An Extensible Editor and Simulation Engine for Petri Nets: RENEW

Olaf Kummer, Frank Wienberg, Michael Duvigneau, Jörn Schumacher,
Michael Köhler, Daniel Moldt, Heiko Rölke, and Rüdiger Valk

University of Hamburg, Department of Computer Science
Vogt-Kölln-Str. 30, D-22527 Hamburg, *surname*@informatik.uni-hamburg.de

Abstract. Renew is a computer tool that supports the development
and execution of object-oriented Petri nets, which include net instances,
synchronous channels, and seamless Java integration for easy modelling.
Renew is available free of charge including the Java source code.
Due to the growing application area more and more requirements had
to be fulfilled by the tool set. Therefore, the architecture of the tool
has been refactored to gain more flexibility. Now new features allow for
plug-ins on the level of concepts (net formalisms) and on the level of
applications (e.g. workflow or agents).

Keywords: Reference nets, RENEW, plug-in, architecture, high-level
Petri nets, nets-within-nets, tool, integrated development environment

1 Introduction

Petri nets are a well established means to describe concurrent systems. Object-
oriented analysis and programming techniques are currently the de-facto stan-
dard of software development. This has lead to the invention of a variety of
object-oriented Petri net formalisms. One of the most widely used object-oriented
languages is Java, which features a relatively clean language design, good porta-
bility, and a powerful set of system libraries.

RENEW [13] is a Java-based high-level Petri net simulator that provides a flex-
ible modelling approach based on reference nets. The publicly available version
1.6 has undergone several improvements and contains many features. However,
different application areas require different functionality of the tool set. There-
fore, we made a major redesign of the tool which will be called RENEW 2.0 or for
short RENEW in the following and is the topic of this paper. Before it is released,
RENEW is undergoing a final comprehensive test in a larger project to ensure
the same high quality as the previous versions.

RENEW is an integrated environment that gives the user access to all required
tools including an editor. Underlying design principles of the editor are: *easy to
use* interface, minimal input for the user, direct relation to the functionality and
provision of a high-level formalism. Fig. 1 shows a screen shot where the main
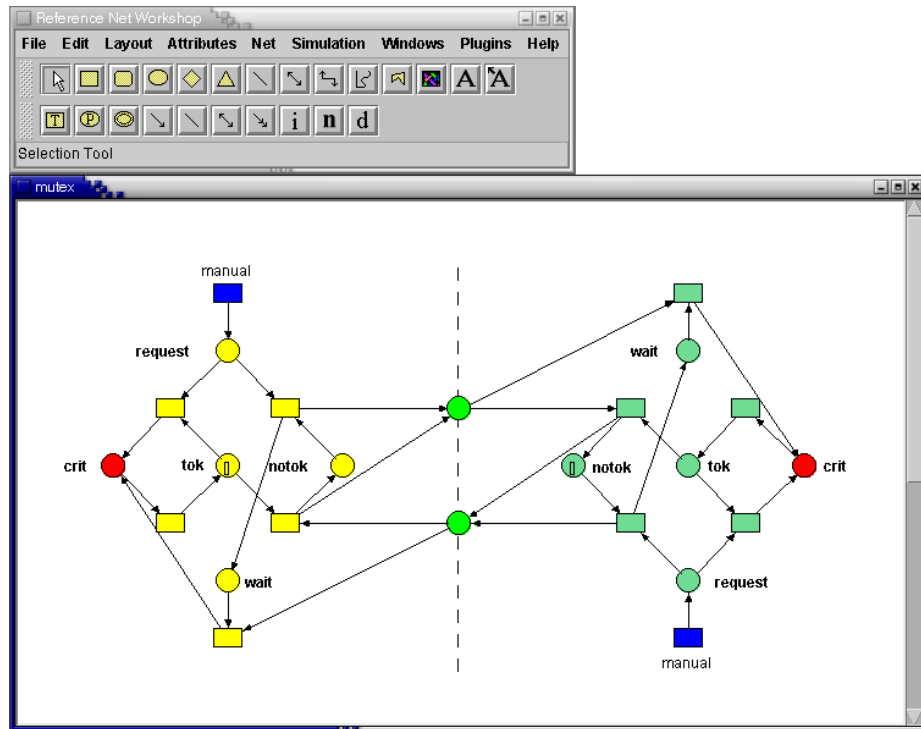tool bars and a net drawing are visible.

