A Proposal for Structuring Petri Net-Based Agent Interaction Protocols

Lawrence Cabac, Daniel Moldt and Heiko Rölke

Department of Computer Science, TGI, University of Hamburg
{6cabac, moldt, roelke}@informatik.uni-hamburg.de

Abstract. In this paper we introduce net components as means for structuring Petri net-based agent interaction protocols. We provide a tool for effortless application of net components to nets. Thus we facilitate the construction of nets and unify their appearance. Net components can be used to derive code for interaction protocols from a subset of extended AUML (Agent Unified Modeling Language) interaction protocol diagrams. This allows for a smooth integration of some traditional software development specification approaches with high-level Petri nets. By using net components we do not only unify the structure of Mulan agent protocols but also succeed to build a common language within a community of developers who share the net components.

Keywords: agents, agent interaction protocols, AUML, high-level Petri nets, Mulan, reference nets, Renew.

1 Introduction

From the beginning of computer science the readability of program code is a well known problem. There are many means of improving readability. The goals are to make code intelligible, easily accessible and clear. Generally, the problem that lies behind the readability is the complexity of code.

Methods to achieve readability are for example structured programming (see [4]) or object-oriented programming (see [19]). Also integrated development environments (IDE), syntax high-lighting and indentation of syntactical entities add to readability. While the first ones of these concepts provide orientation on an abstract level, the latter ones provide orientation on a basic cognitive level. These problems do not only pertain to text but also to other representations of code.

Visual programming languages (VPL) are subject to similar problems. The statement which is known as 'Limit of Deutsch' (see [15]) - the limitation of the number of elements on a screen to fifty graphical objects - gives a glimpse of the problem. Many advances try to hide the complexity by modularizing the views. Complex structures are represented and thus hidden behind simple elements, e.g.: icons. This leads to complex content of simple elements which increases the number of available simple elements. Nevertheless, this is a good

approach, since it clarifies the overall structure of code through abstraction. However, the arrangement of elements of code for visual programming languages like LabVIEW (see [14]) or Prograph (see [20]) is restricted by their development tools to guarantee a unified style.

Workflows are not considered to be programs but they describe processes just like programs. During modeling of workflows very similar problems of complexity occur. Van der Aalst et al. (see [22]) offer a proposal of "Workflow-Pattern" for a unification of recurring patterns in workflows. The unification of parts of the code by modeling with patterns and the naming of those parts of the workflow are useful methods to facilitate the process of understanding the workflow.

Petri nets can also be used to describe processes. They can be simulated (executed) - just like VPL. Thus they can be regarded as programs, and nets themselves can be considered the code of the program.

Petri nets tend to grow in size with their complexity (see [8]). High-level Petri nets use various concepts to handle complex coding. They are capable of expressing complex structures in folded nets by introducing named tokens, thus adding new elements to the formalism. Another approach to handle complexity is to combine Petri nets with concepts of object orientation which leads to the object-oriented Petri nets (OOPN) (see [16]). Agent-oriented Petri nets (AOPN), such as those presented in [10], help develop multi-agent systems, especially through their inherent concurrency.

There are general recommendations for the look and feel of Petri nets. These relate usually to the simple elements of the Petri nets: Transitions, Places, Arcs and Inscriptions. Jensen's recommendations (see [9]) - based on the work of Oberquelle (see [18]) - cover either the elements of the nets, or the rules are fairly general and can be interpreted quite broadly. For special nets and for the arranging of net elements - except for beautification - there exist no additional rules; consequently the appearance of Petri nets varies extremely. Naturally, the appearance depends on the programmer / modeler, the used Petri net tool and the domain. In general, it can be of advantage to have a great variety of appearances of Petri nets. In contrast, the representation of similar nets in the same domain requires conformity. This is the case when implementing application software with Petri nets.

The tool of choice for developing and modeling Petri nets at the Department of Computer Science at the University of Hamburg is *Renew* (see [12]). Not only does it allow to construct Petri nets but it also has the ability to simulate them efficiently. The nets that are processed by Renew are reference nets (see [11]), an extension to Coloured Petri Nets (CPN, see [9]). It is possible to use Java code as inscriptions of net elements which can be of advantage, for example when combining nets with a graphical user interface.

Mulan (Multi-Agent Nets) is based on Renew. It is a multi-agent system that uses the advantages of Petri nets such as concurrency. Together with Renew it provides the possibility to develop software with Petri nets by the agent-oriented paradigm. Since agents are defined as individual and independent components they offer an interesting approach to the development of software for concurrent

processes. Especially for concurrent and adaptive processes strong limitations exist with conventional methods so that building a multi-agent system with inherent concurrency is of advantage. This can easily be achieved by basing the system on Petri nets.

Development of application software based on an agent-oriented software system requires the development of interaction protocols so that the behavior of the agents is well defined. The basic task for application programming with Mulan is to develop Mulan protocols, which are Petri nets. The purpose of the Mulan protocols is to define the behavior and the communication or interactions of agents.

Mulan protocols perform several basic tasks which frequently re-occur. They vary in the overall structure but perform similar tasks within this structure like sending or receiving a message or deciding on a condition. However, those nets can be rather huge so that implementation and debugging can be quite difficult and time consuming. We model agent interaction using AUML (Agent Unified Modeling Language) interaction protocol diagrams (see [7]), which is in the context of UML (Unified Modeling Language) in the version of interaction diagrams a commonly used and elaborated modeling technique.

