# Formal Semantics for AUML Agent Interaction Protocol Diagrams

Lawrence Cabac and Daniel Moldt

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

**Abstract.** In this paper we introduce an approach for defining semantics for AUML agent interaction protocol diagrams using Petri net code structures. This approach is based on the usage of net components which provide basic tasks and the structure for Petri Nets. Agent interaction protocol diagrams are used to model agent conversations on an abstract level. By mapping elements of the diagrams to net components we are able to translate the diagrams into Petri nets, i.e to generate code structures from the drawings. We provide tool support for this approach by combining a tool for net components with a tool for drawing agent interaction protocol diagrams. This combined tool is available as a plug-in for *Renew* (**Re**ference **Ne**t **W**orkshop).

**Keywords:** agents, agent interaction protocols, AUML, *CAPA*, high-level Petri nets, *Mulan*, net components, operational semantics, reference nets, *Renew*.

## 1 Introduction

Computer aided software engineering (CASE) tools are programs that support the development of large software systems. They provide tools for modeling and constructing applications. Furthermore, they provide the possibility to generate code from the models, to facilitate the development and to strip the development process of unnecessary recurrent and error-prone manual tasks. Successful tools for various programming languages exist and are in extensive use.

Especially for the usage of the Agent Unified Modeling Language (AUML) within CASE tools, a well defined semantics is required. However, the semantics of agent interactions protocols (AIP) is usually defined by the semantics of sequence diagrams and descriptions in natural languages. These semantics are usually ambiguous and vague. To address the challenge to define a formal semantics for agent interaction protocols, we use high-level Petri nets to define the semantics of agent interaction protocol diagrams. Since Petri nets not only offer a well defined formal, but also operational semantics, we can by this means, not only give formal semantics, but also operational semantics to agent interaction protocol diagrams.

While modeling with Petri nets is common, the idea of programming with Petri nets has not been widely accepted yet. But especially when it comes to

concurrent and distributed processes, e.g. multi agent systems, the advantages of Petri nets are obvious. For this reason, we build concurrent and distributed software systems as multi agent systems on the basis of reference nets [10] - a high-level Petri net formalism. The framework's reference architecture for the multi agent system is *Mulan*[1] / *CAPA*[2]. It is implemented in reference nets and can be executed efficiently in Renew.[3]

The process of implementing application software in *Mulan* requires the construction of Mulan protocols which define the behavior of the agents. A Mulan protocol is a reference net that describes the communication and the internal behavior of an agent. Since the construction of a large system requires building many Mulan protocols, which frequently require similar parts of functionality, the need for software engineering methods and techniques becomes evident. This means that these methods and techniques should include standardizations, conventions and tool support.

We have established two methods to handle the complexity of Mulan protocols and support their construction. First we use net components [2] to construct the Mulan protocols to achieve a unified and structured form of the protocols. Second we model the agent interaction on an abstract level using agent interaction protocol diagrams [4]. Agent interaction protocol diagrams are defined in the AUML [7] used by the FIPA (Foundation for Intelligent Physical Agents [6]). The advantage of modeling in AUML is its intuitive graphical representation of the architecture and the processes.

By offering tool support for the construction and modeling of Mulan protocols, we have succeeded in speeding up their development. Also, the form and the structure of Mulan protocols have become unified and easily readable. Another advantage is that agents' communications are documented in the agent interaction protocol diagrams. Therefore the overview of the system has been enhanced.

In this paper we want to describe one further step towards an integrated development environment for *Mulan* applications. By combining the two described approaches we are able to generate code (here: Petri net) structures from the agent interaction protocol diagrams.

The following pages briefly introduce the *Mulan* net components and the modeling of agent conversations with agent interaction protocol diagrams. Finally, a prototype tool for code (here: Petri net) generation will be presented together with a simple example.

## 2 AUML and Petri Nets

This section describes how agent interaction protocol diagrams semantics can be defined with the help of Petri nets. Examples for some expressions of the diagrams are given to show the general notion.

[1] **Mul**ti-**A**gent **N**ets, [8]
[2] **C**oncurrent **A**rchitecture for **C**ommunicating **A**gents, [5]
[3] **Re**ference **Ne**t **W**orkshop, [11], [10], [12]

Different versions or flavors of AUML have been presented and discussed. See [15], [14] and [9] for the old version (1) and their extensions. These are also used by the FIPA to describe the Interaction Protocols [7]. The new version (2) is still under development (see [13]). However, we are not concerned with the different flavors of the AUML agent interaction protocol diagrams. Since the meaning behind these flavors is basically the same, which makes the graphical representation interchangeable, it is superfluous to discuss this matter here. The shown examples are given in agent interaction protocol diagrams of the old version (1). If semantics is defined for one of the flavors, it could easily be translated to the other flavors.

