A Proposal for the Modeling of Organizational Structures and Agent Knowledge in MAS

Lawrence Cabac, David Mosteller, Matthias Wester-Ebbinghaus

University of Hamburg
Faculty of Mathematics, Informatics and Natural Sciences
Department of Informatics
{cabac,2mostell,wester}@informatik.uni-hamburg.de
http://www.informatik.uni-hamburg.de/TGI

Abstract. One of the most important tasks when developing multi-agent systems (MAS) is to determine the overall organizational structure of the system. In this paper we present a service-oriented perspective on the organizational structure of MAS and we present modeling techniques and tools for supporting this perspective. We pursue a model-driven approach and a tight integration between various models on the one hand and between the models and the generated code on the other hand. In particular, we combine ontology modeling and organization structure modeling in a way that we can easily generate the initial content of agent knowledge bases in the form of FIPA semantic language (SL) fragments (depending on what positions the agents occupy in the context of the organizational structure). In addition, this allows the agents to reason about and to communicate about their organizational embedding using the same ontology.

Keywords: Renew, Mulan, Paose, Petri nets, multi-agent systems, model-driven development, organizational structure

1 Introduction

The modeling of the fundamental organizational structure is one of the central tasks during the development of a multi-agent system (MAS) [9]. While agents are considered autonomous in their actions, they are also supposed to fulfill certain functions in relation to the purpose of the overall multi-agent application (MAA). A wide spectrum of approaches for organizing multi-agent systems exists [15] and some of them are quite sophisticated in drawing inspiration from organizing principles of social systems (including multiple organizational modeling dimensions like social structures, tasks, social interactions, norms etc., cf. [1,8]). We argue that at the core of most of these approaches lies the determination of an organizational structure in terms of agent functions and agent dependencies based on functional dependencies. This concerns the questions, *which agents* are required / allowed to do *what* (responsibilities / abilities) and *to whom* they can

refer for help in certain cases (support / delegation). Basically, this is a service-oriented perspective on agent relationships. Agents offer functional services to other agents and in turn require the services of other agents in order to fulfill some of their own functionality.

We apply this functional and service-oriented perspective for the design of the basic organizational structure of a MAS in our Paose approach (**P**etri net-based **A**gent- and **O**rganization-oriented **S**oftware **E**ngineering, `http://www.paose.net`). It provides a general basis for MAS organization that can be extended if necessary.[1] We have presented our Paose approach on previous occasions and we have particularly elaborated on the model-driven nature of Paose in [6]. Our multi-agent platform Mulan/Capa [16,22] tightly combines model and code as it is based on a fusion of high-level Petri nets and Java. This allows us to model / implement all processes as directly executable Petri nets. In addition, we use UML-style modeling techniques for development where we need a more declarative perspective than is offered by Petri nets.

In this paper, we specifically refer to the part of Paose that is concerned with modeling organizational structures in terms of agent roles and service dependencies between roles. This part relies on ontology modeling as we explicate the concepts used for organizational structures as an ontology. This has the additional benefit that we can easily translate an organizational structure model into multiple initial knowledge bases for multiple agents (depending on what positions the agents occupy in the organizational structure). The content of the knowledge bases is generated in FIPA semantic language (`http://www.fipa.org`), which provides the technical basis for agents to reason about and to communicate about their organizational embedding. Compared with our previous work presented in [6,7], we present a considerable rework including new tools. Our revision basically takes care of a better and tighter integration between the tools used as well as between the models and the generated code.

In Section 2 we provide an overview of role and service (dependency) modeling in the MAS field and motivate our own approach. In Section 3, we present our concrete models and the supporting tools. We place our contribution in the context of our development process Paose and introduce the agent framework Mulan/Capa. We also describe how the tools fit into the model-driven nature of our Paose approach. Section 4 gives an example of our tools in use, demonstrated in a concrete application scenario. We close with a short summary and some aspects of future research and development that builds upon the results presented in this paper in Section 5.

[1] For example, we have developed the Sonar model [17,18] for multi-agent teamwork support, where we use a more elaborate model of functional service dependencies between agents based on task delegation structures and a behavior-based notion of service refinement.

2 Organizational Structures of Multi-Agent Systems

In this section we elaborate on our conceptual approach to modeling organizational structures in terms of agent roles and service dependencies. We motivate the use of the two core concepts of *roles* and *services* in the context of related work.

2.1 Modeling Agent Roles and Service Dependencies

The interest in establishing organizational structures in a MAS has always been an important part of agent research. One can argue that it is an important part of software design in general (although the architecture metaphor is more established than the organization metaphor). However, in the case of MAS this topic becomes even more imperative. Artificial social agents are regarded as very sophisticated software components with complex knowledge and reasoning mechanisms that often only offer a limited visibility. Consequently, high-level system perspectives are necessary, in which one can abstract from agent-internal details and still comprehend the system on a more abstract level.

The concept of a role has been used extensively in this context and has been established as one of the core concepts of agent-oriented software design [19]. Rights and responsibilities are associated with roles independently from the specific agents that will occupy the roles. Consequently, this leads to a certain degree of predictability and controllability of global MAS behavior without knowing anything about the agents' internals. Examples of bringing the concept of roles to use (cf. [1]) is to enable as well as constrain agent behavior in terms of (1) which roles belong together to a common group context (allowing acquaintance and communication between group members), (2) defining which roles are expected to be associated with which goals, tasks and necessary capabilities and (3) which roles are supposed to take part in which conversations in which way.

Basically, all these efforts boil down to the abstract question what an agent occupying a specific role is supposed to do just because of it taking on that role. We are mainly interested in an explication of a functional perspective on roles and role relationships. Of special interest is the specification of functionality of roles occupants in the context of the wider multi-agent application and the dependencies that exist between different role occupants. Thus, we apply a service-oriented perspective on agent roles: Which roles are associated with the provision of which services and on which other services are they dependent? We are aiming at a rather minimalistic model of agent roles and their relationships in terms of service dependencies that can be enriched with more sophisticated concepts if needed (e.g. goal / task hierarchies, conversation guidelines).