Fig. 1. A screen shot of the RENEW application

The buttons in the visible tool bars mostly deal with the graphical creation of net diagrams. They help the user to create and layout new nets and to illustrate them with additional graphics, as well as to edit existing nets. The creation of new nets is supported by functions that allow the easy creation of new places, transitions, arcs, and textual inscriptions. With a simple mouse drag you can create a new node *and* an arc that connects it to an existing node.

Besides the graphical editor, the tool allows for the use of Petri nets in server-side processes without the graphically animated token game. Based on customised compilers different formalisms were supported in previous versions of RENEW. Several extensions of the Petri net formalism had been integrated that included clear arcs, flexible arcs and inhibitor arcs. The expressiveness of timed Petri nets, where time stamps are attached to tokens and to input and output arcs, was provided by the tool, too.

Our goal for the new Version 2.0 is to add flexibility and extensibility as a key feature at the level of supported formalisms and at the user level to the tool. The desire for more flexibility comes from the different applications of RENEW. Especially its use in the areas of workflow and agent systems required various extensions to the tool set. The major refactoring of RENEW (see [14]) allows such extensions to be plugged in and out of the application without overloading

the tool. Besides the flexibility itself, no new features are added to the tool for now. But the process uncovered some existing hidden features.

In the sequel we will first describe the characteristics of the main formalism, the reference nets. Then a sketch of the underlying architecture of the tool and its extensions to previous versions is given. Besides the architecture some new features are described. To show the applicability some already existing plug-ins are discussed. A summary and an outlook will round out the presentation of the tool.

2   Reference Nets

Reference nets (defined in [10]) start out as ordinary higher-order nets, based on Petri nets whose arcs are annotated by a special inscription language. We choose Java expressions as the primary inscription language, but we add tuples to the language and make some simplifications. As usual, variables are bound to values, expressions are evaluated, and tokens are moved according to the result of arc inscriptions. Additionally, there are also transition inscriptions.

- Guards, notated as `guard` *expr*, require that the expression evaluates to `true` before the transition may fire.
- *expr*=*expr* can be inscribed to a transition, but it does not imply assignment, but rather specification of equality. Variables must be bound to a fixed value during the firing of a transition. This means that modify assignments like `x=x+1` do not make sense.
- Java expressions might be evaluated, even when it turns out that the transition is not enabled and cannot fire. This causes problems for some Java method calls, therefore the notation `action` *expr* is provided. It guarantees that *expr* will be evaluated exactly once, a feature that is needed when side effects (e.g. changes to Java objects) come into play.

When a net is constructed, it is merely a net template without any marking. But it is then possible to create a net instance from the template. In fact, an arbitrary number of net instances can be created dynamically during a simulation. Only net instances, not the templates, have got a marking that can change over time.

- A net instance is created by a transition inscription of the form *var*:`new` *netname*. It means that the variable *var* will be assigned a new net instance of the template *netname*. The net name must be uniquely chosen for each template that we specify.

It should be noted that RENEW supports the concepts of *Nets-within-Nets* (see [16]) which is a major research topic of our group. In order to exchange information between different net instances, synchronous channels were implemented. They provide greater expressiveness compared to message passing. Unlike the synchronous channels from [2], which are completely symmetric, we will impose a direction of invocation. The invoking side of a channel will be known as the downlink, the invoked side is called the uplink.

- An uplink is specified as a transition inscription :*channelname(expr,...)*. It provides a name for the channel and an arbitrary number of parameter expressions.
- A downlink looks like *netexpr:channelname(expr,...)* where *netexpr* is an expression that must evaluate to a net reference. The syntactic difference reflects the semantic difference that the invoked object must be known before the synchronisation starts.

