Modeling the Interaction Between Semantic Agents and Semantic Web Services Using MDA Approach

Geylani Kardas[1], Arda Goknil[2], Oguz Dikenelli[3], and N. Yasemin Topaloglu[3]

[1] Ege University, International Computer Institute, 35100 Bornova, Izmir, Turkey
geylani.kardas@ege.edu.tr
[2] Software Engineering Group, University of Twente, 7500 AE, Enschede, The Netherlands
a.goknil@ewi.utwente.nl
[3] Ege University, Department of Computer Engineering, 35100 Bornova, Izmir, Turkey
{oguz.dikenelli,yasemin.topaloglu}@ege.edu.tr

Abstract. In this paper, we present our metamodeling approach for integrating semantic web services and semantic web enabled agents under *Model Driven Architecture (MDA)* view which defines a conceptual framework to realize model driven development. We believe that agents must have well designed environment specific capabilities to fully utilize the power of semantic web environment. Hence, we first define a conceptual architecture for semantic web enabled agents and then discuss how this conceptual architecture can form the basis of a metamodel that can be used in the development of semantic web enabled agents with a model driven approach. We then zoom into the specific part of the metamodel that defines the interactions between semantic web enabled agents and semantic web services since it is not possible to cover all the aspects of the metamodel at one time. So we extend the metamodel of the conceptual architecture from the point of entity aspect for the interaction between semantic agents and semantic web services. Finally, we discuss the mappings between the entities of this extended metamodel and the implemented entities of SEAGENT framework.

1 Introduction

Recently, model driven approaches have been recognized and become one of the major research topics in agent oriented software engineering community [2] [17] [27]. Model driven development is considered as the most promising generational shift in programming technology [28] and even has been characterized as a paradigm shift [6] by several researchers. Model driven development aims to change the focus of software development from code to models. This would increase the level of abstraction in development. Therefore software products would be less affected from the changes in the technological advancements and also the productivity of software developers would be improved [1]. To work in a higher abstraction level is of critical importance for the development of Multi-agent Systems (MAS) since it is almost impossible to observe code level details of MAS due to their internal complexity, distributedness and openness.

G. O'Hare et al. (Eds.): ESAW 2006, LNAI 4457, pp. 209–228, 2007.

The key activity in model driven development is model transformation [29] and model transformation requires syntactical and semantical definitions of models which are provided by metamodels. Various metamodels have been proposed for specific MAS methodologies like Gaia, Adelfe, PASSI [5] and SODA [21]. These metamodels have been generally used for presenting concepts and only recently they are being considered as a foundation for MAS development tools [26].

Collaborating with Object Management Group's (OMG) Agent SIG, the FIPA Modeling Technical Committee proposes a metamodel called Agent Class Superstructure Metamodel (ACSM) [24] which is based on – and extends – UML 2.0 superstructure [25]. The metamodel presents a formal proposal for agent organizations considering the agent, group and role concepts and their relations. In fact, representing the MAS structure with these main meta-entities is not new and formerly proposed in AALAADIN MAS metamodel [12] but not as formal as FIPA Modeling TC's work.

On the other hand, MetaDIMA [15] is a metamodeling project which aims at bridging the gap between existing agent architectures with their development tools and agent-based methodologies, inspired by the Model Driven Architecture. It deals with metamodeling and transformations for agents. However, the project is currently in its preliminary phase.

In [26], Pavon et al reformulates their agent-oriented methodology called INGENIAS in terms of the Model Driven Development paradigm. This reformulation increases the relevance of the model creation, definition and transformation in the context of multi-agent systems.

However, we believe that a significant deficiency exists in above mentioned agent metamodeling and model-driven MAS development studies when we consider modeling of agent systems working on Semantic Web [4] environment. Near future's agent systems will doubtlessly work in this environment and agents in these systems will have capabilities to interact with other semantic entities such as semantic web services.

In this study, we present our approach for integrating semantic web services and semantic web enabled agents under a model driven view. The primary focus of our work is the semantic web environment. We believe that agents must have well designed environment specific capabilities to fully utilize the power of semantic web environment. Hence in this paper, we first define a conceptual architecture for semantic web enabled agents and then discuss how this conceptual architecture can form the basis of a metamodel that can be used in the development of semantic web enabled agents with a model driven approach.

Model driven architecture (MDA) [23] defines a conceptual framework to realize model driven development. MDA is based on developing Platform Independent Models (PIMs) and then converting these PIMs to Platform Specific Models (PSMs) by model transformation. Therefore definitions of PIM and PSM are required for the development of semantic web enabled MAS with the MDA approach. In this paper, we zoom into the specific part of the metamodel that defines the interactions between semantic web enabled agents and semantic web services since it is not possible to cover all the aspects of the metamodel at one time. So we extend the metamodel of the conceptual architecture from the point of *entity aspect* for the interaction between

semantic agents and semantic web services. We model the agents and the relation between these agents and semantic web services.

The paper is organized as follows: In Section 2, we introduce the proposed approach for Semantic Web enabled MAS modeling. Our conceptual architecture for Semantic Web enabled MASs is discussed within this section. Section 3 introduces our metamodel that extends ACSM. This metamodel is the first step to incorporate a model driven approach to the development of MASs. So, in section 4, we model the interaction between the semantic agents and semantic web services using MDA approach from the entity view. We also discuss a model transformation example for agent plans within this section. Conclusion and future work are given in Section 5.

2 Proposed Approach for Semantic Web Enabled MAS Modeling

The basic entities of a Semantic Web enabled Multiagent System must be defined in order to apply model driven approaches for development of these systems. We believe that these entities can be derived from the conceptual architecture of Semantic Web enabled MASs. These conceptual entities derived from the conceptual architecture will constitute the key point for application models which are defined within the context of model driven software development. For this reason, we introduce the conceptual architecture of Semantic Web enabled MASs in the first following subsection and discuss the use of these conceptual entities and components within the context of model driven approach in the second subsection.

2.1 A Conceptual Architecture for Semantic Web Enabled MASs

As it is mentioned in Berners-Lee et al's study [4], the real power of the Semantic Web will be realized when programs are created that collect Web content from diverse sources, process the information and exchange the results with other programs. The computer programs in question are software agents and their effectiveness will increase exponentially as more machine-readable Web content and automated services (including other agents) become available. First of all, we need to define a conceptual architecture for semantic web enabled MASs to realize this vision. In this MAS architecture, autonomous agents can also evaluate semantic data and collaborate with semantically defined entities such as semantic web services by using content languages.

Our proposed conceptual architecture for Semantic Web enabled MASs is given in Figure 1. The architecture defines three layers: *Architectural Service Layer*, *Agency Layer* and *Communication Infrastructure Layer*. A group of system agents provides services defined in the Architectural Service Layer. Every agent in the system has an inner agent architecture described in the Agency Layer and they communicate with each other according to the protocols defined in the Communication Infrastructure.

Semantic web agents are agents which are initiated by using the platform architecture and able to use semantic services within the service layer. In Architectural Service Layer, services (and/or roles) of semantic web agents inside the platform are described. All services in the Architectural Service Layer use the

capability of the Agency Layer. Besides domain specific agent services, yellow page and mediator services should also be provided.