In this paper we introduce net components as means for structuring Petri net-based agent interaction protocols. We achieve to unify the structure of Mulan protocols which increases their readability and build a common language within a community of developers who share the net components.

In the next section we will present our multi-agent system infrastructure. Then we introduce net components in general, the net components for Mulan protocols and a tool that supports the construction of Petri nets with net components. Finally we present a way to model the communication of agents with AUML interaction protocol diagrams, describe the construction of Mulan protocols with net components and show that net component-based Mulan protocols reveal their structure due to the geometrical forms of the net components.

2 Petri Net-Based Multi-Agent System Infrastructure

In this section we present a short introduction to the Petri net-based multi-agent system Mulan. Mulan is implemented with reference nets and runs within Renew. Renew (Reference Net Workshop, see [12]) is a Petri Net editor and simulator. Mulan (Multi-Agent Nets, see [10]) is a reference architecture to a multi-agent system which complies with the FIPA specification for multi-agent systems. CAPA (Concurrent Agent Platform Architecture, see [5]) extents Mulan to provide FIPA-compliant communication and agent management.

2.1 Renew

With Renew it is possible to draw and simulate Petri nets and reference nets. A net that is loaded into the editor can be executed by the simulation engine. For this an instance of the net is created by the simulator. Any simulated net

can instantiate another net. Hence it is possible to produce many instances of different nets. The relationship between net (also called net template) and net instance can be compared to the relationship of class and object.

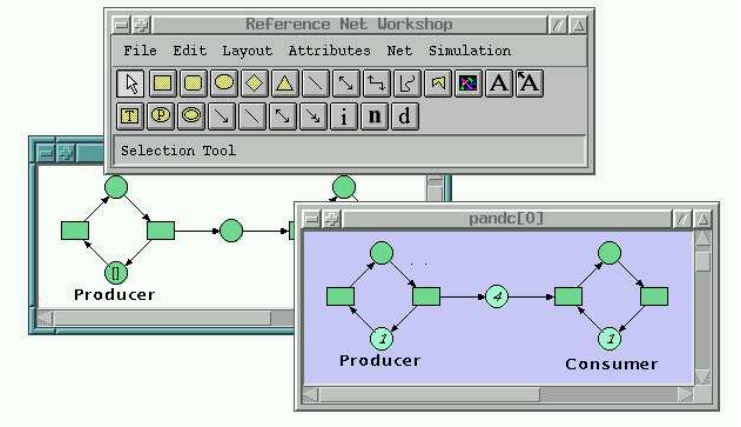

Fig. 1: Renew GUI, Petri net and net instance (producer-consumer example).

Figure 1 shows the graphical user interface (GUI) of Renew, a simple Petri net in the back and a net instance. The user interface consists of the menu bar, two palettes and a status line. The menu bar offers menus for general operations, attribute manipulations, layout adjustment and Petri net-specific operations. It also provides the possibility to handle the simulation. Of the two palettes the first one consists of usual drawing tools while the second one holds the Petri net drawing tools. The latter palette provides the tools for creating transitions, places, virtual places, arcs, test arcs, reserve arcs, inscriptions, names and description nodes. In addition to these tools the editor reacts in a context sensitive manner to facilitate the drawing of nets. One example is the dropping of arcs on the background which can create a new place if the arc starts at a transition and vice versa. Another example is the right click on inscribable elements which produces an inscription for this element with a context sensitive default value.

Nets hold the initial marking where net instances hold the current marking. In figure 1 the producer-consumer example has been started. In the net (background) one of two black tokens of the initial marking can be seen in the place labeled “Producer”. While the net instance by default only shows the number of tokens in a place it is also possible to show the contents of the places by clicking on the numbers.

A special feature of Renew is that it can operate with reference nets. Renew also allows to use any kind of Java objects as tokens. It is implemented in Java and extendible through a plug-in mechanism. A plug-in for net components is presented in section 3.3. Figure 2 shows the architecture of the whole system.

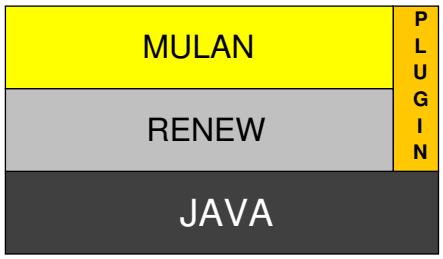

Fig. 2: The architecture of the multi-agent system infrastructure.

2.2 Reference Nets

Reference nets are special high-level Petri nets in which the tokens can be nets again. For these "nets within nets" referential semantics is assumed. Tokens in one net instance can be references to other net instances. The benefit of this feature for Renew is that it is modular and extendible. Nets can call other nets just like method calls of objects by using synchronous channels (see [11]).

A synchronous channel consists of two transitions which are called *down-link* and *up-link*. These two transitions can only fire simultaneously and only if both transitions are activated. *Down-link* and *up-link* can belong to one single net or they can belong to two different nets. In both cases any object can be transferred from either transition to the other. If two different net instances are involved it is thus possible to synchronize those two nets and to transfer objects in either direction by the synchronous channel.[1]

Mulan agents use the synchronous channels to start and stop their protocols. Also the communication between the Mulan agent and its Mulan protocols is realized with them. Messages are transferred from the agents to the protocols and back. While the agent provides the functionality to transmit the message to another agent the protocol is in charge of the processing of the message itself.