2.2 Related Work

Not only in agent-oriented approaches to software development the modeling of component dependencies is one of the major challenges. One main problem (also applying to some of the approaches for role-based specifications mentioned

above) is that dependencies are often hidden underneath quite complex specifications. Ensel and Keller summarize Gopal [13] in the following way: "However, the main problem today lies in the fact that dependencies between services and applications are not made explicit, thus making root cause and impact analysis particularly difficult" [10, p. 148]. Therefore our motivation is to gain the ability to explicitly model dependencies for MAS and our choice is to model component dependencies (agent dependencies) in terms of roles and service dependencies. The actual dependencies between running agents then result from the roles they occupy.

In the context of the different approaches to software development there exist various ways of handling component dependencies. Some of them are restricted to managing service dependencies by utilizing declarative service descriptions, i.e. using XML [10, p. 148], [24]. From our point of view the more promising approach consists in making use of diagram-based methods.

The most obvious benefit lies in the incomparably better visualization of diagram-supported models over declarative service descriptions. This was identified as a central issue, taking up the above mentioned citation by Ensel and Keller again. On the one hand, the diagram is the means to make the dependencies explicit [3] instead of an implicit declaration located in the configuration files of the (distributed) components as it is for example the case in OSGI service descriptions [24]. An explicit representation of the dependencies is of special value during the design phase for a developer / administrator. On the other hand, the capabilities of model transformation are given in both possibilities to describe dependencies as model-based and as declarative descriptions. A similar approach was taken in [26] for Web Services and BDI-Agents. Service dependencies are specified in the model domain and tool support is realized as a Rational Software Modeler extension. "Dependencies between the various components are modeled at the PIM-level and two-way model transformations help us to ensure interoperability at the technical level and consistency at the PIM-level" [26, p. 114]. There are other efforts, which mainly address specification of dependencies between agents and Web Services (e.g. [14]) whereas our work is focused on agent relations.

Most software developing methodologies contain a technique for modeling some kind of dependencies between their components. The Tropos methodology distinguishes four kinds of dependencies between agents, from hard dependencies (resource) to soft ones (soft-goal). Silva and Castro [23] display how Tropos dependency relations can be expressed in UML for real time systems. Ferber et al. [11] show how the organizational structure of an agent-based system can be modeled using the AGR technique. One of the proposed diagrams, the organizational structure diagram, shows roles, interactions and the relations between roles and interactions. This diagram is comparable to the Roles/Dependencies diagram.

In Gaia Zambonelli et al. [25] focus strongly on the organizational modeling. One of the important models is the service model. Our Roles/Dependencies diagram can be regarded as an implementation of the Gaia service model. However,

Gaia does not recognize hierarchical roles. Padgham and Winikoff [21] explicitly model acquaintances in Prometheus. But from these models they do not derive any agent (role) dependencies. Roles are not modeled in Prometheus, instead the focus lies on agents. The system model in Prometheus gives a good overview of the system comparable with the overview of the Roles/Dependencies diagram. It is much more detailed but does not explicitly show any dependencies except the interaction protocols or messages that connect agents. The structure of the system model reflects the one of the acquaintances model.

In the following, we introduce our approach for a minimalistic (but extensible) comprehension of organizational structures of MAS in terms of role descriptions and role dependencies based on service relationships. Our previous work covered details on the conceptual side of modeling the basic organizational structure of MAS, introducing modeling techniques [7,5,4] and tools [6]. In our current work we improve the methods and tools by putting an even stronger focus on the model-driven nature of our approach. We pursue a tighter integration of different tools and to minimize the gap between the models and the code generated from the models. One specific benefit of our approach lies in the fact that the meta-model for organizational structures is expressed in the agents' language – i.e. as an agent ontology. Thus, the agents are able to communicate and reason about their own organizational structures.

3 Role/Dependency Tool Support for Model-Driven Development in PAOSE

In the following we point out how the integration of the conceptional basis we introduced in the previous section is established in our Mulan framework and the Paose development process. We introduce two types of diagrams, namely for ontology modeling and for roles/dependencies modeling. They support the discussed features in a clear and intuitive way, making use of well-known constructs from UML. We also present our tool solution to support our model-driven development approach. All our Paose tools are realized as plugins for the high-level Petri net tool Renew (http://www.renew.de).[2] They extend Renew with modeling techniques that are not based on Petri nets.

3.1 The Paose Development Process

This section puts the subsequent work into the context of the Paose approach, which aims at the development of Mulan applications. The approach focuses on aspects of distribution, concurrency and model-driven development. The framework Mulan offers the basic artifacts and structuring for the application. Its four layered architecture features as basic artifacts the communication infrastructure, the agent platforms, the agents and the agent internals (protocols, decision components and knowledge bases). With the exception of the communication

[2] Renew also provides the virtual machine that executes Mulan applications.

infrastructure, all artifacts are implemented as Java Reference nets. CAPA extends the MULAN architecture with FIPA-compliant communication features, providing inter-platform (IP-based) agent communication. Also the MULAN applications (MAA) are – similar to the MULAN/CAPA framework – implemented in Java Reference nets and Java. They are executed, together with the MULAN/CAPA framework, in the RENEW virtual machine. While the implementation in Java Reference nets introduces concurrency for MULAN applications, the CAPA extension enables the agents to run in distributed environments.