Agent Registry is a system facilitator in which capabilities of agents are semantically defined and advertised for other platform members. We also define a conceptual entity called *Semantic Service Registry* in the proposed architecture in order to provide semantic service discovery and execution for platform agents by advertising semantic capabilities of services. *Ontology Mediator* is another architectural service in which translation and mapping of different ontologies are performed to support interoperability of different agent organizations using different ontologies.

The middle layer of the architecture is the Agency which includes inner structural components of Semantic Web enabled agents. Every agent in the system has a *Semantic Knowledgebase* which stores the agent's local ontologies. Those ontologies are used by the agent during his interaction with other platform agents and semantic web services. Evaluation of the ontologies and primitive inference are realized by the *Reasoner*. *Semantic Knowledge Wrapper* within the Agency provides utilization of above mentioned ontologies by upper-level Agency components.

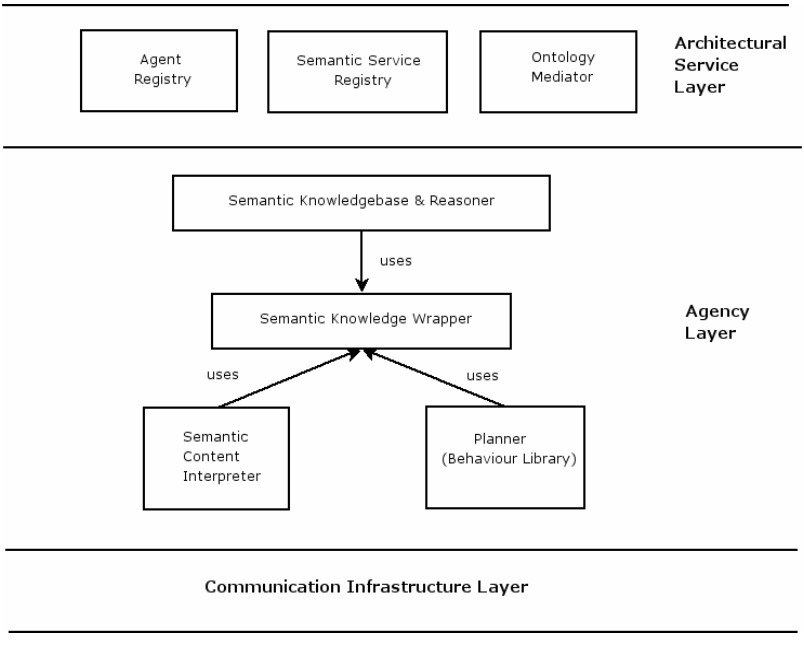

Fig. 1. The conceptual architecture for Semantic Web enabled MASs

The *Planner* of the Agency Layer includes necessary reusable plans with their related behavior libraries. On the other hand, the *Semantic Content Interpreter* module uses the logical foundation of semantic web, ontology and knowledge interpretation in order to check content validity and interpretation of the message during agent communications.

The bottom layer of the architecture is responsible of abstracting the architecture's communication infrastructure implementation. More detailed discussion of this proposed conceptual architecture can be found in [20].

2.2 Model Driven Engineering Approach for Semantic Web Enabled MAS

The implementation of methods and tools for the development of semantic web enabled multi agent systems based on the conceptual architecture discussed in Section 2.1 can be addressed by *Model Driven Engineering (MDE)*. MDE [6] is a recent approach that aims to increase the abstractness level in software development by using models in different phases and therefore by freeing the developers from the code level details. Each conceptual part of the architecture should be analyzed and designed while developing a semantic web enabled multiagent system. Using a model driven approach will enable us to reuse the components of the architecture and to generate the source code of the system from high level abstraction models.

Model Driven Architecture (MDA) [23] is one of the realizations of MDE to support the relations between platform independent and various platform dependent software artifacts. MDA defines several model transformations which are based on the Meta-Object-Facility [22] framework. These transformations are structured in a three-layered architecture: *the Computation Independent Model (CIM)*, *the Platform Independent Model (PIM)*, and *the Platform Specific Model (PSM)*. A CIM is a view of a system from the computation independent viewpoint [23]. Such a model is sometimes called a domain model or a business model. CIM requirements should be traceable to the PIM and PSM constructs by marking the proper elements in CIM. For instance, although the CIM does not have any information about agents and web services, the entities in the CIM are marked in an appropriate notation to trace the agents and semantic web services in the PIM of the semantic web enabled MAS. Bauer and Odell [2] discuss which aspects of a MAS could be considered at CIM and PIM.

The PIM specifies a degree of platform independency to be suitable for use with a number of different platforms of similar type [23]. In our perspective, the PIM of a semantic web enabled MAS should define the main entities and interactions which are derived from the conceptual architecture in Section 2.1. Also, the PIM of semantic web enabled MAS should have different aspects where specific concerns can be addressed. The PIM for service-oriented architecture discussed in [3] identifies four aspects: *information aspect*, *service aspect*, *process aspect* and *Quality of Service aspect*. In our approach, the PIM of semantic web enabled MAS can have mainly two aspects: *entity aspect* and *interaction aspect* in order to avoid decomposing the system into too many views. While the entity aspect combines the information aspect and service aspect defined in [3], the interaction aspect is similar with the process aspect and describes a set of interactions between agents and semantic services in terms of message exchange.

On the other hand, the PSM combines the PIM with the additional details of the platform implementation. The platform independent entities in the PIM of semantic web agents are transformed to the PSM of an implemented semantic web enabled agent framework like SEAGENT [8]. The flexible part of this approach is that the

PIM enables to generate different PSMs of semantic web enabled agent frameworks automatically. These PSMs can be considered as the realizations of our conceptual architecture.

The metamodel proposed in [20] defines the general concepts and entities of the proposed conceptual architecture. This metamodel provides the key point for customizing the entities for PIM and PSM metamodels in the MDA based development of this agent system. However, the current metamodel could not be considered as a complete PIM for semantic web enabled MAS. For instance, Semantic Web Service meta-entity and its related entities such as Service Ontology should be detailed. We believe that new entities for agent - semantic service interaction will be needed to add into the metamodel to provide this metamodel as a PIM for modeling the interaction in question.

Obviously, a Semantic Web Service encapsulates a service interface and a service process mechanism for its discovery and execution by semantic web agents. This interface and the semantic process should also be represented by appropriate entities in the metamodel in order to constitute the PSM of such MAS or directly generate semantic web enabled agent platform source code. Although there are ongoing efforts e.g. OWL-S [31] and WSMO [33] which aim to describe web services semantically, there is currently no platform independent standard for representation of these web services in order to be used in the semantic web environment. Due to the lack of this standard, representation of the service interface and the process mechanism constructs in the metamodel are difficult. Hence, it is not possible to define a PIM metamodel for semantic web enabled MAS without including these appropriate entities in the metamodel.

Another part of the semantic web enabled MAS architecture that must be detailed in the metamodel is the behavior library (planner). One of the implementation of reusable plans in the behavior library is the Hierarchical Task Network (HTN) planning [11] which is an AI planning methodology that creates plans by task decomposition. From the point of MDA based development, this HTN or other realization techniques of planning is defined in PSM level. For instance, SEAGENT which is a semantic web enabled MAS framework is based on the HTN planning framework presented by Sycara et al. [30] and the DECAF architecture [13]. If we consider the metamodel of SEAGENT framework as PSM, the HTN and other specific entities of the SEAGENT framework are defined in this metamodel as PSM entities. In PIM level, the general concepts of planning mechanism should be modeled and any specific component of HTN or other planning mechanisms should not be considered for platform independence.

