Towards a DSML for Semantic Web enabled Multi-agent Systems

Geylani Kardas
International Computer Institute
Ege University
35100 Bornova, Izmir, TURKEY
Tel: +90-232-3423232-103
geylani.kardas@ege.edu.tr

Zekai Demirezen
Department of Computer and Information Sciences
University of Alabama at Birmingham
35294-1170 Birmingham, AL, USA
zekzek@uab.edu

Moharram Challenger
International Computer Institute
Ege University
35100 Bornova, Izmir, TURKEY
Tel: +90-232-3423232-119
challenger@engineer.com

ABSTRACT
Software agents are considered as autonomous software components which are capable of acting to meet its design objectives. To perform their tasks and interact with each other, agents constitute systems called Multi-agent systems (MAS). Although agent researchers have a great effort in MAS metamodeling and model-driven MAS development, a significant deficiency exists in current studies when we consider providing a complete Domain Specific Modeling Language (DSML) for MASs. We believe that a DSML increases the descriptive power of a MAS metamodel, defines the system semantics and hence supports a more fruitful methodology for the development of MASs especially working on the new challenging environments such as the Semantic Web. In this paper, we introduce a new DSML for MASs with its abstract syntax, the textual concrete syntax and the interpreter mechanism. The practical use of the DSML is illustrated with a case study which considers the modeling of a multi-agent based e-barter system.

Categories and Subject Descriptors
D.3.1 [**Programming Languages**]: Formal Definitions and Theory – *semantics, syntax* I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence – *Multiagent systems*

General Terms
Algorithms, Design, Languages.

Keywords
Domain Specific Modeling Language, Metamodel, Model-Driven Engineering, Multi-agent System, Semantic Web.

1. INTRODUCTION
Software agents are considered as autonomous software components which are capable of acting on behalf of their human users in order to perform a group of defined tasks. Many intelligent software agents interact with each other in a system that we call Multi-agent Systems (MASs). The implementation of these autonomous, responsive and proactive software systems is naturally a complex task. In addition, internal agent behavior model and interaction within the agent organizations become even more complex and hard to implement when new requirements and interactions for new agent environments such as the Semantic Web [1] 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 MASs due to their internal complexity, distributedness and openness. Within this context, Model-Driven Engineering (MDE) [2] may provide an infrastructure that simplifies the development of MASs.

MDE has been shown to increase productivity and reduce development costs [3]. It provides higher levels of abstraction to allow such users to focus on the problem, rather than the specific solution [2]. In particular, domain-specific modeling (DSM) is a modeling approach that provides languages, called Domain Specific Modeling Languages (DSMLs), that fit the domain of an end-user by offering intentions, abstractions, and visualizations for domain concepts [4]. Therefore, the part of the success of MDE is also dependent on the descriptive power of DSMLs [5].

Although there exists a great effort of agent researchers in MAS metamodeling (e.g., [6, 7]) and model-driven MAS development (e.g., [8, 9]), a significant deficiency exists in current studies when we consider providing a complete DSML for MASs. In our previous work, we also defined an agent metamodel [10] and have recently presented a complete MDE process [11] for rapid implementation of MASs on various agent software platforms. The proposed metamodel especially supports the Semantic Web constructs and their interactions with the traditional agent system components to provide MDE of the Semantic Web enabled MASs. However, similar to the above referenced studies, both formal specification and descriptive power of the given metamodel are not sufficient enough for the definition of the system semantics and the verification of the MAS models conforming to our proposed metamodel. This is a fundamental requirement especially when we consider dynamic MAS models describing agent behaviors and interactions both with the other agents and the semantic web services. We believe that the definition of a new DSML for such MASs would remove above discussed shortages originating from the existing metamodel and enable agent developers to use a more fruitful MDE methodology for the development of MASs especially working on the new Semantic Web environment. Hence, in this paper we present the initial results of our ongoing study on defining a new DSML that can be used during the model-driven MAS development.

Conference'10, Month 1–2, 2010, City, State, Country.

Based on a revised MAS metamodel, we derive an abstract syntax and a textual concrete syntax for the proposed DSML. We also define an interpreter mechanism that provides the automatic generation of program codes for various MAS development frameworks. These components of the language are discussed in the paper with including their use in the modeling of a multi-agent based e-barter system.

The rest of the paper is organized as follows: The abstract syntax, the concrete syntax and the interpreter mechanism of the proposed DSML are discussed in Sections 2, 3 and 4. Section 5 includes a case study on the development of a MAS by using the DSML. Related work is given in Section 6. Section 7 concludes the paper with a brief discussion of the study.

2. THE ABSTRACT SYNTAX

It is well-known that the abstract syntax of a language describes the vocabulary of concepts provided by the language and how they may be combined to form models or programs. It consists of a set of provided concepts and their relationships to other concepts. A metamodel that describes the meta-entities and their relationships for a domain can naturally provide a base for the definition of such an abstract syntax. For this reason we have revised and extended our MAS metamodel introduced in [11] according to new requirements and provided concepts and their attributes for the abstract syntax of the new DSML.

We call the new modeling language as *SEA_ML (Semantic web Enabled Agent Modeling Language*) that supports modeling of Semantic Web enabled MASs. In our vision, the "*Semantic Web enabled MAS*" means that software agents are planned to collect Web content from diverse sources, process the information and exchange the results on behalf of their human users. Autonomous agents can also evaluate semantic data within these MASs and collaborate with semantically defined entities such as semantic web services by using content languages. We call the software agents with these capabilities as *Semantic Web Agent*s.