2.3 Mulan

Mulan is a multi-agent system that is based on Renew. It is implemented with Petri nets as a system of reference nets (Mulan: Multi-Agent Nets) and it complies with the open specifications of the Foundation for Intelligent Physical Agents (FIPA, see [6]) for multi-agent systems.

The system consists of numerous Petri nets. This is illustrated in figure 3. The figure shows the net within a net hierarchy of the system. Agents are nets which exist on platforms which are also nets. There can be many platforms and the agents can communicate with each other within and across platforms.

[1] Additional information for Renew, reference nets and synchronous channels can be found in [12].

Protocols are nets within the agents and control their behavior. The figure is taken from [10].

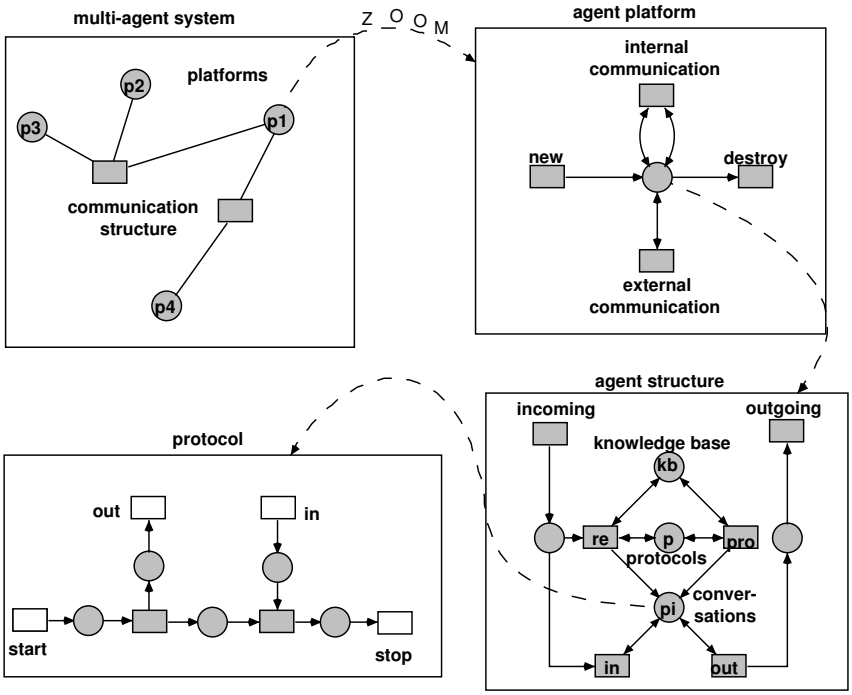

Fig. 3: The structure of Mulan (compare [10]).

Agents communicate by sending messages to each other. A set of messages sent back and forth between two or more agents is called conversation. The conversation is determined by the protocols which the agents use during this communication. So the conversation describes the agents' interactions whereas one or more protocols describe the behavior of one agent during the conversation.[2] Conversations can be compared to FIPA interaction protocols while the Mulan protocols as described here have no equivalent and are parts of the FIPA interaction protocols.

2.4 Terminology

In this section some terminology shall be clarified. Descriptions of the terms agent, protocol and net component are provided to give the reader a first notion of them while the details are postponed to a later section.

[2] This terminology differs slightly from the FIPA terminology.

Agent An agent is an independent software component that follows a goal, which can be achieved alone or in combination with other agents. Agents communicate via messages and can act independently, or reactively. They can change their behavior as needed, i.e. they are adaptive. Each agent has its own knowledge base in which a part of the information of the agent's environment and also its means of reacting is stored. The adaptation of the agent's behavior is achieved by modification of the knowledge base. The interaction with other agents constitutes the social behavior of the agents. All these features are the basis for intelligent behavior. Informally agents can be regarded as software robots.

Protocol A protocol defines a certain behavior. In the agent-oriented view a protocol determines the communicational behavior of agents. Mulan protocols are - just like Mulan agents - Petri nets. An agent can use numerous protocols and it can instantiate any number of instances of protocols.

Net Component A net component is a set of Petri net elements that belong together in a syntactical sense. In addition, it also has a visual meaning. Both the geometrical and the directional arrangements are defined. So in addition to the syntactical unity also a visual unity is achieved. This visual character makes it easy to identify the net component.

3 Net Components

This section introduces net components and their concept. In addition, an example implementation of net components for Mulan protocols is presented. Net components are meant to be combined with each other to form a Petri net.

3.1 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 which results from the arrangement of the net elements. A unique form is intended so that each net component can easily be distinguished from the others and identified. The geometrical figure also holds the potential to provide a defined structure for the Petri net. The structuring of Mulan protocols is achieved by the unique form of the net components and the notion that Mulan protocols can be read from left to right.

In the default implementation a state is added at the outward connecting transition (Interface Place) for convenient net component connection. Only one arc has to be drawn to connect one net component to another. This is a simple and efficient method that also emphasizes the control flow. The connection of net components is provided by this place, which at all times should only contain anonymous tokens.

Direct data exchange between net components is not desired to guarantee an easy connecting interface. Instead data is handed to the data-containing places via *virtual places*[3]. By adding an appropriate *virtual place* to the net component, data can be transferred indirectly to the transition that uses a variable. In the usual case this is done by using a test 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.

