Model Transformation for Model Driven Development of Semantic Web Enabled Multi-Agent Systems

Geylani Kardas[1], Arda Goknil[2], Oguz Dikenelli[3], 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. Model Driven Development (MDD) provides an infrastructure that simplifies Multi-agent System (MAS) development by increasing the abstraction level. In addition to defining models, transformation process for those models is also crucial in MDD. On the other hand, MAS modeling should also take care of emerging requirements of MAS deployment on the Semantic Web environment. Hence, in this paper we propose a model transformation process for MDD of Semantic Web enabled MASs. We first define source and target models for the transformation regarding the modeling of interactions between agents and semantic web services and then grant mappings between these source and model entities to derive transformation rules and constraints. Finally we realize the whole transformation for a real MAS framework by using a well-known model transformation language named ATL.

1 Introduction

The design and implementation of Multi-agent Systems (MAS) becomes more complex and hard to implement when new requirements and interactions for new agent environments such as Semantic Web [2] are considered. To work in a higher abstraction level is of critical importance for the development of MASs since it is almost impossible to observe code level details of MAS due to their internal complexity, distributedness and openness.

Model Driven Development (MDD) [20], which aims to change the focus of software development from code to models, provides an infrastructure that simplifies the development of future's MASs. Such MDD application increases the abstraction level in MAS development. Although there are ongoing efforts in model driven MAS development, a significant deficiency exists in current studies when we consider modeling of agent systems working on Semantic Web environment. The main challenge in here is to provide new entities and define their relations with the traditional MAS entities for MAS metamodels pertaining to the Semantic Web and

employ those new metamodels in a neatly presented model transformation process within the scope of MDD.

In our previous work, we first provided a conceptual MAS architecture [13] in which autonomous agents can also evaluate semantic data and collaborate with semantically defined entities such as semantic web services by using content languages and then we derived entities of a MAS metamodel from the introduced architecture and defined their relations [14]. This new MAS metamodel paves the way for MDD of Semantic Web enabled agent systems in our studies by presenting an alternative for platform independent metamodel of such agent systems.

Definition of such a model is a prerequisite to conduct model transformation which is the key activity in MDD. Hence in this paper, we present a model transformation process for MDD of agent systems working on Semantic Web. A model conforming to above MAS metamodel is transformed into another model conforming to model of a real agent platform within the introduced process. The designed Semantic Web enabled MAS can be implemented on this real platform by applying the transformation. To accomplish this, we first define source and target metamodels for the transformation and then provide mappings between entities of these models to derive transformation rules and constraints. Finally we realize the whole transformation by using a pretty known model transformation language.

The paper is organized as follows: In Section 2, we briefly discuss how MDD can be applied for the development of the Semantic Web enabled agent systems. Models for the related transformation are introduced in Section 3. Application of the model transformation is discussed in Section 4. Section 5 covers related work on MDD of agent systems. Conclusion and future work are given in Section 6.

2 MDD for Semantic Web Enabled MAS Development

MDD approach considers the models as the main artifacts of software development. We use *Model Driven Architecture (MDA)* [15] which is one of the realizations of MDD to support the relations between platform independent and various platform dependent agent artifacts to develop semantic web agents.

MDA defines several model transformations which are based on the Meta-Object-Facility (MOF) [15] 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 [15]. 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 semantic web services, entities in CIM are marked in an appropriate notation to trace the agents and semantic web services in the PIM of the Semantic Web enabled MAS. The PIM specifies a degree of platform independency to be suitable for use with a number of different platforms of similar type [15]. In our perspective, the PIM of a Semantic Web enabled MAS should define the main entities and interactions which are derived from the above mentioned conceptual architecture. Finally, PSM combines PIM with additional details of the platform implementation. The platform

independent entities in PIM of semantic web agents are transformed to PSM of an implemented Semantic Web enabled agent framework like SEAGENT [3]. 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 development process and the MOF based transformations between the MDA models are given in Fig. 1. In the depicted transformation pattern, a source model *sm* is transformed into a target model *tgm*. The transformation is driven by a transformation definition written in a transformation language [5] [12]. The source model, the target model and the transformation definition conform to their metamodels *SMM*, *TgMM* and *TMM* respectively. The transformations defined from CIM to PIM and PIM to PSM use the metamodels of CIMs, PIMs and PSMs for source and target metamodels in the transformation pattern.

