Net Components Revisited

Lawrence Cabac, Michael Duvigneau, Heiko Rölke

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

Summary. This paper introduces net components [1], illustrates the concept, design, and usage of net components, and gives some experiences made in several software development projects over the last years.

Net components are subnets that are meant to be combined with each other to form a Petri net. By this component-based approach of constructing Petri nets, the drawing of the nets is facilitated and the Petri nets are structured in a way that they become easily readable and unified. For the Mulan protocols, a set of net components exists that are called Mulan net components. Besides the fact that they provide the basic functionality regarding the communication that is done by Mulan protocols, also some simple patterns and programming artefacts are introduced into the development process.

Keywords: components, net components, pattern, Petri nets, Renew, structured Petri nets, teaching

1 Introduction

High-level Petri net formalisms like Colored Petri Nets [6] or Reference Nets [9] offer modelling constructs and abstractions comparable to basic programming constructs of high-level programming languages: data types and variables, sequences, branches and iterations, code encapsulation with restricted access, and so on. Additionally, Petri nets allow for an elegant and intuitive modelling of concurrency, which is neglected in most programming language of widespread use. Therefore, is it possible to use an appropriate Petri net formalism not only for modelling and analyzing of systems – as it is usually done – but also for the implementation. This has the additional advantage, that the model can be transformed into an implementation without a change of formalism. The Petri net formalism serves as programming language, the models of the design stage of the software development process can directly be

used – maybe in an refined version – as the implementation.[1] This approach of *implementation through specification* has been sketched in [2, 16].

During the last years we used Reference Nets together with the tool Renew in several advanced student projects (lasting half a year each) that are subordinated to the main topic of Petri net based software development. The projects' application domains ranged from a small stock exchange game over our multi-agent system implementation itself to full-scaled multi-agent applications like a settler game or a distributed workflow management system. Due to an increasing number of participating students, the latter projects grew to a rather big size for this kind of projects[2], so that the demand for conceptual and tool support of the implementation process grew over the years. This paper describes net components, an important means to speed-up the modelling process and standardize the structure of the net models at the same time.

In Section 2, we introduce the concept of net components. An application-specific set of net components – the Mulan protocols – serves as exemplary implementation of the concept. It is presented in Section 3 together with the corresponding toolset. Our experiences after several years of usage of the Mulan protocol net component toolset are the topic of Section 4. Finally, Section 5 summarizes our results and gives an outlook on future work.

2 Concept and Design of Net Components

Net components (NC, see [1] and [4]) are subnets, which can be composed or combined to from a large net. They should provide general functionality that can be commonly used. However, a set of net components is only meant to serve for a special subset of similar Petri nets. Only if many similar nets of the same category are produced, the effort of designing a set of net components is rewarded and the benefits of net components are exploitable.

Net components should not be confused with the software engineering viewpoint on components of large software parts. Instead, the net components presented in this work resemble the control structures of structured programming languages like cycles or conditional statements.

[1] In the Petri net formalisms mentioned above, Petri nets are simulated or interpreted, not compiled. Therefore, one has to accept a loss of performance. This is no problem at early stages of a software development process, when rapid prototyping means rapid implementation of a prototype that offers as much of the desired functionality as possible. In later design stages performance plays a more important role. Using, for example, Reference Nets as modelling formalism, it is possible to switch over to a Java implementation in an easy and organized way. This should not be deepened here.

[2] For example, the multi agent system settler game implementation comprises several hundred different nets as well as several hundred Java classes (mostly data types, but also function libraries).

2.1 Notions

A net component is a set of net elements that fulfills one basic task. The task should be so general that the net component can be applied to a broad variety of nets. Furthermore, the net component can provide additional help, such as a default inscription or comments. One of the used components contains a predefined but adjustable declaration node. In a formal way, net components can be seen as transition-bordered subnets. This suits the notion of net components covering tasks.