### 2.1 AUML Flavors

Each of the different flavors of AUML agent interaction protocol diagrams have advantages and disadvantages. We favor the old version of the AUML agent interaction protocol diagrams for several reasons. First, we think that the old representation of agent interactions is more intuitive and clearer in appearance than the new style that is oriented towards the UML 2.0 standard. Second, through dropping the threads, the new version of AUML (2) does not reflect concurrency in a sufficient way. Third, we have been working with the old version successfully over the last two years in several teaching projects with over hundred students. The modeling technique - although new to the students - was well accepted and successfully used in the development of multi-agent applications.

Figure 1 shows the representation of the alternative to send one message out of two possible messages as an example for the different flavors of AUML agent interaction protocol diagrams.[4] This example shows another advantage of the old long (or explicit) version (figure 1 (a)) of agent interaction protocol diagrams. With the usage of message join figures we are able to represent the fact that only one message is actually received by the receiver of the message. Instead, in the short version (b) and also in the new version (c) of AUML there is no *structural* difference in the representation of receiving one message (of for example two possible messages) and the representation of two concurrently sent / received messages (d), although the syntax itself is unambiguous.

### 2.2 Semantics for AUML

By using Petri nets, which offer a well defined operational semantics, it is possible to describe the operational semantics of agent interaction protocol diagrams. To demonstrate how this is done, the example of figure 1 is used and modeled as Petri net in an abstract (or simplified) fashion. Figure 2 shows the representation of two alternatively sent messages modeled with a Petri net[5] (b). In addition, Petri nets also offer the possibility of coarsening (respectively refining) nets. The coarsened Petri net is shown in (c) which can be interpreted in the coarsened

[4] Only parts of the diagrams are shown in the image.

[5] For the exact semantics including inscriptions and pattern variations see [3].

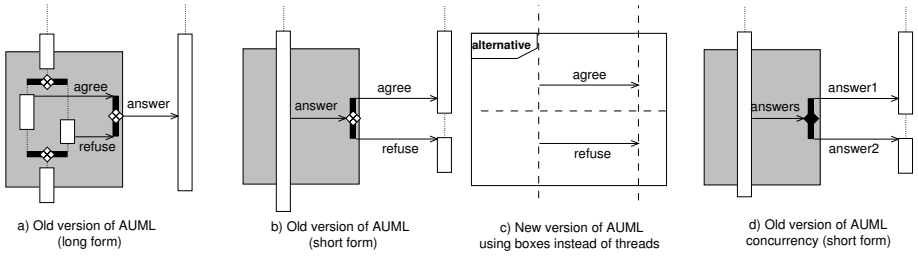


**Fig. 1.** Flavors of AUML, representing the alternative of sending one of two possible (a,b,c) / concurrently sent (d) messages.

agent interaction protocol diagrams as shown in (d). This way of modeling offers the possibility to use abstractions that can clarify the models. It also offers the possibility to exchange one agent's behavior with another possible behavior without the need to alter the behavior of the communicating agent. For instance, the sending agent can always reply with an *agree* and the receiving agent's behavior would not have to be altered.

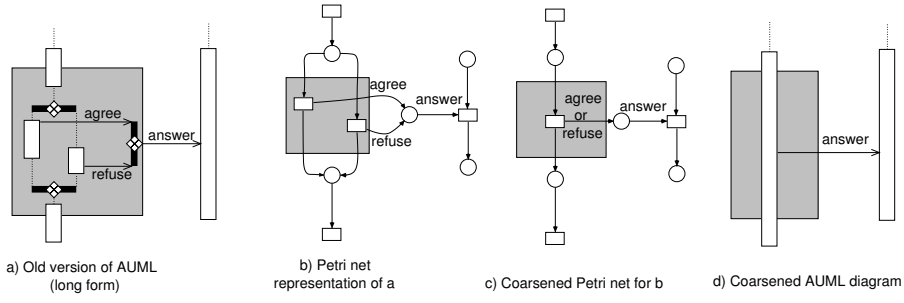


**Fig. 2.** Semantics for alternative message provided by Petri net and coarsened descriptions.

By translating AUML agent interaction protocol diagrams into Petri nets we manage to define the operational semantics of the agent interaction protocol diagrams, i.e. the semantics of the diagrams is defined through the semantics of Petri nets. However, the translation is not done on the abstract level as shown in figure 2. For a translation into a form of Petri nets that is also executable we need to use more elaborate and concrete methods. These methods are a framework architecture for the execution of the resulting protocols and a mapping from agent interaction protocol diagram expressions onto (unified) expressions of Petri net code that can be executed in the frameworks architecture. The first is given