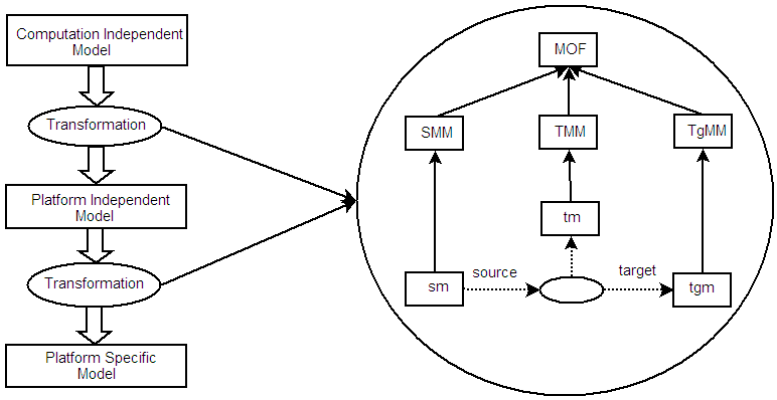

Fig. 1: Transformation Steps in MDA.

We applied the transformation mechanism depicted in Fig. 1 for models conforming to our Semantic Web enabled agent metamodel [14] and SEAGENT [3] model respectively. Due to space limitations, the whole transformation process couldn't be discussed in this paper. However, we believe that interaction between semantic agents and semantic web services is crucial for development of such MASs. Hence, rest of the paper describes modeling of this interaction and whole process of the related transformation.

3 Models for Agent – Semantic Web Service Interaction

Model transformation requires syntactical and semantic definitions of models which are provided by metamodels. We introduced a metamodel for Semantic Web enabled MASs in [14] which extends FIPA Modeling TC's Agent Class Superstructure Metamodel (ACSM) [16]. By extending ACSM, 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. Therefore, 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 [4].

Ontology entities of the metamodel are defined as extensions of the *Ontology* element of the Ontology UML Profile (OUP) defined in [4]. 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.

The aim of this study is to present model transformation for developing Semantic Web enabled MAS by employing our metamodel so full specification of the model is beyond the scope of this paper. The specification of the complete model can be found in [14]. In here, we discuss on its zoomed part in which the interaction between agents and semantic web services is elaborated.

The metamodel given in Fig. 2 is the PIM which will be our source metamodel during the transformation process. This metamodel provides modeling the agent – service interaction from the point of entity aspect.

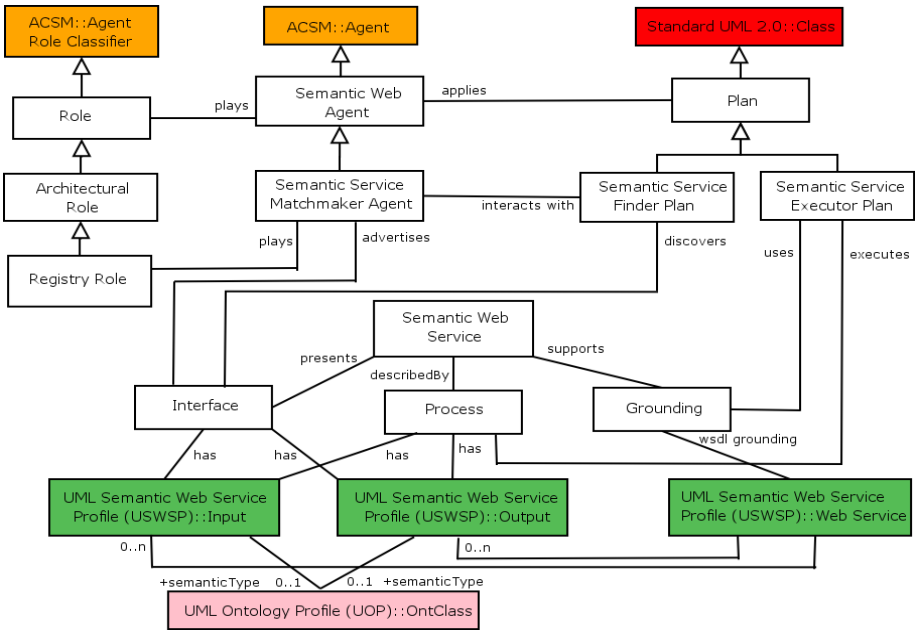

Fig. 2: The metamodel of the interaction between Agents and Semantic Web Services.