Every net component has a unique geometrical form and orientation that results from the arrangement of the net elements. A unique form is intended so that each net component can easily be identified and distinguished from the others. The geometrical figure also holds the potential to provide a defined structure for the Petri net.

In the default implementation of the net components used so far, places are added at the outward connecting transitions (*interface place*) for convenient net component connection. Only one arc has to be drawn to connect one net component with another. This is a simple and efficient method that also emphasizes the control flow by the fact that these simple connecting arcs transport by default only black tokens. The connection of net components is provided by this place, which should not contain other token then anonymous ones.

Direct data exchange between net components is not desired in order to guarantee an easy-connecting interface. Instead, data is handed to the data-containing places via virtual places. [3]. When the programmer adds an appropriate virtual place to the net component, data can be transferred separately from the control flow to the transition that uses a variable. In the usual case the data is read through an additional test arc. The test arc allows for concurrent read-only access on the data. If the data is to be modified a reserve arc prohibits concurrent access, and if the data is not needed anymore it can also be consumed by a directed arc. Data is handled and stored in a data block, which is located above the control flow part of the protocol. Annotations of the data-containing places should be adjusted to the appropriate name as well as the annotations of the corresponding virtual place.

2.2 Structure of Net Components

Net components are transition-bordered[4] subnets that can be composed to form a larger Petri net. Their purpose is to provide pre-manufactured solutions

[3] *Virtual places* can be regarded as references to the original places. Another well known name for this is *fusion-place*. In Renew virtual places can be identified by their doubled outline.

[4] Note that, as mentioned above, for easing the practical use each output transition is supplemented with appropriate output places. Doing so, the net components can be connected just by drawing arcs between such an additional output place and an input transition of another net component.

of reoccurring challenges. Moreover, they also impose their structure onto the constructed net. Like a snowflake's structure is determined through the underlying structure of the water molecules, the net is structured by the net components.

Through their geometric form, the net components are easily identified in a larger net. This adds to the readability of the net and to the clearness of the overall structure of the net, which is an accumulation of substructures.

Jensen [6] describes several design rules for Petri net elements, which are based on work that has been done by Oberquelle [13]. These rules are concerned with the ways of drawing figures and give general advice for Petri net elements such as places, transitions and arcs. They are also concerned about combinations and arrangement of the elements.

Net components extend the rules by giving developer groups the chance to pre-define reusable structures. Within the group of developers, these structures are fixed and well known, although they are open for improvements. Conventions for the design of the code can be introduced into the development process, and for developers it is easy to apply these conventions through the net component-based construction. Furthermore, the developing process is facilitated and the style of the resulting nets is unified. Once a concrete implementation of net components has been incorporated and accepted by the developers, their arrangements (form) will be recognized as conventional symbols. This makes it easier to read a Petri net that is constructed with these net components. Moreover, to understand a net component-based net it is not necessary to read all its net elements, it is sufficient to read the substructures.

2.3 Net Components versus (Design) Patterns

Patterns and design patterns, also in the form of workflow patterns, have been discussed excessively in the past. They are useful elements in software (respectively workflow or business process) development. They help developers naming and communicating important (abstract) concepts in the development phase of process/workflow/software engineering. Often they are visualized with graphical methods, e.g. UML for design patterns or Petri nets for workflow patterns. Many patterns are simple, some are complex. Many of the common patterns are omnipresent in current development. For instance the *Observer* design pattern is implemented in the Java *Listener* interface. Also other patterns are so common that they are sometimes not recognized anymore. Nevertheless, a widespread agreement can be observed that patterns are useful in the development of complex systems.

Net components are instantiations of patterns. They can be instances of commonly known patterns, realizations of trivial patterns or even build from scratch for a special purpose. In addition to the advantages that are offered by patterns, net components have more advantages that are mentioned in the following. These result in part from the fact that net components are

instantiations and in part from the fact that they are implemented and meant for application in Petri nets.

Concreteness: In opposition to a pattern a net component is a concrete graphical component that has a fixed graphical representation.