through *Mulan / CAPA* ([8], [5]), which offers a Petri net-based infrastructure as a reference model for a FIPA-compliant multi-agent system. In addition, these multi-agent applications built on *Mulan / CAPA* are also executable. The second is achieved through mapping the agent interaction protocol diagram expressions onto net components. Net components [4] and the way they can be used to compose Mulan protocols are presented in the next section.

## 3 Net Structures

In this section we introduce net components, show how they provide a structure for Mulan protocols, describe the way we model agent communication with AUML diagrams and present how agent interaction protocol diagrams are mapped to Petri net structures using the net components for Mulan protocols.

### 3.1 Net Components

A net component is a subnet. It consists of net elements and additional elements such as default inscriptions or comments. It fulfills one basic task that is so general that the net component can be applied to a broad variety of nets. A net component is defined by its net elements, but it also has a fixed geometrical structure and orientation. This structure contributes to the net structure of the net in which the net component is used. In addition, the geometrical form makes the net component easily identifiable to the developer.

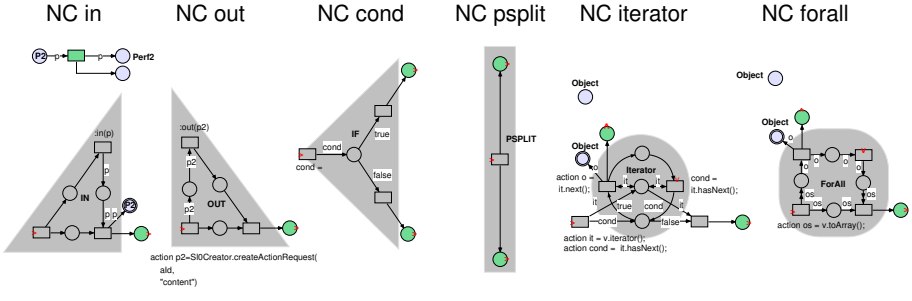


**Fig. 3.** A selection of the *Mulan* net components responsible for message passing, splits and loops.

A set of net components for the Mulan protocols exists that facilitates the construction (modeling) of these Petri nets. Figure 3 shows a selection of the most frequent used *Mulan* net components. The readability of Mulan protocols that are built with net components increases significantly. Furthermore, the structure of the net is unified since it depends on the structure of the net components.

### 3.2 Structured Petri Nets

Petri nets are graphs, i.e. they have a graphical representation. A graphical representation is useful for the understanding of the behavior of a model. A graphic/diagrammatic representation can be more comprehensive than a textual one. Nevertheless a diagram can also be very confusing if it does not provide a clear structure or if substructures of similar behavior are displayed in many different ways. One of the greatest advantages of a diagrammatic representation is the fact that reappearing structures can be perceived by the human cognitive system without any effort.

With the usage of net components, reappearing net structures are effortlessly recognizable and a conventionalized style of the developed Petri nets is achieved.

### 3.3 Modeling Agent Interaction

Modeling agent interaction can be done by using several means. The FIPA [6] uses the AUML agent interaction protocol diagrams [7] for modeling interactions between agents. These diagrams are an extension of the Unified Modeling Language (UML) sequence diagrams [1] but they are more powerful in their expressiveness. They can fold several sequences into one diagram by adding additional elements (AND, XOR and OR) to the usual sequence diagram. Thus they can describe a set of scenarios. Figure 4 shows the FIPA Request Protocol and a compliant Producer Consumer example.