To fire a transition that has a downlink, the referenced net instance must provide an uplink with the same name and parameter count and it must be possible to bind the variables suitably so that the channel expressions evaluate to the same values on both sides. The transitions can then fire simultaneously. Note that channels are bidirectional for all parameters except the downlink's *netexpr*. E.g., a net might pass a value through the first parameter of a downlink and the called net might return a result through the second parameter of the same channel. This is similar to the undirected parameters of Prolog predicates, but different from the invocations of Cooperative Nets by Sibertin-Blanc (see [15]).

A transition may have an arbitrary number of downlinks, but at most one uplink. (Again, the similarity with horn clauses in Prolog does not occur by chance.) A transition without uplinks will be called a spontaneous transition, because it may fire without being invoked by another transition. A transition may have an uplink and downlinks at the same time. A transition may synchronise multiple times with the same net and even with the same transition.

3 Architecture and Extensibility

In the last year, the tool has undergone a large architectural refactoring. The application has been decomposed into several components, each component is seen as a "plug-in". The intention of this decomposition is an increased flexibility and extensibility of the tool so that new features can be added in an easy way. However, in the first step, the architectural changes do not add new features to the tool. But some existing features that were hidden in the system can now be accessed more easily.

All components of the application are now managed by a central plug-in system. The system allows for addition and removal of plug-ins at runtime. Many plug-ins provide extension interfaces where additional features of other plug-ins can be registered.

The basic functionality of RENEW is provided by the plug-ins depicted in Fig. 2. To run a simulation of some reference nets without graphical feedback, the plug-ins Util, Core, Simulator and Formalism are needed. The configuration of the simulation can then be done by setting several properties on the command line.

The main component is the Simulator plug-in. It comprises the simulation engine and some packages providing in- and output abstractions for the simula-tion. Nets can be fed to the simulation engine by using the so-called shadow-API, which abstracts from the layout information and retains only the topological net
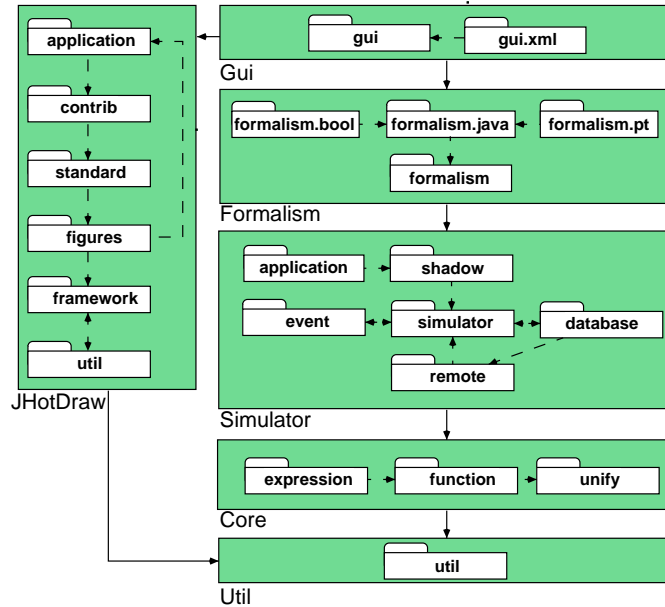
Fig. 2. Basic plug-ins of RENEW: packages and dependencies (based on [14, p. 91])

structure. This provides a convenient way to create non-graphical nets algorithmically.

Other packages provide interfaces to observe and influence the state of a running simulation. Also persistent storage of the simulation state in a relational database engine like mySQL or Oracle is provided, so that the simulation can continue from the last consistent state after a power failure (see [7]).

If graphical feedback is desired, the plug-ins Gui and JHotDraw can be added. This can happen while a simulation is running, the user interface will attach itself to the running simulation engine. It is also possible to quit the user interface without terminating the simulation.

The simulation engine can be enabled for remote access, so that the graphical user interfaces of independent RENEW instances on other hosts can inspect and influence the simulation state. This separation of simulation engine and user interface has already been introduced in RENEW 1.5, but now its use has become easier. If a simulation is started from within the user interface, a configuration dialog (see Fig. 3) provides a convenient way to set the properties of the next simulation run.