Composability: The net components can be composed with each other to form a Petri net. This results partly from the concreteness. However, this has also be considered during the design phase of the net components

Convention: The analogue representation of a net component allows the developers to recognize the implemented pattern in a net component in different environments.

Congenerousness: Since the pattern and its implementation are modeled in the same graphical language there exists no breach between model and implementation.

2.4 Requirements for Net Components

Net components have to be designed for their purpose. In any case, different kinds of nets require different sets of net components. However, within a set of net components that has been designed for a special purpose, the net components should remain as generic as possible. The net components should be easily inter-connectible so that the construction of nets is facilitated. Furthermore, net components should be designed to represent one syntactical entity. This means that a net component should represent one basic task that is decomposable. A net component should be easily identifiable to a reader of the net. This can be achieved by arranging the net elements in a unique (geometrical) form.

Finally, a net component should also provide solutions for challenges that frequently reoccur. Functionality that is thus implemented once can be used again without going through the process of 'low level' implementation again. Altogether these characteristics for net components are shown in Table 1:

Characteristic	**Benefit**
Generic character	Net components are broadly applicable.
Interconnectivity	Net components are easily combinable.
Closeness	Net components have clear semantics.
Unique form	Net components are easily identifiable.
Located in repository	Net components provide pre-manufactured but adaptable solutions.

Table 1. Criteria for net components.

Especially when nets are produced in large numbers while designing Petri net-based applications, some advantages can be seen in a component-based approach. The net components contribute not only to a clear structure of the nets, but also to a faster development of applications.

3 The Mulan Net Components

Mulan (MULti-Agent Nets, see [8, 16]) is a concept model and framework for multi-agent systems desined using reference nets. To build a multi-agent application based on Mulan, numerous protocol nets that implement agent behavior and interactions have to be drawn.

A set of net components for Mulan protocols exists [1, chapter 4.3] that has been successfully tested and used in a teaching project (*settler 2*, [11]) of our group at the University of Hamburg. The set of Mulan net components has been used excessively, and a large number of net component-based Mulan protocols have been designed during the project.

The Mulan net components provide the basic functionality to construct protocols [7]. Those protocols that are constructed with the help of the Mulan net components are not restricted to the exclusive use of net components; however, it is unnecessary to use non component-based net elements, because the set is self-contained. The set provides structures for control flow management that includes alternatives, concurrency, cycles and sequences. In addition, the functionality for exchanging data is provided, which offers receiving or sending of messages. Furthermore, some basic protocol related structures are provided that handle the starting and the stopping of the protocols.

3.1 Generic Net Components

A selection of Mulan net components is presented in this section. This is done to demonstrate what kind of functionality they provide for Mulan protocols and their form – by which they are identified – is introduced. In this section, the essential and most frequent net components for messaging and for basic flow control are presented. Further net components exist that cover sequences, sub calls and manual synchronization.[5]

Control Flow Net Components: Alternatives, Concurrency

The conditional can be used to add an alternative to the protocol. It provides an exclusive or (XOR) situation. To resolve the conflict the `boolean` variable `cond` should be adjusted as desired. As a complement to the *NC if* the *NC ajoin* (alternative join)merges the two alternative lines of the protocol. The *NC psplit* (parallel split) and the *NC pjoin* (parallel join) are provided to enable a concurrent processing within a protocol. Note that the forms of these differ significantly from *NC if* and *NC ajoin* to have a clear separation of parallelism and alternatives.

[5] The full set of net components can be found in [1].

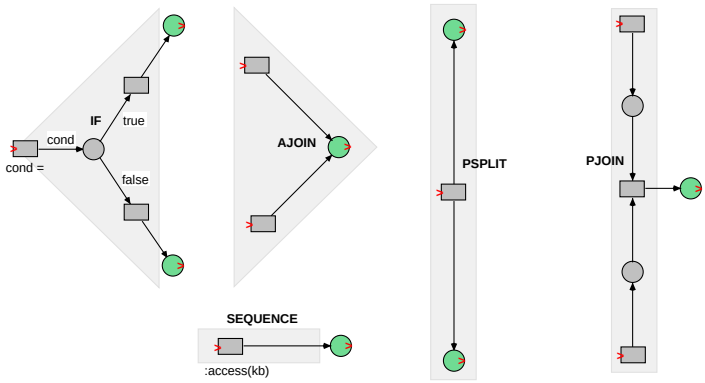

Fig. 1. Conditional and concurrent processing: *if*, *ajoin*, *psplit* and *pjoin*.

