Generating Code Structures for Petri Net-Based Agent Interaction Protocols Using Net Components

Lawrence Cabac

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

March 31, 2004

Abstract

In this paper we introduce a straight forward approach for generating Petri Net code structures from AUML agent interaction protocol diagrams. 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 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 integrated into Renew (The Reference Net Workshop).

Keywords: agents, agent interaction protocols, AUML, high-level Petri nets, Mulan, net components, 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 developing process of unnecessary recurrent and error-prone manual tasks. Successful tools for various programming languages exist and are in extensive use.

While modeling with Petri nets is common, still the idea of programming with Petri nets is a possibility that is not well accepted. But especially when it comes to concurrent and distributed processes, e.g. multi agent systems, the advantages of Petri nets are undeniable. For this reason we build concurrent and distributed software systems as multi agent systems on the basis of reference nets [8] - a high-level Petri net formalism. The framework's reference architecture for the multi agent system is Mulan (MULti Agent Nets, [7], [4]). It is implemented in reference nets and can be executed efficiently in Renew (The Reference Net Workshop, [9], [8]).

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 use similar parts of functionality, the need for software engineering methods and techniques becomes evident. This includes standardization, 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 [3]. Agent interaction protocol diagrams are defined in the AUML (Agent Unified Modeling Language [6]) standardized by the FIPA (Foundation of Intelligent Physical Agents [5]). The advantages of modeling in AUML are its standardization and the 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 oversight over 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 (Petri net) structures from the agent interaction protocol diagrams.

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

2 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.

2.1 Net Components

A net component is a subnet. It consists of net elements and additional elements such as default inscription 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.

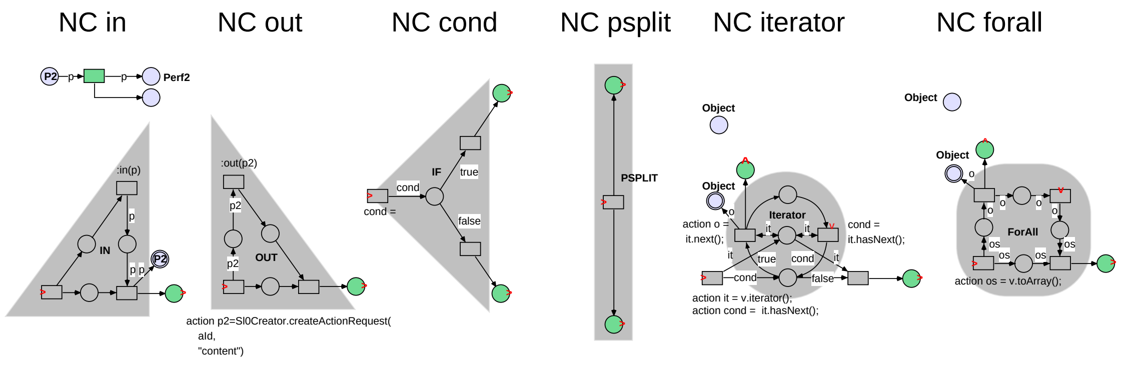


Figure 1: A selection of the Mulan net components responsible for message passing, splits and loops.

A set of net components for the Mulan protocols exists that facilitates the construction (modeling) of these Petri nets. Figure 1 shows a selection of the most frequent used Mulan net components. The readability of Mulan protocols that are built with net components is increased significantly. Furthermore the structure of the net is unified since it depends on the structure of the net components.

2.2 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. 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 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.

2.3 Modeling Agent Interaction

Modeling agent interaction can be done by using several means. The FIPA [5] uses the AUML agent interaction protocol diagrams [6] 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 2 shows the FIPA Request Protocol and a compliant Producer Consumer example.