In the design of RENEW it has been clear from the very beginning that the tool must be able to handle different formalisms by just exchanging a compiler component. A compiler converts a shadow net into an internal format by parsing the textual net inscriptions. A custom compiler can reuse the existing classes of
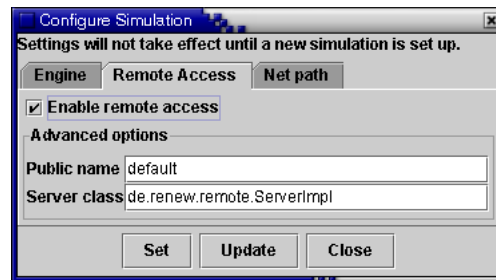
Fig. 3. Simulation configuration dialog

the Core plug-in which provides predefined building blocks for expressions, places, arcs, and so on. The compiler just needs to compose the compiled nets out of these classes. Custom classes are only needed for non-standard net components.

Some exemplary formalisms existed in previous versions of the tool, but without much help for their configuration. Now the Formalism plug-in provides a registry for known compilers. All registered compilers are presented to the user by a menu in the user interface, if the Gui plug-in is loaded (see Fig. 4). As a formalism can be accompanied by new graphical elements to be used in net drawings, the plug-in can add tool bars or menu entries to the application's user interface. New formalisms can be easily included in the system by creating a new plug-in that comprises the compiler and related classes.

A refactoring of the simulation engine has clarified the separation of formalisms (compilers), common Petri net abstractions (like transitions and places), and the core engine, which in fact is just a generic binding search algorithm (see [10]). With some changes to the user interface, we will soon have a real multi-formalism tool that supports a different formalism for each net template within one simulation. This means that some parts of a system can be modelled as simple P/T-nets and some parts as coloured Petri nets.

4 Development support

Renew is used in our group to develop and run medium-sized applications implemented in a mixture of reference nets and Java code (e.g. roundabout 100 nets and the same number of Java classes). The mix of reference nets and Java
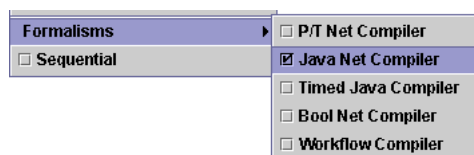

Fig. 4. Compiler selection menu

is well supported because any Java object can be used as a token in nets and because any net can be wrapped by a so-called "stub" to make it appear as a Java class. When mixing nets and classic code, the developer can benefit from both sides: he has the clarity of nets at his hand, when it comes to concurrency and synchronisation, and he has access to the rich functionality of the Java class libraries.

Pursuant of additional ease for application development, many other features have found their way into RENEW throughout the last releases. Among the developer-driven features there are:

- Nets can be loaded not only from files, but also from arbitrary locations by specifying their URL. Many common ways of retrieving files can be used this way, as long as the Java runtime system knows a way to access the specified resource (e.g. via HTTP over Internet or by extraction out of `.jar` files).
- It is not necessary to open all nets belonging to an application prior to the start of the simulation engine. A net loading mechanism fetches nets on demand during the running simulation. The nets can be loaded from shadow net files without showing up in the graphical user interface. The graphical representation of the net can also be loaded on demand when the user inspects a net instance during the simulation.
- An implementation of last year's PNML-standard draft (*P*etri *N*et *M*arkup *L*anguage, see [6]) is provided, allowing import and export of nets from or to other tools.
- Breakpoints can be attached to transitions or places, causing the simulation to halt when a transition fires or the marking of a place changes.
- Transitions can be marked as "manual". The simulation engine will not fire these transitions during automatic runs. But such a transition can be fired explicitly from the graphical user interface at any time (given it is activated).
- The interactive debugging of complex mixed systems using nets and Java code is supported by an in-depth inspection of Java token objects.

Of course, the database backing and remote access features mentioned in the previous section also have their origins in application development. All these features are of great value for our existing plug-ins, like those presented in the next section.

5   Plug-In Examples

This section describes some current plug-ins that have been built for RENEW. The plug-ins presented in the following are not all available to the general public, some being experimental or highly application specific. But they can serve as examples for the extensibility of the tool.

Workflow. Workflow support requires a workflow management system. A standard has been defined by the WfMC (Workflow management coalition, see [17]).

