

Independence Analysis of Firing and Rule-based Net Transformations in Reconfigurable Object Nets

E. Biermann¹ and T. Modica²

Institut für Softwaretechnik und Theoretische Informatik
Technische Universität Berlin, Germany

¹enrico@cs.tu-berlin.de, ²modica@cs.tu-berlin.de

Abstract: The main idea behind *Reconfigurable Object Nets (RONs)* is to support the visual specification of controlled rule-based net transformations of place/transition nets (P/T nets). RONs are high-level nets with two types of tokens: 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 tokens through the high-level net, and applying net transformation rules to object nets, e.g. to model net reconfigurations. 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.

The problem in this context is to analyze under which conditions net transformations and token firing can be executed in arbitrary order. This problem has been solved formally in a previous paper. In this contribution we present an extension of our RON tool which implements the analysis of conflicts between parallel enabled transitions, between parallel applicable net transformation rules (Church-Rosser property), and between transition firing and net transformation steps. The conflict analysis is applied to a RON simulating a distributed producer-consumer system.

Keywords: Petri nets, net transformation, graph transformation, visual editor, reconfigurable object nets, conflict analysis, independence analysis, Eclipse, GEF

1 Introduction

Modelling the adaption of a system to a changing environment has become a significant topic in recent years. Application areas cover e.g. computer supported cooperative work, multi agent systems, dynamic process mining or mobile ad-hoc networks (MANETs). Especially in the context of our project *Formal modeling and analysis of flexible processes in mobile ad-hoc networks* [PEH07, For06] we aim to develop a formal technique which on the one hand enables the modeling of flexible processes in MANETs and on the other hand supports changes of the network topology and the transformation of processes. This can be achieved by an appropriate integration of graph transformation, nets and processes in high-level net classes.

The main idea behind *Reconfigurable Object Nets (RONs)* is the integration of transition firing and rule-based net structure transformation of place/transition nets (P/T nets) during system simulation. This approach increases the expressiveness of Petri nets and allows a formal description

of dynamic system changes.

RONs are high-level nets with two types of tokens: object nets (which are P/T nets) and net transformation rules (a dedicated type of graph transformation rules). Thus, on the one hand, RONs follow the paradigm “nets as tokens”, introduced by Valk in [Val98], and, on the other hand, extend this paradigm to “nets and rules as tokens” in order to allow for modelling net structure changes (reconfigurations) of object nets. The high-level net constitutes the control frame for object net behavior and rule-based reconfiguration of object nets. Firing of high-level transitions may involve firing of object net transitions, transporting object net tokens through the high-level net, and applying net transformation rules to object nets. Net transformation rules model net reconfigurations such as merging or splitting of object nets, and net refinements.

The formal basis for RONs is given in [HME05], where high-level nets with nets and rules as tokens are defined algebraically, based on algebraic high-level nets [PER95]. The basic idea behind net transformation is the stepwise development of P/T systems by given rules consisting of a left-hand side *LHS* related to a right-hand side *RHS*. Think of these rules as replacement systems where the *LHS* is replaced by the *RHS* preserving a context. Similar to the concept of graph transformations [EEPT06], each application of a rule $r = (LHS \rightarrow RHS)$ to a source net N_1 leads to a net reconfiguration step $N_1 \xrightarrow{r} N_2$, where in the source net N_1 the subnet corresponding to the *LHS* is replaced by the subnet corresponding to the *RHS*, yielding the target net N_2 . Rule-based Petri net transformations have been treated in depth in e.g. [EP04, PU03].

The visual editor and simulator for RONs has been realized as a plug-in for ECLIPSE using the ECLIPSE Modeling Framework (EMF) [EMF06] and Graphical Editor Framework (GEF) [GEF06] plug-ins. In RONs, the algebraic operations defined for rule applications and transition firing are modeled as special RON-transition types which have a fixed firing semantics. It turned out that four RON-transition types for composition, decomposition, firing and rule-based reconfiguration are sufficient to model various interesting examples (Case studies and downloads of the RON tool are available on our RON homepage [RON07]).

Recently, work has been done to formalize independence conditions for reconfigurable P/T systems (i.e. P/T systems together with reconfiguration rules) [EHP⁺07, EEH⁺07]. While independence conditions for two firing steps of P/T-systems are well-known (transitions in conflict), independence of net reconfiguration steps is closely related to local Church-Rosser properties for graph transformations that are valid in the case of parallel and sequential independence of rule-based transformations. In [EEPT06], conditions for two transformation steps are given in the framework of high-level replacement systems with application to net transformations, so that these transformation steps applied to the same P/T-system can be executed in arbitrary order, leading to the same result. In [EHP⁺07] we state under which conditions a net transformation step and a firing step are independent of each other. The subject of this paper is the implementation of the different formal notions of conflict and independence analysis in the RON tool.

