Integrating Verification into the Paose Approach

Marcin Hewelt, Thomas Wagner, and Lawrence Cabac

University of Hamburg, Department of Informatics
http://www.informatik.uni-hamburg.de/TGI/

Abstract. The Paose approach to software engineering combines Petri nets with the paradigm of agent-orientation and utilises the agent metaphor to structure large software systems and their development. Up until now the Paose approach only exhibited aspects of verification in a rudimentary way. This is due to the complexity of the systems to be verified on the one hand, and the expressiveness of the employed Petri net formalism of reference nets on the other hand. This contribution deals with enhancing the tool support for Paose in this regard.
We present how we technically integrate the functionality of LoLA, a sophisticated verification tool, into Renew, the development and runtime environment that backs Paose. Furthermore we sketch how structural aspects of multi-agent systems developed with the agent framework of Mulan can be exploited for verification. The results of the integration are applied in the context of distributed network security for the Herold research project.

Keywords: Development approach, Petri nets, Petri net tools, verification

1 Introduction

In order to successfully tackle the challenges of creating large, complex software systems, the adoption of a development approach is almost mandatory. Paose (**P**etri Net-Based **A**gent-**O**riented **S**oftware **E**ngineering, [3]) is a sophisticated development approach that utilises Petri nets and the multi-agent system (MAS) metaphor to structure software systems and their development. It has been used in a variety of projects and has proven to be suitable for developing large and complex systems with large groups of developers.

One aspect that has, up until now, been largely left out of the Paose approach is the verification of the resulting software systems. The main challenge in this regard is posed by the highly expressive nature of the Petri net formalism of reference nets [12], utilised in Paose. Generally speaking, (low-level) Petri nets offer a formal and clear way of specifying complex systems, while at the same time retaining an intuitive and compact graphical representation. This allows for the use of analysis techniques, which can verify certain properties of a Petri net specification. Reachability of certain markings, boundedness of places and liveness of transitions are just some examples of properties that can be verified

for some Petri net classes. Reference nets, however follow the nets-within-nets paradigm. This allows tokens to be nets themselves and consequently leads to a nested system of nets. In addition tokens can also refer to Java objects which can be manipulated using (Java) inscriptions on the transitions. Finally reference nets employ so-called *test arcs*, which allow concurrent reading access to a resource. These features enable us, on the one hand, to use reference nets as a full-fledged programming language to develop software systems. But on the other hand classical verification algorithms can no longer be directly applied to reference net systems. Adaptions and abstractions have to be undertaken in order to make verification available in the context of Paose. Our current work constitutes the first steps in this endeavour.

Verification of a software system is especially important, when certain properties of the system need to be guaranteed after deployment. Aspects, like deadlock-freeness or boundedness, are often not just simple inconveniences, but can endanger critical components. This is especially true in the case of the Herold project [17], which researches the application of agent-orientation in the domain of distributed network security. In this project we employ the Paose approach to develop a MAS for the management of network security components (e.g. firewalls), because a MAS is capable of capturing the distributed nature of the domain. If the deployed system contained deadlocks or interactions between agents failed to terminate the result could endanger the entire network under management by leaving it open to attacks. Because this has to be prevented under all circumstances, the need for verification support is clear and the integration of such functionality into Paose becomes a priority.

The goal of the particular research presented in this contribution is to enhance the tool support for Paose with regards to verification. The designated tool for Paose is Renew (**Re**ference **ne**t **w**orkshop, [13]), which was especially developed to simulate reference nets, although it is not restricted to this formalism. It has served as the build- and run-time environment for many software systems using reference nets [4, 23].

We have chosen to integrate an existing, external tool for verification, instead of developing this functionality from scratch. This tool is LoLA (Low Level Net Analyser, [21]). It provides the desired verification functionality in an efficient and accessible way. This contribution presents the technical and conceptual enhancements to Paose and Renew, especially the integration of LoLA.

Concerning related work, we researched other verification tools prior to choosing LoLA for the integration. These tools included Netlab (see [8, 18]) and Maria (see [15]). Netlab offers functionality to edit, simulate and analyse place/transition nets. Analysis functions include computation of S- and T-invariants and the reachability/coverability graphs of a given net. Maria (The Modular Reachability Analyzer) employs algebraic system nets. For verification functionality it features reachability analysis and LTL model checking. Both Netlab and Maria are sophisticated verification tools, which provide extensive techniques. In the end, we chose LoLA since it provides comprehensive verification functionality, is quite efficient and its integration is straight-forward.