Loops

These are the equivalent to the basic loops. The *NC iterator* provides a loop through all elements of a set described by the `java.util.Iterator`. It processes the core of the loop in a sequential order. The *NC forall* uses flexible arcs to provide a concurrent processing of all elements of an array. Flexible arcs allow the movement of multiple tokens with one single arc (see [15] and [10]). The number of tokens moved by the flexible arc may vary, thus its name. In Renew two arrowheads indicate the flexible arcs. A flexible arc puts all elements of an array into the output place and it removes all elements of a pre-known array from the input place. The cores of both loops, *NC iterator* and *NC forall*, are marked with $\wedge$ (beginning) $\vee$ (ending).

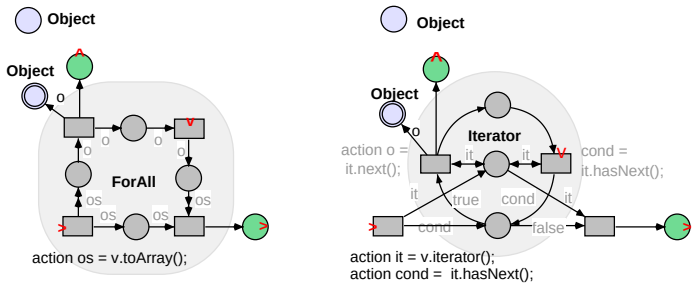

Fig. 2. Loops. *NC iterator and NC forall.*

3.2 Mulan Protocol Specific Net Components

Some net components for Mulan protocols are specialized for the use within Mulan agents. These are protocol management net components and messaging net components.

Protocol Management Net Components

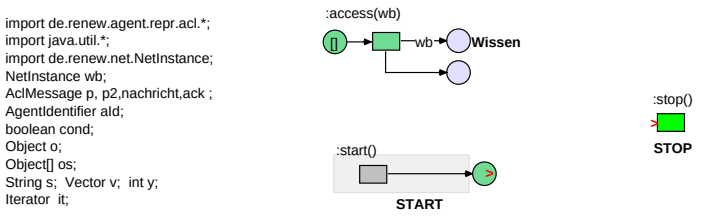

Fig. 3. Protocol Management net components: *NC start* and *NC stop*.

Beginning (*NC start*) and Ending (*NC stop*) are needed in all protocols. There is exactly one start in every Mulan protocol, but there may be more than one stop. The protocol is started when the transition with the channel inscription `:start()` is fired and stopped when one transition with the inscription `:stop()` is fired. In addition, the *NC start* also provides the declaration of the imports, all variables that are used by the net components and the access to the knowledge base (`:access(wb)`). The transitions with the inscriptions `:start()`, `:stop()`, and `:access(wb)` are up-links of synchronous channels. Interfaces of the net components – i.e. the elements that can connect to other net components – are marked with '>'.

Furthermore, the *NC start* provides a declaration for the net and the access of the knowledge base. The declaration already declares the variables that are used in all Mulan net components and the import statements. It can be supplemented with other variables or imports by the developer. The access to the knowledge base is realized as a synchronous channel. It has to be supplemented with access methods for the data that is stored in or retrieved from the knowledge base.

Messaging Net Components

These are the net components that provide the means of communication. The *NC in* receives a message in the same manner as the *NC start* (described in the preceding section). The message is handed to the data block of the net component.