The paper is structured as follows: A running example (a distributed producer-consumer system) is introduced in Section 2. Section 3 introduces the RON tool, and Section 4 describes the extension of our tool with conflict and independence analysis based on the theoretical results from [EHP⁺07, EEH⁺07]. These new features of the tool are used to analyze the producer-consumer system. Section 5 concludes the paper with an outlook on future work.

2 Example: A Producer-Consumer System

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 Fig. 1 potential producers and consumers are distributed on different Net places as independent object nets without interaction. Producer nets may fire, e.g. they can produce

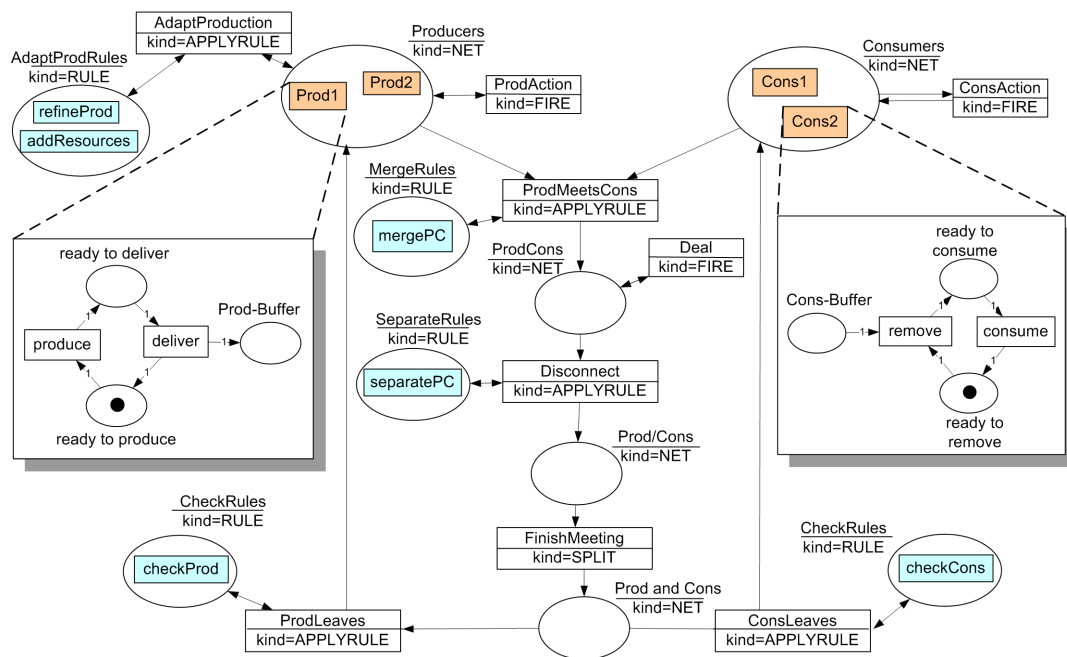


Figure 1: Distributed Producers/Consumers modelled as RON

items and place them on the buffer place. Firing in object nets is triggered by firing a RON high-level (HL) 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. Producer nets can also be refined by firing the RON HL-transition *AdaptProduction* of type APPLYRULE. 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. Note that a transition is preserved by a rule only if its pre- and postdomains are preserved as well.

Rule *refineProd*, depicted in Fig. 2, refines the transition produce by two transitions prepare production and a new transition produce. Transition produce is deleted by rule *refineProd* and generated again with a different pre-domain place. Rule *refineProd* can be applied only once to the same object net since the NAC forbids its application if there is already a place called production prepared in the net. Rule *addResources* (also Fig. 2) replaces the produce transition by two alternative production procedures, each using a different resource.

For producer-consumer interaction, a producer net can be merged with a consumer net by

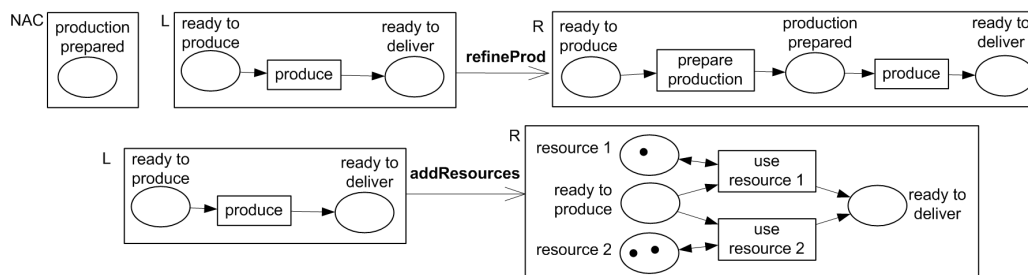


Figure 2: Rules for adapting a producer net

firing the RON HL-transition *ProdMeetsCons* of type APPLYRULE. Rule *merge-PC*, depicted in Fig. 3, glues a producer object net and a consumer object net by inserting a *connect* transition between both buffers. A so-called negative application condition (NAC) forbids the application of the rule if there already exists a *connect* transition. Note that the transition *ProdMeetsCons* controls which producer interacts with which consumer.