3.2 The Mulan Protocol Net Components

We explain a selection of the net components so that their form and application will be clear. In this section the net components for messaging and for basic flow control are presented. There exist further net components which cover sequences, sub calls and manual synchronization.[4]

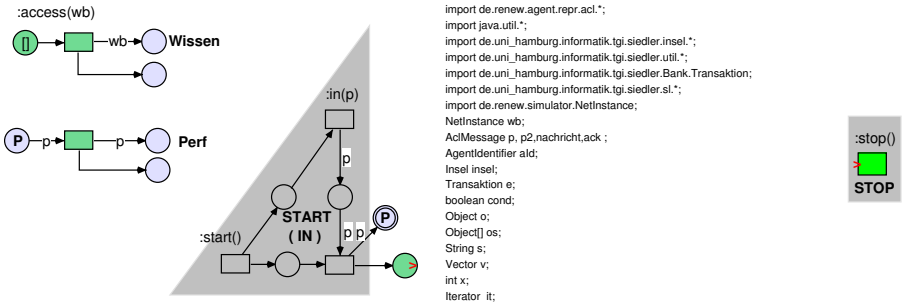

Fig. 4: Essential net components: *NC start* and *NC stop*.

Essential Net Components Beginning (*NC start*) and Ending (*NC stop*) are needed in all protocols. There is exactly one start in every Mulan protocol, but

[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] The full set of net components can be found in [3].

there may be more than one stop. The protocol is started when the transition *: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 and all variables which are used by the net components and the access to the knowledge base (*:access(wb)*). The *NC start* always receives a message so this functionality is also provided together with a data block, which holds the received message. The message (performative *"p"*) is received by the transition *:in(p)* and is handed to the data block of the net component by a virtual place (*"P"*). The message is finally held in the place *Perf* and information from the message can be extracted at the preceding transition and stored in additional places. The four transitions *:start()*, *:stop()*, *:access(wb)* and *:in(p)* are the uplinks of synchronous channels. Interfaces of the net components - i.e. the elements that can connect to other net components - are marked with *">"*.

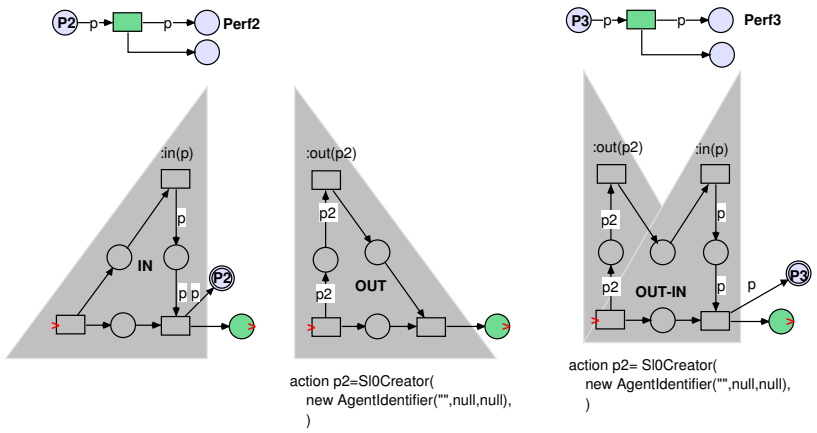

Fig. 5: The net components for message transport: *NC in, NC out, NC out-in.*

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. 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 shorter implementation for the combination of both *NC out* and *NC in* which provides a send request and wait-for-answer situation. It does not add functionality but shortens the protocol significantly.

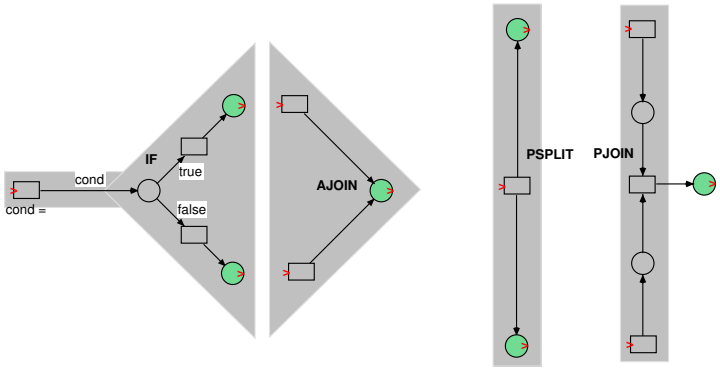

Fig. 6: Conditional and concurrent processing: *NC cond*, *NC ajoin*, *NC psplit* and *NC pjoin*.

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. The *NC psplit* (parallel split) and the *NC pjoin* are provided to enable a concurrent processing within a protocol. Note that the forms of these differ significantly from *NC cond* and *NC ajoin* to have a clear separation.

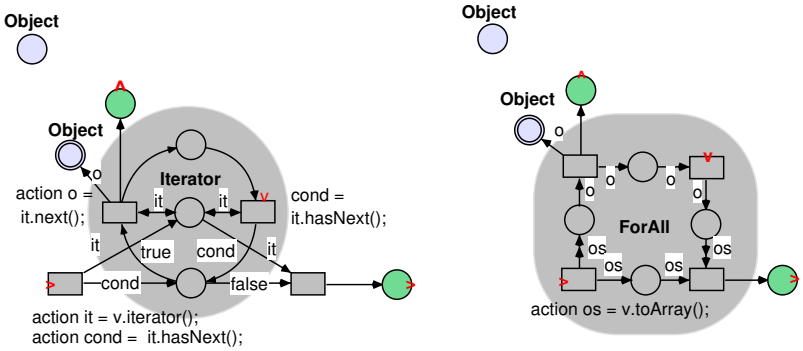

Fig. 7: Loops. *NC iterator and NC forall.*

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 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 to move multiple tokens with one single arc (see [21] and [13]). The number of tokens moved by the flexible arc may vary, thus its name. In Renew the flexible arcs are indicated by two arrowheads. 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 the loops are marked with $\wedge$ (beginning) $\vee$ (ending).