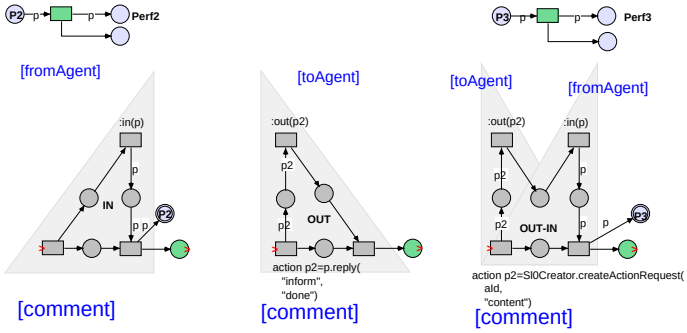

Fig. 4. The net components for message transport: *NC in*, *NC out*, *NC out-in*.

Additional data containing places can be added to the data block as desired. These places can contain elements that were extracted from the messages, for example the name of the sender or the type of the performative. The *NC out* provides the outgoing message task. The *NC out-in* is a short implementation for the combination of both *NC out* and *NC in*. It provides a send request and wait-for-answer situation but does not add functionality other than *NC out* and *NC in*. However, it shortens the protocol significantly.

3.3 Realization

Petri nets can be drawn with Renew in a fast and comfortable way. To be able to use net components in a similar way, it is desirable to have a seamless integration of net components in Renew. This is provided by a simple palette that is the usual container for the buttons of all drawing tools for net elements.

Renew supports a highly sophisticated plug-in architecture [17]. It is appropriate to extend Renew with a plug-in, so that the usual functionality is still completely available. The net components can be drawn in the same way as simple drawing elements by selecting the tool from a tool palette. Once the palette is loaded into the system, the net components are always available for drawing until the palette is unloaded again. Figure 5 shows the graphical user interface with the extension palette loaded.

All net components are realized as Renew drawings, so they can easily be adjusted to the need of the programmer by editing within Renew. The net component drawings are held in a repository, thus a general set of net components can be shared by a group of programmers. Nevertheless, users can also copy and modify the repository to adjust the net components to their needs, or build new net components with Renew. It is also possible to use multiple palettes of different repositories.

Net components are added to the drawing in the same way as the usual net elements. The only difference is that after the new net component is drawn

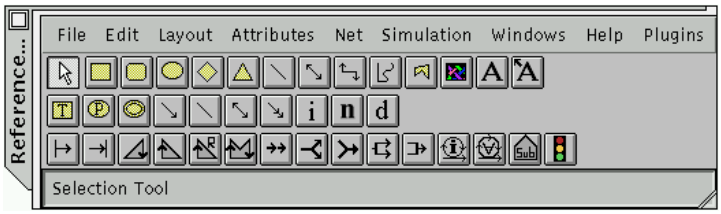

Fig. 5. The graphical user interface of Renew with the net component extension as the bottom palette of tool buttons.

all elements of it are selected automatically. This provides the possibility to adjust the position of the net component in relation to the rest of the drawing.

4 Experiences with Net Components

Net components have been used for about four years in our Petri net based software development projects. In this section, we give an overview of our experiences how the availability of net components and their tool support influenced the net code created by students in comparison to earlier projects.

To illustrate our experiences, Figure 6 shows an example net created by students in our most recent project. The net implements a plan of a *User* agent in a workflow management system to obtain addresses of service providers in the system. The plan initiates an instance of the `queryService` agent interaction protocol diagram with the *WFMS* agent for each required service. We have to stress the point that the net *is* real executable code from the multi-agent application implementation, and that it is shown unmodified, as the students drew it.

The control flow of the net starts in the lower left with a *NC start* and goes in a straight line to the right. In the middle, a *NC forall* splits the control flow in n independent flows, one for each service provider to retrieve. The initiation of the individual agent interaction protocol diagram instances and the corresponding result evaluation is implemented in the shadowed box in the upper part of the net. A protocol instance is initiated by sending a local message to the agent itself that comprises all instance-specific data and waiting for the local response. As the net is a prototype, the reaction to unexpected results is currently not fully implemented.