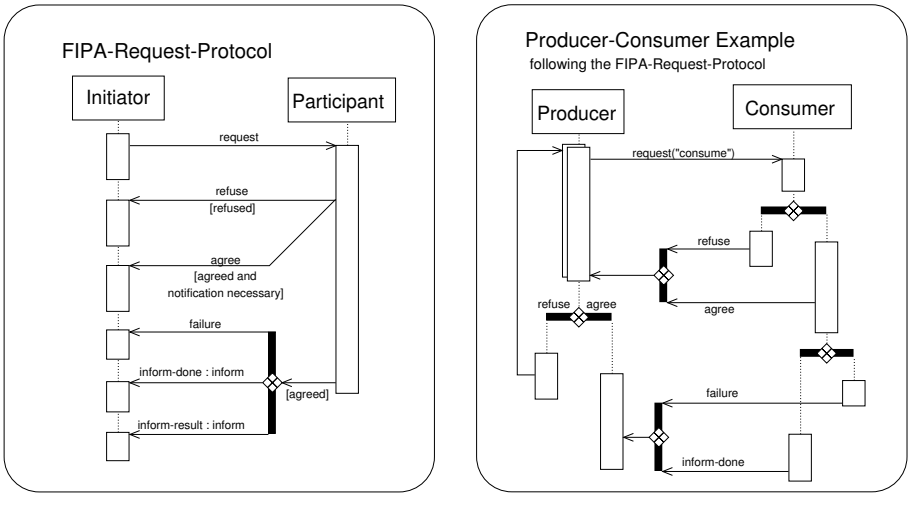


**Fig. 4.** Agent interaction protocol diagrams of the FIPA Request Protocol and a compliant Producer Consumer example.

There are several advantages in the method of modeling agent interactions with agent interaction protocol diagrams. Three of them are:

- The models are easily readable by all participants, because they are close to UML.
- Abstract modeling increases the overview over the system.
- A means of communication, specification and documentation is established.

### 3.4 Mapping Agent Interaction Protocol Diagrams to *Mulan* Protocols

The combination of the two tools – the tool for applying net components and the tool for drawing diagrams – is done as follows. By using agent interaction protocol diagrams for modeling agent communication the structure of the Mulan protocols can be derived directly from the diagram. This is done by mapping the relating elements in the agent interaction protocol diagrams to the net components. In detail this means (compare with figures 3 and 6 to 11):

- A message arc is the abstract representation of the basic messaging net components (*NC out* and *NC in*).
- A split figure is the abstract representation of the conditional (*NC cond*) or a parallel split (*NC psplit*).
- A life line between a role descriptor and an activation marks the start of a protocol (*NC start*)

Several other net components are not yet represented in the abstract model since no elements exist to represent their functionality in agent interaction protocol diagrams. It seems that for some of these basic tasks the notation of the agent interaction protocol diagrams has to be extended.

- Loops are not well represented yet. Proposals exist for their representation, but so far there has been no way to determine whether a sequential or a concurrent process is desired.
- Sub-calls: It is possible to nest agent interaction protocol diagrams, but semantics is ambiguous.

In general, the main problem is the vague semantics of the agent interaction protocol diagrams. However, this can be of advantage while modeling. The process of modeling can be accelerated by postponing the description of details to the implementation or some implicit knowledge defines the missing semantics. In contrast, if there is the need to define a specific mapping, clear semantics is desired. For us this becomes necessary because we follow the metaphor of "Implementing by Modeling".

## 4 From Model to Net

This section describes the tool support for mapping agent interaction protocol diagrams to Mulan protocol structures. The tool generates Petri net structures

that can be compared to program source code skeletons. To achieve a functional Mulan protocol, the inscriptions have to be adjusted and - if needed - the classes for the messages have to be implemented. Furthermore, the net has to be adjusted (refactored) if an element has to be used that is not yet provided, e.g. loops.

### 4.1 Code Generation

In the last developer version of *Renew*[6], a tool for applying net components to nets and a tool to draw agent interaction protocol diagrams were included. The developers of Mulan protocols were able to draw diagrams to model the behavior of agents with the diagram tool. Diagrams were used as means of specification, documentation and communication among the developers of *Mulan* applications. The basic communication protocols were established and defined using these diagrams. So agent interaction protocol diagrams only defined the way of communication between the agents, but not the internal behavior. Usually, but not necessarily, different developers implement the Mulan protocols for each agent defining the external and the internal behavior of an agent. As long as the different developers constructed the Mulan protocol according to the given agent interaction protocol diagrams, the agents could communicate in a correct way.

Figure 5 displays the *Renew* GUI including the control elements of the diagram plug-in. The last palette contains buttons for the drawing of role figures, activations, messages, life lines, split and join figures, note figures, frames and inscriptions.

