Agent Models for Concurrent Software Systems

Lawrence Cabac, Till Dörges, Michael Duvigneau, Daniel Moldt,
Christine Reese, Matthias Wester-Ebbinghaus

University of Hamburg, Department of Computer Science,
Vogt-Kölln-Str. 30, D-22527 Hamburg
http://www.informatik.uni-hamburg.de/TGI

Abstract In this work we present modeling techniques for the development of multi-agent applications within the reference architecture for multi-agent system Mulan. Our approach can be characterized as model driven development by using models in all stages and levels of abstraction regarding design, implementation and documentation. Both, standard techniques from software development as well as customized ones are used to satisfy the needs of multi-agent system development. To illustrate the techniques and models within this paper we use diagrams created during the development of an agent-based distributed Workflow Management System (WFMS).

Keywords: high-level Petri nets, nets-within-nets, reference nets, net components, Renew, modeling, agents, multi-agent systems, Paose

1 Introduction

The agent metaphor is highly abstract and it is necessary to develop software engineering techniques and methodologies that particularly fit the agent-oriented paradigm. They must capture the flexibility and autonomy of an agent's problem-solving capabilities, the richness of agent interactions and the (social) organizational structure of a multi-agent system as a whole.

Many agent-oriented software development methodologies have been brought forward over the past decade, many of them already in a mature state. Here, we present our contribution to this rapidly evolving field of research by describing agent models and their usage during the development of multi-agent systems with Mulan (Multi-Agent Nets [7]). As a matter of course there exist many analogies to related agent-oriented development techniques and methodologies like Gaia [15], MaSE [4] or Prometheus [11]. This concerns development methods and abstractions like use cases, system structure (organization) diagrams, role models, interaction diagrams and interaction protocols as well as more fine-grained models of agents' internal events, data structures and decision making capabilities.

Our approach Paose (Petri net-based AOSE) facilitates the metaphor of multi-agent systems in a formally precise and coherent way throughout all aspects of software development as well as a concurrency-aware (Petri net-based) modeling and programming language. The metaphor of multi-agent systems is

R. Bergmann, G. Lindemann (Eds.): Multiagent System Technologies, LNAI 5244, pp. 37–48, 2008.

formalized by the MULAN reference architecture, which is modeled using reference nets. We integrate several ideas from the methodologies mentioned above as well as concepts from conventional modeling techniques (UML). The result of those efforts is a development methodology that continuously integrates our philosophy of Petri net-based and model-driven software engineering in the context of multi-agent systems.

This paper focuses on the set of modeling techniques used within the PAOSE approach. Other aspects have already been presented, for example *the multi-agent system as a guiding metaphor* for development processes in [1]. In Section 2 we introduce the basic conceptual features of multi-agent application development with MULAN. The particular techniques, models and tools are presented in Section 3.

2 Concepts of Application Development with Mulan

Reference nets[1] and thus also MULAN run in the virtual machine provided by RENEW [9], which also includes an editor and runtime support for several kinds of Petri nets. Since reference nets may carry complex JAVA-instructions as inscriptions and thereby offer the possibility of Petri net-based programming, the MULAN models have been extended to a fully elaborated and running software architecture, the FIPA[2]-compliant re-implementation CAPA [5].

Reference nets can be regarded as a concurrency extension to *Java*, which allows for easy implementation of concurrent systems in regard to modeling (implementation) and synchronization aspects. Those – often tedious – aspects of implementation regarding concurrency are handled by the formalism as well as by the underlying virtual machine. In this aspect lies the advantage of our approach. We rely on a formal background, which is at the same time tightly coupled with the programming environment Java. MULAN can be regarded as a reference architecture for concurrent systems providing a highly structured approach using the multi-agent system metaphor.

We describe the internal components of the MULAN agent followed by an investigation of the interrelations between them, which results in the organizational structure of the system. For the details of further aspects of the MULAN architecture we refer to Rölke et. al. [7].

2.1 The Mulan Agent

The reference net-based multi-agent system architecture MULAN (Multi Agent Nets) structures a multi-agent system in four layers, namely *infrastructure*, *platform*, *agent* and *protocol* [7,12]. Figure 1 shows a schematic net model of a

[1] Reference nets [8] are high-level Petri nets comparable to colored Petri nets. In addition they implement the nets-within-nets paradigm where tokens are active elements (token refinement). Reference semantics is applied, so tokens are *references* to *net instances*. *Synchronous channels* allow for communication between net instances.