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 which is the usual container for the buttons of all drawing tools for net elements.

3.3 Realization

Renew supports a highly sophisticated plug-in architecture.[5] It is appropriate to extend Renew with a plug-in, so that the usual functionality is still completely available. Once the palette is loaded into the system the net components are always available for drawing until the palette is unloaded again. Figure 8 shows the graphical user interface with the extension palette loaded.

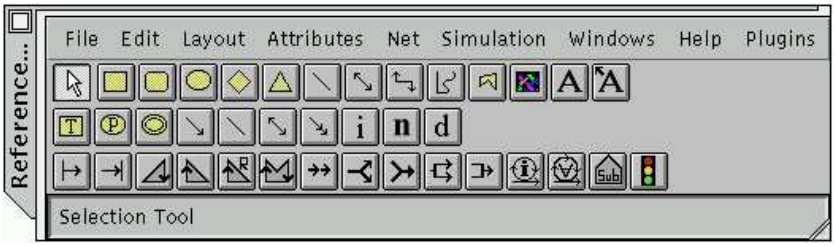

Fig. 8: The graphical user interface of Renew with the net component extension.

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 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.

[5] Ongoing work of Jörn Schumacher.

4 Application of Net Components

This section deals with the modeling and the implementation of Mulan protocols. We show the advantages of structured net component-based development of Petri nets for Mulan protocols. Furthermore we investigate how to achieve a suitable structure. First we show how modeling agent interaction can be done with AUML interaction protocol diagrams. Then we present an example to show how to derive code from the agent interaction protocol diagram. By joining net components together we build the Mulan protocols for the agents. At last we want show that net component-based Mulan protocols reveal their structure due to the geometrical forms of the net components.

4.1 Modeling Agent Interactions

Modeling agent interaction can be done by using several means. The FIPA defines the AUML interaction protocol diagrams for modeling interactions between agents. These diagrams are an extension of the Unified Modeling Language (UML) sequence diagrams (see [1]) but they are more powerful in their expressiveness. They can fold several sequences into one diagram. Thus they can describe a set of scenarios.

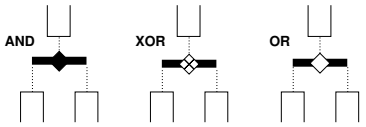

Fig. 9: New elements of interaction protocol diagrams: AND, XOR, OR

Some additional elements are added to the usual sequence diagram. Those additional elements provide alternative, concurrent and arbitrary splitting in a manner of the three gates AND, XOR and OR.

Figure 9 shows the new elements in a horizontal version which are applied to split the life line of an agent. In addition the FIPA also defines the vertical versions of those three elements to split the messages. Figure 10 shows an example protocol diagram for the contract net protocol as presented in [7]. It shows the other variant of the additional elements.

We use AUML interaction protocol diagrams to model the behavior of the Mulan agents. The models of agent interaction are then implemented as reference nets by using the net components.

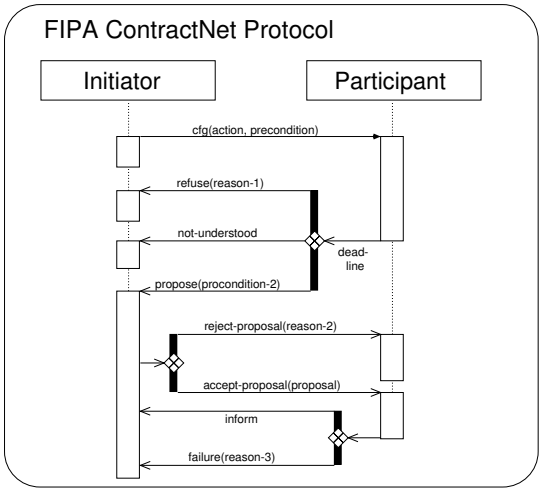

Fig. 10: The FIPA contract net protocol diagram.

4.2 Using and Applying Net Components

We restrict the interaction protocol diagrams to the usage of split figures for the life line. By doing this we can achieve a structure that can directly be transformed into a Petri net by using net components. Instead of using message split figures we favor the opposite, i.e. message join figures.

Especially for alternatively sent messages a message join figure can at times be of advantage, for example when the message is a reply. This is also intuitive since the receiver of a reply expects only one answer.

The development of a Mulan protocol using net components illustrates the procedure. For the reasons of clarity we present an agent version of the *producer-consumer* example which is adjusted to be compliant with the FIPA request protocol. Figure 11 shows the FIPA request protocol (see [7]) and an interaction protocol diagram of the FIPA-request compliant *producer-consumer* example. This version of the diagram only uses the split of the life line and the join of the message arc in addition to the usual UML diagram elements.

To demonstrate the process of transforming the diagram into a net we would like to show the development in detail in three steps. First, we divide the protocol diagram into the two parts which belong to each agent and rotate the resulting two diagrams by 90° so that they can be read from left to right.

As the second step we add the geometrical forms of the net component to the appropriate parts of the diagrams. This shows that the net components can be used for the implementation and it also shows the overall structure of the Petri nets. Figure 13 shows the same diagram augmented with the geometrical symbols of the net components.