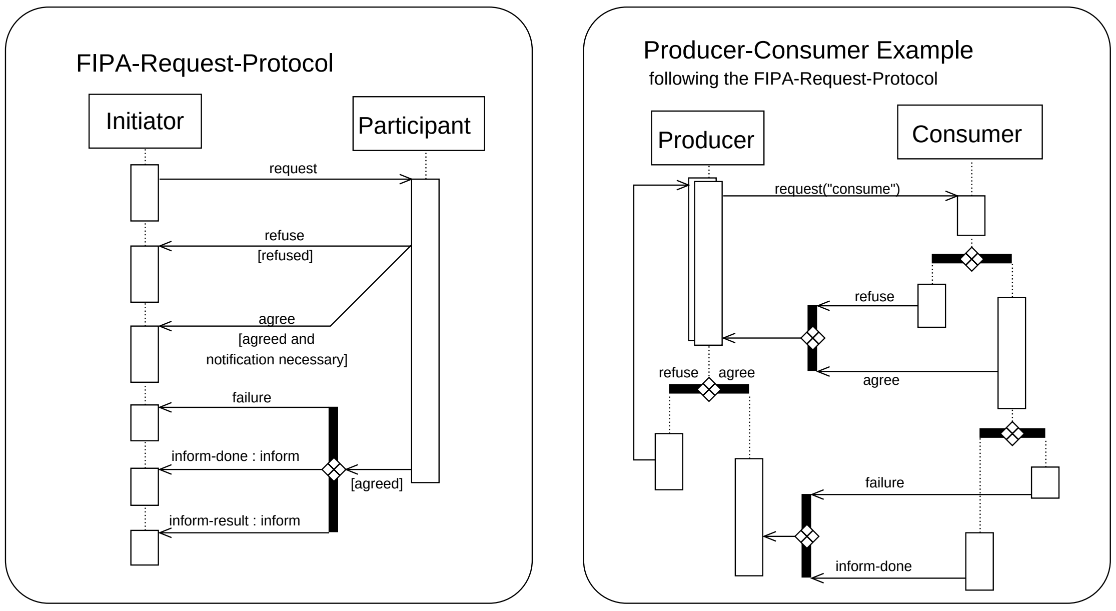


Figure 2: Agent interaction protocol diagrams of the FIPA Request Protocol and a compliant Producer Consumer example.

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 oversight over the system.
- A means of communication, specification and documentation is established.

2.4 Mapping Agent Interaction Protocol Diagrams to Mulan Protocols

The combination of the two introduced tools 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 1 and 3 to 8):

- 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 there exist no elements 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. There exist proposals for their representation, but there is no way yet to determine whether a sequential or a concurrent process is desired.
- Sub-calls: It is possible to nest agent interaction protocol diagrams, but the semantics for that is not fixed.

In general the main problem is that the semantics of the agent interaction protocol diagrams are not very clear nor fixed. 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.

3 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.

3.1 Code Generation

In the last developer version of Renew, 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.

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 2.4.

It seems obvious that this task can be performed automatically by the here 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.

3.2 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 to 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 just like the structure of a snowflake results from the structures of atoms of water. So net components provide the structure by imposing their own structure onto the net structure.

3.3 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 3 shows the diagram from which the code is generated.

Figure 4 shows the source of the model augmented with the geometrical representation of the corresponding net components and figures 5 and 7 show the two parts of the model that match the two Mulan protocols, rotated by ninety degrees. The resulting skeletons are shown in Figures 6 and 8. 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 6 and 8)as they are generated without any modification of the nets or the inscriptions.

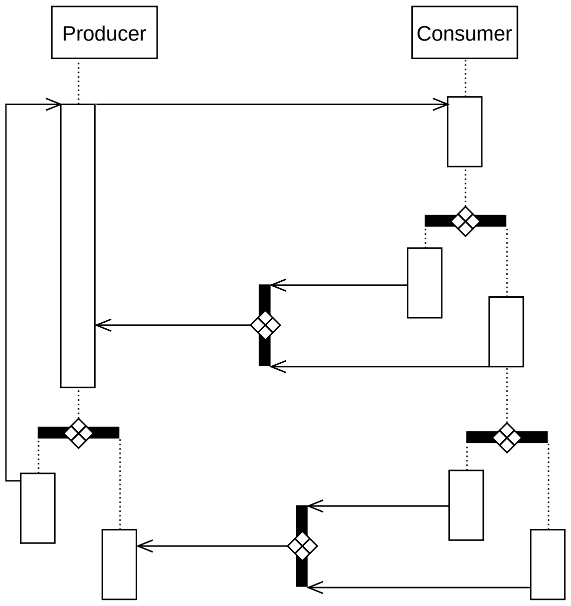


Figure 3: Source for generation of the Producer Consumer example.