HL-transitions of type FIRE trigger the firing of object net transitions. Note that for firing object net transitions, no net transformation rule is applied. The firing semantics of object nets is the usual P/T net behavior. By firing the FIRE HL-transition *Deal*, in the glued 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 HL-transition *Disconnect*. This applies rule *separate-PC* in Fig. 3 to the glued net which deletes the *connect* transition from the net.

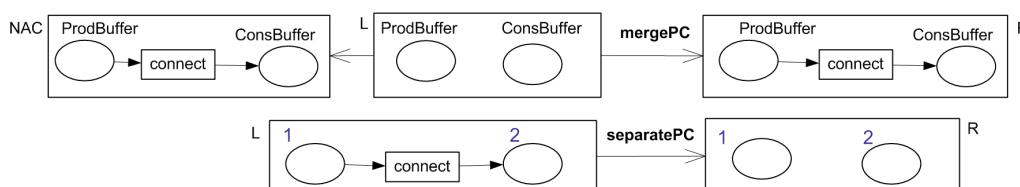


Figure 3: Rules for gluing and for separating a producer and a consumer net

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 these components into two object nets, a HL-transition of type SPLIT has to be fired. Firing RON HL-transition *FinishMeeting* results in two separate object nets on place *Prod* and *Cons*. In the last step, we put the now separated producer and consumer nets back to their initial places. To prevent them to return to “wrong” initial places we again use APPLYRULE HL-transitions. These HL-transitions apply the rules *checkProd* and *checkCons*, respectively, which do not change the object nets they are applied to but simply check them for the occurrence of a *producer* or *consumer* place, respectively. Apart from HL-transitions of type FIRE, APPLYRULE and SPLIT, RONs provide a fourth HL-transition type, called STANDARD (not used in the producer-consumer example). STANDARD HL-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.

3 The RON Environment: Editor and Simulator

The RON environment [BEHM07] is divided into four main components, i.e. the RON tree view based on an EMF model for RONs, and the visual editors for object nets, for transformation rules and for high-level nets with the four HL-transition types FIRE, APPLYRULE, SPLIT and STANDARD. The visual net editors also support the simulation of an edited object net or high-level net. Fig. 4 shows a screenshot of the RON environment showing all views and editors.

RON Tree View. View 1 in Fig. 4 shows the main editor component, a tree view for the complete RON model from which the graphical views can be opened by double-click.