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

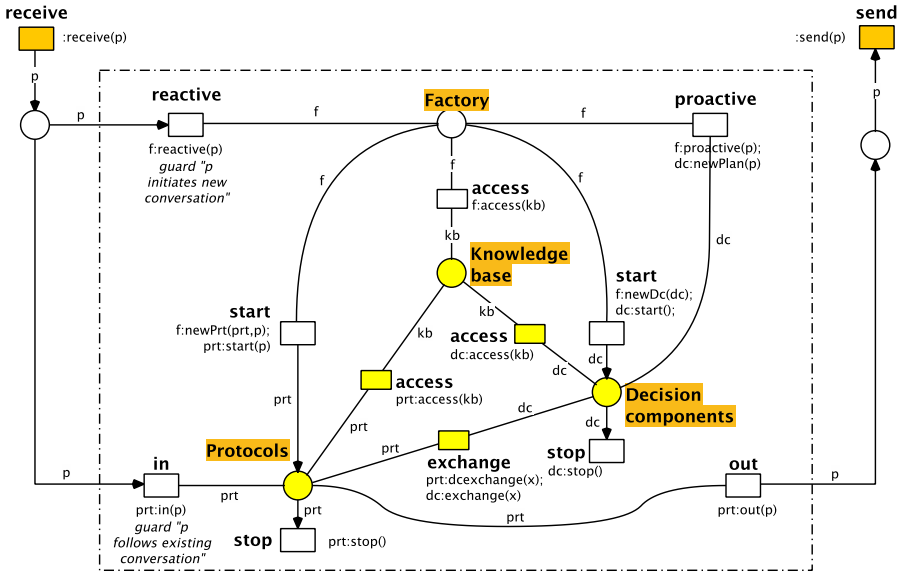

Figure 1. Agent net

MULAN agent. Several parts of the operational model, such as inscriptions, synchronous channels and initialization, are omitted for clearness. Instead, descriptive names have been given to the net elements representing synchronous channels or place contents. The model stresses that the agent is a communicating agent being able to receive and send messages. The labeled places store references to net instances that provide or refine the main functionality of the agent. These are the Factory, the Knowledge base, the Decision components, and the Protocols. Protocol and decision component nets comprise parts of the domain-specific agent behavior, the two corresponding places in the agent net may contain numerous net instances (compare *nets-within-nets* [14]).

The *factory* produces net instances from net patterns of protocols and decision components. It realizes reactive and proactive behavior by examining incoming messages and the agent's knowledge.

The *knowledge base* offers database functionality including atomic query, create, remove and modify operations to other subnets of the agent. It is used to store persistent information to be shared by protocol nets and decision components, for example the agent's representation of the environment. The knowledge base also stores the agent's configuration. It holds information about provided and required services, and a mapping of incoming messages to protocol nets.

Protocol nets implement domain-specific agent behavior. Each protocol net template models the participation of an agent role in a multi-agent interaction protocol. Instantiated protocol nets reside on the place Protocols of the agent, handle the processing of received messages and may generate outgoing messages.

Protocol net instances are the manifestations of the agent's involvement in an interaction with one or more other agents. They can access the knowledge base and exchange information with decision components through the *exchange* channel.

Decision components implement, like protocol nets, domain-specific agent behavior. A decision component net instance can be queried by protocol net instances to add flexibility to the static, workflow-like character of protocol nets. Decision components can also initiate proactive agent behavior by requesting the factory to instantiate protocol nets. Thus an AI-like planning component can be attached to an agent as a decision component or the functionality can be implemented directly as reference nets. Decision components may also encapsulate external tools or legacy code as well as a graphical user interface whereby the external feedback is transformed into proactive agent behavior.

2.2 Organizational Structure

In a multi-agent application the organizational structure has to be defined, such that responsibilities for all aspects of the system are specified. The general perspectives in the area of a multi-agent systems are the structure, the interactions, and the terminology. These perspectives are orthogonal with connecting points at some intersections (compare Figure 2).

The structure of a multi-agent system is given by the agents, their roles, knowledge bases and decision components. The behavior of a multi-agent system is given by the interactions of the agents, their communicative acts and the internal actions related to the interactions. The terminology of a multi-agent system is given as a domain-specific ontology definition that enables agents and interactions to refer to the same objects, actions and facts. Without a common ontology successful interactions are impossible.