In this study, we model the planning mechanism and the relation between this planning mechanism and semantic web service from the point of entity aspect. That is why we use the Class diagram to represent the model of this relation. In semantic web enabled MAS architecture, planner mechanism has the capability of executing plans consisting of special tasks for semantic service agents in a way described in [16]. The agents in the system can discover the appropriate service and invoke this service through the planning mechanism. The metamodel of the semantic web enabled MAS should consider the general entities of planner mechanism, semantic web service profile parameters and the relation between these entities. While the PIM metamodel

does not have any platform specific entities like HTN, OWL-S or WSMO, the implementation of this mechanism in SEAGENT could be considered as platform specific realization.

3 A Metamodel for Semantic Web Enabled MASs

In [20], we introduced a core agent metamodel superstructure to define elements and their relationships of a Semantic Web enabled MAS depending on the previously discussed conceptual architecture. However, the metamodel in question was improper to be used in model transformations and it was too primitive to support widely-accepted software modeling tools due to its arbitrary formalism. Therefore, in this study, we present one representation of the above metamodel by extending FIPA Modeling TC's Agent Class Superstructure Metamodel (ACSM) [24]. Although ACSM is currently also in its preliminary phase, we believe that it neatly presents an appropriate superstructure specification that defines the user-level constructs required to model agents, their roles and their groups. By extending this superstructure we do not need to re-define basic entities of the agent domain. Also, ACSM models assignment of agents to roles by taking into consideration of group context. Hence, extending ACSM clarifies relatively blurred associations between Semantic Organization, Semantic Agent and Role concepts in our metamodel by appropriate inclusion of ACSM's Agent Role Assignment entity. However, ACSM extension is not sufficient and we provide new constructs for our metamodel by extending UML 2.0 Superstructure and Ontology UML Profile which is defined by Djuric [9].

Before discussing our metamodel, ACSM is briefly mentioned below. More information about ACSM can be found in [24] and [25]. ACSM has a specification which is based on –and extends- UML superstructure. It proposes a superstructure for modeling agents, agent roles and agent groups. Its class model is illustrated in Figure 2.

ACSM utilizes the distinction between UML Classifier and UML Class. The agent classification in the model is based on an extension of Classifier. This provides omitting features of object-orientation (such as object-based messaging and polymorphism) which are troublesome for agents.

An Agent Classifier in the model defines various ways in which agents will be classified. It has two subclasses: Agent Physical Classifier which defines the primitive or basic classes describing core requirements of an agent and Agent Role Classifier which classifies agents by the various kinds of roles agents may play.

The Agent class defines the set of all agents that populate a system. Each instance of an Agent is associated with one or more Agent Classifiers that define its necessary features.

Group is defined as a set of agents which have been collected together for some reason. Within a group, its member agents interact according to the roles that they play. Groups are partitioned into Agentified Groups and Non-Agentified Groups according to whether or not they are addressable as an agent and can act as an agent in their own right.

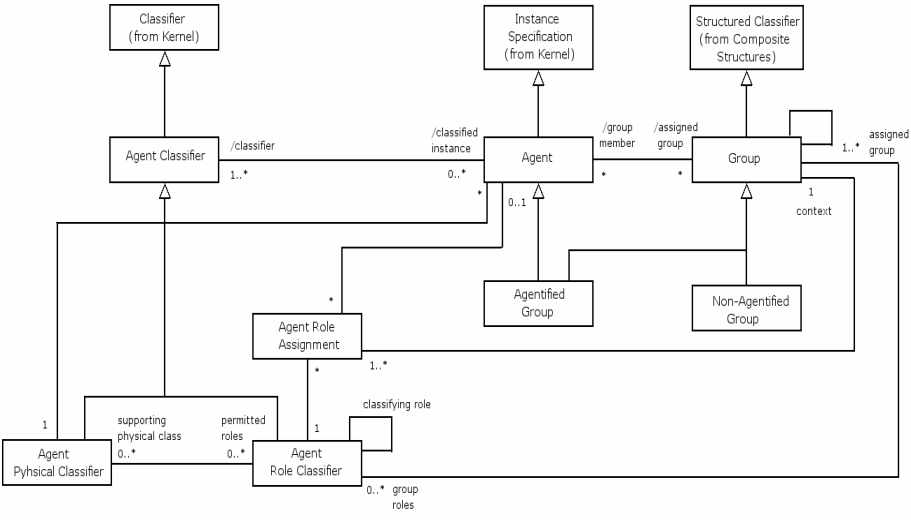

Fig. 2. FIPA Modeling TC's Agent Class Superstructure Metamodel [24]

Besides UML Classifier utilization, another noteworthy feature of the ACSM is modeling Agent – Role assignment as a ternary association. The fact that assignment of Agents to Roles is dynamic, required association is modeled by the Agent Role Assignment entity. A Role assignment between an agent and its role must be qualified by a group context. Hence, an Agent Role Assignment is a Class in the model whose associated instances associate Roles, Groups and Agents. Each instance of the ternary Agent Role Assignment associates a role, a group and an agent.

The Semantic Web enabled MAS metamodel being proposed in this study is given in Figure 3. The model extends FIPA Modeling TC's Agent Class, UML 2.0 superstructures and Ontology UML Profile.

As given in [20] a *Semantic Web Agent* is an autonomous entity which is capable of interaction with both other agents and semantic web services within the environment. It is a special form of the ACSM's Agent class due to its entity capabilities. It includes new features in addition to Agent classified instance.

Roles provide both the building blocks for agent social systems and the requirements by which agents interact as it has been remarked in [25]. We believe that the same is true for roles played in Semantic Web enabled agent environments. However, this general model entity should be specialized in the metamodel according to task definitions of architectural and domain based roles: An *Architectural Role* defines a mandatory Semantic Web enabled MAS role that should be played at least one agent inside the platform regardless of the organization context whereas a *Domain Role* completely depends on the requirements and task definitions of a specific Semantic Organization created for a specific business domain.