The paper is structured in the following way. Section 2 discusses the technical aspects of the integration of LoLA into Renew. Section 3 then discusses, how this integration can conceptually be used within Paose. Finally Section 4 summarises and concludes the paper.

2 Technical Integration

Renew, the **Re**ference **Ne**t **W**orkshop, is a high-level Petri net editor and simulator created at the University of Hamburg. It is described in [13] and was developed alongside the reference net formalism [12]. It supports multiple formalisms, including the aforementioned reference nets, workflow nets [7, 22] and place/transition nets. With regards to verification it provides a type and syntax check when editing nets and some effort has been made to support external verification tools (e.g. [16]).

LoLA, the **Lo**w **L**evel Petri Net **A**nalyser, is a verification tool for place/transition and coloured nets described, for example, in [21]. It is based on state space exploration exploiting state-of-the-art reduction techniques and has a high standing in industry and academia, see e.g. [6, 14, 10] for recent applications. Functionality includes the creation of useful additional information about nets, such as reachability and coverability graphs, and the verification of deadlock-freeness, reachability of markings, non-deadness of transitions, state predicates, CTL formulas and other properties.

In order to use LoLA within the Renew tool, we implemented an export from Renew nets to the textual net representation that LoLA requires. Figure 1 gives an example of a simple Renew net and the corresponding textual LoLA net file, as generated by the presented Renew plugin.

Renew nets consist of an unsorted set of figure objects, each of which represents one net element[1]. The figures of a net can be accessed by means of a Java enumeration, which is processed to generate the representation LoLA understands. In order to do so, we have to identify the places, the transitions (together with their pre- and post-set) and the initial marking of the net[2]. These are extracted from the Renew net and written into a new file that complies with the LoLA syntax.

The export of Renew nets is restricted to those concepts that can be expressed within LoLA. Therefore we employ the following *projection*, which strips some semantic information from the reference net. We first need to ignore all inscriptions on places, transitions and arcs (type declarations, guards, synchronous channels, Java code and variables), except for names of places and transitions.

[1] Net elements in this context are not only transitions, places and tokens, but *everything* in the net drawing, including inscriptions, names and comments.

[2] We can restrict ourselves to only these three, because, for now, we are only using the P/T-net functionality of LoLA. However, we continue to work on mapping Renew's type declarations to LoLA's specification format for high-level nets, which allows the declaration of sorts and operations. These can later on be referred in the verification requests.

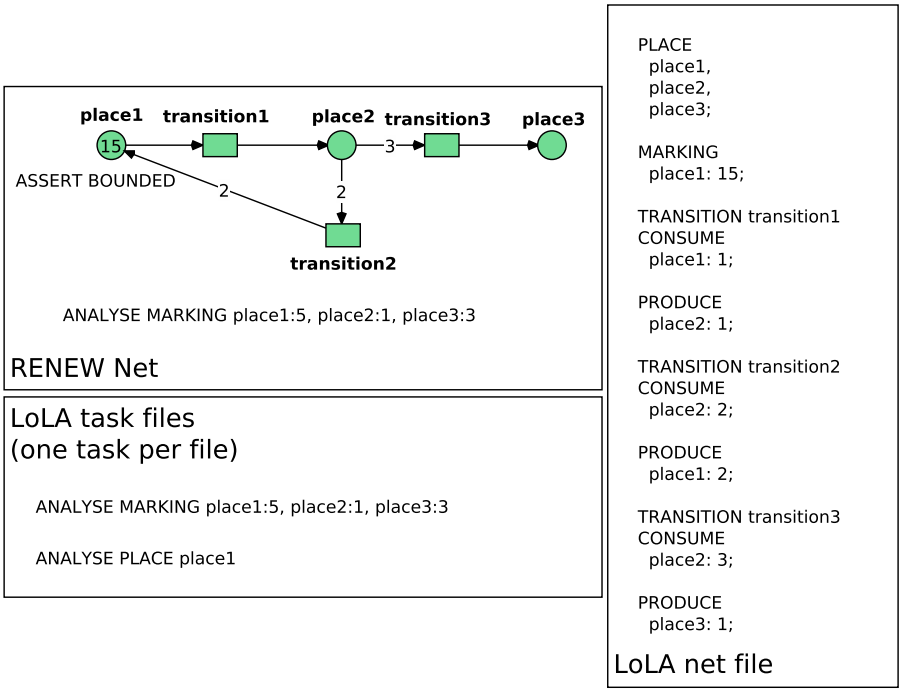

Fig. 1. A Renew net and the corresponding LoLA files.