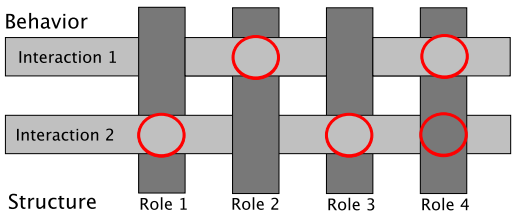

Figure 2. Two dimensional matrix showing perspectives (*behavior, structure*).

A schematic two dimensional matrix is depicted in Figure 2 showing the independence and interconnection of agents and interactions. Neither is there any direct relationship between any pair of agents, nor between any pair of interactions. Thus these architectural elements are independent and depicted in parallel to each other. Agents and interactions are shown as orthogonal because

each agent is involved in some interactions. The general case for any two structural and/or behavioral elements is independence, but interconnections exist. Coupled agents and interactions are marked by circles in the figure.

The terminology defined as ontology is the third dimension of perspectives (omitted in the diagram). It is orthogonal to the other two, but it tends to have many interconnecting points because each interaction and each agent uses parts of the ontology definition to fulfill its purpose.

Since the three perspectives are orthogonal to each other and independent within the perspective, it is easily possible to divide the tasks of design and implementation into independent parts. This means that different interactions can be developed by independent sub-teams and different agents can be designed by other independent sub-teams. Between agent teams and interaction teams, coordination is needed for the crucial parts only (circles in the diagram).

These three perspectives enable us to develop the parts of the system independently and concurrently – thus also distributedly – as long as there is enough coordination / synchronization between intersecting groups.

3 Techniques, Models and Development Tools

In this section we describe the techniques applied during the various stages of multi-agent application development with MULAN. An agent-based Workflow Management System serves as an example application to provide real-world models. However, since the WFMS is not the objective here, we will not go into detail of its design.

We present the applied techniques and resulting models starting with the coarse design giving an overview over the system, continuing with the definition of the structure of the multi-agent application, the ontology and the behavior of the agents.

3.1 Coarse Design

The requirements analysis is done mainly in open discussions. The results are captured in simple lists of system components and agent interactions. This culminates in a use case diagram as shown in Figure 3. Of course other methods to derive use cases can also be applied.

A use case diagram is especially useful to derive the multi-agent application matrix because we depict agent roles in the system as actors in the diagram. In contrast, usually in use case models the actors represent real world users.

Figure 3 shows the Account Manager (AM) role, the Workflow Data Base (WFDB) role, the Workflow Management System (WFMS) role and the User role together with several interactions. Already the use case diagram reveals the matrix structure in two dimensions. Agent roles form the multi-agent application structure while interactions form the behavior of the system. Arcs in the diagram correspond to the matrix interconnection points from Section 2.2. Use

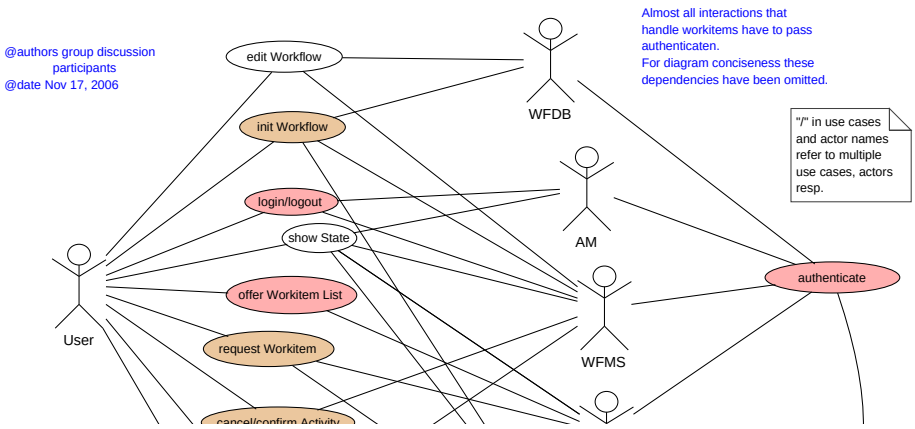

Figure 3. Fragment of a use case diagram showing the system's coarse design.

case diagrams are drawn directly in RENEW. The use case plugin provides the functionality by adding a palette of drawing tools to the editor.