The Role concept in the metamodel is an extension of Agent Role Classifier due to its classification for roles the semantic agents are capable of playing at a given time. This conforms to the Agent – Agent Role Classifier association defined in ACSM

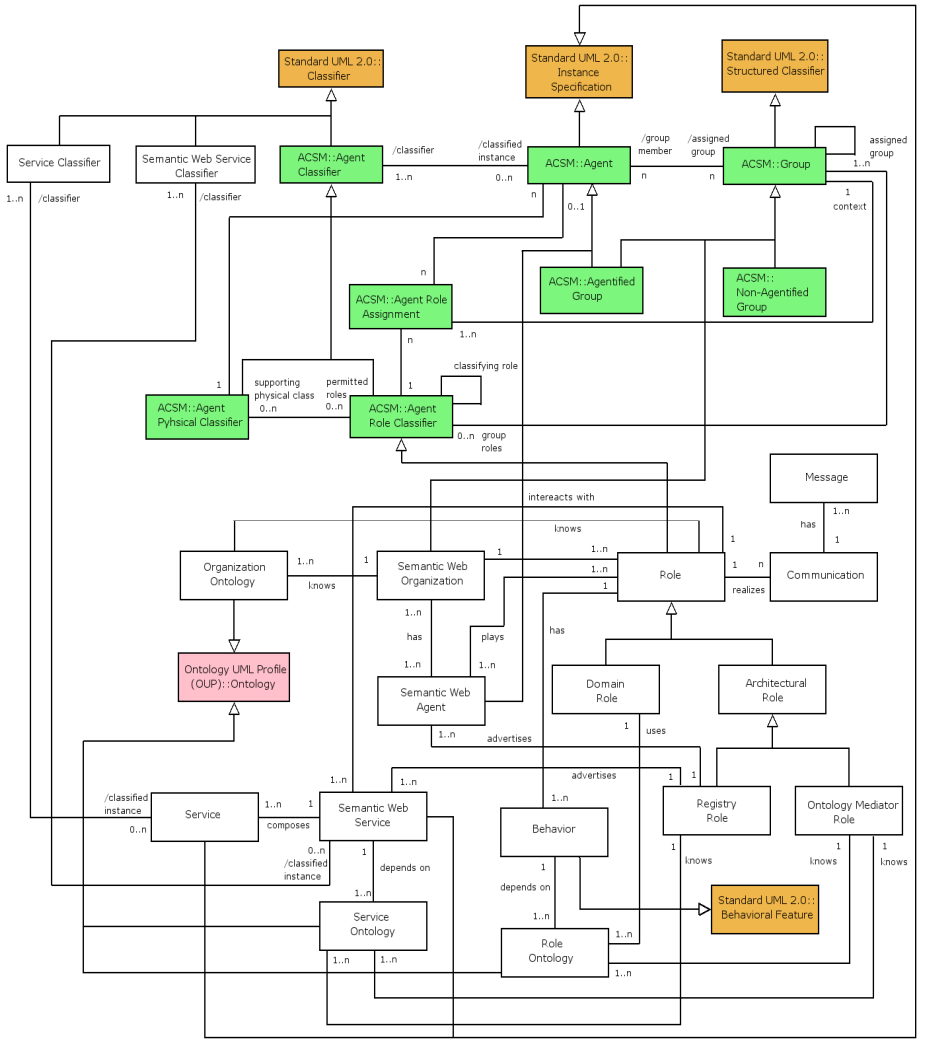

Fig. 3. The metamodel for Semantic Web enabled MASs which extends FIPA Modeling TC's Agent Class, UML 2.0 Superstructure and Ontology UML Profile

[25]: *Semantic Web Agents can be associated with more than one Role (which is also an Agent Role Classifier) at the same point in time (multiple classification) and can change roles over time (dynamic classification).*

Agent Role Classifiers form a generalization hierarchy. This is also valid for Semantic environment's Role elements. For example, in Figure 4, a hierarchy of Architectural Roles in SEAGENT [8] MAS framework is given. Due to its FIPA compliancy, related framework also defines a Registry Role called Directory Facilitator (DF). However, it also includes a service role called Semantic Service

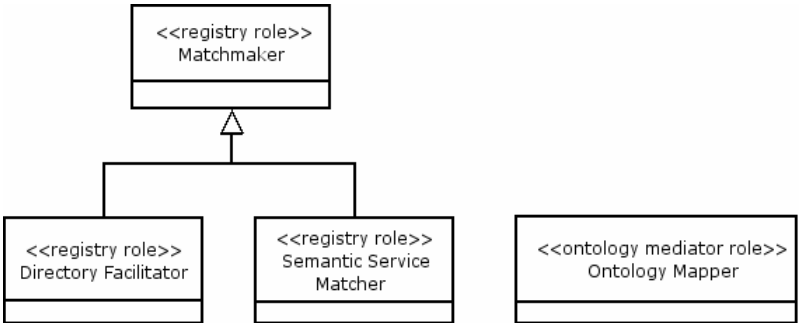

Fig. 4. A generalization hierarchy of Architectural Roles in SEAGENT MAS

Matcher (SSM) which should be played by some of the platform agents in order to realize Semantic Web Service – Agent interaction.

On the other hand, *Semantic Web Organization* is defined as a specialization of the ACSM's Group entity in the proposed model because it should be implemented as only a composition of Semantic Web Agents. However, a Semantic Web Organization may or may not behave as a Semantic Web Agent in overall manner. Hence, it shouldn't be defined neither as Agentified nor Non-Agentified Group. It is a direct extension of the Group Composite Structure.

Above discussed ACSM extensions provide clarification of the relations between Semantic Web Agent, Role and Semantic Web Organization in our model by presenting practicability of ACSM's Agent Role Assignment ternary association between Agent, Agent Role Classifier and Group.

The metamodel is also based on – and extends – UML 2.0 Superstructure to define meta-elements of the Semantic Web environment. For example, we have defined a first-class entity called Semantic Web Service Classifier in our core model. This entity is defined in the final model as a UML 2.0 Classifier extension.

A *Semantic Web Service* represents any service (except agent services) whose capabilities and interactions are semantically described within a Semantic Web enabled MAS. A Semantic Web Service composes one or more *Service* entities. Each service may be a web service or another service with predefined invocation protocol in real-life implementation. But they should have a semantic web interface to be used by autonomous agents of the platform.

Like agents, semantic web services have also capabilities and features which could not be just based on object-oriented paradigm. Hence, we define new Classifiers and their related Instance Specifications in the metamodel to encapsulate semantic web entities. We have applied classifier – classified instance association between *Semantic Web Service Classifier* and *Semantic Web Service*. Same is valid for *Service Classifier – Service* relationship.

Ontology entities (*Organization Ontology*, *Service Ontology* and *Role Ontology*) are defined as extensions of the *Ontology* element of the Ontology UML Profile (OUP) defined in [9]. OUP captures ontology concepts with properties and relationships and provides a set of UML elements available to use as semantic types in our metamodel. By deriving the semantic concepts from OUP, there will be already-defined UML elements to use as semantic concepts within the metamodel.

One Role is composed of one or more *Behavior*s. Task definitions and related task execution processes of Semantic Web agents are modeled inside Behavior entities. The Behavior entity is defined in the metamodel as a UML 2.0 Behavioral Feature because it refers to a dynamic feature of a Semantic Web Agent (e.g. an agent task which realizes agent interaction with other agents).

According to played roles, agents inevitably communicate with other agents to perform desired tasks. Each *Communication* entity defines a specific interaction between two agents of the platform which takes place in proper to predefined agent interaction protocol. One Communication is composed of one or more *Message*s whose content can be expressed in a RDF based semantic content language.

Figure 5 portrays an example semantic role assignment considering a MAS working in Tourism domain.