We do not show an example for a net without net components from earlier projects here. It should be obvious that Petri nets allow the creation of spaghetti code literally. Although the students tried not to draw such nets at first, often additional places and transitions were added ad-hoc in the debugging phase to get a protocol net running. The rearrangement of all surrounding net elements is a very time-consuming process that has often been skipped. With net components, such situations are mostly eliminated due to several reasons:

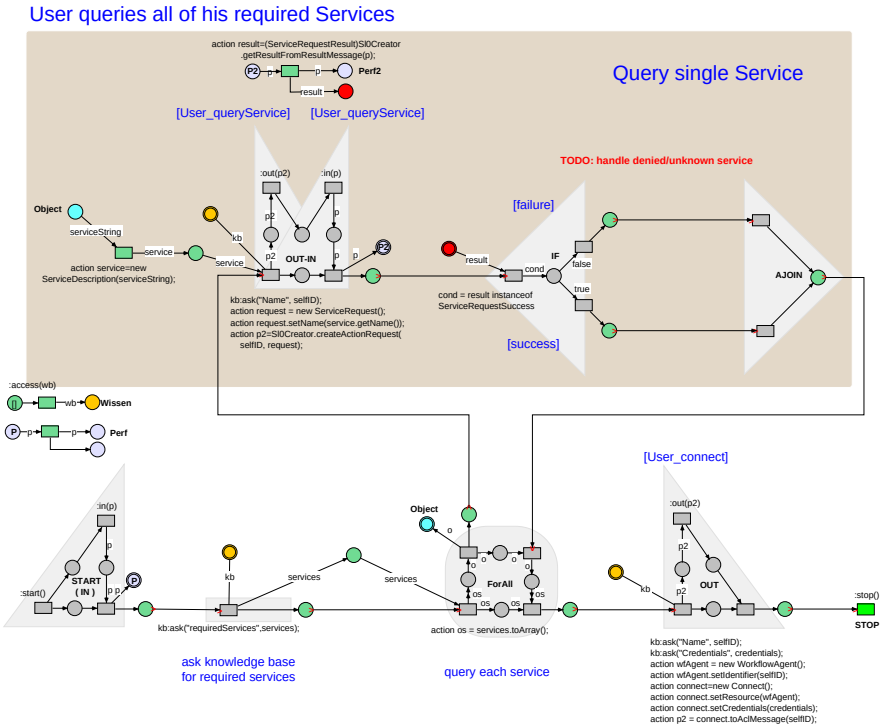

Fig. 6. Example Mulan protocol from the most recent student project

- The Mulan protocol net components promote a left-to-right net layout just because the components themselves are designed horizontally.
- In most net components there is exactly one element to be customized. Therefore, additional data places are forced to be located near this element.
- Ad-hoc transitions are not needed any more because many net components already come with a skeleton for data flow and manipulation.
- Net components are supported in the tool by a weak and flat grouping mechanism.[6] This improvement eases the insertion of new elements in the middle of an existing net without destroying the layout of individual net components while it retains full modifiability of net component details.
- Documentation of nets is promoted by standardized comment templates that are attached to the net components.

[6] The weak and flat grouping mechanism allows the movement of all elements of the component as a group while individual net elements of the net component are still accessible to any kind of manipulation. There is no hierarchic grouping, grouped groups become fused instead. The classical grouping feature is more restrictive since single elements of a group can not be manipulated nor selected.

In general, we observed that nets built with net components are tidier than nets without net components. Additionally, the readability of nets is improved significantly. With a little experience the comprehension of the nets is reduced to the reading of two elements per net component: the shape of the component and the customized inscription. Without the need to examine every net element, it is thus possible to understand the described process.