The use case plugin (UC-Plugin) integrates a generator feature, which generates the complete folder structure of the application necessary for the implementation of a multi-agent application. This includes a standard source package folder structure, skeletons for all agent interactions, role diagram and ontology files as well as configuration files and build / start skripts. The generator utilizes the Velocity[3] template engine.

3.2 Multi-Agent Application Structure

The structure of the multi-agent application is refined using a R/D diagram (role/dependency diagram). This kind of diagram uses features from class diagrams and component diagrams. Class diagrams provide inheritance arcs to denote role hierarchies. Component diagrams provide explicit nodes for services as well as arcs with *uses* and *offers* semantics to denote dependencies between roles. Initial values for role-specific knowledge bases are included through refinement of nodes.

Figure 4 shows a fragment of the WFMS R/D diagram. The fragment depicts several roles marked `«AgentRole»`: CapaAgent, AuthenticationNeeder, AccountManager and WFEngine. Also some services marked `«Interface»` are depicted: SessionManagement, Authentication etc. As an example, the service Authentication is offered by the AccountManager and used by each agent that holds the role AuthenticationNeeder.

The agent role descriptions are automatically generated from the R/D diagram. Role descriptions are combined to form agent descriptions (initial knowledge bases). Roles can easily be assembled to form the multi-agent application

[3] The Apache Velocity Project `http://velocity.apache.org/`

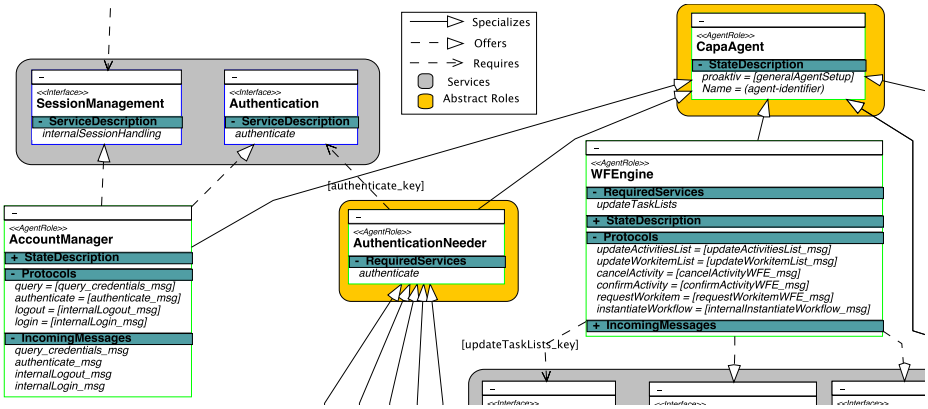

Figure 4. Fragment of a R/D diagram (agents, roles, services).

using the graphical user interface. The multi-agent application is started either from within the tool, by a startup script or by a Petri net.

3.3 Terminology

The terminology of a multi-agent system is used in a twofold way. First, it is used in form of an ontology definition by the agents to communicate with each other and for their internal representation of the environment. Second, it is used among the developers to communicate about the system and its design.

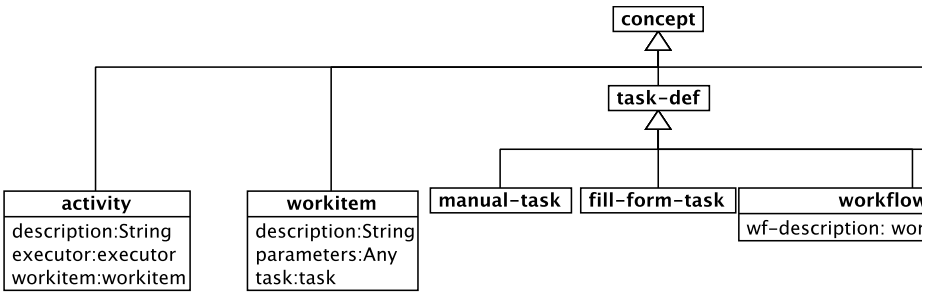

Figure 5. Fragment of the WFMS ontology.

To define the ontology of our multi-agent applications we have been using Protégé[4] for over two years now. Ontologies are defined in Protégé and then translated by a generator into Java classes. Protégé is a very powerful tool, but it features a completely different user interface design than Renew.

[4] Protégé `http://protege.stanford.edu/`.

The Renew feature structure plugin allows to explicitly model the ontology as a concept diagram as shown in Figure 5. These are class diagrams restricted to inheritance and association. The concept diagrams can easily be understood by all sub-teams to capture the context of the concepts in use.