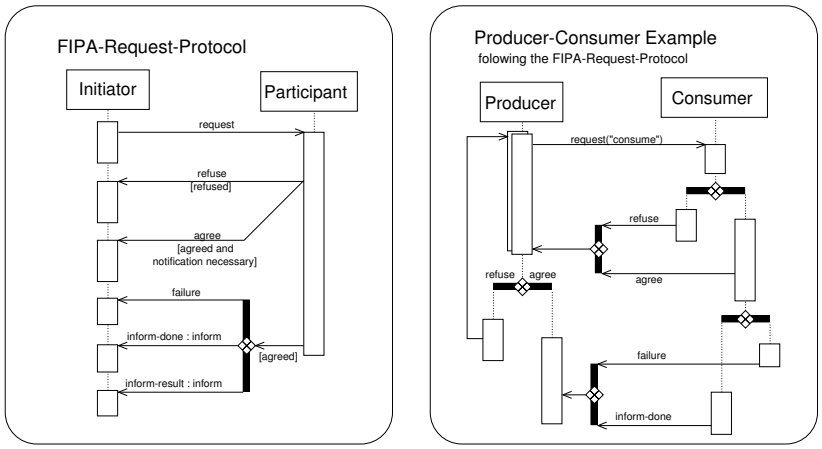

Fig. 11: Interaction protocol diagram of the FIPA Request and a FIPA Request-compliant *producer-consumer* example.

In the third and last step we use the net component extension of Renew to draw the Mulan protocols with the net components analogous to the symbols in Figure 13. Still some work has to be done regarding the actions, the adjustments of messages and other inscriptions. Both figures 14 and 15 show resulting Petri nets of the *produce* and the *consume* protocol. Note that the nets are not shown here to be read as Petri nets, although they are fully operational and can be executed in Renew and Mulan without any changes. Instead they are presented to get an impression of the structured layout, the application of the net components and the analogies to the structure of the interaction protocol diagrams.

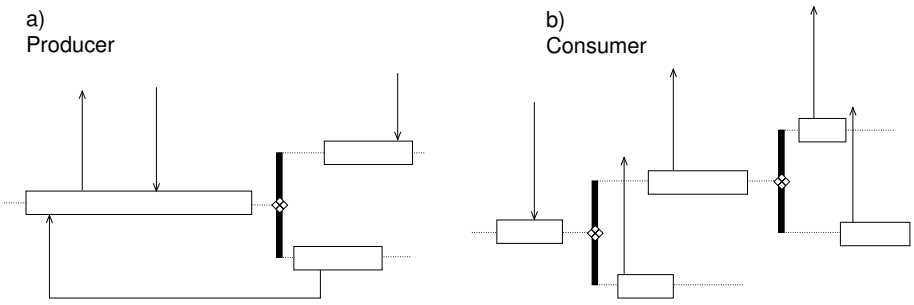

Fig. 12: Two parts of the example diagram; one for each agent.

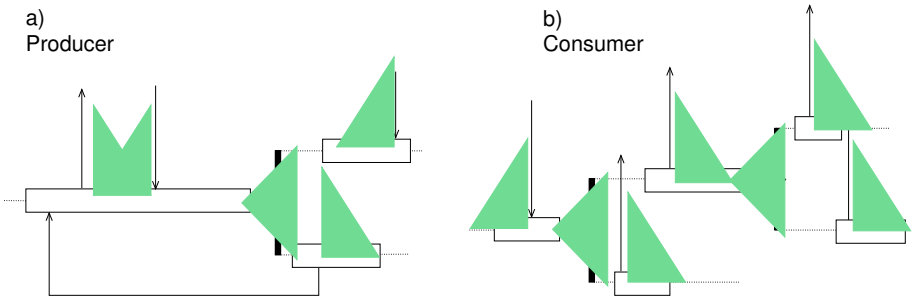

Fig. 13: Both parts augmented by the symbols for the net components.

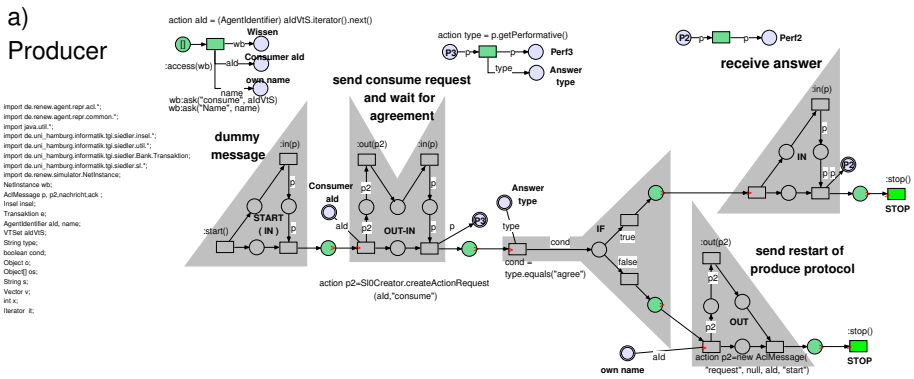

Fig. 14: The resulting *produce* Mulan protocol.