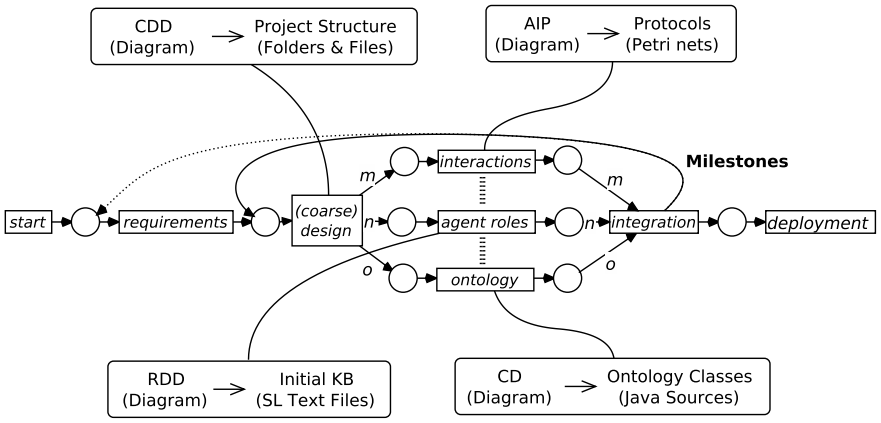

Figure 1. The PAOSE development process and techniques. Modified from [3, p. 133]

The organization of MAS can be explicitly modeled using model-driven techniques [6], as described in the following sections. However, in addition to the organizational structure of the MAA, we apply the agent-oriented view onto the organizational structure of the development team through the metaphor of the *multi-agent system of developers* [2]. The metaphor provides the perspective that human participants of the development team form an organization, similar to agents in an MAA, and their collaborative efforts constitute the development process, similar to the MAA process. During development the responsibilities for the diverse tasks are distributed among the developers, which allows for concurrent and distributed collaboration as well as explicit identification of dependencies between the team participants. In the previous sections we motivated a service-oriented composition of MAS based on roles and service dependencies. Here we argue that developers dependencies result from the organizational structure and the application's dependencies. These dependencies are also reflected in the PAOSE development process, which consists in iterative repetitions of specific fundamental steps of design and implementation, as shown in Figure 1. The figure depicts a simplified Petri-net process of the PAOSE design cycle.

A project starts with the requirements analysis resulting in a coarse design of the overall structure of the MAA. The coarse design identifies essential roles and interactions of the organization. It is used to generate the initial structure (development artifacts) of a project. The main step of an iteration consists of three tasks of modeling and implementation. These are the modeling of interactions, agent roles and ontologies, as well as generating sources from the models and refining the implementation. The integration of the resulting artifacts completes an iteration. In the diagram annotations refer to modeling techniques, which are utilized to carry out a corresponding task and the artifacts, which are generated from the design models. Taking up the aforementioned view on the organization of a development team, the completion of an iteration requires the synchronized, collaborative effort of the participants.

In the context of this work we introduce a technique and a tool for the modeling of agent roles. To this end, we utilize the ontology model used in Paose as a meta-model. In this sense the following section describes how our integration approach essentially applies ontology concepts for the design of a new modeling technique and a corresponding tool – in this case the modeling of organizational structures of MAA.

3.2 Integration Approach

Within our development process we apply a few presumptions. The approach taken relies on two fundamental ideas. These are the support for the development process by making use of methods from model-driven development (MDD) and the tightening of the integration of models (diagrams), generated code and serialized representations. This leads to a threefold integrative approach that is illustrated in Figure 2 and that we discuss in the following.

Integration encompasses three parts: (1) an ontology including multiple concepts, (2) the code generated from the ontology and (3) the serialized representations of concept instances in FIPA Semantic Language (SL) format [12].

Ontologies are modeled using a light-weight technique called *Concept Diagram*[3]. The concepts defined in the ontology are transformed into Java classes, one class for each concept. Instances of these classes (ontology objects) can be extracted into SL-formatted text. Through an SL parser the serialized SL text representations can be used to instantiate Java ontology objects in the reverse direction. Consequently, we utilize three tools for these three tasks: (1) the *ConceptDiagramModeler*, (2) the *OntologyGenerator* and (3) the *SL parser*.

This basic integration approach described so far has several benefits. The development process takes on a model-driven approach, which allows for the specification of ontological concepts in a graphical notation. Concept Diagrams are very similar to the widely-used UML Class Diagrams. They are quite intuitively comprehensible and easy to manage. Manipulation of attributes can be carried out directly in the diagram. Additionally, by making use of code

[3] An example of a Concept Diagram will be discussed in the context of defining a knowledge base format in the following section (3.3).

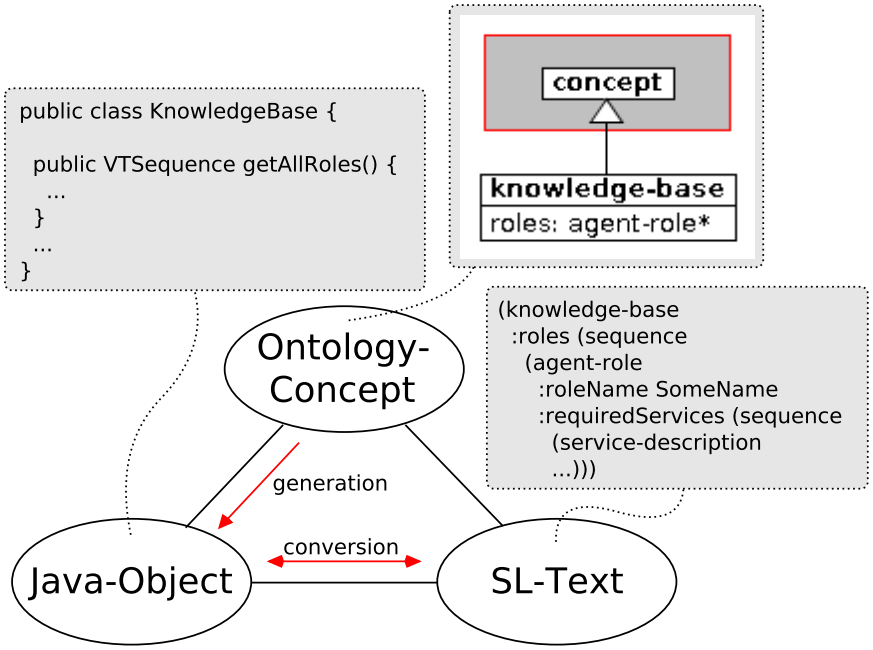

Figure 2. The three-part basic model of agent knowledge

generation and the bi-directional conversion between Java objects and SL text representations for concept instances, the integration of the different representations is very tight, i.e. transformation is transparent to a user. By using SL for the text representations of ontology objects we employ an agent-comprehensible format, as MULAN agents use SL text representations for message encoding. In addition, our experience has indicated that SL text is also better human-readable in comparison to an equivalent XML representation and shows a lower overhead.