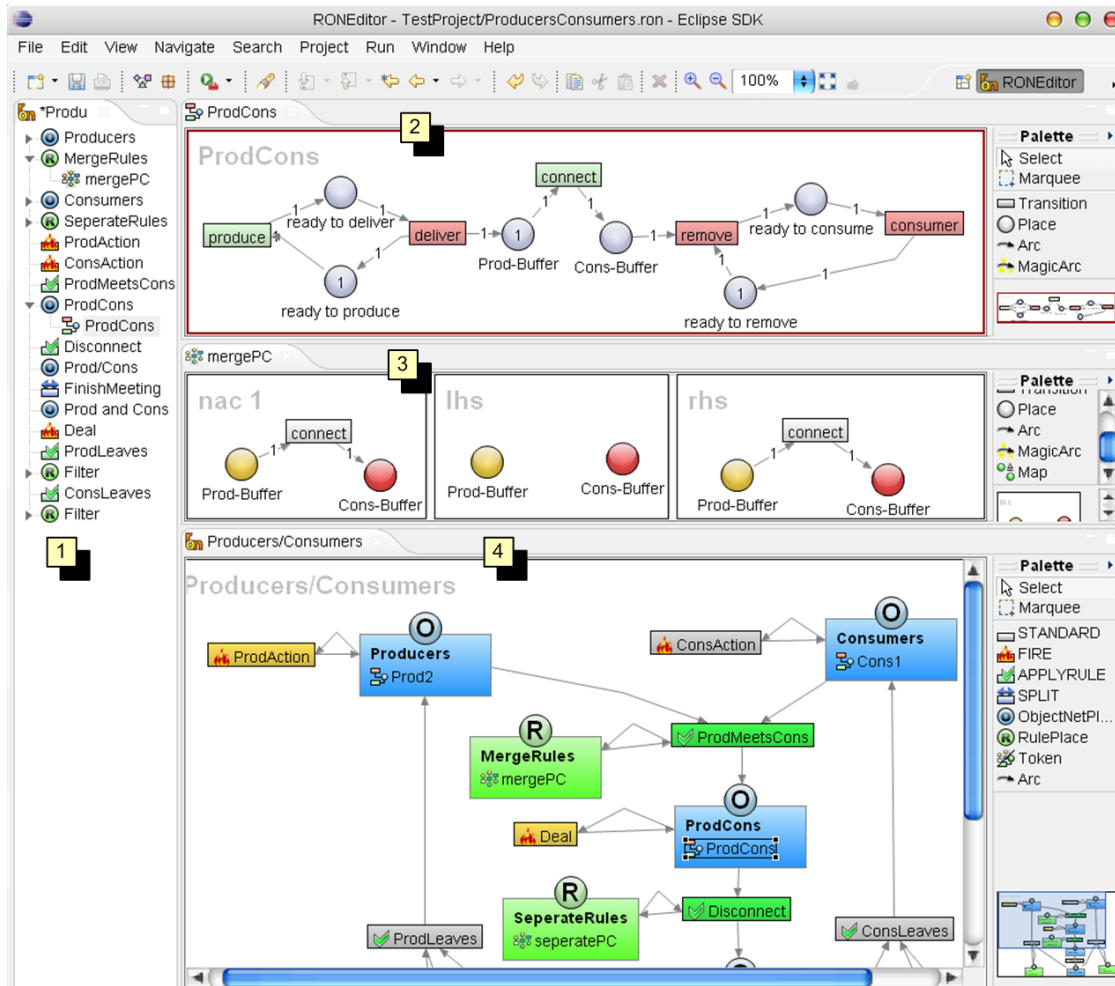






Figure 4: The RON Environment for Editing and Simulating Reconfigurable Object Nets

Object Net Editor. The kernel component is a graphical editor for object nets, i.e. the net tokens on the RON's NET-typed places. This component is actually a place/transition net editor, allowing the simulation of firing transitions. An object editor panel is shown in Fig. 4, View [2], holding the object net *ProdCons*, which models producer-consumer interaction.

Transformation Rule Editor. The editor for transformation rules mainly consists of three editor panels, one for the left-hand side (LHS), one for the right-hand side (RHS) and one for a negative application condition (NAC) (see view [3] in Fig. 4). Each editor panel is basically an object net editor itself, but with the additional possibility to relate the object nets by defining mappings on places and transitions. Mappings are realized by the *mapping tool* of the rule editor that allows the matching of LHS objects to RHS objects to define which objects are preserved by the rule, or to NAC objects to define which objects are connected to additional forbidden objects. In the editor, mappings are shown by object colouring. In order to ensure that the mapping specified by the mapping tool is also a valid Petri net morphism, it is checked for each mapped transition that all places in its pre- (post-)domain in the LHS are mapped to the corresponding places in the pre- (post-)domain of in the RHS. Another restriction is that all rules are injective, so different LHS objects must be mapped to different RHS objects. Note that the object net *ProdCons* in Fig. 4, View [2], is the result of applying rule *mergePC* to two object nets *Prod1* and *Cons2* from the places *Producers* and *Consumers*, respectively. (For the situation before the rule is applied, see Fig. 1).

High-Level Net Editor. A high-level net controls object net behaviour and rule applications to object nets. Such a high-level net is drawn in the high-level net editor panel, shown in Fig. 4, View [4]. Here, NET places carrying object net tokens are blue containers marked by an "O" for *Object Nets*. RULE places carrying transformation rules are green containers marked by an "R" for *Rules*. Each transition type has a special graphical icon as visualization:  for FIRE,  for APPLYRULE,  for SPLIT, and  for STANDARD. Enabled HL-transitions are coloured, disabled ones are gray.

Simulation of RONS. A RON HL-transition is fired when double-clicked. The simulation of firing HL-transitions of kinds STANDARD, FIRE, and SPLIT has been implemented directly in the editor. In order to simulate firing of APPLYRULE HL-transitions, internally the RON editor was extended by a converter to AGG, an engine to perform and analyze algebraic graph transformations [AGG]. If the user gives the command to fire an APPLYRULE HL-transition he has to select the rule and the object net token(s) in the pre-domain the rule should be applied to. This is realized in the user interface shown in Fig. 4, View [4], by ordering the tokens in the corresponding NET and RULE containers in a way that the uppermost tokens are the ones considered by the rule application. Furthermore, the user is asked for a match defining the occurrence of the rule's left-hand side in the selected object net. Optionally, AGG can find or complete partial matches and propose them to the user in the RON editor. With the selected rule, match, and object net AGG computes the result of the transformation which is put on the post-domain places according to the firing semantics explained in Section 2.