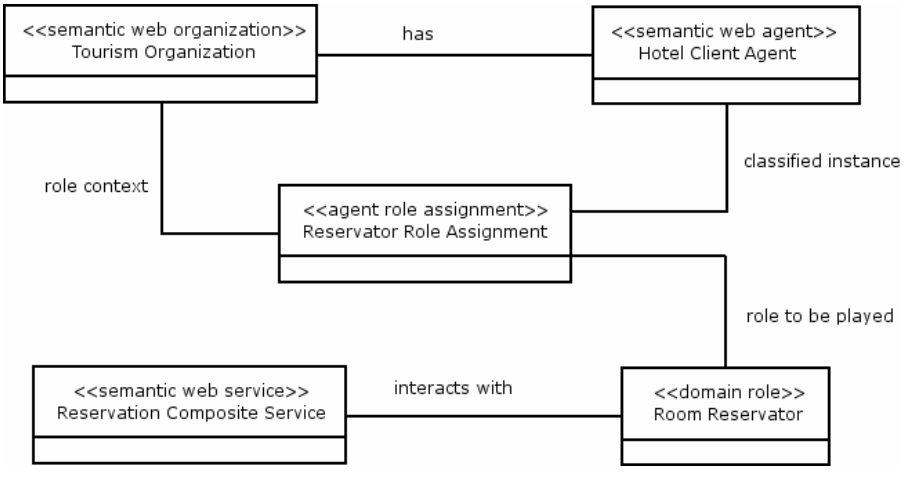

Fig. 5. A Semantic Role Assignment for a MAS working in Tourism domain

In this system, there exists an Agent Role Assignment Class called "Reservator Role Assignment" which represents the three-way association between a Hotel Client Agent, the Room Reservator Role and Tourism Organization. Hotel Client is a Semantic Web Agent which reserves hotel rooms on behalf of its human users. Within the Semantic Web Organization called Tourism Organization, the semantic web agent plays a Room Reservator Role. The related role includes a semantic web service interaction during its task execution: Hotel Client Agent uses Reservation Composite semantic web service which may be a composition of discovery, engagement and invocation services for hotel room reservation.

4 Elaboration of the Metamodel by Considering the Interaction Between Semantic Agents and Semantic Web Services

The metamodel discussed in the previous section defines required meta-entities and entity relations of a Semantic Web enabled MAS architecture. However, interaction

between semantic agents and external services needs to be studied in more detail in order to realize model transformations during system development. Such a study also provides a practical evaluation of the proposed metamodel. The extended model given in Figure 6 elaborates the agent – service interaction from the point of entity aspect.

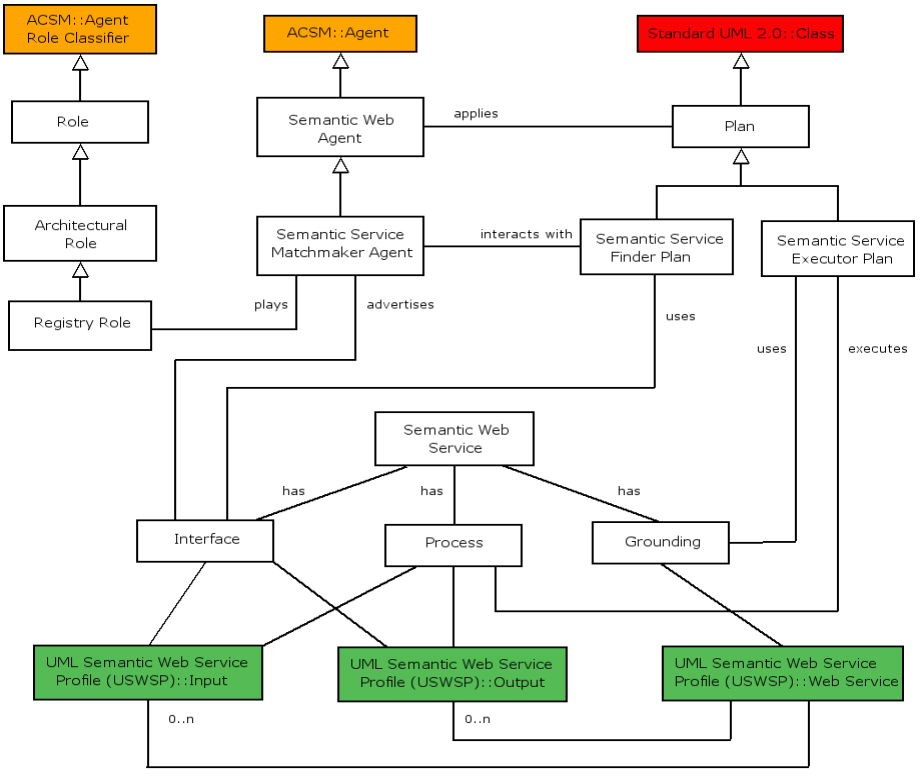

Fig. 6. The extended metamodel of the interaction between Semantic Agents and Semantic Web Services

Semantic Web Agents have *Plan*s to discover and execute Semantic Web Services dynamically. In order to discover service capabilities, agents need to communicate with a service registry. For this reason, the model includes a specialized agent entity, called *Semantic Service Matchmaker Agent*. This meta-entity represents matchmaker agents which store capability advertisements of semantic web services within a MAS and match those capabilities with service requirements sent by the other platform agents.

When we consider various semantic web service modeling languages such as OWL-S [31] and WSMO [33], it is clear that services are represented by three semantic documents: *Service Interface*, *Process Model* and *Physical Grounding*. Service Interface is the capability representation of the service in which service inputs, outputs and any other necessary service descriptions are listed. Process Model describes internal composition and execution dynamics of the service. Finally

Physical Grounding defines invocation protocol of the web service. These Semantic Web Service components are given in the metamodel with *Interface*, *Process* and *Grounding* entities respectively. Semantic input, output and web service definitions used by those service components are exported from the UML Semantic Web Service Profile proposed in [14].

Semantic Web Agents have two consecutive plans to interact with Semantic Web Services. *Semantic Service Finder Plan* is a Plan in which discovery of candidate semantic web services takes place. During this plan execution, the agent communicates with the service matchmaker of the platform to determine proper semantic services. After service discovery, the agent applies the *Semantic Service Executor Plan* in order to execute appropriate semantic web services. Process model and grounding mechanism of the service are used within the plan. An instance model of the above metamodel is given in Figure 7 for the interaction between a Hotel Client Agent and a Reservation Service within a MAS working in Tourism domain.

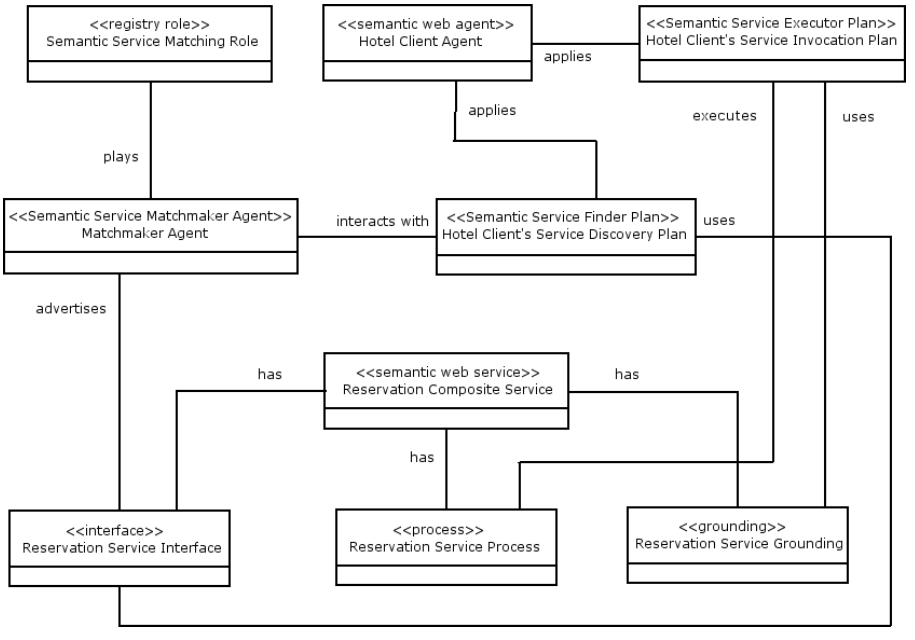

Fig. 7. An instance model for the agent – service interaction within a MAS working in Tourism domain