In the following, we show how we apply this method in the case of modeling service dependencies and agent knowledge.

3.3 Concept Diagrams for Role and Knowledge Base Concepts

The previous section provides a general overview of our model-driven approach based on the integration of multiple representations. Now we describe how the three basic parts (ontology, Java code, SL text representation) are applied in the case of modeling as well as establishing organizational structures in multi-agent applications (MAA). The model-driven approach starts with the specification of an ontology encompassing organizational concepts (roles, services, protocols) and concepts necessary for the generation of agent knowledge from organizational structure models (knowledge base as an aggregation of role definitions resulting

from a mapping between agents and roles, in which multiple roles can be assigned to each agent). The ontology we use is shown in Figure 3 as a Concept Diagram. It is created with the ConceptDiagramModeler.

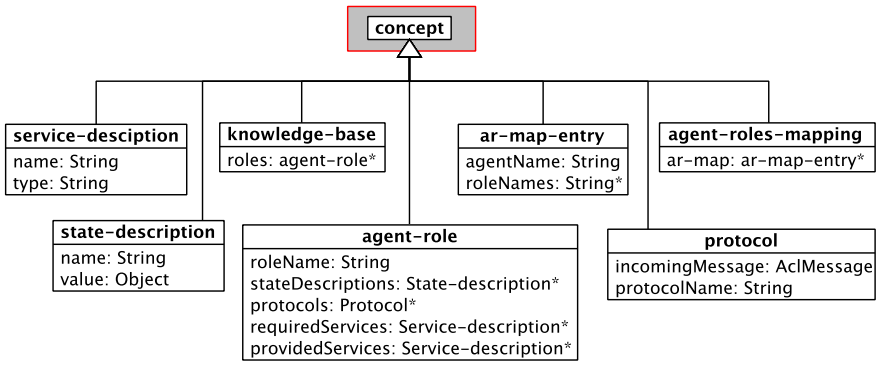

Figure 3. A Concept Diagram for agent knowledge base concepts

The Concept Diagram serves in a twofold way. First, it defines all the content types of the agent communication in an application. Second, it serves as a metamodel for the tools that handle the modeled contents.

From here on, we rely on further tool support for code generation and conversion between the different representations of concepts and concept instances. We use the OntologyGenerator (based on Velocity, `http://velocity.apache.org`) and the SL parser provided by the MULAN framework. Ontology modeling in terms of Concept Diagrams and code generation from these models is already a part of the PAOSE development process (cf. [3, p. 173]). Thus, the approach described here for handling organizational structures and agent knowledge fits neatly into the context of our wider work.

Basically, Figure 3 can be regarded as capturing the ontology for knowledge bases of MULAN agents (the schema of a knowledge base). It illustrates the modeling technique of Concept Diagrams in terms of inheritance and the use of concepts for the definition of other concepts (this could also be modeled via associations between different concepts).

Besides capturing the ontology for knowledge bases, the ontology is also used by the AgentRoleModeler tool presented in the next subsection. Java ontology classes that are generated by the OntologyGenerator tool are specializations of generic ValueTuple (VT) and KeyValueTuple (KVT) classes. VT and KVT structures are the root interfaces of an implementation of the FIPA SL.

As mentioned above, by using an SL parser, ontology objects can be instantiated from their SL string representations. This is a feature that lies at the heart of creating *agent instances* from knowledge base patterns (because knowledge-base is a concept in the diagram from Figure 3 and thus each knowledge base

as a whole has an SL representation). Ontology classes provide *getters*, *setters* and convenience methods for operations on the data structures. The Ontology-Generator tool is integrated into the build environment of the MULAN framework and the SL parser can be used on the fly in a running MULAN MAA. All in all, this supports our ambition of realizing a tight integration of different models, tools and code.

Using the knowledge base ontology shown in Figure 3 the following section explains how we model roles and dependencies in multi-agent applications.

3.4 Roles/Dependencies Diagrams

Roles and role dependencies are modeled with the AgentRoleModeler tool. The corresponding Roles/Dependencies Diagrams combine notations from Class Diagrams and Communication Diagrams. The tool is embedded in our model-driven development approach. The content of Roles/Dependencies Diagrams (Figure 5) is based on the concepts that were already defined in the ontology from Figure 3. Because of this, the AgentRoleModeler tool allows for the generation of knowledge base descriptions in FIPA SL from Roles/Dependencies Diagrams using the knowledge base ontology from Figure 3 as a meta-model. Thus the concepts from the Concept Diagram reappear as stereotypes in the Roles/Dependencies Diagram. The knowledge base descriptions resulting from a Roles/Dependencies Diagram are used as patterns for the initialization of agent instances in the MULAN multi-agent framework.