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.

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 [16]. In here, we also define its one sub-entity called *Architectural Role*. This 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.

Semantic Web Agents have *Plan*s to discover and execute *Semantic Web Service*s 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. This agent plays the *Registry Role* which is a specialized Architectural Role.

A *Semantic Web Service* represents any service (except agent services) whose capabilities and interactions are semantically described within a Semantic Web enabled MAS. 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.

When we consider various semantic web service modeling languages such as OWL-S [21] and WSMO [22], 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 [8].

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.

The input model of our transformation process is an instance model which conforms to the above mentioned interaction metamodel. This source model for the transformation is given in Fig. 3. The model depicts the interaction between a Hotel Client Agent and a Reservation Service within a MAS working in Tourism domain. 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. Hotel Client Agent determines appropriate semantic service by asking the Matchmaker Agent and interacts with the determined

semantic service by executing service's process description and using service's grounding.

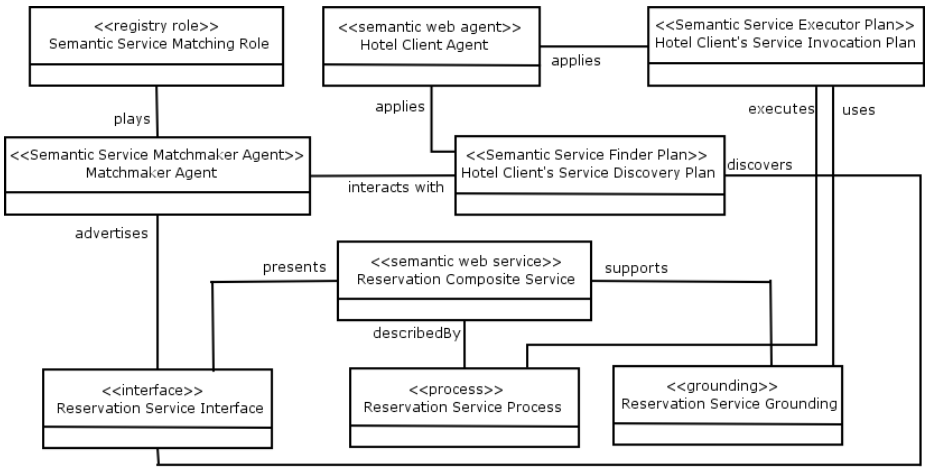

Fig. 3: An instance model for the agent – service interaction within a MAS working in Tourism domain. The model is used in the transformation process as the source model.

To realize MDD of the MAS defined in Fig. 3, we employ the transformation between PIM and PSM shown in Fig. 1. We can facilitate implementation of the specified agent system in various Semantic Web enabled agent development environments such as SEAGENT [3] if we provide metamodels of the corresponding frameworks as platform specific metamodels and define transformation rules.

In this study, our target platform for platform specific models is the SEAGENT framework. SEAGENT is implemented in Java and provides libraries to develop Semantic Web enabled MASs also in Java. Java classes and objects are concrete realizations of our PIM entities in the platform specific level and target (output) model of the transformation will be a Java model (composed of SEAGENT classes and their associations). This Java model conforms to the metamodel of Java [9].

Table 1: Mappings between the metamodel entities and SEAGENT classes

Metamodel Entity	SEAGENT Class	Explanation
Role Semantic Web Agent (SWA)	Agent	Both Role and SWA in the metamodel corresponds to the Agent in SEAGENT.
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.
Semantic Service Finder Plan	DiscoverCandidateService	Corresponding SEAGENT entities are Behaviour classes.
Semantic Service Executor Plan	EnactService	
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	

The crucial part of the transformation process is to define transformation rules in a predefined transformation language. Those rules are based on the mappings between source and target model entities. The rules also include formal representation of mapping constraints which are applied during transformation. In our case, we have to define mappings between entities of the interaction metamodel and SEAGENT framework. In Table 1, some of the entity mappings are listed. After execution of the whole transformation process, we achieved platform specific model of our MAS. This output (target) model is given at the end of the following section (in Fig. 4).