Up to now, the translation of models from the feature structure concepts to Protégé ontologies is a manual task. The Protégé model can then be used to generate the *Java* ontology classes. However, we have also developed a prototypical implementation of an ontology classes generator (directly) from concept diagrams. We are also working on transformations from and to Protégé models.

3.4 Knowledge and Decisions

While the agent's interactive behavior is defined in the interaction protocols (see next section), the facts about its environment are located in the agent's knowledge base. The initial knowledge of the agent is defined in its initial knowledge base file, constructed by joining information from the role definitions, which have been defined in the R/D diagram (introduced in Section 3.2). This XML document that can also be customized apart from the R/D diagram is parsed to build the initial knowledge of the agent during its initialization. Alternatively, a text file in the style of properties files suffices for the same purpose.

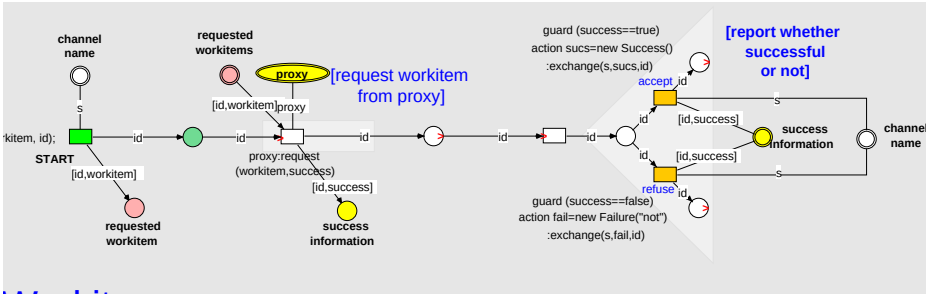

Figure 6. Fragment of a decision component net: RequestWorkitemHandling

Decision components (DC) are constructed as reference nets. There exists a generalized form of a DC providing GUI interface connection. Also net components [2] for the development of DCs are provided.

Figure 6 shows a fragment of the DC net handling the request of a user for a workitem in the workitem dispatcher agent. The net holds the proxy net which implements the interface to the workflow engine. A request starts at the left of the image and is handed over to the proxy, which holds a list of available work items for the given user. The result of the request is handed back to the DC net and passed (via the *exchange* channel) on to the requester, a protocol net, which in turn sends an appropriate message to the requesting agent.

3.5 Behavior

The interactive behavior of the system components is specified using agent interaction protocol diagrams (AIP, introduced in [10], integrated in PAOSE in [3]).

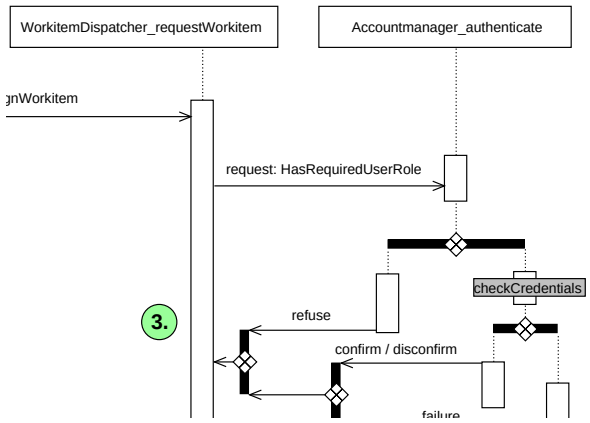

Figure 7. Fragment of an agent interaction protocol diagram.

Figure 7 depicts a fragment of an AIP involving the two roles AccountManager and WorkitemDispatcher in the authenticate interaction. Agent interaction protocol diagrams are integrated in our tool set through the RENEW Diagram plugin which is also capable of generating functional skeletons for protocol nets. As described in Section 2, protocol nets are reference nets that directly define the behavior of a MULAN agent. Protocol nets are composed of *net components* [2].