Then we need to treat object and net tokens (i.e. tokens that represent Java objects or itself nets) as black tokens, wherever they appear. Finally we need to replace test arcs (which are bidirectional) with normal arcs leading to and from the elements involved.

Additionally we have implemented the possibility to annotate verification tasks inside a Renew net. The corresponding text figures are written into separate files, since LoLA requires them to be independent from the textual net representation[3].

Verification tasks for LoLA can be defined in two ways in the Renew tool. They can either be added directly as unconnected text figures, using the LoLA Syntax (e.g. the "ANALYSE MARKING . . . " text in Figure 1, which corresponds to the task of checking if the specified marking is reachable) or, more comfortably, as a special inscription for the concerning element. For example, the inscription of the place *place1* in Figure 1 stating "ASSERT BOUNDED" is automatically translated into a verification task to check the boundedness of the place. The latter is more convenient for the user, since it does not require him to keep track of the names of the net elements for verification.

[3] It is also possible to add some verification tasks directly to the net file. Another viable alternative is to provide the tasks to LoLA as streams.

Providing LoLA functionality with the exported nets from within Renew is handled by calling LoLA externally and feeding the results back into Renew. There are other options how to integrate LoLA, however missing insight into the internal workings of LoLA we were not able to implement them yet. The work on this is still ongoing.

Now that we have outlined *how* LoLA was technically integrated into Renew we can proceed to discuss in *which ways* the functionality is used. Apart from simply being able to specify nets and manually exporting and verifying them, we have decided to provide a more convenient tool set for the Renew user to access the functionality. Therefore we implemented a plugin for Renew that offers the described export capabilities and aggregates some of the functionality offered by LoLA into a graphical view, which makes it comfortably accessible in the Renew context. The work on this plugin is still ongoing. Some features have already been implemented while others are considered work-in-progress.

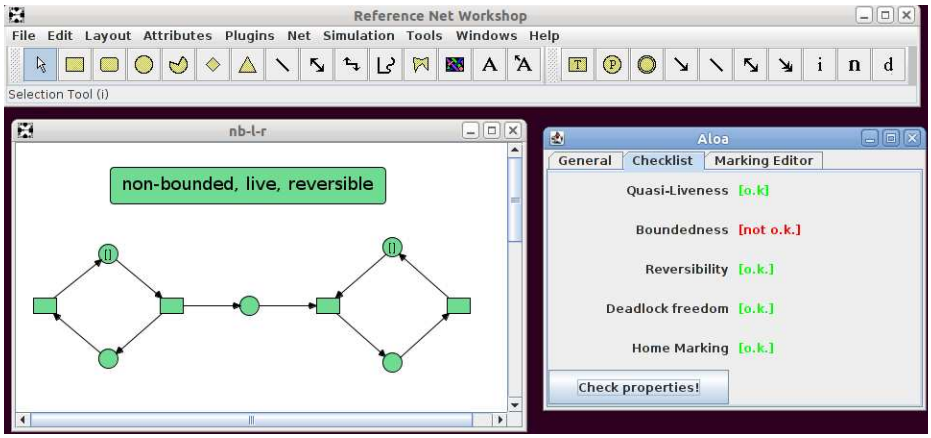

Fig. 2. Screenshot of checklist

On-the-fly check for transitions and places:
The plugin provides the means to check the transitions and places of a net currently being drawn. It will check if transitions are dead and places are bounded. Dead transitions and unbounded places will be marked in the net drawing, so the user can revise the relevant parts of the net.

Net "checklist":
The plugin offers a kind of checklist for a net. In this checklist the properties of quasi-liveness, reversibility, deadlock freedom, existence of home markings and boundedness are displayed. For now, for each of these properties an indicator is given (either positive or negative), but the checklist can be extended to supply additional information, such as concrete home markings

or witness paths. In Figure 2 we included a screenshot of the net checklist that shows the result we obtained for the net on the left.