4 Application of the Transformation using ATL

We implemented the whole transformation process discussed in this study by using ATLAS INRIA & LINA research group's ATL (Atlas Transformation Language) [12]. ATL is a widely accepted model transformation language, specified as both a metamodel and a textual concrete syntax. It also provides a development environment as a plugin in Eclipse [6]. These advantages cause us to prefer ATL.

Referring to transformation process depicted in Fig. 1, transformation metamodel (TMM) is ATL and source, target and transformation metamodels conform to Ecore metametamodel [6] in our case. Our source model (SM) is the platform independent model given in Fig. 3 which comforms to metamodel given in Fig. 2. Our target metamodel is the metamodel of the Java language [9].

In order to use ATL engine, we need to prepare Eclipse Modeling Framework (EMF) encodings -ecore files- of both metamodels (SMM and TgMM). EMF provides its own file format (.ecore) for model and metamodel encoding. However 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) [11]. This textual notation eases the edition of metamodels. Once edited, KM3 metamodels can be injected into Ecore format using ADT integrated injectors. More information about such injections can be found in [11].

Due to space limitations, it is impossible to give whole KM3 representations and ATL rule definitions of our implementation. To give some flavor of the implementation in here, we describe transformation of the Semantic Web Agent source entity into its corresponding entity in Java based SEAGENT framework.

Following is the part of the KM3 file in which Semantic Web Agent is represented with its associations for Role and Plan entities:

```
class SemanticWebAgent {
       attribute name : String;
       reference apply[0-*] : Plan oppositeOf appliedBy;
       reference play[0-*] : Role oppositeOf playedBy;       }
class Role {
       attribute name : String;
       reference playedBy[0-*] : SemanticWebAgent oppositeOf play;  }
class Plan {
       attribute name : String;
       reference appliedBy [0-*] : SemanticWebAgent oppositeOf apply; }
```

According to the entity mappings, heuristic rules for the transformation should be given in ATL. Each ATL rule for the transformation defines a source model element in its source part and has the full definition of constraints to query the whole source pattern in the model. For instance, the Semantic Web Agent class in the source part of `SemanticWebAgent2Agent` rule needs the full constraint definition of the source pattern to match in the model because the constraint part requires constraints of other source pattern elements related to the Semantic Web Agent class to bind the appropriate model element. The helper rules are required in the constraint part to define the relationships between the pattern elements. Following is the `SemanticWebAgent2Agent` ATL rule:

```
1 rule SemanticWebAgent2Agent {
2      from ag: Agent!SemanticWebAgent(
3             ag.partofPatternforWebAgent  )
4      to c:JAVA!Class (
5          name<- ag.name,
6          associatedClass<-Sequence{ag.executorPlans, ag.finderPlans} )
7 }
```

In rule `SemanticWebAgent2Agent`, we need to call helper rule for the relations of the SemanticWebAgent Class with its role and plan attributes. We also use another rule in order to realize mapping of the SemanticWebAgent class into its corresponding target model entity (a JAVA class in here). The same helper rules and constraint repetitions may be required for other rules in the transformation. Hence this kind of rule decomposition makes the definitions easier. The helper `partofPatternforWebAgent` called in line 3 of the rule `SemanticWebAgent2Agent` is given below:

```
1 helper context Agent!SemanticWebAgent def:
2   partofPatternforWebAgent : Boolean =
3     if not self.oclIsTypeOf(Agent!SemanticServiceMatchmakerAgent)
4      and not self.play.oclIsTypeOf(Agent!RegistryRole)
5      and self.apply->
6           select(p|p.oclIsTypeOf(Agent!SemanticServiceExecutorPlan))->
7             forAll(p|p.execute.owner = p.use.owner)
8      and self.apply->
9           select(p|p.oclIsTypeOf(Agent!SemanticServiceFinderPlan))->
10            forAll(p|p.interact.advertise->
11                  exists(intfc|intfc=p.discover))
12     then true
13     else false
14    endif;
```