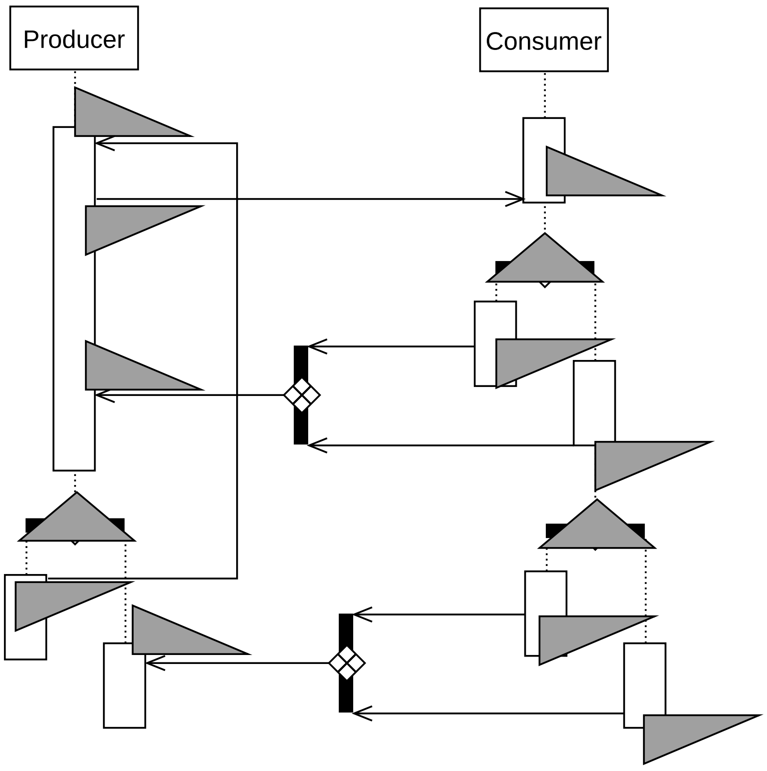


Figure 4: The source from figure 3, with the geometrical representation of the corresponding net components.

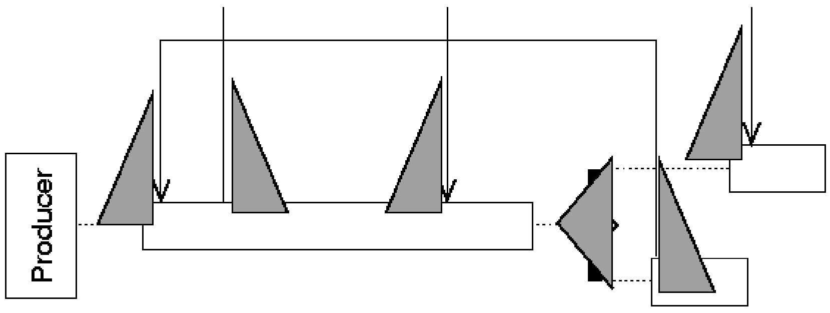
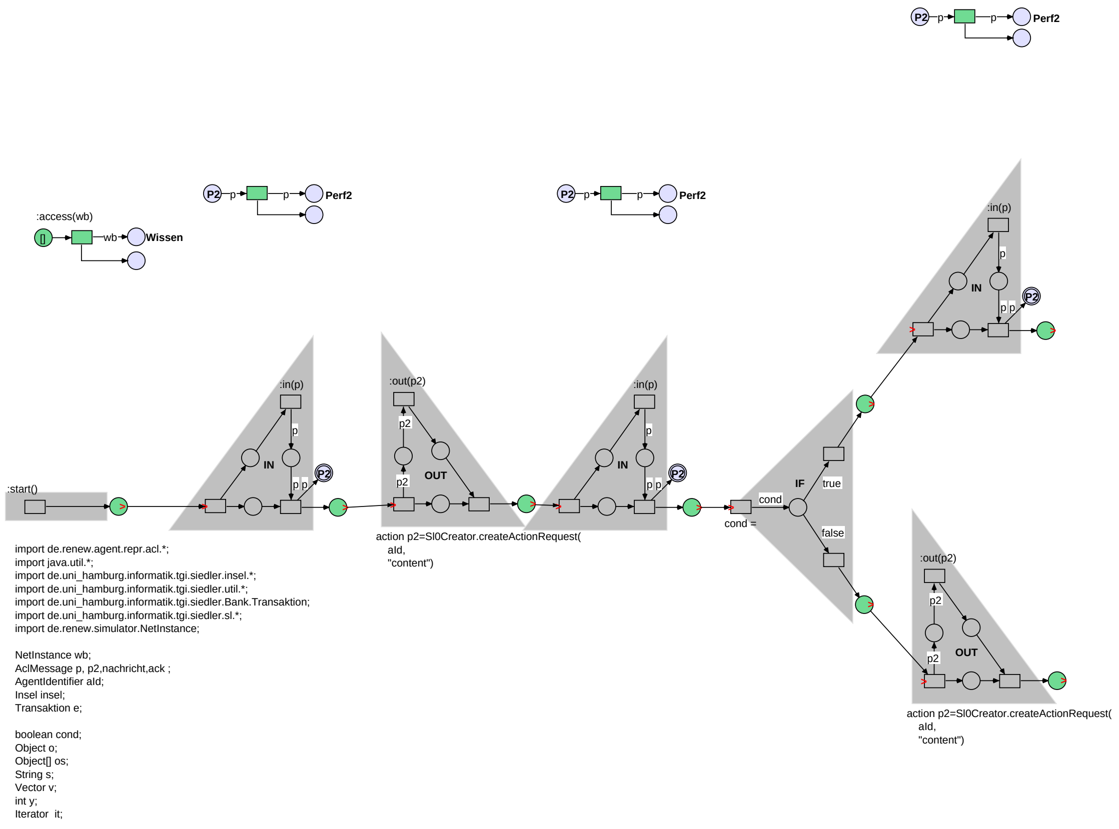