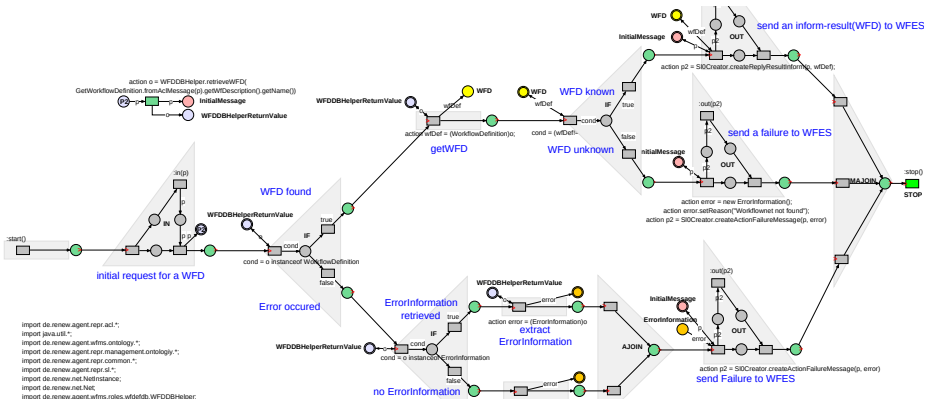

Figure 8. A protocol net constructed with net components.

Net components are also used for automatic generation of protocol net skeletons from agent interaction protocol diagrams. The protocol nets are then refined during the implementation phase by adding inscriptions to the nets. Figure 8 shows an example protocol net.[5] Several decisions are made after receiving a request message. Finally, the appropriate answer is sent back.

With the implementation of interactions as protocol nets, the internal processes as decision components and the knowledge bases through the description of the role diagram, the whole multi-agent application is defined.

Additionally, all diagrams presented here serve as documentation elements and are included in the API-documentation of the system (MulanDoc Plugin).

3.6 Summary

In the context of Mulan and Paose we can identify three basic dimensions in which the perspectives on the system can be categorized. *Structure* relates to roles and knowledge, which is functionally decomposed. *Behavior* relates to interactions and internal processes, which reflects the natural view via Petri nets onto systems with respect to behavior. *Terminology* is covered by ontologies and provides the glue between the different perspectives. Organizational embedding is covered by the matrix-like treatment, which provides the relationships between entities in the organizational context including the involved people. In addition, Table 1 shows a table of relations between task types, modeling techniques, applied tools and resulting artifact.

Task	**Model**	**Tool**	**Result**
Coarse Design	Use Case Diagram	UC Plugin	Plugin Structure
Ontology Design	Concept Diagram	FS-Nets/Protégé	Generated Classes
Role Design	R/D Diagram	KBE Plugin	Knowledge Bases
Internal Processes	Petri Net Diagram[6]	Renew[6]	Decision Components
Interaction Design	AIP Diagram	Diagram Plugin	Protocol Nets

Table 1. Overview over the contiguous techniques.

3.7 Experiences

The presented approach has been applied to several teaching projects consisting of twenty to forty students, tutors and lecturers. The approach has been further developed over the years, which resulted in better tool support and further elaboration of methods and techniques (many of which were presented earlier). After a phase of learning the concepts, methods and techniques, the students

[5] The net components are recognizable and show the structure of the protocol net.

[6] For the internal processes no abstract modeling technique has been presented. Several proposals exist, but have not resulted in tool support, yet. However, those processes can be modeled directly as reference nets in Renew or can be externalized.

were able to design and construct rather complex concurrent and distributed software systems. For example, an agent-based workflow management system (compare with the diagrams of this paper) was developed using this approach.

The results of 5 weeks of teaching and 9 weeks of implementation include about 10 agent roles, more than 20 interactions and almost 70 concepts in the ontology. The outcome is a running prototype of an distributed agent based workflow management system, where a user is represented by an agent and the basic features are provided through a user GUI: Authentication, workflow instantiation, offering of available tasks according to application roles and task rules, accepting, cancellation and conclusion of tasks during the progress of a workflow. Workflows themselves are specified with Petri nets using a special task transition which provides cancellation (compare [6]). Thus synchronization and conflict solving are provided by the inherent features of the Renew simulation engine. This example and our other previous projects show that Paose together with the guiding metaphor of a *multi-agent system of developers* [1] enable us to develop multi-agent applications with Mulan. The developed methods and the tool support have proven to be effective in supporting the development process.

4 Conclusion

In this paper we present the modeling techniques used within the Paose approach to build agent models. The tools that are used during the development process support all tasks of development with modeling power, code generation and deployment facilities. Still some of the tools have prototypical character. Specifically, we have presented techniques to model structure, behavior and terminology of concurrent software systems in a coherent way following the multi-agent paradigm. All techniques and tools own semantics built upon the unique, concurrency-oriented modeling and programming language of reference nets, either directly or by referring to the Mulan reference architecture.