Reachability/coverability graphs: *Work-in-progress*
The plugin will provide a window in which the reachability and coverability graphs of the net currently being drawn are displayed. This creation of the graphs is initiated manually by the user.

Marking editor: *Work-in-progress*
The plugin will provide an editor for entering a specific marking for the current net. The fields of the editor are updated to display all current places. The user can enter the tokens of each place and then check the marking for example for its reachability (from the initial marking or a second marking input into the plugin window), coverability or status as a home state. Feedback is immediately given to the user within the plugin window.

CTL formula editor: *Work-in-progress*
Similar to the marking editor, the plugin will provide the means to input a CTL formula for the current net, which can then be checked directly from within Renew.

3 Conceptual Integration into Paose approach

Applying Petri nets for software engineering (SE) is a common practice (see e.g. [19]). Nonetheless the Paose approach developed at the theoretical foundations group (TGI) in Hamburg is quite unique, in that it combines Petri nets with the paradigm of agent-orientation to build software systems. We first give a short introduction to the Paose approach and then discuss the role of verification. Finally we show how the results of the technical integration presented in this paper can be applied to the verification of multi-agent systems (MAS).

3.1 Petri net agents

Paose employs concepts from agent-orientation to structure software systems on four different levels, which together form the Mulan architecture (**Mul**ti-**a**gent **n**ets, see [20]). Agents are active, goal-oriented and autonomous software components that only communicate by means of asynchronous messages. They are situated in a logical environment, called platform, that allows for agent migration, facilitates message transport and provides a service registry. The behaviour of agents is autonomous, so unlike function calls, agents choose how to answer to a received message (if at all). For the concrete handling of a message the agent executes a protocol[4] from his knowledge base, which constitutes a repository of known behaviours and data.

Each of the four levels is realised with reference nets, with higher levels embedding the lower ones as net tokens. In this way a platform contains references to agent nets, which, in turn, hold references to protocol nets. Due to the

[4] The term does not refer to a communication protocol that encompasses several agents, but rather stands for a specification of a specific behaviour of an agent.

operational semantics of reference nets, a modelled multi-agent system can be simulated directly.

Being an exhaustive SE-approach PAOSE also encompasses techniques for the specification of interactions between agents, for the definition of ontologies and for the definition of agent roles and knowledge bases, all of which are fully supported by internally and externally developed tools. A model for the development process, as well as an agile method and facilities for the automatic generation of nets and documentation are available. However, we will not go into detail here, but refer the reader to [2] for a complete, detailed presentation of the PAOSE approach.

3.2 Role of Verification

Up until the research presented here, the PAOSE approach did not include formal verification, but instead relied on the sophisticated type and syntax checking offered by RENEW. Because the ontology of a multi-agent application is transformed into a Java type hierarchy, the RENEW tool can provide the developer with type checking for ontology objects. The same is true for the meta-ontology describing platforms, agents, services and messages among other MULAN-relevant objects.

An additional feature at least partially fills the role of verification. During the development process the multi-agent application can be simulated and inspected thoroughly, and can even be modified on the fly during a simulation run [5]. All these measures reduce the cost of development, as they make the process less error-prone.

This contribution addresses the afore-mentioned shortcoming of PAOSE, by providing "classical" Petri net verification. However, we tailor the functionality to the specific context of PAOSE. Due to the Turing completeness of reference nets [11] the properties of liveness and boundedness, among others, are undecidable and thus the problem of general verification has not been tackled before. We have come to realise that although no general verification can be achieved, interesting properties can be validated to some degree. To make reference nets suitable for verification with the LoLA plugin, we employ the projection described in Section 2, which replaces object and net tokens with black tokens and strips away certain inscriptions e.g. Java calls.

It needs to be examined what assertions can be made about the original net. One can observe that if a transition is activated in the original net, it is also activated in the projection[5]. On the other hand, if a transition in the projected net is not activated, it will not be activated under any circumstances in the original net. So we can conclude that if a deadlock occurs in the projected net, it also occurs in the original one. The opposite does not hold true however.

[5] There is a minor exception to this. In the case where several transitions are connected to the same place with a test arc, they can fire concurrently in the original net. Because test arcs are replaced by usual arcs, this no longer can happen in the projected net.

We will now detail how the LoLA plugin can play an important role in supporting the development of multi-agent systems by utilising the structure of Mulan nets for verification.