The helpers correspond to the constraint part of the related rules. There are two types of helper in our transformations. The first type helpers like `partofPatternforWebAgent` are used to check if the model element is the part of the pattern or not. The second type helpers (e.g. `finderPlans` and `executorPlans`) are used to select the appropriate elements for the associations between target elements within the transformation. Following is the `finderPlans` helper which is called in line 6 of the rule `SemanticWebAgent2Agent`:

```
1 helper context Agent!SemanticWebAgent def:
2   finderPlans : Sequence(Agent!SemanticServiceFinderPlan) =
3     self.apply->select(fp|fp.oclIsTypeOf(
4       Agent!SemanticServiceFinderPlan))->select(fndpln|
5         fndpln.appliedBy->forAll(agnt| not
6           agnt.oclIsTypeOf(Agent!SemanticServiceMatchmakerAgent)
7           and not agnt.play.oclIsTypeOf(Agent!RegistryRole))
8           and fndpln.interact.play.oclIsTypeOf(Agent!RegistryRole)
9           and fndpln.interact.advertise->
10              exists(intfc|intfc.discoveredBy=fndpln)
11          and fndpln.discover.advertisedBy.interactedBy=fndpln);
```

The ecore model conforming to source metamodel includes the following model instance in which the Semantic Web Agent called “Hotel Client Agent” is defined. References to the other instances are omitted.

```
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns="Agent">
   <SemanticWebAgent name="Hotel Client Agent" apply="#/2 #/3" play="#/1" />
   <Role name="Hotel Client Role" playedBy="#/0" />
   <SemanticServiceFinderPlan name="Hotel Client's Service Discovery Plan"
                              appliedBy="#/0" interact="..."  discover="..." />
   <SemanticServiceExecutorPlan name="Hotel Client's Service Invocation Plan"
                              appliedBy="#/0" execute="..." use="..." />
</xmi:XMI>
```

During the transformation process, the ATL engine applies the above rule (SemanticWebAgent2Agent) in order to transform “Hotel Client Agent” into a SEAGENT Agent class. The ecore representation of this obtained target instance is given below. References to the other instances are omitted again:

```
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns="JAVA">
<Class name="Hotel Client Agent" associatedClass="/1 /2"/>
<Class name="Hotel Client's Service Discovery Plan" superClass=".." associatedClass="/0"/>
<Class name="Hotel Client's Service Invocation Plan" superClass=".." associatedClass="/0"/>
</xmi:XMI>
```

After execution of the whole process in ATL environment, we obtained the platform specific (SEAGENT) model of the tourism MAS given in Fig. 4. Each entity of the model is a Java class. Upper part of the model includes the SEAGENT planner components. In the SEAGENT framework, agents execute their tasks according to Hierarchical Task Networks (HTN) [23]. As a requirement of HTN, tasks might be either complex (called behaviors) or primitive (called actions). Tasks have a name describing what they are supposed to do and have zero or more provisions (information needs) and outcomes (execution results). Classes for the tourism MAS take place beneath the agent plan components. Model includes the `Hotel_Client_Agent` that discovers hotel reservation services with semantic capability interfaces according to its `Hotel_Client_Service_Discovery_Plan`. It communicates with the `Matchmaker_Agent` of the system during execution of this plan. Discovery plan extends `DiscoverCandidateService` behavior. This behavior is the corresponding entity for the “Semantic Service Finder Plan” meta-entity given in our PIM. Similarly, agent’s service execution plan

(`Hotel_Client_Service_Invocation_Plan`) is an `EnactService` behavior and is the counterpart of our PIM's "Semantic Service Executor Plan" meta-entity. Semantic web services are OWL-S services in SEAGENT. Hence, our reservation service is a subclass of `OWL_S_Service` class after the transformation as expected.