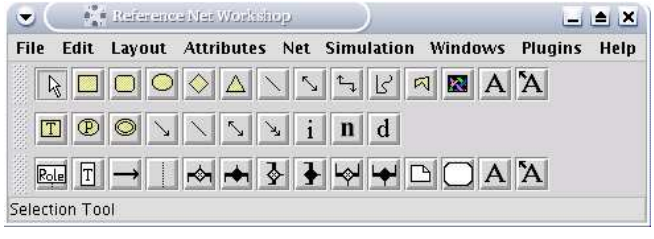


**Fig. 5.** The GUI of *Renew* with the tool support for drawing diagrams.

The process of constructing the Mulan protocol requires the manual task of mapping the diagram structures to each Mulan protocol. This was done by connecting net components with each other using the net components tool. Many elements in the agent interaction protocol diagrams could be mapped onto net components in a straight forward fashion as described in section 3.4.

[6] See also *Renew* 2.0, [12].

It seems obvious that this task can be performed automatically by the introduced tool. Since agent interaction protocol diagrams describe the interactions and the splitting of activities, we decided to implement a prototype that is capable of generating Petri net skeletons from the diagrams that reflect these structures. To be able to execute the generated code, it has to be refactored and adjusted with additional functionality. This is a common approach for code generation: The parts that can be derived from the model are generated and the rest is added manually.

### 4.2 Geometrical Arrangement of *Mulan* Protocols

In addition to textual code generation, the construction of Petri nets also has to deal with the layout of the generated nets. The structure of nets is crucial to readability. If the code is used as it is generated, there is no need to design the layout of the code. But if the code has to be adjusted, the programmer has to understand the code. So the layout becomes important.

Net components provide a structure for Petri nets. This is not only true for the manually made nets but also for generated code. For each net component only some additional information is needed that provides the knowledge of how it can be connected to other net components and how this is reflected in the layout. The net structure results from the smaller structure of the net components similar to the structure of a snowflake, which results from the structures of molecules of water. So net components provide the structure by imposing their own structure onto the net structure. However, the generated Petri net code structures have always the same form due to the automated generation.

### 4.3 Example: Producer-Consumer

Generating code skeletons from the Producer-Consumer example agent interaction protocol diagram is possible and results in two Mulan protocol skeletons. Figure 6 shows the diagram from which the code is generated.

Figure 7 shows the source of the model augmented with the geometrical representation of the corresponding net components and Figures 8 and 10 show the two parts of the model that match the two Mulan protocols, rotated by ninety degrees. The resulting skeletons are shown in Figures 9 and 11. All augmented models are just presented here to illustrate the matching of diagram elements to net components. They are not necessary for the generation of the Mulan protocols. The generated Mulan protocol skeletons are shown (Figures 9 and 11) as they have been generated, without any modification of the nets or the inscriptions.