Besides the graphical benefits, the teaching staff observed an increase in code development speed and student's learning curve in Petri net design. There are multiple explanations:

- It is obviously faster to compose a net from larger blocks than to repeat every small step every time (reuse of code).
- The first set of net components covered mostly well-known constructs from classic sequential programming languages like conditionals and loops. So the transition from classical programming to Petri net programming became easier.
- Moreover, net components enable automation of net construction resulting in generation of nets or round-trip engineering techniques.
- Existing net components come with transitions and places already inscribed and connected correctly, thus they show examples of correct code to the students.
- For the attending teachers and tutors the results of the students' designs are much easier reviewed due to the structural clearness mentioned above.
- Reuse of concepts and solutions have been intensified. The original set of net components is based on patterns of net elements that have been recognized in Mulan protocols. Now they are in use, and we detect more advanced patterns that can also become net components. Design patterns become graspable for students.
- Reuse of concepts is facilitated. It is easy to cut a recognized pattern from an existing net and turn it into a new net component.

The overall acceleration of net design leaves more time to discuss e.g. important architectural matters of the software. These benefits do not only apply to software development, net components have also been applied to other areas. Besides the workflow patterns presented in [12], an example is a set of net components for the construction of Petri net-based plans which are automatically assembled during runtime and executed on the fly.

It has to be admitted that some of the advantages mentioned above entail a trade-off in flexibility of Petri net engineering. The strong form of net components restricts the overall net layout, and structured net component-based Petri nets tend to be larger then simple unstructured nets. Students tend to stick to the predefined solutions and are very conservative with the introduction of new patterns. Because many net components are oriented along classical sequential programming language constructs, resulting nets sometimes include less concurrency than Petri nets would allow. However, depending on

the stage of modelling expertise the students gained, parts of this flexibility trade-off may also be seen as an advantage rather than a disadvantage.

However, some of the original and some later proposed net components have been used only very seldom, if at all. This results from manifold reasons, which can not be extensively discussed in this paper. Some of the reasons of rejection of net components are bad integration and too specialized purpose. Also, frequently the ability of net components to be parameterizable, for the automation of e.g. inscription adaption or for dynamically adaptable net structures (number of choices in a switch / case component), is requested.

The implementation as a Renew plugin allows for an easy management of the net components and their repository. Recent further development of the net components plugin allows repositories of net components to dynamically extend the tool set, even allowing to plugin, plugout and – after redefining the repository – re-plugin with the altered functionality at run-time [2, 3].

5 Conclusion

5.1 Summary

Net components are sets of net elements with geometrical arrangements that make it easy to identify them and to distinguish them form each other. Each net component is bounded by transitions and fulfills one basic task. As a means of structuring, the Mulan net components are capable of accelerating the development of Mulan protocols. The readability of net component-based Mulan protocols is increased significantly in comparison to Mulan protocols that do not use net components.

The net components tool that is implemented as a plug-in for Renew makes it possible to apply net components easily to Petri nets. The net components that are held in a repository are editable in Renew, thus adaptable to the needs of the development team. Moreover, net components plugin itself is extensible by repository plugins that are dynamically pluggable an unpluggable during runtime.

The Mulan net components are being successfully used since the second teaching project of the ongoing series of multi-agent system development projects in our research group and have eased the teaching and development of Petri net-based Mulan protocols.

5.2 Outlook

In the future the influence of data flow on control flow is a topic that will be further investigated. In question is still whether a strict separation of data flow and control flow is desired or whether this can and should be relaxed. Another question is whether a structuring of Petri net source code similar to

(and influenced by) the usual concepts of programming languages or Petri net-specific features have more advantages during the development of nets. It seems save to claim that in general this question can only be answered within the context of the development domain of the Petri nets.

We are looking forward to apply the mentioned workflow patterns [12] – and improve the implementation as net components – to the currently developed agent-based distributed workflow engine [14]. This integrates a reference net-based workflow engine [5] into the multi-agent system Mulan. In this context net component-based development can improve the development of workflows. A first design of those net components has already been integrated into the net component tool set [12].