RENEW-Editor-Palette

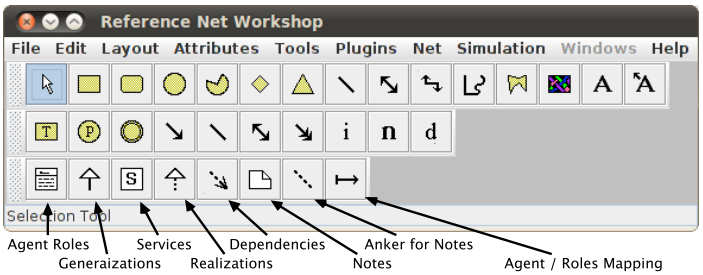

Figure 4. The RENEW-Editor-Palette

The AgentRoleModeler is a drawing plugin for RENEW and adds a custom palette for drawing elements of Roles/Dependencies Diagrams as shown in Figure 4 (the AgentRoleModeler palette is shown at the bottom, under RENEW's standard palettes). The graphical representation of Roles/Dependencies Diagram elements is displayed in Figure 5. The nodes of Roles/Dependencies Diagrams (roles and services) contain the text in FIPA SL format, specifying the corresponding attributes of the element. For a compact representation all drawing elements can be collapsed to a smaller view. This provides a very compact

and high-level view of an organizational structure in terms of roles and role dependencies based on service dependencies. Expanding the drawing elements allows manipulation of their attributes.

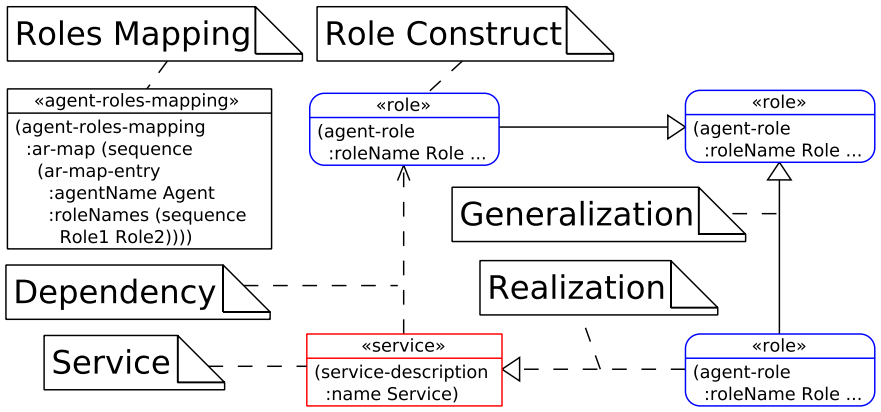

Figure 5. Constructs of a Roles/Dependencies Diagram.

Following Figure 3, the knowledge base of a MULAN agent contains an arbitrary number of agent role descriptions, depending on which roles the agent occupies. The attributes of agent roles (besides having a role name) are basically of three different types: (1) service dependencies, (2) protocol triggers and (3) state descriptions. Such (initial) knowledge base content can be generated from Roles/Dependencies Diagrams. Required and provided services of a role (i.e. hard dependencies) are shown explicitly as independent service nodes and offer / use associations connected to role nodes[4]. Protocol triggers are key-value tuples that define, which conversation protocol (value) an agent should initiate in reaction to incoming messages of a certain message pattern (key). They are not represented in a Roles/Dependencies Diagram as explicit nodes but are inserted directly into the corresponding role description. Further, state descriptions for a role may contain any kind of key-value tuples that shall serve as initial knowledge for role occupants. In addition to this flat specification of role dependencies, it is also possible to define inheritance relationships between roles. This introduces hierarchical relationships.

A Roles/Dependencies Diagram contains exactly one node that defines an *agent-role mapping*. Basically, this node serves to define *agent types* in terms of what roles a specific agent type should encompass. For each such agent type, a pattern in FIPA SL can be generated that serves as the basis for the initial knowledge of instantiated agents of that type.

[4] Refer to [7] for a discussion of our view on hard and soft dependencies.

4 Application

Aforementioned, we motivate the modeling of organizational structures in MAS. Our approach to modeling organizations grounds on a ontological content definition, namely the three-part basic model of agent knowledge. We introduced the three-part basic model in Section 3.2. We showed how the concrete realization of a tool for modeling organizations utilizes the ontological content definition. In this spirit the previous section introduced the notation of the Roles/Dependencies diagram and the AgentRoleModeler tool. We will now use the technique of a Roles/Dependencies diagram to model a sample application. The organizational structure is brought into the MAS by the means of generic knowledge base patterns. In the following example we demonstrate our method of extracting initial agent knowledge and structural information from the graphical model and show how they are brought into the running system. We illustrate the modeling of organizational structures and agent knowledge in sample applications.

The AgentRoleModeler tool was developed in the context of a bachelors thesis [20] at the University of Hamburg. After its completion it was used in several student projects for agent-oriented software development. In one of these projects about 20 team members worked on implementing applications for an agent-based collaboration platform. The following example from Figure 6 is taken from this project. It displays the scenario of a chat application, in which agents occur in the roles of chat senders and receivers.

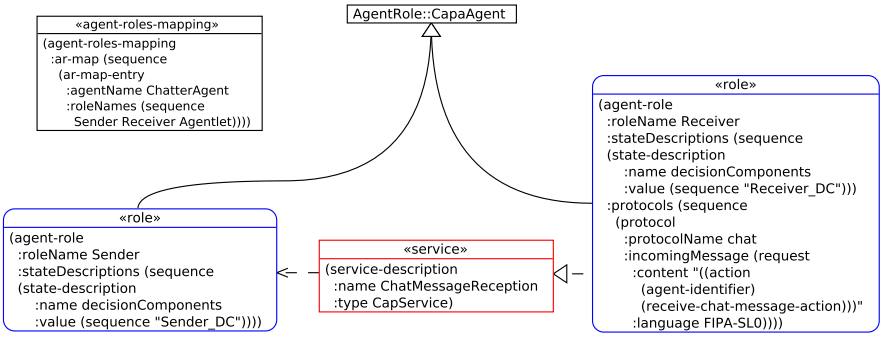

Figure 6. ARM of WebChat.