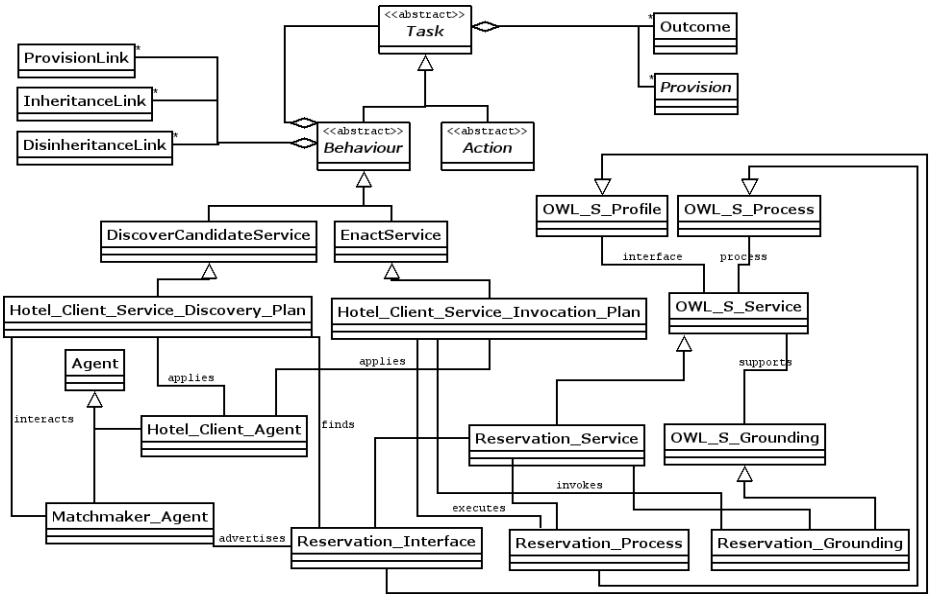

Fig. 4: The target MAS model obtained after the transformation between PIM and PSM.

5 Related Work

Recently, model driven approaches have been recognized and become one of the major research topics in agent oriented software engineering (AOSE) community. As briefly mentioned below, some of the studies intend to apply the whole MDD process for MAS development while some of them only utilize either metamodels or model transformation as needed. Conceptual MDA definitions and study on MDA based MAS research directions are also discussed in some of the studies e.g. [1] [7]. Bauer and Odell discuss the use of UML 2.0 and MDA for agent-based systems in [1]. They also discuss which aspects of a MAS could be considered at CIM and PIM. The Cougaar MDA discussed in [7] provides a higher application composition for agent systems by elevating the composition level from individual components to domain level model specifications in order to generate software artifacts. Jayatilleke et al. [10] provide a toolkit for their conceptual framework of domain independent component types in order to make their approach consistent with MDD and use agent models to generate executable codes.

On the other hand, the study defined in [19] is a good example that applies the transformation process of MDA which is depicted in Fig. 1. In that study, Perini and

Susi [19] use TEFKAT model transformation language [5] to implement the transformation process in automating conversions from their methodology structures to UML models. In [17], Pavon and his friends reformulate 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. A similar MAS methodology revision is discussed in [18]. Ideas and standards from MDA are adopted both in refining the modeling process algorithm and building tools within this study.

Regarding all of the above studies, it can be said that current application of the MDD on MAS development is in its preliminary phase. Neither a complete MDD process nor a common MAS metamodel has been developed. On the other hand, Semantic Web [2] technology and its required constructs on MASs are not supported within those studies. We believe this shortage in question is crucial when development of future MASs is considered. Therefore providing a Semantic Web enabled MDD process for MAS development is the key difference between our study and those previous studies.

6 Conclusion and Future Work

A model transformation process for the model driven development of Semantic Web enabled MASs is discussed in this paper. The study in here presents description of a whole process in which the source and the target metamodels, entity mappings and the implementation of the transformation for a real MAS framework are all included.

In fact, our aim is to enhance this study by providing code generation (at least in template level) for Semantic Web enabled MAS implementations. That means a MAS developer just creates a model of the MAS conforming to the platform independent model and then chooses the desired physical implementation environment (e.g. SEAGENT) for the system. Finally, our tool generates template codes for the developer by using target environment's metamodel, model and transformations. The developer completes the software for the full deployment of the system. Therefore, in addition to improvement studies on model transformation (e.g. elaborating mappings in entity attribute level, clarifying input/output and precondition/effect representations of semantic web service entities on the model), we are currently working on code generation from target models we gained. We intend to employ a source code generator such as JET (Java Emitter Templates) Engine [6] in order to generate platform specific MAS software as the final product of our MDD process.