References

1. Cabac, L. Entwicklung von geometrisch unterscheidbaren Komponenten zur Vereinheitlichung von Mulan-Protokollen. Studienarbeit, Universität Hamburg, Fachbereich Informatik, 2002.
2. Cabac, L., Duvigneau, M., Moldt, D., and Rölke, H. Modeling dynamic architectures using nets-within-nets. In *Applications and Theory of Petri Nets 2005. 26th International Conference, ICATPN 2005, Miami, USA, June 2005. Proceedings* (2005), pp. 148–167.
3. Cabac, L., Duvigneau, M., Moldt, D., and Rölke, H. Applying multi-agent concepts to dynamic plug-in architectures. In *Agent-Oriented Software Engineering VI: 6th International Workshop, AOSE 2005, Utrecht, Netherlands, July 21, 2005. Revised Selected Papers* (June 2006), J. Mueller and F. Zambonelli, Eds., vol. 3950 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 190–204.
4. Cabac, L., Moldt, D., and Rölke, H. A proposal for structuring Petri net-based agent interaction protocols. In *24th International Conference on Application and Theory of Petri Nets, Eindhoven, Netherlands, June 2003* (June 2003), W. v. d. Aalst and E. Best, Eds., vol. 2679 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 102–120.
5. Jacob, T., Kummer, O., Moldt, D., and Ultes-Nitsche, U. Implementation of workflow systems using reference nets – security and operability aspects. In *Fourth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools* (Ny Munkegade, Bldg. 540, DK-8000 Aarhus C, Denmark, Aug. 2002), K. Jensen, Ed., University of Aarhus, Department of Computer Science. DAIMI PB: Aarhus, Denmark, August 28–30, number 560.
6. Jensen, K. *Coloured Petri Nets*, 2nd ed., vol. 1. Springer-Verlag, Berlin, 1996.
7. Köhler, M., Moldt, D., and Rölke, H. Modelling the structure and behaviour of Petri net agents. In *Proceedings of the 22nd Conference on Application and Theory of Petri Nets 2001* (2001), J. Colom and M. Koutny, Eds., vol. 2075 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 224–241.
8. Köhler, M., Moldt, D., and Rölke, H. Modelling mobility and mobile agents using nets within nets. In *Proceedings of the 24th International Conference on Application and Theory of Petri Nets 2003 (ICATPN 2003)* (2003),

W. van der Aalst and E. Best, Eds., vol. 2679 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 121–139.

9. Kummer, O. *Referenznetze*. Logos Verlag, Berlin, 2002.
10. Kummer, O., Wienberg, F., and Duvigneau, M. Renew – the Reference Net Workshop. Available at: `http://www.renew.de/`, May 2006. Release 2.1.
11. Moldt, D., Köhler, M., and Rölke, H. Projekt: Agenten-orientierte Softwareentwicklung, WS 2001.
12. Moldt, D., and Rölke, H. Pattern based workflow design using reference nets. In *Proceedings of International Conference on BUSINESS PROCESS MANAGEMENT, Eindhoven, NL* (2003), W. v. d. Aalst, A. t. Hofstede, and M. Weske, Eds., vol. 2678 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 246–260.
13. Oberquelle, H. Communication by graphic net representations. Fachbereichsbericht IFI-HH-B-75/81, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, D-22527 Hamburg, 1981.
14. Reese, C., Markwardt, K., Offermann, S., and Moldt, D. Distributed business processes in open agent environments. In *International Conference on Electronic Information Systems (ICEIS) 2006* (2006). Accepted paper.
15. Reisig, W. *Elements of Distributed Algorithms: Modeling and Analysis with Petri Nets*. Springer-Verlag New York, Oct. 1997.
16. Rölke, H. *Modellierung von Agenten und Multiagentensystemen – Grundlagen und Anwendungen*, vol. 2 of *Agent Technology – Theory and Applications*. Logos Verlag, Berlin, 2004.
17. Schumacher, J. Eine Plugin-Architektur für Renew – Konzepte, Methoden, Umsetzung. Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, D-22527 Hamburg, Oct. 2003.