A RON HL-transition of type FIRE triggers the firing of an object net transition for an object net

in the FIRE transition's pre-domain. We decided to implement object net transition firing directly in Java instead of modeling firing steps by graph transformation rules and using AGG to compute the successor markings. This design decision is based on the complexity of encoding P/T net behavior in graph transformation systems: there are two possibilities: 1) we could translate each single object net transition into a (model-dependent) graph rule [Erm06]; 2) we could envisage a more general encoding resulting in more than one rule for a single firing step due to the arbitrary number of input and output places for each transition. In case 1) we have the problem that our net transformation rules may delete / add / replace transitions. So, after each rule application we might be forced to adapt the set of model-dependent graph rules representing our object net. In case 2), we cannot make use of AGG's analysis features to find transitions in conflict, since the analysis of graph transformation steps is based on finding critical pairs of rules and needs one firing step to be modeled by one rule only. Thus, it proved to be more natural and straightforward to implement both the object net firing behavior and the check for conflicting object net transitions directly in Java.

4 Extending the RON Environment by Independence Analysis

4.1 Independence in Reconfigurable P/T Systems

In this section we give a brief overview on the different analysis cases and demonstrate them by conflict examples in our producer-consumer system (Fig. 1).

As object nets in RONS can evolve in two different ways (by firing object net transitions and by applying net transformation rules), the notions of conflict and concurrency become quite complex. We illustrate the situation in Fig.5, where we have in the center an object net PN_0 , with marking M_0 and two transitions that are both enabled leading to firing steps $(PN_0, M_0) \xrightarrow{t_1} (PN_0, M'_0)$ and $(PN_0, M_0) \xrightarrow{t_2} (PN_0, M''_0)$, and two transformations $(PN_0, M_0) \xrightarrow{prod_1, m_1} (PN_1, M_1)$ and $(PN_0, M_0) \xrightarrow{prod_2, m_2} (PN_2, M_2)$ via the corresponding rules and matches.

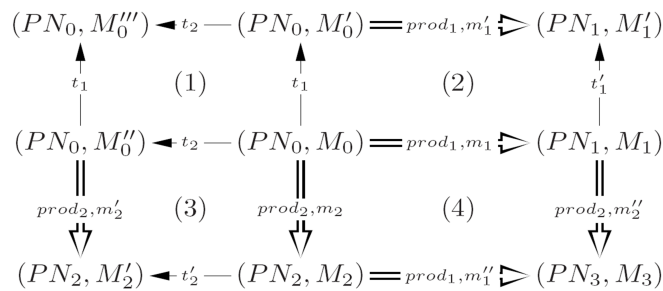


Figure 5: Concurrency in RONS

Hence, we distinguish three kinds of conflicts corresponding to the squares (1), (2/3) and (4):

Transition / Transition: The classic concurrency situation in P/T systems without place capacities regards two transitions with overlapping predomains. Such transitions are in conflict if

they are both enabled and would require more tokens for firing than are available in the current marking. Hence, for square (1), we have the usual condition that t_1 and t_2 need to be conflict free, so that both can fire in arbitrary order or in parallel, yielding the same marking.

Example: Consider the net which results from applying rule *addResource* to the *Producer* net in Fig. 1: here the two transitions *UseResource1* and *UseResource2* are in conflict because there is only one token on the place *ready to produce*.

Rule / Transition: The application of a rule might remove an enabled transition from a given net. In this case one might be able to fire the transition first and apply the rule afterwards, but not the other way round. Vice versa, a rule might require a token on a place which might have been removed by a preceding transition firing step. Hence, for squares (2) and (3), we require parallel independence which allows the execution of the transformation step and the firing step in arbitrary order leading to the same object net. Intuitively, a transition firing and a transformation are parallel independent if the transition is not deleted by the transformation and the successor marking is still sufficient for the match of the transformation (see [EHP⁺07]).

Example: Firing the transition *produce* in net *Prod1* on the RON place *Producers* is in conflict with applying the net transformation rule *refineProducer* to the net *Prod1*. Since the net transformation rule removes the transition *produce*, the firing step cannot take place after the net transformation step. Vice versa, the transition can fire first without disabling the application of rule *refineProducer*.

Rule / Rule: Two rules are in conflict with each other if one of them deletes certain parts of the object net which the other rule needs for its application. Another conflict possibility is the creation of net structures by the first rule which are forbidden by a NAC of the second one. Hence, for square (4), we require parallel independence of both rules. In [EHP⁺07, EEH⁺07] it has been shown that reconfigurable P/T nets fulfill the formal conditions of a weak adhesive HLR category¹. Using such a category not only allows the notions of rules and transformations, but in addition provides a large amount of results such as:

- **Parallelism:** The Church-Rosser Theorem states a local confluence in the sense of formal languages. The Parallelism Theorem states that sequential or parallel independent transformations can be carried out either in arbitrary sequential order or in parallel. In the context of step-by-step development these theorems are important as they provide conditions for the independent development of different parts or views of the system.
- **Concurrency and pair factorization:** The Concurrency Theorem handles general transformations, which may be non-sequentially independent. Roughly speaking, for a sequence there is a concurrent rule that allows the construction of a corresponding direct transformation.
- **Embedding and local confluence:** Further important results for transformation systems are the Embedding, Extension and the Local Confluence Theorems [EEPT06]. The first two

¹ Adhesive High-Level Replacement (HLR) systems have been established as a suitable categorical framework for double-pushout transformations [EEPT06].

allow to embed transformations into larger contexts and with the third one we are able to show local confluence of transformation systems based on the confluence of critical pairs.