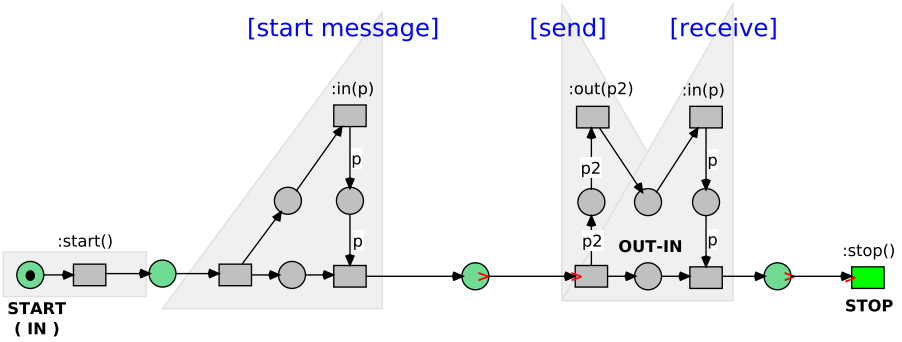

Fig. 3. A Mulan protocol net

3.3 Towards MAS verification

We claim that the reference nets used in the Paose context exhibit some structural features that make them well suited for verification. For this we have to take a more detailed look at the utilised net systems. Agent and platform nets are fixed, this means all agents and platforms share a common structure, while application specific functionality is added through protocol nets and the knowledge base (and application specific infrastructure). While agent and platform nets need to be live, it is essential for protocol nets to terminate, i.e. to reach a terminal state. Some groundwork on this topic has been presented in [9].

Figure 3 illustrates the typical form of a protocol net used in Mulan. Structurally speaking, protocol nets start and end with a transition, which synchronises with the embedding agent net. The initial synchronisation causes the newly instantiated protocol net to be put on a special place in the agent net, dedicated for ongoing conversations. Messages an agent receives can be directly passed to an ongoing conversation, if their type matches the type expected in the protocol net. The final synchronisation removes the instance of the protocol net from the conversation place, once it is finished (meaning it reached its terminal state). In this way, it is essentially a workflow net [22]. Using this perspective the closure of the workflow net would be provided by the agent, which is capable of re-instantiating the protocol/workflow.

While a conversation is ongoing, agents send and receive messages in their corresponding protocol nets that together form the conversation. The platform net is also involved in the conversation, because it routes the message to its destination. Deadlock-freeness and thus termination can only be verified under the

assumption that the communication partners and the message transport service behave well. On the other hand, stripping the inscriptions from the protocol net, one can verify deadlock-freeness of the projected protocol net independently of its environment. Using the LoLA tool in combination with the existing type check we can enhance verification in cases where alternative behaviours are chosen according to message content (which is the standard case).

Some other verification tasks for multi-agent systems spring to mind and will be the focus of future work:

- *Do agents possess matching protocols so that they can communicate?*
  Given two, or more, agents this task can determine if they possess matching protocols. This means that an outgoing message in one agent's protocol has to correspond to an incoming message in another agent's protocol. In terms of practical implementation this boils down to matching the outgoing message as a type to the incoming message as a type, possibly by merging the protocol nets.
- *Does an agent have required protocols to be embedded on a platform?*
  While similar to the previous task, this task is a bit more general. Instead of explicitly matching a number of agents for communication, this task determines if an agent possesses the required functionality (i.e. protocols) needed to be active on a specific platform. This task is more comprehensive and complex, since it addresses further aspects concerning functionality, not just communication. In practical terms, more nets have to be matched and certain assumptions about the content of protocols (for example through type matching) have to be verified.
- *Is the overall communication structure deadlock-free?*
  Another related task is to check the overall interaction and communication structure of the multi-agent application for deadlocks. Taking the agent interactions into account we could generate a simplified net version of the overall communication, assuming that the internal workings of the agents are correct and do not deadlock. The check would then again be confined to the messages and their types, but instead of having a particular set of agents and protocols in the focus, it would examine the overall multi-agent application.

These last points dealing with the orchestration of protocol nets illustrates the intimate relation between protocol nets in the Mulan architecture and open nets of [24]. Open nets are used in the context of service synthesis, composition and orchestration and are supported by an exhaustive tool suite[6] that includes LoLA. Therefore we started our investigation into verification tools with LoLA and plan to extend it to other tools from the suite.