Figure 6 shows an instance of a Roles/Dependencies diagram. The model consists in the concepts that were already used in the previous section, introduced in Section 3.2 and illustrated in Figure 5. The Sender/Receiver-Scenario is an example we regularly use to demonstrate the Mulan-Framework in student projects. It consists in two roles, a Sender and a Receiver. Both roles inherit attributes from the generic CapaAgent role, thus they are specializations of this role. We will go into more detail about this later. The Sender is in possession of

a decision component SenderDC, which enables him to sporadically participate in conversation. He is dependent on a service (ChatMessageReception) allowing him to find and address chat partners. The Receiver is a role, which provides such a service, as can be seen by the realization relation between the Receiver role and the ChatMessageReception service. Upon receiving a chat message, the Receiver role reacts by initiating a chat protocol. The role specification formalizes reactive behavior as a protocol trigger, which can be seen on the lower right part of the above figure. A protocol trigger maps a type of message, identified by a message pattern, to a protocol. Every time a message of the defined type is received, the protocol will be triggered. The chat protocol passes chat messages to a decision component of the Receiver (ReceiverDC) allowing him to process the message. He can carry out internal reasoning about the conversation and decide on his further actions, such as creating a response or initiating a new conversation. The diagram constructs described up to this point make up the Roles/Dependencies model. There is a part we have not yet discussed. The agent-roles mapping construct shown on the upper left formalizes an instance specification. It determines, which agents occupy a previously defined set of roles. The agent-roles mapping in this case maps both roles to one type of agent, a ChatterAgent. The reason is that a ChatterAgent should naturally have the ability to do both, send and receive chat messages.

The example displays our notion of functional rights and responsibilities in terms of services dependencies. This specification of roles and services in form of the Roles/Dependencies diagram can be used to generate initial knowledge base contents for the agent instances dedicated to fulfill the corresponding roles. The following example focuses on the succeeding step of extracting the information required to initialize agent instances from the model.

Figure 7 shows a fragment of the Roles/Dependencies diagram that basically refers to one of the roles from the example above. The blue-bordered role figure (round corners) displays the attributes for the role name, protocol triggers and state descriptions. On the right hand side one can see a snippet of the FIPA SL code generated from the Roles/Dependencies diagram. Here, the service provided by the Receiver role (ChatMessageReception) is also included directly in the FIPA SL text. It can also be seen that the FIPA SL fragment contains more than one state descriptions. The additional state descriptions are inherited from the CapaAgent super role.

The roles and their mutual service dependencies are compiled into knowledge base patterns. The knowledge base patterns are in FIPA SL text, so they can directly be used to initialize agent instances, as they are specified in the language (ontology) of MULAN-agents. The example shows how this information is extracted from the model. It also shows how the model can express hierarchies of roles in terms of role specializations, enabling inheritance of attributes. Besides using the specialization relation between role constructs inside one single diagram, the mechanism implemented in the AgentRoleModeler tool also allows using inheritance across diagrams. This is also shown in the above figure. The CapaAgent role is accessed from the Roles/Dependencies diagram named Agent-

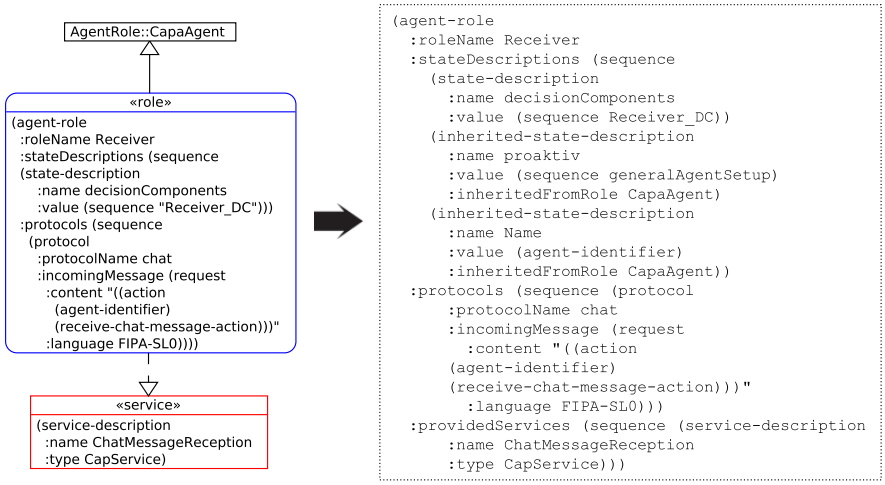

Figure 7. Role attributes and FIPA SL code generation.

Role. This is denoted by the displayed notation containing double colons. With the support for expressing specializations with the AgentRoleModeler tool it is possible to build graphical models containing hierarchies and compile them into knowledge base patterns, which allows us to project the overall organizational structure onto the MAA.

The approach for modeling basic organizational structures of MAS in terms of roles and role relationships fits neatly into the general model-driven nature of our Paose approach. In particular, in this case it helps to generate initial agent knowledge.

5 Conclusion

In agent-oriented software engineering and especially in the context of developing Mulan applications two essential design aspects are the modeling of the organizational structures and the initial knowledge of the agents. For the purpose of modeling these fundamental features it requires a conceptional basis as well as corresponding techniques, methods and tools.

5.1 Summary

In the context of the Paose approach we utilize the technique of Concept Diagrams and the OntologyGenerator tool to specify ontology concepts in order to design multi-agent applications. In the context of this paper this technique and tool is introduced together with the method of modeling agent roles and service dependencies. Furthermore we present a technique and a supporting tool

– also implemented as plugin for our IDE Renew – for a light-weight modeling of service dependencies and agent roles.