Example: Both rule *addResources* and rule *refineProducer* delete the original transition *produce*, when applied to net *Prod1*. Hence, only one of these rules can be applied, disabling the application of the other one (delete-use-conflict). Another example for a conflict would be the rule *mergePC*. The NAC of this rule forbids the application if there already exists a *connect* transition between the two buffers which results in a conflict of the rule with itself. However the structure of the high-level net will prevent two consecutive applications of this rule to the same net.

4.2 Implementation

The editor described in the previous chapter offers support to model reconfigurable object nets and to perform hand-triggered simulation steps.

We extended the basic editing and simulation features of our RON environment by the analysis techniques explained in the previous section. To perform the analysis we employ the attributed graph grammar system AGG [AGG], an environment for the execution and analysis of graph transformations. Since Petri nets and net transformation rules can be simulated as special graphs and graph transformation rules, we use AGG to compute possible rule matches, to apply rules and to perform rule/rule conflict analysis. More specifically, we implemented three methods for the three analysis cases:

1. Method `analyzeTransitionTransition(objectnet)` is implemented directly and searches the given object net for conflicting transitions. The result is a vector of transition pairs which are in conflict. We implemented this analysis directly like the simulation of object net transition firing described in Section 3.
2. Method `analyzeRuleTransition(objectnet, rule)` collects all possible matches of the given rule into the given object net and checks whether there exist enabled transitions in the object net whose firing would delete tokens which are parts of the match. Vice versa, the method checks whether the rule application would delete any of the previously enabled transitions. The resulting pairs, each consisting of an enabled transition and a match from the LHS of a rule to the object net, describe situations where the transition will no longer be enabled after applying the rule. We use AGG for match-computing and implement the formal criteria for conflicts in [EHP⁺07].
3. Method `analyzeRuleRule(rule1, rule2, objectnet)` computes all critical pairs for `rule1` and `rule2` and checks whether the overlapping graphs are parts of the intersection of matches from `rule1` and `rule2` into the object net. Here we use AGG critical pair analysis for this. The result of the computation will be a vector containing pairs of matches of the LHS of both rules into the object net. Semantically each pair describe a situation where the application of the first rule with the first given match will prevent the application of the second rule under the second given match.

Given a specific RON, our editor offers two modes to perform a conflict analysis: single conflict mode and local place mode.

4.3 Single conflict analysis

This mode allows to specify in detail which conflicts are interesting and should be computed. In our producer-consumer system for example, we would like to know either whether our rules for adapting a producer's production workflow might lead to object nets with transition/transition conflicts, or we might be interested in transition/rule conflicts with the transitions of the producer net itself or in possible rule/rule conflicts.

For a transition/transition analysis, the user of the RON environment selects an object net and chooses Analyze Conflicts T/T from its context menu. Window (a) in Fig. 6 shows the result of the T/T analysis of the object net Prod1, where the two transitions in conflict are highlighted.