As previously mentioned, the client agent is a Semantic Web Agent which reserves hotel rooms on behalf of its human users. During its task execution, it needs to interact with a semantic web service called Reservation Composite Service. Matchmaker Agent is the service matcher of the related agent platform.

When we consider the metamodel as a PIM of the agent – service interaction, we should give the corresponding PSM entities in an implemented Semantic Web enabled MAS environment. As previously mentioned, SEAGENT [8] is a MAS development framework which provides built-in components for Semantic Web enabled MASs. Hence, in Table 1, we give the mappings between entities of our proposed metamodel and SEAGENT framework. These mappings precede the transformation between PIM and PSMs of such kind of MASs according to MDA approach.

Table 1. Mappings between the metamodel and SEAGENT framework entities

Metamodel Entity	SEAGENT Entity	Explanation
Registry Role Semantic Service Matchmaker Agent (SSMA)	Semantic Service Matcher (SSM)	Both Registry role and SSMA in the metamodel corresponds to the SSM in SEAGENT.
Plan	HTN Plan	In SEAGENT MAS, agent plans are designed as hierarchical task networks.
Semantic Service Finder Plan	HTN Finder Task	
Semantic Service Executor Plan	HTN Executor Task	
Semantic Web Agent	Agent	
Semantic Web Service	OWL-S Service	In SEAGENT, capabilities and process models of semantic web services are defined by using OWL-S markup language.
Interface	OWL-S Profile	
Process	OWL-S Process	
Grounding	OWL-S Grounding	

To derive a transformation (based on the mappings listed in Table 1) from metamodel entities depicted in Figure 6 to SEAGENT entities, we first define a platform dependent metamodel and instance models of SEAGENT. Since SEAGENT is implemented in Java, we do not need a customized metamodel instead of Java metamodel. All SEAGENT entities defined in Table 1 are a realization of the meta classes in Java metamodel. In our transformation we use a Kernel MetaMetaModel (KM3) based on the Java metamodel which is defined in a metamodel zoo [32]. All the metamodels available in this zoo [32] are expressed in KM3 [18] metamodel format and can be injected to an “ecore” file which is an Eclipse Modeling Framework (EMF) [10] format.

SEAGENT dependent plan and semantic web agent models based on this Java metamodel for the interaction between semantic web agents and semantic web services contain the components of SEAGENT plan structure. Gürcan et al [16] define a software platform which fulfills fundamental requirements of Semantic Web Services Architecture's (SWSA) [7] conceptual model including all its sub-processes

and a planner that has the capability of reusable plans in which these sub-processes are modeled for development of semantic service agents. This plan structure [16] is similar to the frameworks presented by Sycara et al [30] and the DECAF architecture [13]. As a requirement of HTN, tasks might be either complex (called behaviors) or primitive (called actions). Each plan consists of a complex root task consisting of sub-tasks to achieve a predefined goal.

Components of our plan structure are shown in Figure 8. Tasks have a name describing what they are supposed to do and have zero or more provisions (information needs) and outcomes (execution results). The provision information is supplied dynamically during plan execution. Tasks are ready, and thus eligible for execution, when there is a value for each of its provisions. Related control is done via *isAllProvisionsAreSet()* method. The more detailed information about SEAGENT plan structure can be found in [16].

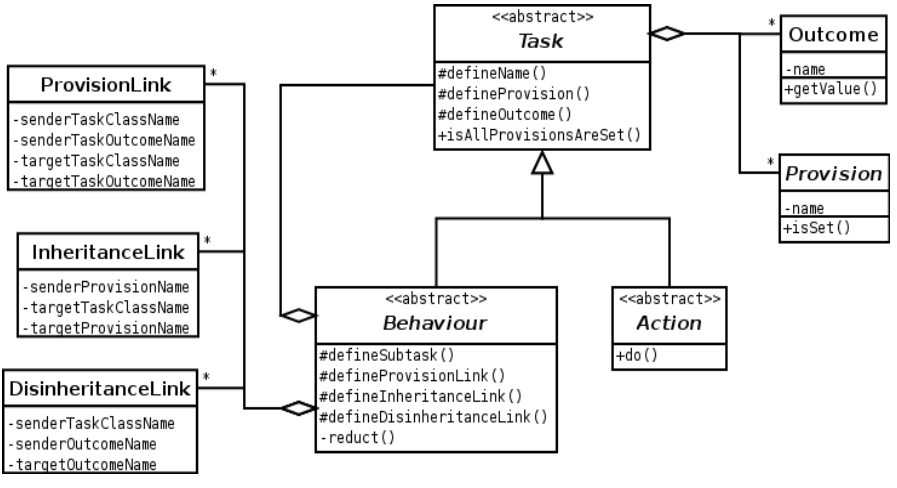

Fig. 8. Components of SEAGENT Plan Structure [16]

According to the plan structure depicted in Figure 8, we define an instance SEAGENT plan model based on Java metamodel for the agent – service interaction within a MAS working in Tourism domain whose platform independent model is shown in Figure 7.

Figure 9 shows an instance plan model in SEAGENT for the agent – service interaction within a MAS working in Tourism domain. This is the corresponding platform dependent model of the plan part of the platform independent model depicted in Figure 7. Every entity in this model is a Java class which is defined as a meta class in the Java metamodel. In this plan, *FindaHotel*, *FindaRoom*, and *MakeRoomReservation* sub-tasks of the plan are concrete realizations of *ExecuteService* task. They are connected with their provisions and outcome slots, and because they are domain dependent plans they know what input parameters they will take. Since the realization of a Plan from another plan is done through inheritance