The concurrency-awareness in development process and modeling techniques distinguishes our approach from most of the methodologies mentioned in the introduction since they usually do not address true concurrency explicitly (compare [13]). The advantage of tight integration of abstract modeling techniques with the conceptual framework given through the formal model of Mulan is responsible for the clearness and the effectivity of our approach.

For the future, we follow several directions to refine the approach. On the practical side, we look into further developments, improvements and integration of tools and techniques. On the conceptual side, we work on expanding the multi-agent-oriented approach to other aspects of the development process like project organization and agent-oriented tool support. Following these directions, we want to achieve symmetrical structures in all three aspects of software development: the system, the development process and the project organization (compare [1]).

References

1. Lawrence Cabac. Multi-agent system: A guiding metaphor for the organization of software development projects. In Petta Paolo, editor, *Proceedings of the Fifth German Conference on Multiagent System Technologies*, volume 4687 of *LNCS*, pages 1–12, Leipzig, Germany, 2007. Springer-Verlag.
2. Lawrence Cabac, Michael Duvigneau, and Heiko Rölke. Net components revisited. In Daniel Moldt, editor, *Fourth International Workshop on Modelling of Objects, Components, and Agents. MOCA 2006*, pages 87–102, 2006.
3. Lawrence Cabac, Daniel Moldt, and Heiko Rölke. A proposal for structuring Petri net-based agent interaction protocols. In Wil van der Aalst and E. Best, editors, *24th International Conference on Application and Theory of Petri Nets, Eindhoven, Netherlands*, volume 2679 of *LNCS*, pages 102–120. Springer-Verlag, 2003.
4. Scott DeLoach. Engineering organization-based multiagent systems. In *Software Engineering for Large-Scale Multi-Agent Systems (SELMAS)*, volume 3914 of *Lecture Notes in Computer Science*, pages 109–125. Springer Verlag, 2005.
5. M. Duvigneau, D. Moldt, and H. Rölke. Concurrent architecture for a multi-agent platform. In Fausto Giunchiglia, James Odell, and Gerhard Weiß, editors, *Proc. of AOSE 2002*, volume 2585 of *LNCS*, Berlin, 2003. Springer Verlag.
6. Thomas Jacob, Olaf Kummer, Daniel Moldt, and Ulrich Ultes-Nitsche. Implementation of workflow systems using reference nets – security and operability aspects. In Kurt Jensen, editor, *Fourth Workshop on Practical Use of Coloured Petri Nets*. University of Aarhus, Department of Computer Science, August 2002.
7. Michael Köhler, Daniel Moldt, and Heiko Rölke. Modelling the structure and behaviour of Petri net agents. In J.M. Colom and M. Koutny, editors, *Proceedings of the 22nd Conference on Application and Theory of Petri Nets 2001*, volume 2075 of *LNCS*, pages 224–241. Springer-Verlag, 2001.
8. Olaf Kummer. Introduction to Petri nets and reference nets. *Sozionik Aktuell*, 1:1–9, 2001. ISSN 1617-2477.
9. Olaf Kummer, Frank Wienberg, and Michael Duvigneau. Renew – The Reference Net Workshop. `http://www.renew.de`, March 2007. Release 2.1.
10. 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.
11. Lin Padgham and Michael Winikoff. Prometheus: A pragmatic methodology for engineering intelligent agents. In *Proceedings of the OOPSLA 2002 Workshop on Agent–Oriented Methodologies*, pages 97–108, 2002.
12. Heiko Rölke. *Modellierung von Agenten und Multiagentensystemen – Grundlagen und Anwendungen*, volume 2 of *Agent Technology – Theory and Applications*. Logos Verlag, Berlin, 2004.
13. Onn Shehory and Arnon Sturm. Evaluation of modeling techniques for agent-based systems. In *Agents*, pages 624–631, 2001.
14. Rüdiger Valk. Petri nets as token objects - an introduction to elementary object nets. In Jörg Desel and Manuel Silva, editors, *19th International Conference on Application and Theory of Petri nets, Lisbon, Portugal*, number 1420 in LNCS, pages 1–25, Berlin, Heidelberg, New York, 1998. Springer-Verlag.
15. Franco Zambonelli, Nicholas Jennings, and Michael Wooldridge. Developing multi-agent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3):317–370, 2003.