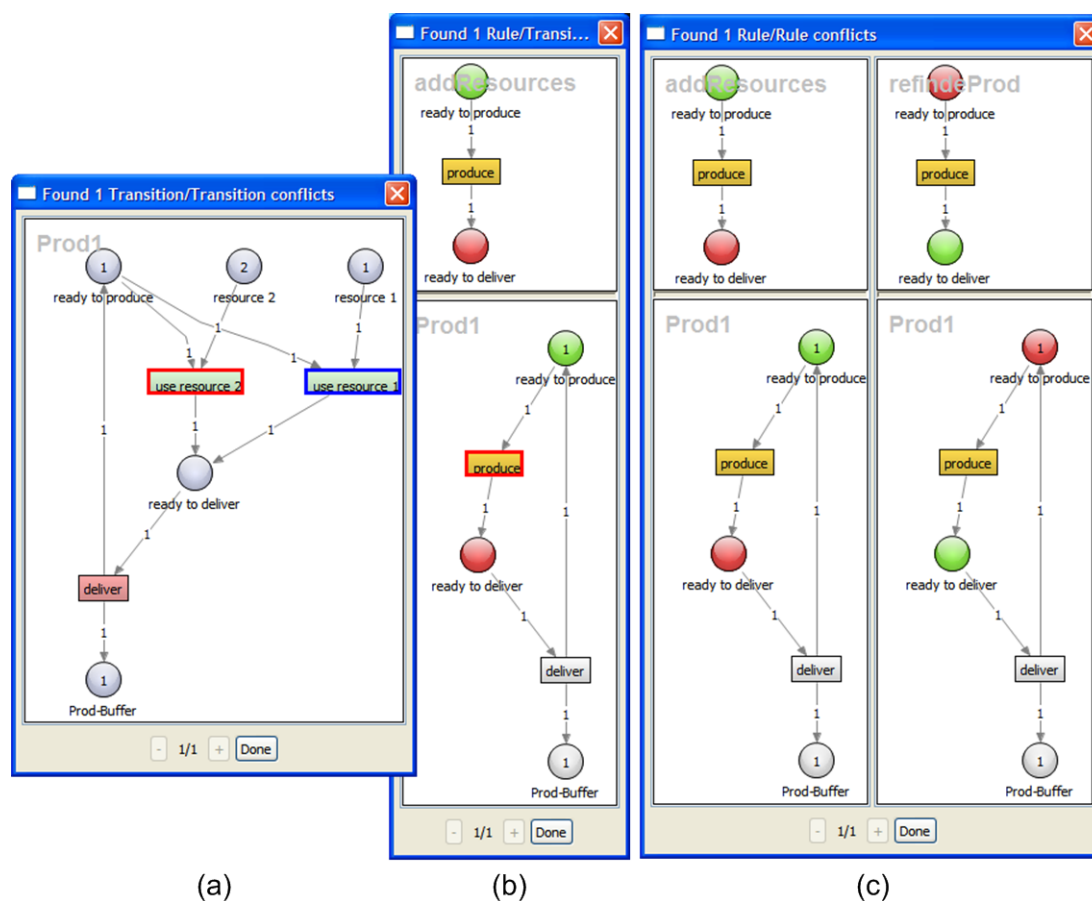


Figure 6: A Transition/Transition Conflict (a), a Rule/Transition Conflict (b), and a Rule/Rule Conflict (c) in the Producer-Consumer System

For a rule/transition analysis, the user of the RON environment selects a rule and an object net which marks a NET place connected by an APPLYRULE transition to the RULE place which contains the selected rule. Afterwards, the item Analyze Conflicts R/T can be chosen from the context menu. Window (b) in Fig. 6 shows that rule *refineProd* and transition *produce* are in R/T conflict with each other because transition *produce* is deleted by the rule. Note that the steps

still can be performed in reversed order: after firing transition *produce*, rule *refineProd* is still applicable because it does not depend on the tokens which are consumed by the transition.

For a rule/rule analysis, two rules and an object net have to be selected in order to be able to evoke the context menu item Analyze Conflicts R/R. Window (c) in Fig. 6 shows that rules *refineProd* and *addResources* (see Fig. 2) are in conflict with each other. Since the matches of the rules overlap, both rules replace the same transition *produce* by a more complex structure.

4.4 Local place conflict analysis

In this analysis mode, a comprehensive direct analysis is evoked for a selected high-level NET place. This means, all possible local conflicts are computed, taking into account the current object nets comprising the marking of the selected NET place, the marking of these object nets, and the rules on RULE places which are connected by an APPLYRULE transition to the selected NET place. Fig. 7 shows the result of a local place analysis, evoked for the NET place Producer in our Producer-Consumer-RON from Fig. 1.

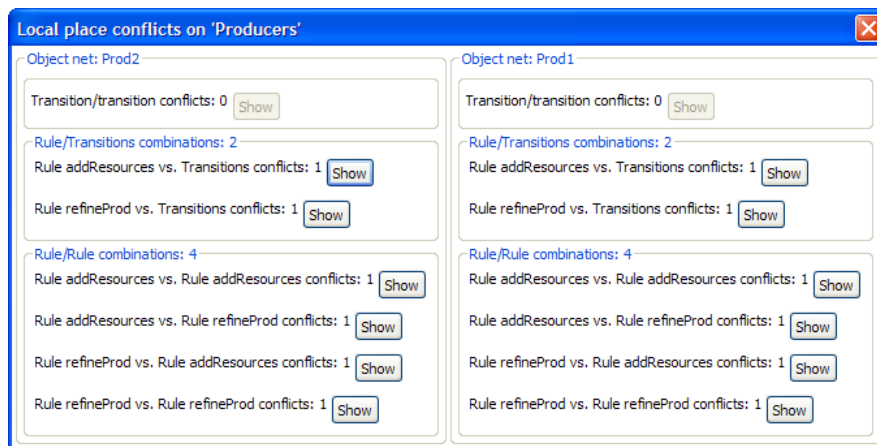


Figure 7: Local Place Analysis Results for the NET Place Producer

For all object nets being tokens on the selected high-level NET place, the calculated conflicts are displayed, sorted by conflict types transition/transition, rule/transition, and rule/rule. Pressing the corresponding Show button shows the respective conflict in detail, i.e. the conflicting transitions are highlighted (Fig. 6 (a)), or a match from a rule is shown together with a highlighted transition this rule is in conflict with (Fig. 6 (b)), or the overlapping matches of two conflicting rules being in conflict into an object net are shown (Fig. 6 (c)).