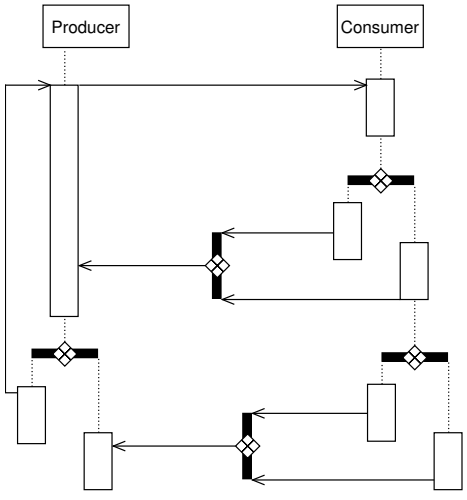


**Fig. 6.** Source for generation of the Producer-Consumer example.

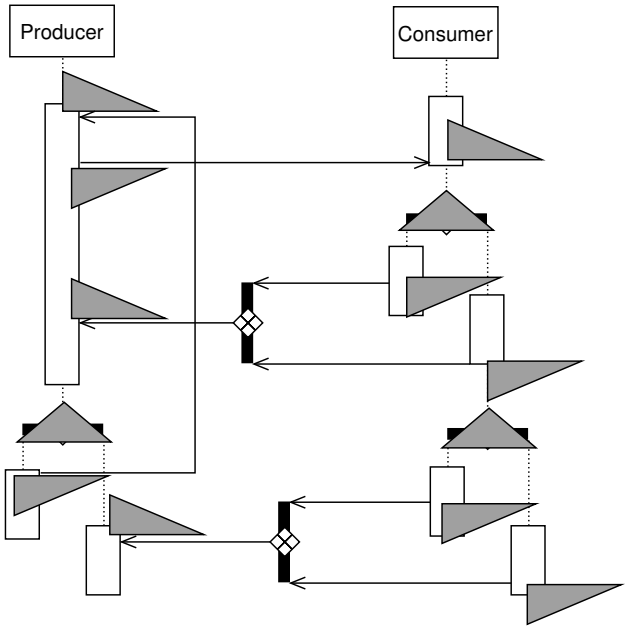


**Fig. 7.** The source from Figure 6, with the geometrical representation of the corresponding net components.

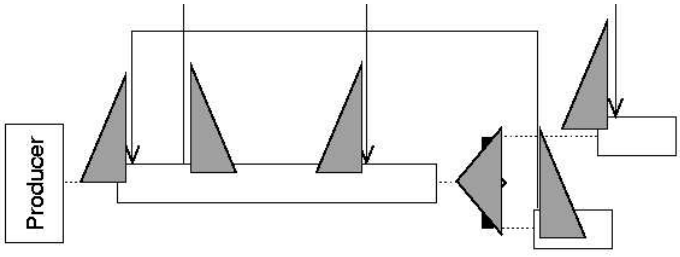


**Fig. 8.** The Producer part of the source from Figure 6, with the geometrical representation of the corresponding net components. Rotated by ninety degrees to fit the orientation of the resulting Mulan protocol.

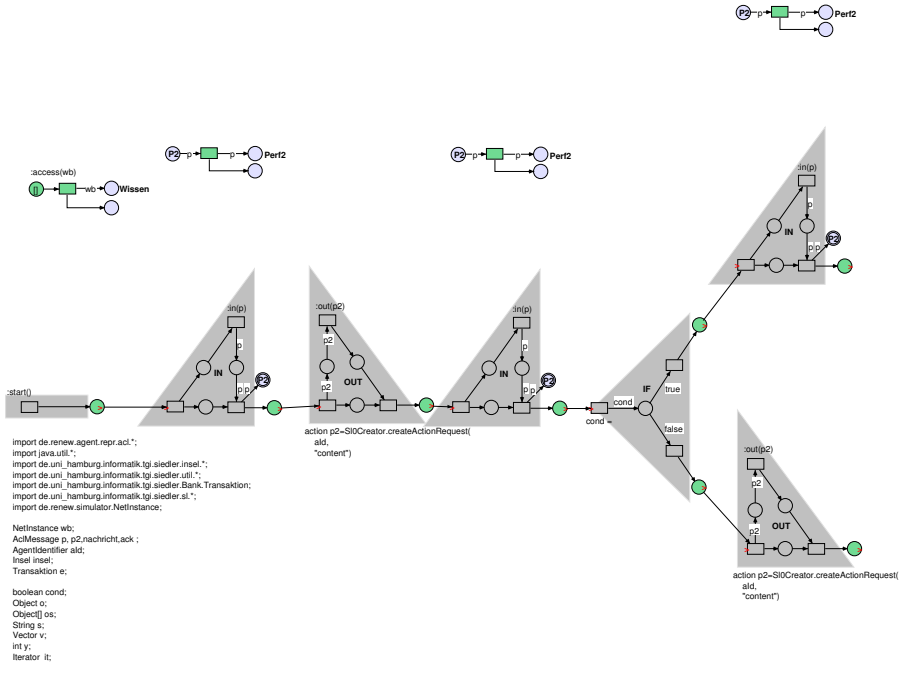


**Fig. 9.** Generated Producer Mulan protocol skeleton

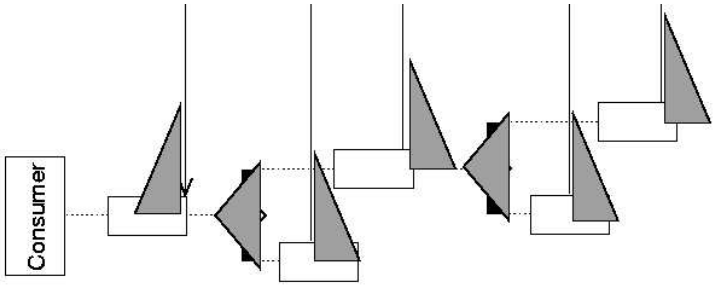


**Fig. 10.** The Consumer part of the source from Figure 6, with the geometrical representation of the corresponding net components. Rotated by ninety degrees to fit the orientation of the resulting Mulan protocol.