4 Conclusion

We have presented a way to make use of an existing tool as a plugin for Renew and how the Paose approach profits from this integration. The technical real-

[6] Available online as open source from http://service-technology.org/.

isation was easily achieved due to the flexible plugin mechanisms provided by Renew. So far we have used the tool to check for very basic properties, but as was advocated in Section 3, we are working on applying it to multi-agent systems. Conceptually we gain enhanced tool support for Paose, making it easier to develop functioning-as-designed multi-agent applications.

Concerning future work, the efforts of enhancing Renew and Paose with the help of LoLA are still ongoing. We are looking at further ways of extending the functionality of the LoLA plugin for Renew and are researching additional areas of our multi-agent systems for verification, as was outlined in Section 3.3. As for the technical aspects, a promising avenue of work is to also incorporate the support for sorts and operations offered by LoLA into the plugin. This will improve the modelling effort for verification and, in turn, make using LoLA within Renew more comfortable.

From a practical point of view, we will use the results we have obtained within the Herold research project[7]. Herold deals with distributed network security and the management of network security components (see [1] for more information). The complex processes within the project, coupled with the critical nature of the application domain, require the use of verification techniques in order to ensure correct execution of the produced multi-agent system. The LoLA plugin will play an invaluable role for the verification aspects of the project.

References

1. Simon Adameit, Tobias Betz, Lawrence Cabac, Florian Hars, Marcin Hewelt, Michael Köhler-Bußmeier, Daniel Moldt, Dimitri Popov, Jose Quenum, Axel Theilmann, Thomas Wagner, Timo Warns, and Lars Wüstenberg. Herold - agent-oriented, policy-based network security management. In *Future Security, 5th Security Research Conference, Berlin*, 2010.
2. Lawrence Cabac. *Modeling Petri Net-Based Multi-Agent Applications*, volume 5 of *Agent Technology – Theory and Applications*. Logos Verlag, Berlin, 2010.
3. Lawrence Cabac, Till Dörges, Michael Duvigneau, Christine Reese, and Matthias Wester-Ebbinghaus. Application development with Mulan. In Daniel Moldt, Fabrice Kordon, Kees van Hee, José-Manuel Colom, and Rémi Bastide, editors, *Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE'07)*, pages 145–159, Siedlce, Poland, June 2007. Akademia Podlaska.
4. Lawrence Cabac, Michael Duvigneau, Michael Köhler, Kolja Lehmann, Daniel Moldt, Sven Offermann, Jan Ortmann, Christine Reese, Heiko Rölke, and Volker Tell. PAOSE Settler demo. In *First Workshop on High-Level Petri Nets and Distributed Systems (PNDS) 2005*, Vogt-Kölln Str. 30, D-22527 Hamburg, March 2005. University of Hamburg, Department of Computer Science.
5. Lawrence Cabac, Daniel Moldt, and Jan Schlüter. Adding runtime net manipulation features to MulanViewer. In *15. Workshop Algorithmen und Werkzeuge für Petrinetze, AWPN'08*, volume 380 of *CEUR Workshop Proceedings*, pages 87–92. Universität Rostock, September 2008.

[7] The Herold project is funded by the German Federal Government, through its Ministry of Education and Research (Grant No. 01BS0901).

