability graph and hence to the corresponding possibilities of model checking.

Acknowledgements: This tool is the result the student's efforts in two courses given at the HAW Hamburg. We want to thank Nora Berg, Mathias Blumreiter, Andreas Jäckel, Tobias Markmann, Maximilian Meyer, Dmytro Oleiynyk, Alexander Schulz, Oliver Willhöft, Pierre-Henri Nodin (fall semester 2011) and Steffen Brauer, Roman Geez, Till Gerken, Christian Götz,

Philipp Kühn, Thorsten Paech, Florian Reiter, Zabihullah Safai, Niklas Schreiber, Moritz Uhlig (fall semester 2010).

References

[BEHM07] E. Biermann, C. Ermel, F. Hermann, T. Modica. A Visual Editor for Reconfigurable Object Nets based on the ECLIPSE Graphical Editor Framework. In Juhas and Desel (eds.), *Proc. 14th Workshop on Algorithms and Tools for Petri Nets (AWPN'07)*. GI Special Interest Group on Petri Nets and Related System Models, Universität Koblenz-Landau, Germany, 2007.

[BEMS08] E. Biermann, C. Ermel, T. Modica, P. Sylopp. Implementing Petri Net Transformations using Graph Transformation Tools. *ECEASST* 14, 2008.

[BM08] E. Biermann, T. Modica. Independence Analysis of Firing and Rule-based Net Transformations in Reconfigurable Object Nets. *ECEASST* 10, 2008.

[DRR04] J. Desel, W. Reisig, G. Rozenberg (eds.). *Lectures on Concurrency and Petri Nets, Advances in Petri Nets*. Lecture Notes in Computer Science 3098. Springer, 2004.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs in TCS. Springer, 2006.

[EGH$^+$12] H. Ehrig, U. Golas, A. Habel, L. Lambers, F. Orejas. M-AdhesiveTransformation Systems with Nested Application Conditions. Part 2: Embedding, Critical Pairs and Local Confluence. *Fundam. Inform.* 118(1-2):35–63, 2012.

[EHP$^+$07] H. Ehrig, K. Hoffmann, J. Padberg, U. Prange, C. Ermel. Independence of Net Transformations and Token Firing in Reconfigurable Place/Transition Systems. In Kleijn and Yakovlev (eds.), *ICATPN*. Lecture Notes in Computer Science 4546, pp. 104–123. Springer, 2007.

[EHP09] H. Ehrig, F. Hermann, U. Prange. Cospan DPO Approach: An Alternative for DPO Graph Transformations. *Bulletin of the EATCS* 98:139–149, 2009.

[EP03] H. Ehrig, J. Padberg. Graph Grammars and Petri Net Transformations. Pp. 496–536 in [DRR04].

[GN11] S. Gottmann, N. Nachtigall. Modelling the Living Place Project using Algebraic Higher Order Nets. Diploma Thesis, Technische Universität Berlin, 2011. http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/diplom/gottmann-nachtigall.pdf

[HEH10] K. Hoffmann, H. Ehrig, F. Hermann. Flexible Independence of Net Transformations and Token Firing in the Cospan DPO Approach. In *Proceedings of the 3rd International Symposium of Multiagent Systems (MAS), Robotics and Cybernetics: Theory and Practice, 2009*. IIAS, 2010.

[HKPT10] L.-M. Hillah, F. Kordon, L. Petrucci, N. Trèves. PNML Framework: An Extendable Reference Implementation of the Petri Net Markup Language. In Lilius and Penczek (eds.), *Petri Nets*. Lecture Notes in Computer Science 6128, pp. 318–327. Springer, 2010.

[JAX] Java Architecture for XML Binding.
http://www.oracle.com/technetwork/articles/javase/index-140168.html

[JUN] Java Universal Network/Graph Framework.
http://jung.sourceforge.net/

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

[LO05] M. Llorens, J. Oliver. MCReNet: a tool for Marked-Controlled Reconfigurable Nets. *Quantitative Evaluation of Systems, International Conference on* 0:255–256, 2005. doi:http://doi.ieeecomputersociety.org/10.1109/QEST.2005.18

[Pad12] J. Padberg. Abstract Interleaving Semantics for Reconfigurable Petri Nets. In *PNGT 2012*. Volume 51. 2012.

[PEHP08] U. Prange, H. Ehrig, K. Hoffman, J. Padberg. Transformations in Reconfigurable Place/Transition Systems. In Degano et al. (eds.), *Concurrency, Graphs and Models: Essays Dedicated to Ugo Montanari on the Occasion of His 65th Birthday*. Lecture Notes in Computer Science 5065, pp. 96–113. Springer Verlag, 2008.

[Rei12] F. Reiter. Modellierung und Analyse von Szenarien des Living Place mit rekonfigurierbaren Petrinetzen. Bachelor Thesis, Hochschule für Angewandte Wissenschaften Hamburg, 2012.
http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/bachelor/reiter.pdf

[SWI] Swing Enhancements in the Java TM Standard Edition 6.0.
http://docs.oracle.com/javase/6/docs/technotes/guides/swing/6.0/index.html

Appendix

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<pnml nodeSize="50.0" type="rule" xmlns="http://www.pnml.org/version
    -2009/grammar/pnml">
    <net id="8" nettype="L">
        <page>
            <arc id="21" source="17" target="13">
                <graphics>
                    <dimension/>
                    <position/>
                </graphics>
                <inscription>
                    <text>undefined</text>
                </inscription>
            </arc>
            ...
            <place id="10">
                <graphics>
                    <color b="0" g="0" r="255"/>
                    <position x="68.0" y="23.0"/>
                </graphics>
                <initialMarking>
                    <text>1</text>
                </initialMarking>
                <placeName>
                    <text>with alarm</text>
                </placeName>
            </place>
            <place id="13">
                <graphics>
                    <color b="255" g="0" r="0"/>
                    <position x="281.0" y="215.0"/>
                </graphics>
                <initialMarking>
                    <text>0</text>
                </initialMarking>
                <placeName>
                    <text>awoken</text>
                </placeName>
            </place>
      ...
        </page>
    </net>
    <net id="7" nettype="K">
        <page>
         ...
        </page>
    </net>
</pnml>
```

Figure 3: Left-hand side of rule `ringAlarm` as XML document