The Workflow plug-in extends Renew in two ways: It uses an interface provided by the engine to monitor and control the activation of transitions in general, and it provides a specialised workflow compiler to the Formalism plug-in. Some sketches of the resulting workflow engine have been given in [8].

With the plug-in, Renew can serve as a development environment and execution engine for workflow systems, where the firing of transitions is coupled with the execution of workflow tasks. Overall, the suitability of Renew for workflow support and the tool's extensibility is demonstrated by the Workflow plug-in.

Multi-Agent Systems. Since multi-agent systems (MAS) are inherently concurrent, Petri nets recommend themselves to be used in that area. In our group, we have designed a MAS architecture called Mulan (short for *Mul*ti *A*gent *N*ets, see [9]) with reference nets. This architecture is implemented with a mixture of nets and Java code by the Capa plug-in (*C*oncurrent *A*gent *P*latform *A*rchitecture, presented in [4]). This plug-in mostly runs within the Renew simulation engine, but in addition provides customised graphical representations for special token objects.

The MulanViewer plug-in reflects the high potential of the whole architecture. It enhances the user interface of Renew by an overview window that summarises the state of certain places in certain nets of the Capa plug-in. The viewer also provides fast navigation through the important net instances of the MAS to inspect the token game.

A different kind of plug-in is the Diagram plug-in: It extends the editor component of Renew by a special mode to draw Agent Interaction Protocols (AIPs) as they are proposed by AUML (see [12]). These protocol diagrams are not Petri nets, but the plug-in prototype can generate net structures reflecting the control flow corresponding to the AIP for each participating agent.

Modelling Support. The task of modelling benefits enormously from elaborated tool support. Possible features are the direct input of model elements or flexible menu bars that adapt to the current task. The menus of the former Renew were static and could only be extended by releasing a new version of the tool. By making the menu bar flexible, we can add any tool bar the installed plug-ins provide.

An example is the NetComponents plug-in, as presented in [1]. It allows to add new tool bars to the user interface at runtime, where each button creates a small component of net structure that is often used during a development process. Different component sets are available, depending on the application area: the tool bar from [1] comprises patterns for creating Mulan/Capa protocol nets. A different tool bar covers some workflow patterns, as presented in [11].

6   Conclusions

Renew, based on its new architecture, provides all major features of the old versions up to now. This is the result of a refactoring process. We concentrated

on the aspect of flexibility to provide a better tool set to the RENEW-users without introducing new features besides the plug-in mechanism. However, as a result of the new architecture as a side effect the tool can now easily be extended based on the plug-in concept.

The underlying algorithms and the support of concurrency, which is even present inside the tool itself, has not been tackled in this paper. However, the basic architecture and important newly added features with respect to usability have been presented.

The plug-in architecture now allows for the easy use of multiple formalisms and net variants. Basis of this is the clean separation between the underlying simulation engine and the formalisms on top. The former versions allowed the use of customised compilers that had to be specified when starting the tool. Now the refactored tool allows for an easy formalism setup between simulation runs. In the near future, we will be able to combine different formalisms within one simulation environment.

The multi-formalism simulation can be useful within our agent-oriented modelling approach, where we consider open systems. Each agent can come along to an agent platform, which we consider to be a RENEW-simulator, with its own formalism. This depends on the level of abstraction when building the models. The idea is to use the weakest formalism (with respect to the expressibility) to reach a certain task during modelling. In combination with the support for the PNML-standard this allows the modeller to exchange the nets with other tools adequate for the chosen formalisms. Especially in the area of verification a weaker formalism commonly allows the use of more powerful tools.

An important outcome of the work done for the new architecture is that we are able to build a powerful agent development and simulation tool set. The integration into the FIPA-standard environment (see [5]) has been functionally reached with [4], followed by an integration into the Agentcities context (see [3]). The functionality built into plug-ins will enhance the flexibility of our tool set to become a full agent execution environment which is autonomous, open, concurrent, adaptive and mobile.

In the future the number of plug-ins will increase. Any programmer may build those plug-ins that support his own needs in the directions of a development environment that is based on high-level Petri nets, as we do. Furthermore, others might realize that they need their own special plug-in to support a certain net formalism or even other formalisms. They can then easily add their special drawing tools and compilers. The main restriction is that they have to stick to our architecture and the interfaces provided so far. Due to the underlying design principles the coupling, however, will be loose.