In Section 2 we elaborate on organizational concepts in MAS research and motivate our approach to modeling organizational structures. The main part of our contribution is preceded by an introduction to the Paose development process and the Mulan framework (Section 3.1), which constitute the context of our work. Our approach to modeling roles and dependencies is introduced in three steps. First, we introduce the three-part basic model of our approach (Section 3.2). Second, we illustrate the modeling of ontology concepts utilizing Concept Diagrams and the OntologyGenerator (Section 3.3). Third, we present the modeling of agent roles and dependencies with Roles/Dependencies Diagrams and the AgentRoleModeler (Section 3.4). The presented technique is an occurrence of the Roles/Dependencies Diagram – a Class Diagram that includes notations from Communication Diagrams for the modeling of agent role dependencies – that makes use of the Semantic Language (SL) as a description language for the agents' initial knowledge base contents. Finally, the techniques and tools presented in the course of this contribution are demonstrated in a application scenario in Section 4.

5.2 Future Work

In the context of our current research we elaborate on generalizing the approach that was presented in this paper to a further step. The idea is not only to apply the model-driven approach for code generation, conversion and transformation of models, but to generate special purpose tools from ontology diagrams as well. A step in this direction is generalizing the AgentRoleModeler tool to a generic SLEditor tool. The UI of a current prototype is shown in Figure 8. It displays the previously introduced role description of a Receiver from the Sender/Receiver application in a nested graphical figure. The outer frame is that of the agent-role. The highlighted constructs (in gray) indicate *ValueTuples*. This representation allows for displaying any nested structure of *KeyValueTuples* and *ValueTuples*. It can be seen as an alternative view to the plain text representation in Semantic Language. Further efforts are being made to utilize an ontology – modeled with the technique of a Concept Diagram – as a meta-model to generate specialized structures and at the same time generate the modeling tools by using the generic SLEditor. We are occasionally confronted with criticism against grounding our work on an *outdated* infrastructure, because we still rely on the FIPA Semantic Language for the specification of MAA. We address this subject with our development plan on extending the SLEditor. With the generic SLEditor tool we can support the modeling of MAA using other languages for content specification, such as XML. This can be achieved by extending the ontology to support XML to express knowledge contents.

Taking up the Figure 2 from Section 3.2 the three-part basic model of agent knowledge is extended to display an XML ontology of an agent role. The Figure reveals what is required to enable this feature: an XML schema definition specifying the format of our ontologies and the methods for conversion of representa-

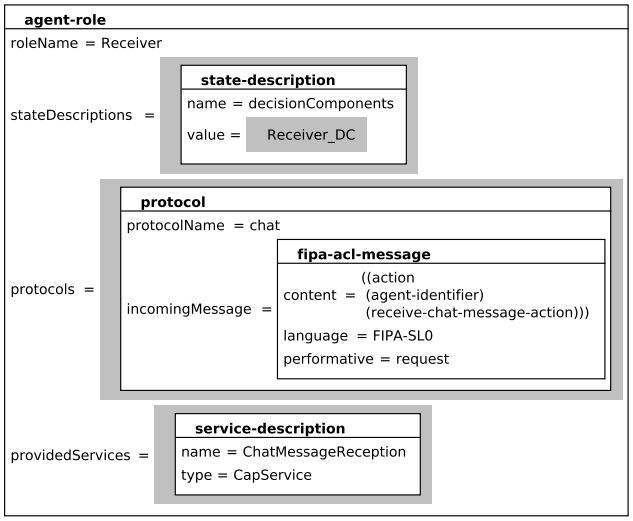

Figure 8. Alternative representation of SL content.

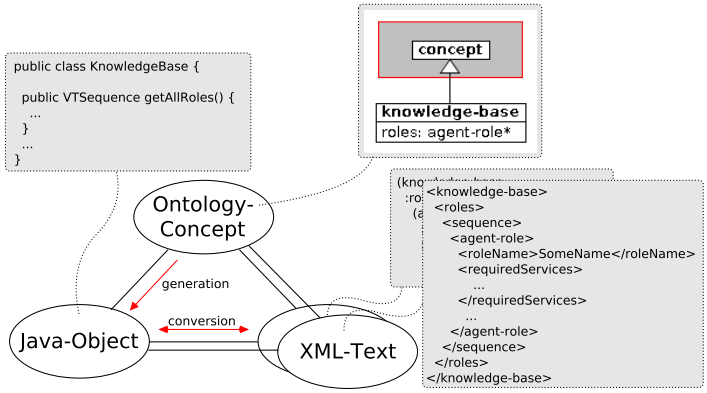

Figure 9. The three-part basic model of agent knowledge, extended to support XML for content specification.

tions between XML-Text and Java-Object. They are at this time not integrated in Mulan, but there exist standard tools that support XML conversion.

References

1. Olivier Boissier, Jomi Hübner, and Jaime Simão Sichman. Organization oriented programming: From closed to open organizations. In G. O'Hare, A. Ricci, M. O'Grady, and O. Dikenelli, editors, *Engineering Societies in the Agents World*