References

[1] Bauer, B., Odell, J.: UML 2.0 and Agents: How to Build Agent-based Systems with the New UML Standard. Eng. Appl. Artif. Intel., 18(2), 141-157 (2005)

[2] Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web, Sci. Am., 284(5), 34-43, (2001)
[3] Dikenelli, O., Erdur, R.C., Kardas, G., Gümüs, Ö., Seylan, I., Gürcan, Ö., 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)
[4] Djuric, D.: MDA-based Ontology Infrastructure. Comput. Sci. Inf. Syst. 1(1), 91-116, (2004)
[5] Duddy, K., Gerber A., Lawley, M.: Model Transformation: A declarative, reusable patterns approach. In: 7th International Enterprise Distributed Object Computing Conference, pp. 174-185. IEEE Computer Society (2003)
[6] Eclipse Open Development Platform, http://www.eclipse.org
[7] Gracanin, D., Singh, H.L., Bohner, S.A., Hinchey, M.G.: Model-Driven Architecture for Agent-Based Systems. In: Hinchey, M.G., Rash, J.L., Truszkowski, W., Rouff, C. (eds.) FAABS 2004. LNCS (LNAI), vol. 3228, pp. 249-261, Springer, Heidelberg (2005)
[8] 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)
[9] Java Metamodel, http://www.eclipse.org/gmt/am3/zoos/atlanticUMLZoo/#JAVA
[10] Jayatilleke, G.B., Padgham, L., and Winikoff, M.: A Model Driven Development Toolkit for Domain Experts to Modify Agent Based Systems. In: Padgham, L., Zambonelli, F. (eds.) AOSE 2006. LNCS, vol. 4405, Springer, Heidelberg (2007)
[11] 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)
[12] Jouault, F., Kurtev, I.: Transforming Models with ATL. In: Bruel, J. (eds.) MoDELS Satellite Events 2005. LNCS, vol. 3844, pp. 128-138, Springer, Heidelberg (2006)
[13] Kardas, G., Goknil, A., Dikenelli, O., Topaloglu, N.Y.: Metamodeling of Semantic Web Enabled Multiagent Systems. In: Multiagent Systems and Software Architecture 2006, pp. 79-86 (2006)
[14] Kardas, G., Goknil, A., Dikenelli, O., Topaloglu, N. Y.: Modeling the Interaction between Semantic Agents and Semantic Web Services using MDA Approach. In: O'Hare, G., et al. (eds.) ESAW 2006, LNCS (LNAI), vol. 4457, pp. 209-228, Springer, Heidelberg (2007)
[15] OMG Specifications, http://www.omg.org
[16] Odell, J., Levy, R., Nodine M.: FIPA Modeling TC: Agent Class Superstructure Metamodel, http://www.omg.org/docs/agent/04-12-02.pdf
[17] 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)
[18] Penserini, L., Perini, A., Susi, A., Mylopoulos, J.: From Stakeholder Intentions to Software Agent Implementations. In: Dubois, E., Pohl, K. (eds.) CAiSE 2006. LNCS, vol. 4001, pp. 465-479, Springer, Heidelberg (2006)
[19] Perini, A., Susi, A.: Automating Model Transformations in Agent-Oriented Modeling. In: Müller, J.P., Zambonelli, F. (eds.) AOSE 2005. LNCS, vol. 3950, pp. 167-178, Springer, Heidelberg (2006)
[20] Selic, B.: The Pragmatics of Model-Driven Development. IEEE Software 20, 19-25 (2003)
[21] The Semantic Markup for Web Services (OWL-S), http://www.daml.org/services/owl-s
[22] Web Service Modeling Ontology, http://www.wsmo.org/
[23] Williamson, M., Decker, K., Sycara, K.: Unified Information and Control Flow in Hierarchical Task Networks. In: AAAI-96 Workshop, pp. 142-150 (1996)