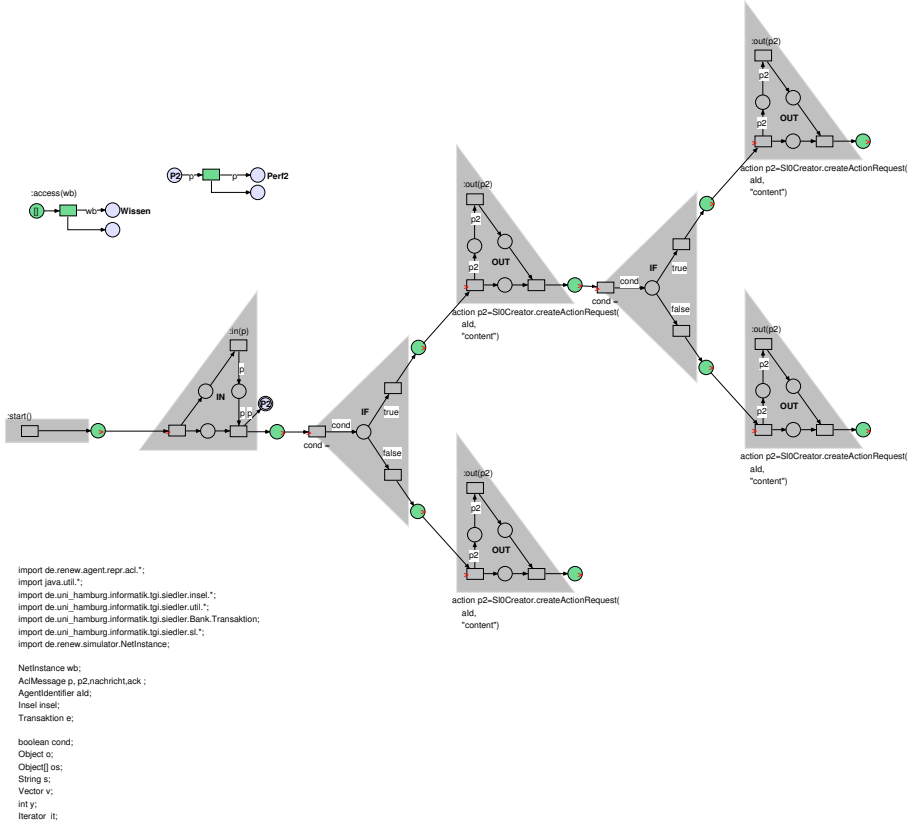


**Fig. 11.** Generated Consumer Mulan protocol skeleton.

The results of this simple example are satisfying. The Mulan protocols do not need to be refactored because the conversation deals only with communication and decisions. However, in order to convert these skeletons into executable Mulan protocols, we still have to work on them. The relevant data has to be extracted from the messages and from the agents' knowledge bases. Furthermore, we have to define the decisions and the outgoing messages.

It seems that for more complex communication protocols, dealing with internal behavior, loops or sub-calls, this simple approach is not powerful enough. But since most of the used net components deal with message passing, splits, starting and stopping, this approach will already generate more than ninety percent of the Petri net code structure. Only the parts that deal with broadcasting or multi-casting messages, or the parts that deal with internal behavior have to be adjusted manually.

## 5 Conclusion

Software engineering methods have been developed to enhance the construction of large software systems and are used and applied successfully. These methods can also be applied for software development based on high-level Petri nets. With more extensive use of these conventional techniques the process of Petri net-based software developing can be improved. The advantages of Petri nets lie in their inherent concurrency; UML is a powerful modeling language that is well accepted and widely spread. Both – UML and Petri nets – can contribute to the construction of large distributed and / or concurrent systems. Combining their advantages results in a powerful method to develop applications.

A crucial point in this is that the semantics of the used AUML agent interaction protocol diagrams has to be well defined. This holds for developing / designing as well as for the generation of code (structures). We showed that the defining of agent interaction protocol diagram semantics can be achieved by mapping the diagrams onto Petri nets. For this a net component-based approach was used, which enables us to generate Petri net code structures from the diagrams. The net components provide the functionality of syntactic / semantic unities as well as the structure of the resulting Mulan protocols. However, the definition of semantics is realized in the tool and not explicitly given here.