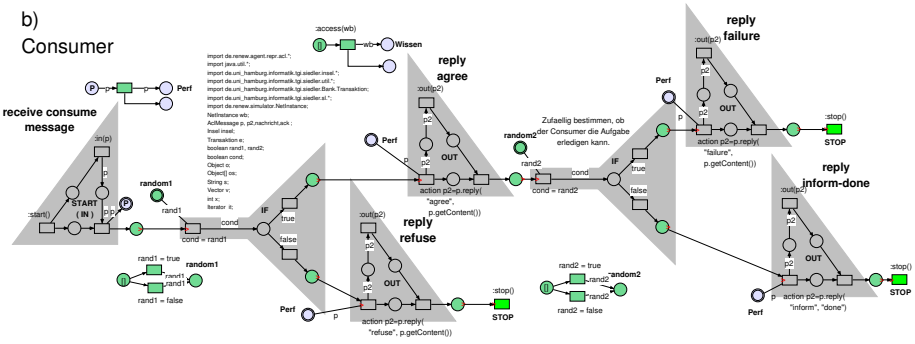

Fig. 15: The resulting *consume* Mulan protocol.

4.3 Mulan Protocols Structured by Net Components

The visual aspects of net components play a crucial role in recognition and thus in readability. Since net components have a fixed geometrical structure they can always be identified without reading any details of the net elements. The geometrical form of the net itself becomes readable to the programmer without using any modeling abstraction. Nevertheless a net constructed with net components can be transformed directly into an interaction protocol diagram. We claim that a well-structured Mulan protocol that uses net components exclusively is readable without reading any of the net elements. Figure 16 shows the registration of a player at the game control.[6] Again this net is presented here not to be read; instead the net components should be regarded. Once they are identified they can easily be mapped onto an interaction protocol diagram.

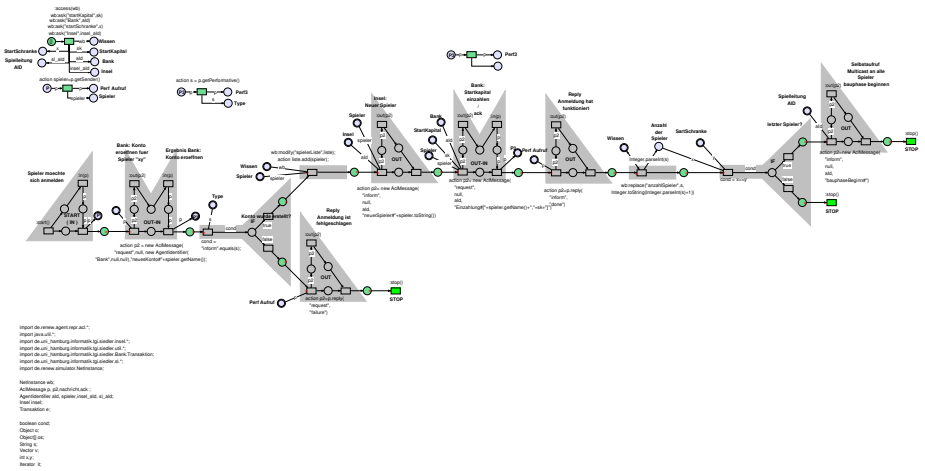

Fig. 16: Mulan protocol for the registration of a player.

Similar to the *producer-consumer* example in section 4.2 the registration protocol in figure 16 can be read like a sentence from left to right. Due to the usage of net components the basic tasks performed by this protocol and the structure of the interaction can be read without the need to interpret the details of the net itself. It can be seen that the game control is involved in several communicative acts and two decisions. The general structure of the interaction can thus be derived from the net itself. Only the participants of the conversation are not identified yet. For this we recommend to add a comment for each net component. Figure 17 shows the full conversation between the four agents as the final interaction protocol augmented with the symbols of the net components.

[6] For the multi-agent version of the "The Settlers of Catan" board game (see [2]).

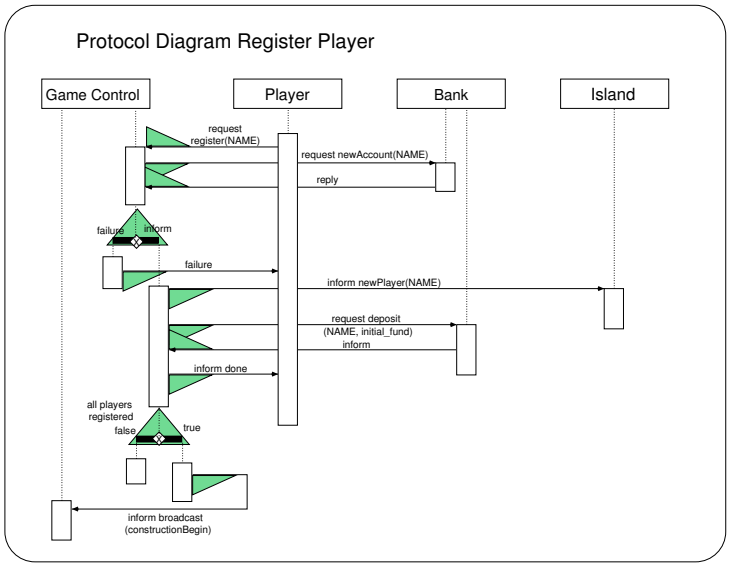

Fig. 17: Mulan conform-structured agent interaction protocol for the registration of a player.

5 Conclusion

The gap between traditional system modeling and the use of high-level Petri nets is relatively large. Both areas can contribute to the requirements of actual software development today. The latest developments in this area are AUML (see www.auml.org) where concepts from agent orientation are integrated mostly for a more compact representation. However, distributed systems require more elaborated features. These are covered by high-level Petri nets like reference nets (see [11]) in combination with an agent-oriented architecture (see [10]).