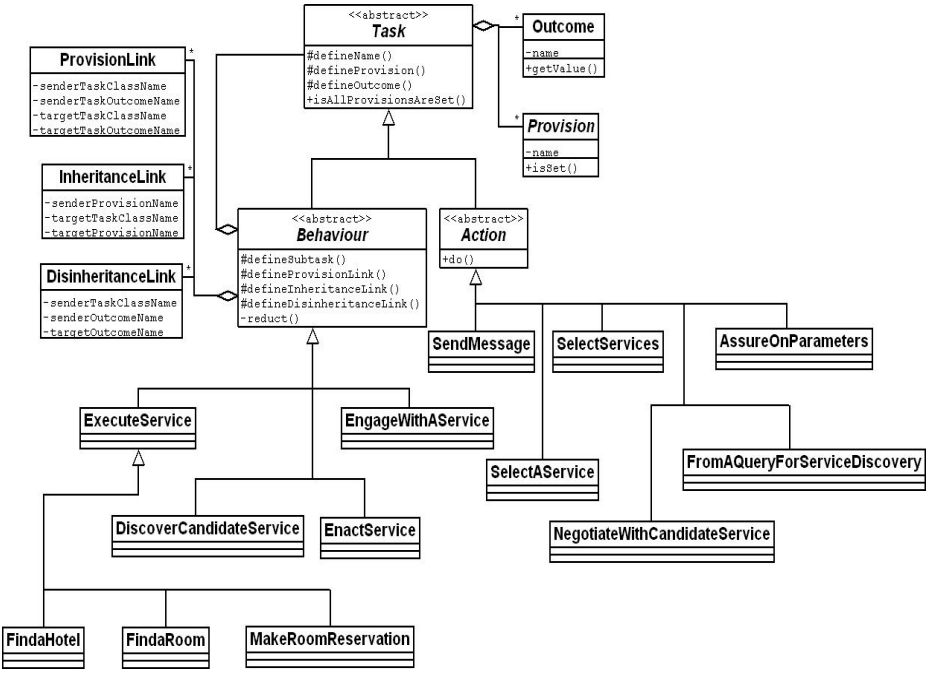

Fig. 9. An instance plan model in SEAGENT for the agent – service interaction within a MAS working in Tourism domain

relations between Java classes in SEAGENT, *FindaHotel* class is extended from *ExecuteService* class and *ExecuteService* class is extended from *Behaviour* class.

Currently, we are working on implementing the transformations derived from the mappings given in this study to realize the model driven development of Semantic Web enabled MASs using MDA approach. For this purpose, we use ATLAS INRIA & LINA research group's ATL (Atlas Transformation Language) which is a model transformation language specified as both a metamodel and a textual concrete syntax [19].

Figure 10 summarizes the full model transformation process. A model M_a, conforming to a metamodel MM_a, is here transformed into a model M_b that conforms to a metamodel MM_b. The transformation is defined by the model transformation model M_t which itself conforms to a model transformation metamodel MM_t. This last metamodel, along with the MM_a and MM_b metamodels, has to conform to a metametamodel MMM such as MOF (Meta Object Facility) or Ecore [10].

In our transformation case, MMM is Ecore and MM_t is ATL. Our source model (M_a) is the model given in Figure 7 which conforms to metamodel (MM_a) given in Figure 6. When we apply a transformation into our source model, we aim to obtain the platform specific destination model (M_b) which conforms to metamodel of the SEAGENT planner depicted in Figure 9.

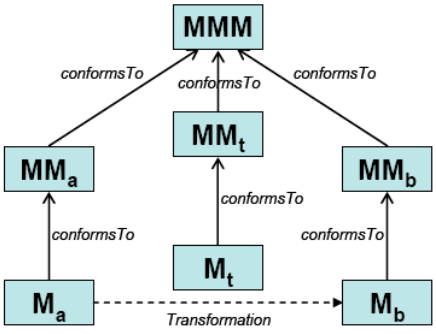

Fig. 10. An overview of model transformation [19]

Consider the simple example in which we transform a Semantic Service Finder Plan (in Figure 6) into its corresponding SEAGENT plan which is DiscoverCandidateService (in Figure 9) within the ATL environment. To do this we have to create EMF encodings -.ecore files- of both models and use them in ATL transformation.

EMF provides its own file format (.ecore) for model and metamodel encoding. However the manual edition of Ecore metamodels is particularly difficult with EMF. In order to make this common kind of editions easier, the ATL Development Tools (ADT) include a simple textual notation dedicated to metamodel edition: the Kernel MetaMetaModel (KM3) [18]. This textual notation eases the edition of metamodels. Once edited, KM3 metamodels can be injected into the Ecore format using ADT integrated injectors. More information about KM3 and Ecore injection can be found in [18, 19].

Following is the part of the KM3 file in which Semantic Service Finder Plan is represented:

```
package SemanticServiceFinderPlan {
      class SemanticServiceFinderPlan {
            attribute plan_name : String;
            reference desiredServiceInterface : Interface;
      }
      class Interface {
            attribute input: String;
            attribute output: String;
            attribute precondition: String;
            attribute effect: String;
      }
}

package PrimitiveTypes {      datatype String;      }
```

Notice that service interface metamodel definition in here is extremely simplified for the demonstration purposes. In a real transformation, IOPE (Input, Output, Precondition and Effect) attributes of a semantic service interface would have complex types. The ecore model conforming to above metamodel includes the following model instance which will be given into transformation process:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI  xmlns:xmi="http://www.omg.org/XMI" xmlns="SemanticServiceFinderPlan">
    <SemanticServiceFinderPlan>
        <name>Hotel Client's Service Discovery Plan</name>
        <Interface>ReservationServiceInterface</Interface>
    </SemanticServiceFinderPlan>
</xmi:XMI>
```

The KM3 representation of the destination model's metamodel is given below:

```
package DiscoverCandidateService {
      class DiscoverCandidateService {
            attribute name : String;
            attribute candidateServiceInputList : String;
            attribute candidateServiceOutputList : String;
      }
}

package PrimitiveTypes {      datatype String;      }
```

Finally, here is the transformation rule written in ATL which will be used by the ATL engine in order to generate the model conforming to DiscoverCandidateService's metamodel:

```
module SemanticServiceFinderPlan2DiscoverCandidateService;
create OUT : DiscoverCandidateService from IN : SemanticServiceFinderPlan;
rule SemanticServiceFinderPlan {
    from
      ssfp : SemanticServiceFinderPlan!SemanticServiceFinderPlan
    to
      dcs : DiscoverCandidateService!DiscoverCandidateService (
          name <- ssfp.name,
          candidateServiceInputList <- ssfp. desiredServiceInterface.input,
          candidateServiceOutputList <- ssfp. desiredServiceInterface.output
      )
}
```

The engine applies the above rule in order to transform "Hotel Client's Service Discovery Plan" model which conforms to SemanticServiceFinderPlan metamodel into a model instance that can be used within the SEAGENT environment conforming to plan metamodel of DiscoverCandidateService.

5 Conclusion and Future Work

A metamodel for Semantic Web enabled MASs and the extended part of this metamodel for the interaction between semantic agents and semantic web services are introduced in this paper. This extended metamodel can be considered as a part of *Platform Independent Model* within the context of MDA approach. This PIM models the planning mechanism and the relation between this planning mechanism and semantic web service from the point of entity aspect. The agents in the system can discover the appropriate semantic services and invoke these services through the planning mechanism. General entities of the planner mechanism, semantic web

service profile parameters and the relation between these entities are considered. While the PIM does not have any platform specific entities of HTN, OWL-S or WSMO, the implementation of these mechanisms in SEAGENT could be considered as platform specific realization. The mappings between the entities of the metamodel and the implemented entities of SEAGENT framework in Section 4 show the practical relevance of the metamodel.

In our future work, we aim to define interaction aspect of this extended metamodel at first. Meanwhile, we also intend to improve mappings and model transformations introduced in this study. The metamodel in here is only extended for interaction between semantic agents and semantic web services. Hence, as our further work, we plan to extend other parts of the metamodel according to the components of the layered conceptual architecture and provide tool support for the proposed metamodel.

References

[1] Atkinson, C., Kühne, T.: Model-Driven Development: A Metamodeling Foundation. IEEE Software 20, 36–41 (2003)

[2] Bauer, B., Odell, J.: UML 2.0 and Agents: How to Build Agent-based Systems with the New UML Standard. Journal of Engineering Applications of Artificial Intelligence 18(2), 141–157 (2005)

[3] Benguria, G., Larrucea, X., Elvesaeter, B., Neple, T., Beardsmore, A., Winchester, M.: A Platform Independent Model for Service Oriented Architectures. In: Interoperability for Enterprise Software and Applications Conference (I-ESA'06), Bordeaux, France (2006)

[4] Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American 284(5), 34–43 (2001)

[5] Bernon, C., Cossentino, M., Gleizes, M., Turci, P., Zambonelli, F.: A Study of some Multi-Agent Meta-Models. In: Odell, J.J., Giorgini, P., Müller, J.P. (eds.) AOSE 2004. LNCS, vol. 3382. Springer, Heidelberg (2005)

[6] Bezivin, J.: Model Driven Engineering: Principles, Scope, Deployment and Applicability. In: Proceedings of 2005 Summer School on Generative and Transformational Techniques in Software Engineering (July 2005)

[7] Burstein, M., Bussler, C., Zaremba, M., Finin, T., Huhns, M., Paolucci, M., Sheth, A., Williams, S.: A semantic web services architecture. IEEE Internet Computing 9(5), 72–81 (2005)

[8] Dikenelli, O., Erdur, R.C., Kardas, G., Gümüs, O., Seylan, I., Gurcan, O., Tiryaki, A.M., Ekinci, E.E.: Developing Multi Agent Systems on Semantic Web Environment using SEAGENT Platform. In: Dikenelli, O., Gleizes, M.-P., Ricci, A. (eds.) ESAW 2005. LNCS (LNAI), vol. 3963, pp. 1–13. Springer, Heidelberg (2006)

[9] Djuric, D.: MDA-based Ontology Infrastructure. Computer Science Information Systems (ComSIS) 1(1), 91–116 (2004)

[10] Eclipse Modeling Framework (2006) available at: http://www.eclipse.org/emf

[11] Erol, K., Hendler, J.A., Nau, D.S.: Complexity Results for HTN Planning. Ann. Math. Artif. Intell. (1996)

[12] Ferber, J., Gutknecht, O.: A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems. In: Proc. 3rd International Conference on Multi-Agent Systems, pp. 128–135. IEEE Computer Society Press, Los Alamitos (1998)

[13] Graham, J.R., Decker, K., Mersic, M.: DECAF – a flexible multi agent system architecture. Autonomous Agents and Multi-Agent Systems (2003)

[14] Gronmo, R., Jaeger, M.C., Hoff, H.: Transformations between UML and OWL-S. In: Hartman, A., Kreische, D. (eds.) ECMDA-FA 2005. LNCS, vol. 3748, pp. 269–283. Springer, Heidelberg (2005)
[15] Guessoum, Z., Thiefaine, A., Perrot, J., Blain, G.: META-DIMA: a Model-Driven Architecture for Multi-Agent Systems, (last accessed: 2006), http://www-poleia.lip6.fr/%7Eguessoum/MetaDima.html
[16] Gürcan, Ö., Kardas, G., Gümüs, Ö., Ekinci, E.E., Dikenelli, O.: A Planner for Implementing Semantic Service Agents based on Semantic Web Services Initiative Architecture. In: Fourth European Workshop on Multi-Agent Systems, Lisbon, Portugal (2006)
[17] Jayatilleke, G.B., Padgham, L., Winikoff, M.: Towards a Component Based Development Framework for Agents. In: Lindemann, G., Denzinger, J., Timm, I.J., Unland, R. (eds.) MATES 2004. LNCS (LNAI), vol. 3187, pp. 183–197. Springer, Heidelberg (2004)
[18] Jouault, F., Bezivin, J.: KM3: A DSL for Metamodel Specification. In: Gorrieri, R., Wehrheim, H. (eds.) FMOODS 2006. LNCS, vol. 4037, pp. 171–185. Springer, Heidelberg (2006)
[19] Jouault, F., Kurtev, I.: Transforming Models with ATL. In: Bruel, J.-M. (ed.) MoDELS 2005. LNCS, vol. 3844, pp. 128–138. Springer, Heidelberg (2006)
[20] Kardas, G., Goknil, A., Dikenelli, O., Topaloglu, N.Y.: Metamodeling of Semantic Web Enabled Multiagent Systems. In: Multiagent Systems and Software Architecture (MASSA), Erfurt, Germany, pp. 79–86 (2006)
[21] Molesini, A., Denti, E., Omicini, A.: MAS Meta-models on Test: UML vs. OPM in the SODA Case Study. In: Pěchouček, M., Petta, P., Varga, L.Z. (eds.) CEEMAS 2005. LNCS (LNAI), vol. 3690. Springer, Heidelberg (2005)
[22] Object Management Group (OMG): Meta Object Facility (MOF) Specification. OMG Document AD/97-08-14, (September 1997)
[23] Object Management Group (OMG): MDA Guide Version 1.0.1. Document Number: omg/2003-06-01 (2003)
[24] Odell, J., Levy, R., Nodine, M.: FIPA Modeling TC: Agent Class Superstructure Metamodel (2004), available at: http://www.omg.org/docs/agent/04-12-02.pdf
[25] Odell, J., Nodine, M., Levy, R.: A Metamodel for Agents, Roles and Groups. In: Odell, J.J., Giorgini, P., Müller, J.P. (eds.) AOSE 2004. LNCS, vol. 3382. Springer, Heidelberg (2005)
[26] Pavon, J., Gomez, J., Fuentes, R.: Model Driven Development of Multi-Agent Systems. In: Rensink, A., Warmer, J. (eds.) ECMDA-FA 2006. LNCS, vol. 4066, pp. 284–298. Springer, Heidelberg (2006)
[27] Perini, A., Susi, A.: Automating Model Transformations in Agent-Oriented Modeling. In: Odell, J.J., Giorgini, P., Müller, J.P. (eds.) AOSE 2004. LNCS, vol. 3382. Springer, Heidelberg (2005)
[28] Selic, B.: The Pragmatics of Model-Driven Development. IEEE Software 20, 19–25 (2003)
[29] Sendall, S., Kozaczynski, W.: Model Transformation – the Heart and Soul of Model-Driven Software Development. IEEE Software 20, 42–45 (2003)
[30] Sycara, K., Williamson, M., Decker, K.: Unified information and control workflow in hierarchical task networks. In: Working Notes of the AAAI-96 workshop 'Theories of Action, Planning, and Control' (1996)
[31] The OWL Services Coalition: Semantic Markup for Web Services (OWL-S) (2004), http://www.daml.org/services/owl-s/1.1/
[32] The Atlantic Zoo: Metamodels expressed in KM3 (2006), available at: http://www.eclipse.org/gmt/am3/zoos/atlanticZoo/
[33] Web Service Modeling Ontology (2005), http://www.wsmo.org/