References

1. L. Cabac, D. Moldt, and H. Rölke. A Proposal for Structuring Petri Net-Based Agent Interaction Protocols. In W. v.d. Aalst and E. Best, editors, *Applications and Theory of Petri Nets. ICATPN 2003, Eindhoven. Proceedings*, number 2679 in LNCS, pages 102–120. Springer, 2003.

2. S. Christensen and N. Damgaard Hansen. Coloured Petri Nets Extended with Channels for Synchronous Communication. Report DAIMI PB–390, Aarhus University, 1992.

3. M. Duvigneau, M. Köhler, D. Moldt, C. Reese, and H. Rölke. Agent-based Settler Game. In *Agentcities Agent Technology Competition, Barcelona, Spain. Proceedings*, February 2003.

4. M. Duvigneau, D. Moldt, and H. Rölke. Concurrent Architecture for a Multi-agent Platform. In F. Giunchiglia, J. Odell, and G. Weiss, editors, *Agent-Oriented Software Engineering III. Workshop, AOSE 2002, Bologna.*, number 1420 in LNCS, pages 59–72. Springer, 2003.

5. Foundation for Intelligent Physical Agents (FIPA). `http://www.fipa.org`.

6. J. Billington et. al. The Petri Net Markup Language: Concepts, Technology, and Tools. In W. v.d. Aalst and E. Best, editors, *Applications and Theory of Petri Nets. ICATPN 2003, Eindhoven. Proceedings*, number 2679 in LNCS, pages 483–505. Springer, 2003.

7. T. Jacob, O. Kummer, and D. Moldt. Persistent Petri Net Execution. *Petri Net Newsletter*, 61:18–26, October 2001.

8. T. Jacob, O. Kummer, D. Moldt, and U. Ultes-Nitsche. Implementation of Workflow Systems using Reference Nets – Security and Operability Aspects. In K. Jensen, editor, *4th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools.* University of Aarhus, 2002. Report DAIMI PB-560.

9. M. Köhler, D. Moldt, and H. Rölke. Modelling the Structure and Behaviour of Petri Net Agents. In J.-M. Colom and M. Koutny, editors, *Application and Theory of Petri Nets. ICATPN 2001, Newcastle upon Tyne. Proceedings*, number 2075 in LNCS, pages 224–241. Springer, 2001.

10. O. Kummer. *Referenznetze.* Logos-Verlag, Berlin, 2002.

11. D. Moldt and H. Rölke. Pattern Based Workflow Design Using Reference Nets. In W. v.d. Aalst, A. ter Hofstede, and M. Weske, editors, *Business Process Management. BPM 2003, Eindhoven. Proceedings*, number 2678 in LNCS, pages 246–260. Springer, 2003.

12. J. Odell, H. Van Dyke Parunak, and B. Bauer. Extending UML for Agents. In G. Wagner, Y. Lesperance, and E. Yu, editors, *Agent-Oriented Information Systems. Workshop at the 17th National Conference on Artificial Intelligence (AAAI), AOIS 2000, Austin. Proceedings*, pages 3–17, Austin, TX, 2000.

13. Renew – The Reference Net Workshop. WWW page at `http://renew.de/`. Contains the documentation for Renew and an introduction to reference nets.

14. J. Schumacher. Eine Plugin-Architektur für Renew. Konzepte, Methoden, Umsetzung. Diplomarbeit, Universität Hamburg, October 2003.

15. C. Sibertin-Blanc. Cooperative Nets. In R. Valette, editor, *Application and Theory of Petri Nets. ICATPN '94, Zaragoza. Proceedings*, volume 815 of *LNCS*, pages 471–490. Springer, 1994.

16. R. Valk. Petri nets as token objects: An introduction to elementary object nets. In J. Desel and M. Silva, editors, *Application and Theory of Petri Nets. ICATPN '98, Lisbon. Proceedings*, number 1420 in LNCS, pages 1–25. Springer, June 1998.

17. Workflow Management Coalition Homepage. URL: `http://www.wfmc.org/`, 2003.