In this paper we were able to integrate all directions. For this we started with the introduction of our multi-agent system infrastructure. The advantage of using a Petri net-based system is its inherent concurrency. The implementation of a multi-agent system with Petri nets is only possible if it can be backed up with a powerful and efficient simulation tool. This was achieved by building the system on Renew.

We showed how modeling of agent interaction can be done by using FIPA interaction protocol diagrams. The notion of net components and a corresponding set for Mulan agent interaction were presented. Together with the seamless integration of the net components into Renew we now have a simple but powerful tool to support net component-based Mulan protocol implementation. AUML provides methods for modeling interactions which can be used to facilitate and structure the agent interaction. The structure of the protocols can be unified and

clarified through the usage of net components. Unification is especially desired when implementing in project groups. Through the net components a common language and style can be achieved. Refactoring of protocols is facilitated because the components are loosely interconnected. So remodeling of Mulan protocols is supported.

We want to point out that by using net components we achieve a structure for a Petri net layout that improves the readability significantly. Furthermore the overall structure of the model is retained through the analogous construction of interaction protocol diagrams and Mulan protocols. Developing and debugging time for Mulan protocols can be reduced significantly and reuse of code is facilitated. Although the structuring of the Mulan protocols is not achieved automatically, the results can be seen in analogy to the structuring of program code by using indentation, syntax highlighting or conventions like capitalizing.

Net components were presented here as a part of Mulan protocols. Nevertheless, it is possible to apply the same principles to other domains. As an example we would like to mention the obvious solution for a Petri net-based workflow engine (see [17]). It is possible to realize workflow patterns for this domain using the Renew extension presented in section 3.3.

References

1. Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, Reading, Massachusetts, 1996.
2. Tobias Bosch, Oliver Gries, Heiko Kausch, Maxim Klenski, Kolja Lehmann, Michael Morales, Valentin Seegert, and Anatolij Vilner. *Agentenorientierte Implementierung des Spiels "Die Siedler von Catan"*. Internal report, University of Hamburg, Department of Computer Science, 2002.
3. Lawrence Cabac. *Entwicklung von geometrisch unterscheidbaren Komponenten zur Vereinheitlichung von Mulan-Protokollen*. Studienarbeit, University of Hamburg, Department of Computer Science, 2002.
4. O.-J. Dahl, E. W. Dijkstra, and C. A. R. Hoare. *Structured Programming*. Acad. Press, London, 7th edition, 1975.
5. Michael Duvigneau, Daniel Moldt, and Heiko Rölke. Concurrent architecture for a multi-agent platform. In *Proceedings of the 2002 Workshop on Agent-Oriented Software Engineering (AOSE'02)*. Springer Lecture Notes, 2002.
6. Foundation for Intelligent Physical Agents. http://www.fipa.org.
7. FIPA. FIPA Interaction Protocol Library Specification, August 2001. http://www.fipa.org/specs/fipa00025/XC00025E.pdf.
8. K. Jensen and G. Rozenberg, editors. *High-level Petri Nets – Theory and Application*. Springer-Verlag, Berlin Heidelberg, 1991.
9. Kurt Jensen. *Coloured Petri Nets*, volume 1. Springer-Verlag, Berlin, 2nd edition, 1996.
10. Michael Köhler, Daniel Moldt, and Heiko Rölke. Modeling the behaviour of Petri net agents. In *Proceedings of the 22nd Conference on Application and Theory of Petri Nets*, pages 224–241, 2001.
11. Olaf Kummer. *Referenznetze*. PhD thesis, University of Hamburg, Department of Computer Science, Logos-Verlag, Berlin, 2002. R35896-7.

12. Olaf Kummer, Frank Wienberg, and Michael Duvigneau. Renew - The Reference Net Workshop. In *Tool Demonstrations - 22nd International Conference on Application and Theory of Petri Nets*, 2001. See also http://www.renew.de.
13. Olaf Kummer, Frank Wienberg, and Michael Duvigneau. Renew - user guide. Dokumentation, University of Hamburg, Department of Computer Science, 2001. http://www.renew.de.
14. LabVIEW. Labview home, 2002. http://www.labview.com.
15. David McIntyre. Comp.lang.visual - Frequently Asked Questions List, 1998. ftp://rtfm.mit.edu/pub/usenet/comp.lang.visual/comp.lang.visual_Frequently-Asked_Questions_(FAQ).
16. Daniel Moldt. *Höhere Petrinetze als Grundlage für Systemspezifikationen*. PhD thesis, University of Hamburg, Department of Computer Science, August 1996.
17. Daniel Moldt and Heiko Rölke. Pattern based workflow design using reference nets. In W.M.P. van der Aalst, A.H.M. ter Hofstede, and M. Weske, editors, *International Conference on Business Process Management*, 2003.
18. Horst Oberquelle. *Sprachkonzepte für benutzergerechte Systeme*. Springer-Verlag, Berlin, 1987.
19. Kirsten Nygaard Ole-Johan Dahl. SIMULA: An ALGOL-based Simulation Language. Communication of the ACM, September 1966.
20. Inc. Pictorius. The home of visual object-oriented development environments., 2002. http://www.pictorius.com/home.html.
21. Wolfgang Reisig. *Elements of Distributed Algorithms: Modeling and Analysis with Petri Nets*. Springer-Verlag New York, October 1997.
22. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns, 2000. http://tmitwww.tm.tue.nl/research/patterns/wfs-pat-2000.pdf.