Renew – The Reference Net Workshop

Olaf Kummer Frank Wienberg Michael Duvigneau
Michael Köhler Daniel Moldt Heiko Rölke
Universität Hamburg, Fachbereich Informatik
Vogt-Kölln-Straße 30, D-22527 Hamburg
{kummer,wienberg,duvigneau,koehler,moldt,roelke}@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.

1 Introduction

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

Renew [5] is a Java-based high-level Petri net simulator that provides a flexible modelling approach based on reference nets. Renew is an integrated environment that gives the user access to all required tools. Fig. 1 shows a screenshot where the main toolbar and a net drawing are visible.

The visible toolset mostly deals with the graphical creation of net diagrams. It helps the user to create and layout new nets and to illustrate them with additional graphics, as well as the editing of 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 simple mouse drag you can create a new node *and* an arc that connects it to an existing node.

The simulation engine is shielded from the user interface by means of the so-called `shadow`-API, which abstracts from the layout information and retains only the topological net structure. This provides a convenient way to create non-graphical nets algorithmically. These nets can then be compiled and executed easily. In fact, it is conceivable to use Petri nets in server-side processes, e.g., for the control of workflow processes.

The compilation is done by a compiler that can be plugged into the existing application. It converts the shadow net into an internal format by parsing the textual net inscriptions. This allows the easy creation of new net formalisms, because only the compiler needs to be rewritten. E.g., a compiler for boolean Petri nets was created in a single day. A compiler for feature structure nets was written, too, which supports artificial intelligence applications.

A custom compiler can reuse the existing simulation engine by composing the compiled nets out of predefined building blocks for expressions, places, arcs, and so on. Custom classes are only needed for non-standard net components. An architecture overview is given in Fig. 2.

In the sequel we will first describe the main characteristics of the net formalism, before elaborating on the changes that were applied to Renew since its release 1.4, which was presented at the ICATPN'2001.

2 Reference Nets

Reference nets (defined in [4]) 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.

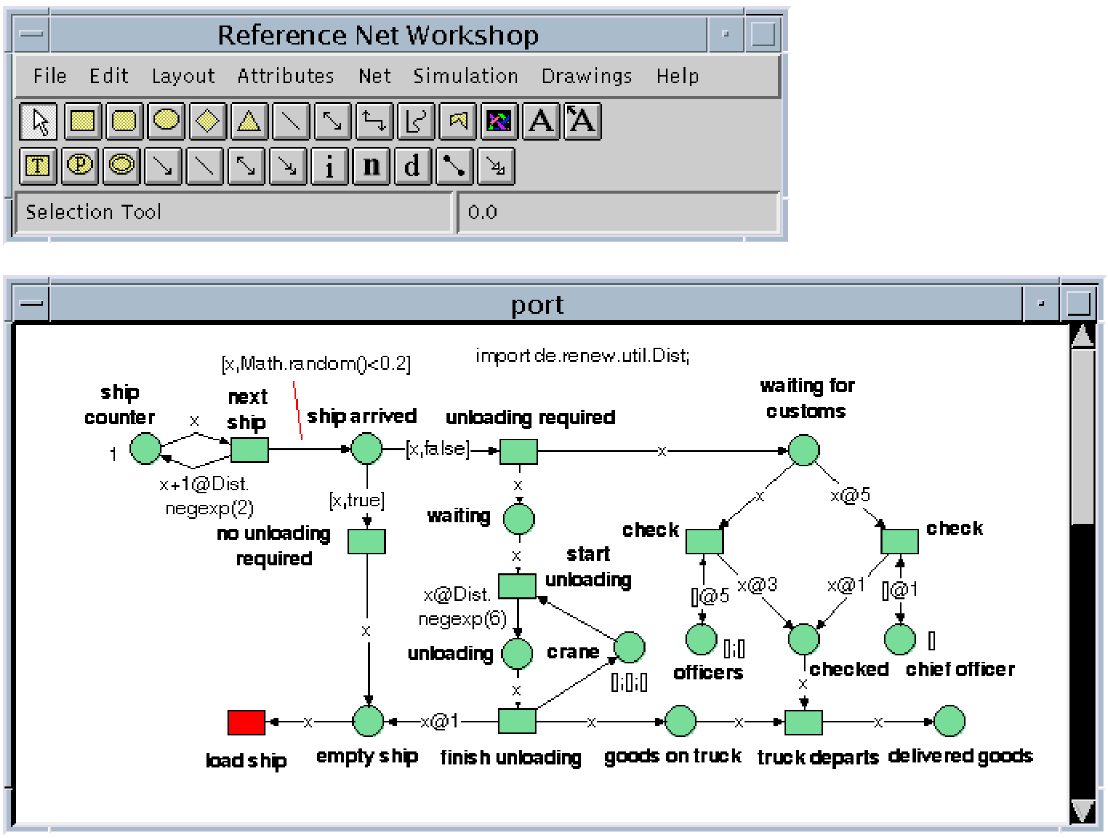

Figure 1: A screenshot of the Renew application

- 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.

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 [1], 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*`,`*expr*`,...)`. It provides a name for the channel and an arbitrary number of parameter expressions.