5 Conclusion

Modeling mobile and distributed systems requires a modeling language which covers both reconfiguration and coordination. RONS meet these conditions. The abstract high-level net controls the flow, selection, manipulation and behavior of object nets (P/T nets) which are tokens on the high-level net places. For object net reconfiguration at runtime, net transformation rules are

used, the application of which is also controlled by the high-level net. In this paper, the RON environment for editing and simulating RONs has been extended with conflict analysis. Apart from the classical situation of two transitions in a P/T net being in conflict, it is now also possible to analyze conflicts between parallel applicable net transformation rules, as well as between enabled transitions and net transformation rules. The knowledge about conflicts helps to detect potential problems in system behavior, e.g. "a produce transition can be deleted even if there are still resources for productions available".

To the best of our knowledge, no other Petri net tool offers the possibility to define and analyze net transformations by rules represented as tokens in a high-level net.

Work is in progress to optimize the analysis result visualization (e.g. by showing the overlapping part of two rule matches only once, and by indicating which are the critical objects causing the conflict). Furthermore, we plan to improve the conflict analysis performance for analyzing also more complex case studies.

Bibliography

- [AGG] AGG Homepage. <http://tfs.cs.tu-berlin.de/agg>.
- [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, 2007. <http://tfs.cs.tu-berlin.de/publikationen/Papers07/BEHM07.pdf>
- [EEH⁺07] H. Ehrig, C. Ermel, K. Hoffmann, J. Padberg, U. Prange. Concurrency in Reconfigurable Place/Transition Systems: Independence of Net Transformations as Well as Net Transformations and Token Firing. Technical report 2007/02, TU Berlin, 2007. <http://iv.tu-berlin.de/TechnBerichte/2007/2007-02.pdf>
- [EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs. Springer Verlag, 2006.
- [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.), *Petri Nets and Other Models of Concurrency. Proceedings of ICATPN 2007*. LNCS 4546, pp. 104–123. Springer Verlag, 2007. <http://tfs.cs.tu-berlin.de/publikationen/Papers07/EHP+07.pdf>
- [EMF06] Eclipse Consortium. Eclipse Modeling Framework (EMF) – Version 2.2.0. 2006. <http://www.eclipse.org/emf>.
- [EP04] H. Ehrig, J. Padberg. Graph Grammars and Petri Net Transformations. In *Lectures on Concurrency and Petri Nets Special Issue Advanced Course PNT*. LNCS 3098, pp. 496–536. Springer Verlag, 2004.

- [Erm06] C. ErmeI. *Simulation and Animation of Visual Languages based on Typed Algebraic Graph Transformation*. PhD thesis, Technische Universität Berlin, Fak. IV, Books on Demand, Norderstedt, 2006.
- [For06] ForMA₁NET. DFG Project, Technical University of Berlin. Formal Modeling and Analysis of Flexible Processes in Mobile Ad-hoc Networks. 2006.
<http://www.tfs.cs.tu-berlin.de/formalnet>
- [GEF06] Eclipse Consortium. Eclipse Graphical Editing Framework (GEF) – Version 3.2. 2006. <http://www.eclipse.org/gef>.
- [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 Verlag, 2005.
<http://tfs.cs.tu-berlin.de/publikationen/Papers05/HEM05.pdf>
- [PEH07] J. Padberg, H. Ehrig, K. Hoffmann. Formal Modeling and Analysis of flexible Processes in Mobile Ad-Hoc Networks. *EATCS Bulletin* 91:128–132, 2007.
<http://tfs.cs.tu-berlin.de/publikationen/Papers07/PEH07.pdf>
- [PER95] J. Padberg, H. Ehrig, L. Ribeiro. Algebraic High-Level Net Transformation Systems. *Mathematical Structures in Computer Science* 5:217–256, 1995.
- [PU03] J. Padberg, M. Urbášek. Rule-Based Refinement of Petri Nets: A Survey. In Ehrig et al. (eds.), *Advances in Petri Nets: Petri Net Technology for Communication Based Systems*. LNCS 2472, pp. 161–196. Springer Verlag, 2003.
- [RON07] Student’s Visual Language Project. TFS, TU Berlin. Reconfigurable Object Nets. 2007.
<http://www.tfs.cs.tu-berlin.de/roneditor>
- [Val98] R. Valk. Petri Nets as Token Objects: An Introduction to Elementary Object Nets. In *ICATPN ’98: Proceedings of the 19th International Conference on Application and Theory of Petri Nets*. LNCS 2987, pp. 1–25. Springer, 1998.