Figure 5: The Producer part of the source from figure 3, with the geometrical representation of the corresponding net components. Rotated by ninety degrees to fit the orientation of the resulting Mulan protocol.



```
import de.renew.agent.repr.acl.*;
import java.util.*;
import de.uni_hamburg.informatik.tgi.siedler.insel.*;
import de.uni_hamburg.informatik.tgi.siedler.util.*;
import de.uni_hamburg.informatik.tgi.siedler.Bank.Transaktion;
import de.uni_hamburg.informatik.tgi.siedler.sl.*;
import de.renew.simulator.NetInstance;

NetInstance wb;
AclMessage p, p2,nachricht,ack ;
AgentIdentifier aId;
Insel insel;
Transaktion e;

boolean cond;
Object o;
Object[] os;
String s;
Vector v;
int y;
Iterator  it;
```

```
action p2=Sl0Creator.createActionRequest(
   aId,
   "content")
```

Figure 6: Generated Producer Mulan protocol skeleton

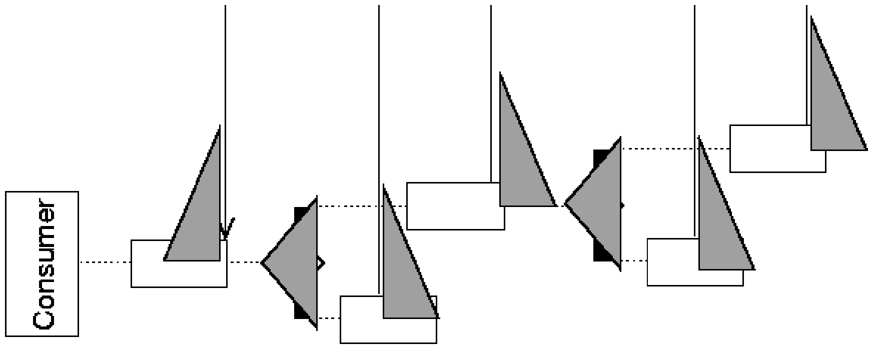
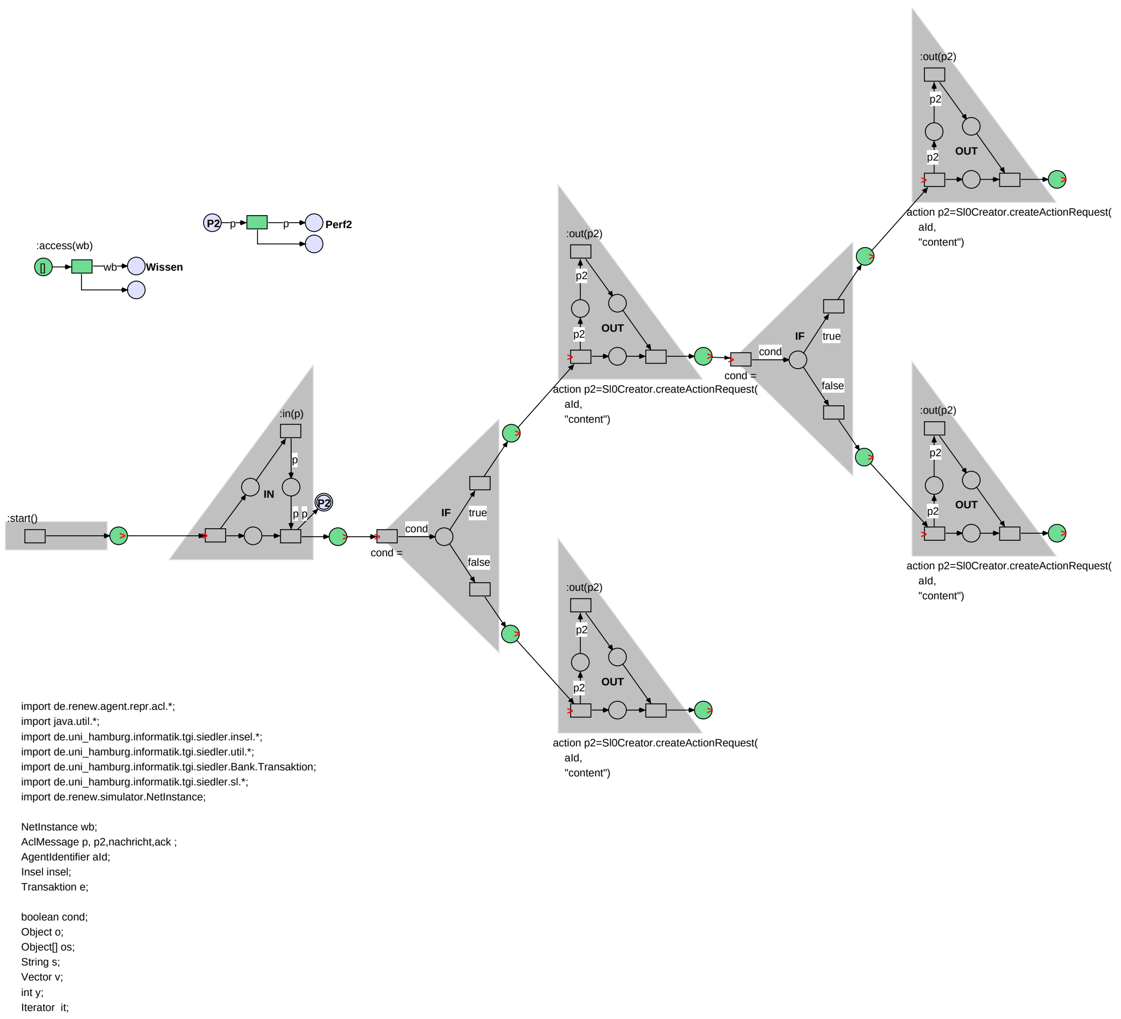


Figure 7: The Consumer part of the source from figure 3, with the geometrical representation of the corresponding net components. Rotated by ninety degrees to fit the orientation of the resulting Mulan protocol.



```
import de.renew.agent.repr.acl.*;
import java.util.*;
import de.uni_hamburg.informatik.tgi.siedler.insel.*;
import de.uni_hamburg.informatik.tgi.siedler.util.*;
import de.uni_hamburg.informatik.tgi.siedler.Bank.Transaktion;
import de.uni_hamburg.informatik.tgi.siedler.sl.*;
import de.renew.simulator.NetInstance;

NetInstance wb;
AclMessage p, p2,nachricht,ack ;
AgentIdentifier aId;
Insel insel;
Transaktion e;

boolean cond;
Object o;
Object[] os;
String s;
Vector v;
int y;
Iterator  it;
```

Figure 8: 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. But 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.

4 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 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.

The advantages of tool supported Mulan protocol development are clear. 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. Besides the developing process, also the focus of development was altered by using AUML. The center of focus shifted from the agent's process 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 (UML) and development of software with Petri nets can be driven further. In this paper we presented one step towards an integrated development environment (IDE) 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 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 now can also ease the developing process by generating code (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, 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.

[4] Michael Duvigneau, Daniel Moldt, and Heiko Rölke. Concurrent architecture for a multiagent platform. In *Proceedings of the 2002 Workshop on Agent-Oriented Software Engineering (AOSE'02).* Springer Lecture Notes, 2002.

[5] Foundation for Intelligent Physical Agents. http://www.fipa.org.

[6] FIPA. FIPA Interaction Protocol Library Specification, August 2001. http://www.fipa.org/specs/fipa00025/XC00025E.pdf.

[7] 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.

[8] Olaf Kummer. *Referenznetze.* PhD thesis, University of Hamburg, Department of Computer Science, Logos-Verlag, Berlin, 2002. R35896-7.

[9] 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.