In addition, by using net components the Mulan protocols are structured and their structure is unified. This increases the readability of Mulan protocols and the software development is accelerated. The integration of UML-based modeling into the developing process has contributed to the clearness of the system and its overall structure. Apart from the development process, the focus of development was also altered by using AUML. The center of focus shifted from the agents' processes to the communication between the agents.

The introduction of UML-based modeling into the developing process and the unification of net structures turned out to be a successful approach. Nevertheless, the integration of conventional methods as UML and development of software with Petri nets can be driven further. In this paper we presented one step towards

an integrated development environment for the construction of *Mulan*-based application software. By merging the two approaches – net components and agent interaction protocol diagrams – we are able to generate skeletons of Mulan protocols from interaction diagrams.

For the development of large applications on the basis of the Petri net-based multi-agent system *Mulan / CAPA*, tool support is needed on different levels of abstraction. This includes the construction of Mulan protocols, the modeling of agent interaction and the debugging of the system during development. The first two points are covered by the tool support for net components and agent interaction protocol diagrams. Additionally, we can now also ease the developing process by generating code in the form of Petri net structures from diagrams.

The integration of methods and techniques can also be driven further. For a representation of all *Mulan* net components in the agent interaction protocol diagrams the notation for these diagrams has to be augmented by corresponding elements. Another useful approach is to integrate a round-trip engineering functionality into the diagram tool so that changes that are made in the Mulan protocols are reflected in the diagrams.

## References


1. Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, Reading, Massachusetts, 1996.
2. Lawrence Cabac. *Entwicklung von geometrisch unterscheidbaren Komponenten zur Vereinheitlichung von Mulan-Protokollen*. Studienarbeit, University of Hamburg, Department of Computer Science, 2002.
3. Lawrence Cabac. *Modeling Agent Interaction with AUML Diagrams and Petri Nets*. Diplomarbeit, University of Hamburg, Department of Computer Science, Vogt-Kölln Str. 30, 22527 Hamburg, Germany, 2003.
4. Lawrence Cabac, Daniel Moldt, and Heiko Rölke. A proposal for structuring Petri net-based agent interaction protocols. In *Lecture Notes in Computer Science: 24th International Conference on Application and Theory of Petri Nets, Eindhoven, Netherlands, June 2003*. Springer-Verlag, June 2003.
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. Michael Köhler, Daniel Moldt, and Heiko Rölke. Modeling the structure and behaviour of Petri net agents. In *Proceedings of the 22nd Conference on Application and Theory of Petri Nets*, pages 224–241, 2001.
9. Jean-Luc Koning, Marc-Philippe Huget, Jun Wei, and Xu Wang. Extended modeling languages for interaction protocol design. *Lecture Notes in Computer Science*, 2222:68–76, 2002.
10. Olaf Kummer. *Referenznetze*. Dissertation, University of Hamburg, Department of Computer Science, Logos-Verlag, Berlin, 2002.

11. 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.
12. Olaf Kummer, Frank Wienberg, Michael Duvigneau, Jörn Schumacher, Michael Köhler, Daniel Moldt, Heiko Rölke, and Rüdiger Valk. An Extensible Editor and Simulation Engine for Petri Nets: Renew. In W.Reisig J. Cortadella, editor, *International Conference on Application and Theory of Petri Nets 2004*. Springer-Verlag, 2004.
13. James Odell and Marc-Philippe Huget. FIPA Modeling: Interaction Diagrams. Working draft, Foundation for Intelligent Physical Agents, July 2003. `http://www.auml.org/auml/documents/ID-03-07-02.pdf`.
14. James Odell, H. Van Dyke Parunak, and Bernhard Bauer. Extending UML for agents. In Gerd Wagner, Yves Lesperance, and Eric Yu, editors, *Proc. of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence*, pages 3–17, 2000. `http://www.jamesodell.com/ExtendingUML.pdf`.
15. James Odell, H. Van Dyke Parunak, and Bernhard Bauer. Representing agent interaction protocols in UML. In Paolo Ciancarini and Michael Wooldridge, editors, *Agent-Oriented Software Engineering*, pages 121–140. Springer, Berlin, 2001. `http://www.auml.org/auml/supplements/Odell-AOSE2000.pdf`.