VII, volume 4457 of *Lecture Notes in Computer Science*, pages 86–105. Springer, 2007.
2. Lawrence Cabac. Multi-agent system: A guiding metaphor for the organization of software development projects. In Paolo Petta, editor, *Proceedings of the Fifth German Conference on Multiagent System Technologies*, volume 4687 of *Lecture Notes in Computer Science*, pages 1–12, Leipzig, Germany, 2007. Springer-Verlag.
3. Lawrence Cabac. *Modeling Petri Net-Based Multi-Agent Applications*, volume 5 of *Agent Technology – Theory and Applications*. Logos Verlag, Berlin, 2010.
4. Lawrence Cabac, Ragna Dirkner, and Daniel Moldt. Modeling with service dependency diagrams. In Daniel Moldt, Ulrich Ultes-Nitsche, and Juan Carlos Augusto, editors, *Proceedings of the 6th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems, MSVVEIS-2008, In conjunction with ICEIS 2008, Barcelona, Spain, June 2008*, pages 109–118, Portugal, 2008. INSTICC PRESS.
5. Lawrence Cabac, Ragna Dirkner, and Heiko Rölke. Modelling service dependencies for the analysis and design of multi-agent applications. In Daniel Moldt, editor, *Proceedings of the Fourth International Workshop on Modelling of Objects, Components, and Agents. MOCA'06*, number FBI-HH-B-272/06 in Report of the Department of Informatics, pages 291–298, Vogt-Kölln Str. 30, D-22527 Hamburg, Germany, June 2006. University of Hamburg, Department of Informatics.
6. Lawrence Cabac, Till Dörges, Michael Duvigneau, Daniel Moldt, Christine Reese, and Matthias Wester-Ebbinghaus. Agent models for concurrent software systems. In Ralph Bergmann and Gabriela Lindemann, editors, *Proceedings of the Sixth German Conference on Multiagent System Technologies, MATES'08*, volume 5244 of *Lecture Notes in Artificial Intelligence*, pages 37–48, Berlin Heidelberg New York, 2008. Springer-Verlag.
7. Lawrence Cabac and Daniel Moldt. Support for modeling roles and dependencies in multi-agent systems. In Michael Köhler-Bußmeier, Daniel Moldt, and Olivier Boissier, editors, *Organizational Modelling, International Workshop, OrgMod'09. Proceedings*, Technical Reports Université Paris 13, pages 15–33, 99, avenue Jean-Baptiste Clément, 93 430 Villetaneuse, June 2009. Université Paris 13. Preeproceedings available online at `http://www.informatik.uni-hamburg.de/TGI/events/orgmod09/#proceedings`.
8. Luciano Coutinho, Jaime Sichmann, and Olivier Boissier. Modelling dimensions for agent organizations. In Virginia Dignum, editor, *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, pages 18–50. Information Science Reference, 2009.
9. Virginia Dignum. The role of organization in agent systems. In Virginia Dignum, editor, *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, pages 1–16. Information Science Reference, 2009.
10. Christian Ensel and Alexander Keller. An approach for managing service dependencies with xml and the resource description framework. *Journal of Network and Systems Management*, 10:147–170, 2002.
11. Jacques Ferber, Olivier Gutknecht, and Fabien Michel. From agents to organizations: An organizational view of multi-agent systems. In Paolo Giorgini, Jörg P. Müller, and James Odell, editors, *Agent-Oriented Software Engineering IV, 4th International Workshop, AOSE 2003, Melbourne, Australia, Ju ly 15, 2003, Revised Papers*, volume 2935 of *Lecture Notes in Computer Science*, pages 214–230. Springer-Verlag, 2003.
12. Foundation for Intelligent Physical Agents (FIPA). FIPA SL Content Language Specification. http://www.fipa.org/specs/fipa00008/index.html, 2002.

13. R. Gopal. Layered model for supporting fault isolation and recovery. In *Network Operations and Management Symposium, 2000. NOMS 2000. 2000 IEEE/IFIP*, pages 729–742. IEEE, 2000.
14. C. Hahn, S. Jacobi, and D. Raber. Enhancing the interoperability between multiagent systems and service-oriented architectures through a model-driven approach. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*, volume 2, pages 415–422. IEEE, 2010.
15. Bryan Horling and Victor Lesser. A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review*, 19(4):281–316, 2005.
16. 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 *Lecture Notes in Computer Science*, pages 224–241. Springer-Verlag, 2001.
17. Michael Köhler-Bußmeier, Daniel Moldt, and Matthias Wester-Ebbinghaus. A formal model for organisational structures behind process-aware information systems. volume 5460 of *Lecture Notes in Computer Science*, pages 98–115. Springer-Verlag, 2009.
18. Michael Köhler-Bußmeier, Matthias Wester-Ebbinghaus, and Daniel Moldt. Generating executable multi-agent system prototypes from sonar specifications. In Nicoletta Fornara and George Vouros, editors, *11th Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems, COIN@Mallow 2010*, pages 82–97, 2010.
19. Xinjun Mao and Eric Yu. Organizational and social concepts in agent oriented software engineering. In James Odell, Paolo Giorgini, and Jörg Müller, editors, *Agent-Oriented Software Engineering V*, volume 3382 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin / Heidelberg, 2005.
20. David Mosteller. Entwicklung eines Werkzeugs zur Modellierung der initialen Wissensbasen und Rollen-Abhängigkeiten in Multiagentenanwendungen im Kontext von PAOSE / MULAN. Bachelor's thesis, University of Hamburg, Department of Informatics, December 2010.
21. Lin Padgham and Michael Winikoff. *Developing Intelligent Agent Systems : A Practical Guide*. Wiley Series in Agent Technology. Chichester [u.a.] : Wiley, 2004. isbn:0-470-86120-7, Pages 225.
22. Heiko Rölke. *Modellierung von Agenten und Multiagentensystemen – Grundlagen und Anwendungen*, volume 2 of *Agent Technology – Theory and Applications*. Logos Verlag, Berlin, 2004.
23. Carla T. L. L. Silva and Jaelson Castro. Modeling organizational architectural styles in UML: The tropos case. In Oscar Pastor and Juan Sánchez Díaz, editors, *Anais do WER02 - Workshop em Engenharia de Requisitos*, pages 162–176, 11 2002.
24. Andre L.C. Tavares and Marco Tulio Valente. A gentle introduction to OSGi. *SIGSOFT Softw. Eng. Notes*, 33:8:1–8:5, August 2008.
25. Franco Zambonelli, Nicholas Jennings, and Michael Wooldridge. Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3):317–370, 2003.
26. Ingo Zinnikus, Gorka Benguria, Brian Elvesæter, Klaus Fischer, and Julien Vayssière. A model driven approach to agent-based service-oriented architectures. In Klaus Fischer, Ingo Timm, Elisabeth André, and Ning Zhong, editors, *Multiagent System Technologies*, volume 4196 of *Lecture Notes in Computer Science*, pages 110–122. Springer Berlin / Heidelberg, 2006.