6. Sebastian Hinz, Karsten Schmidt, and Christian Stahl. Transforming BPEL to Petri nets. In Wil M. P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, *Proceedings of the Third International Conference on Business Process Management (BPM 2005)*, volume 3649 of *Lecture Notes in Computer Science*, pages 220–235, Nancy, France, September 2005. Springer-Verlag.
7. Thomas Jacob. Implementierung einer sicheren und rollenbasierten Workflowmanagement-Komponente für ein Petrinetzwerkzeug. Diploma thesis, University of Hamburg, Department of Computer Science, Vogt-Kölln Str. 30, D-22527 Hamburg, 2002.
8. Stephan Kleuker. *Formale Modelle der Softwareentwicklung. Model-Checking, Verifikation, Analyse und Simulation.* Vieweg+Teubner, Wiesbaden, 2009.
9. Michael Köhler, Daniel Moldt, and Heiko Rölke. Liveness preserving composition of behaviour protocols for Petri net agents. Report of the research program: Act in Social Contexts FBI-HH-M-316/02, University of Hamburg, Department of Computer Science, June 2002.
10. Milos Krstic, Eckhard Grass, and Christian Stahl. Request-driven GALS technique for wireless communication system. In *Proceedings of the 11th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 76–85, Washington, DC, USA, 2005. IEEE Computer Society.
11. Olaf Kummer. Undecidability in object-oriented Petri nets. *Petri Net Newsletter*, 59:18–23, 2000.
12. Olaf Kummer. *Referenznetze.* Logos Verlag, Berlin, 2002.
13. Olaf Kummer, Frank Wienberg, Michael Duvigneau, Michael Köhler, Daniel Moldt, and Heiko Rölke. Renew – the Reference Net Workshop. In Eric Veerbeek, editor, *Tool Demonstrations. 24th International Conference on Application and Theory of Petri Nets (ATPN 2003). International Conference on Business Process Management (BPM 2003)*, pages 99–102. Department of Technology Management, Technische Universiteit Eindhoven, Beta Research School for Operations Management and Logistics, June 2003.
14. Niels Lohmann, Oliver Kopp, Frank Leymann, and Wolfgang Reisig. Analyzing BPEL4Chor: Verification and participant synthesis. In Marlon Dumas and Reiko Heckel, editors, *Web Services and Formal Methods*, volume 4937 of *Lecture Notes in Computer Science*, pages 46–60. Springer-Verlag, 2008.
15. Marko Mäkelä. Maria: Modular reachability analyser for algebraic system nets. In *Proceedings of the 23rd International Conference on Applications and Theory of Petri Nets*, ICATPN '02, pages 434–444. Springer-Verlag, 2002.
16. Marco Mascheroni, Thomas Wagner, and Lars Wüstenberg. Verifying reference nets by means of hypernets: A plugin for Renew. In Michael Duvigneau and Daniel Moldt, editors, *Proceedings of the International Workshop on Petri Nets and Software Engineering, PNSE'10, Braga, Portugal*, number FBI-HH-B-294/10 in Bericht, pages 39–54, Vogt-Kölln Str. 30, D-22527 Hamburg, June 2010. University of Hamburg, Department of Informatics.
17. Daniel Moldt, Michael Köhler-Bußmeier, Axel Theilmann, Simon Adameit, Tobias Betz, Lawrence Cabac, Florian Hars, Marcin Hewelt, Dimitri Popov, José Quenum, Thomas Wagner, Timo Warns, and Lars Wüstenberg. Modelling distributed network security in a Petri net and agent-based approach. In Jürgen Dix and Witteveen Cees, editors, *Multiagent System Technologies. 8th German Conference, MATES 2010,Leipzig, Germany, September 27-28, 2010. Proceedings*, volume 6251 of *Lecture Notes in Artificial Intelligence*, pages 209–220, Berlin Heidelberg New York, September 2010. Springer-Verlag.

18. Philipp Orth and Dirk Abe. Rapid Control Prototyping petrinetzbasierter Steuerungen mit dem Tool NETLAB. *at-Automatisierungstechnik*, 54, Issue: 5:222–227, 2006.
19. Wolfgang Reisig. Petri nets in software engineering. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Applications and Relationships to Other Models of Concurrency*, volume 255 of *Lecture Notes in Computer Science*, pages 62–96. Springer, 1987.
20. Heiko Rölke. *Modellierung von Agenten und Multiagentensystemen – Grundlagen und Anwendungen*, volume 2 of *Agent Technology – Theory and Applications*. Logos Verlag, Berlin, 2004.
21. Karsten Schmidt. LoLA: A Low Level Analyser. In Mogens Nielsen and Dan Simpson, editors, *Application and Theory of Petri Nets 2000: 21st International Conference, ICATPN 2000, Aarhus, Denmark, Proceedings*, volume 1825 of *Lecture Notes in Computer Science*, pages 465–474. Springer-Verlag, June 2000.
22. Wil M. P. van der Aalst. Verification of workflow nets. In Pierre Azéma and Gianfranco Balbo, editors, *ICATPN*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer, 1997.
23. Thomas Wagner. Prototypische Realisierung einer Integration von Agenten und Workflows. Diploma thesis, University of Hamburg, Department of Informatics, Vogt-Kölln Str. 30, D-22527 Hamburg, 2009.
24. Karsten Wolf. Does my service have partners? *Transactions on Petri Nets and Other Models of Concurrency*, 2:152–171, 2009.