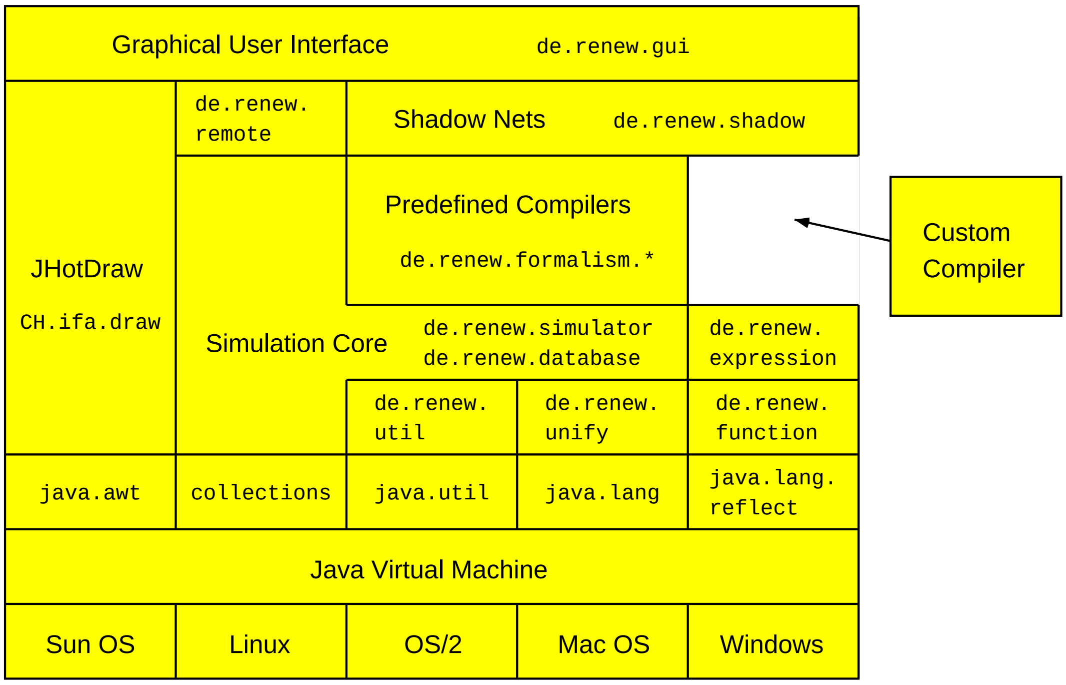

Figure 2: The basic architecture of Renew

- A downlink looks like `netexpr:channelname(expr,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 although channels are asymmetric, they are still bidirectional for all parameters except the invoked net. 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 quite different from the invocations of Cooperative Nets as described by Sibertin-Blanc in [6].

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 synchronize multiple times with the same net and even with the same transition.

3 Recent Improvements

Already release 1.4 of Renew featured several extensions of the Petri net formalism that included clear arcs, flexible arcs and inhibitor arcs. Also already available was the expressiveness of timed Petri nets, where time stamps are attached to tokens and to input and output arcs. The interactive debugging of complex net systems was already supported by breakpoints and an in-depth inspection of Java token objects.

The current release 1.6 comes with several technical improvements suitable for production environments. These are remote simulation access, database backing and a net loading mechanism.

A new remote layer separates the graphical user interface from the simulation engine. Now a stand-alone simulation engine can run permanently on one host without graphical feedback or interaction. Users can connect from a Renew application on any host to the running simulation engine and watch the token game. Most interactions with the token game that are available in a local Renew engine can also be used on the remote simulation.

The Renew simulation engine can be coupled with existing relational database applications like mySQL or Oracle. The marking of the net system can be made persistent in the database, each firing of a transition is accompanied by an equivalent database transaction. The persistence provided by the database backs the Renew simulation engine, so that in case of a system failure the simulation can be continued from the last recorded state (see [2]).

Since applications built with reference nets easily grow in the number of involved net templates, the user's screen could get cluttered with net windows. A net loading mechanism allows the loading of nets on demand during the running simulation. The nets can be loaded from shadow net files without interfering with the user's graphical display. The graphical representation of the net can also be loaded on demand when the user inspects a net instance during the simulation.

These productive features were added to Renew mostly as support for another extension. Renew can serve as development environment and execution engine for workflow systems, where the firing of transitions is coupled with the execution of workflow tasks. Some sketches of the resulting workflow engine have been given in [3]. This extension is not made available for download yet, but is shown on the demonstration session.

References

[1] Søren Christensen and Niels Damgaard Hansen. Coloured Petri nets extended with channels for synchronous communication. Technical Report DAIMI PB–390, Aarhus University, 1992.

[2] Thomas Jacob, Olaf Kummer, and Daniel Moldt. Persistent Petri Net Execution. *Petri Net Newsletter*, 61:18–26, October 2001.

[3] Thomas Jacob, Olaf Kummer, Daniel Moldt, and Ulrich Ultes-Nitsche. Implementation of workflow systems using reference nets – security and operability aspects. In Kurt Jensen, editor, *Fourth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, Ny Munkegade, Bldg. 540, DK-8000 Aarhus C, Denmark, August 2002. University of Aarhus, Department of Computer Science. DAIMI PB: Aarhus, Denmark, August 28–30, number 560.

[4] Olaf Kummer. *Referenznetze.* Logos-Verlag, Berlin, 2002.

[5] Renew – the reference net workshop. WWW page at `http://renew.de/`. Contains the documentation for Renew and an introduction to reference nets.

[6] Christophe Sibertin-Blanc. Cooperative nets. In Robert Valette, editor, *15th Int. Conf. on Application and Theory of Petri Nets*, volume 815 of *Lecture Notes in Computer Science*, pages 471–490. Springer-Verlag, 1994.