Our metamodel for the Semantic Web enabled MASs considers various aspects of MAS development (e.g., behavioral, organizational and protocol) and provides both internal modeling of a software agent and interaction of agents and semantic web services within the environment. With more than 50 concepts and 80 relations, the exact metamodel is too big to completely discuss in this paper. Interested readers may find the whole metamodel in [11]. Here, we can only discuss one of the major parts of the metamodel; the service interaction viewpoint of the metamodel in which the interaction of agents and the semantic web services is modeled. Figure 1 depicts this partial metamodel.

A *Semantic Web Agent* is an autonomous entity which is capable of interaction with both other agents and semantic web services within the environment. They play roles and use ontologies to maintain their internal knowledge and infer about the environment based on the known facts. Semantic Web Agents can be associated with more than one *Role* at the same point in time and can change roles over time. Task definitions and related task execution processes of the Semantic Web agents are modeled inside the *Behavior* entities.

A *Semantic Web Service* represents any service (except agent services) whose capabilities and interactions are semantically described. It should be noted that the association between the semantic web agents and the services is provided over the agent role entities in the metamodel. Because agents interact with semantic web services depending on their roles defined inside the MAS organization.

An ontology represents any information gathering and reasoning resource for MAS members. Collection of the ontologies creates the knowledgebase of the MAS that provides domain context. Specializations of the ontology called *Role Ontology* and *Service Ontology* are utilized by the Semantic Web Agents and Semantic Web Services respectively. Because the Web Ontology Language (OWL) is now both the premier and the W3C standard semantic markup language for publishing and sharing ontologies, the revised metamodel define ontology entities as extensions of the *OWLOntology* concept defined in the OMG's Ontology Definition Metamodel (ODM) [12].

Semantic web service modeling approaches (e.g., OWL-S [13]) mostly describe services by three semantic documents: Service Interface, Process Model and Physical Grounding. Service Interface is the capability representation of the service. Process Model describes internal composition of the service. Finally, Physical Grounding defines invocation protocol of the service. These Semantic Web Service components are given in our metamodel with *Interface*, *Process* and *Grounding* entities respectively. *Input*, *Output*, *Precondition* and *Effect* definitions used by the Semantic Web Service components are also defined in the metamodel. The revised metamodel inherits *OWLClass* meta-entity from ODM as the base class for these semantic properties.

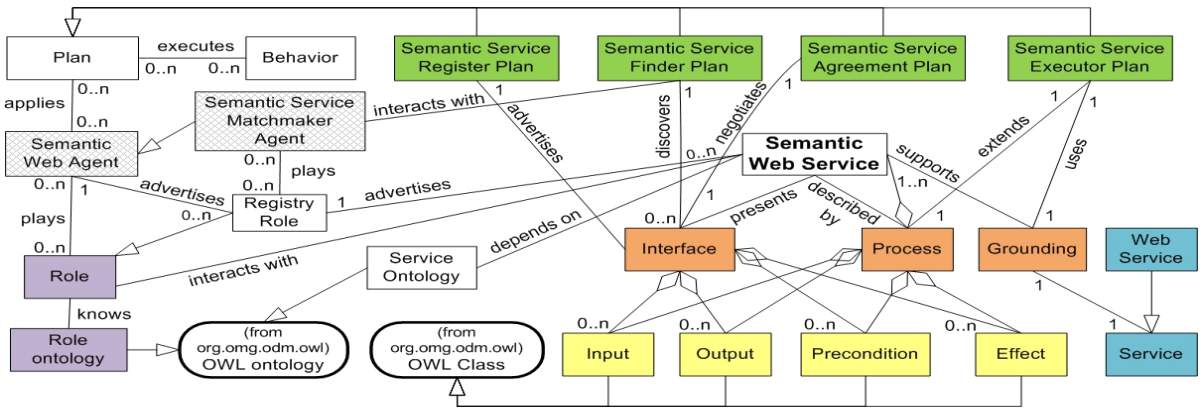

Figure 1. The partial MAS metamodel for agent-semantic web service interaction.

Semantic Web Agents apply *Plan*s to perform their tasks. In order to discover, negotiate and execute Semantic Web Services dynamically, three extensions of the Plan entity are defined in the metamodel. *Semantic Service Finder Plan* is a Plan in which discovery of candidate semantic web services takes place. *Semantic Service Agreement Plan* involves the negotiation on QoS metrics of the service (e.g., service execution cost, running time, location) and agreement settlement. After service discovery and negotiation, the agent applies the *Semantic Service Executor Plan* for executing appropriate semantic web services.

On the other hand, agents need to communicate with a service registry in order to discover service capabilities. For this reason, the metamodel includes a specialized agent entity, called *Semantic Service Matchmaker Agent*. This meta-entity represents the matchmaker agents that store the capability advertisements of semantic web services within a MAS and match those capabilities with service requirements sent by the other platform agents.

Based on the above discussed metamodel, the abstract syntax of the SEA_ML has been defined by using the Kernel MetaMetaModel (KM3) [14]. In addition to neat presentation of our abstract syntax, the utilization of the KM3 notation also enables us to employ our MAS metamodel in various model-to-model transformations. Listing 1 shows an excerpt taken from the abstract syntax of the SEA_ML defined in KM3 notation. The excerpt includes definitions of the SemanticWebAgent and SemanticWebService concepts with their relations.

```
class SemanticWebAgent {
  attribute name: String;
  ...
  reference apply[0-*]: Plan oppositeOf appliedBy;
  reference play: Role oppositeOf playedBy;
  reference advertisedBy[0-1]: RegistryRole
oppositeOf advertiseAgent;
}
...
class SemanticWebService {
  attribute name: String;
  ...
  reference interface: Interface oppositeOf owner;
  reference process: Process oppositeOf owner;
  reference grounding: Grounding oppositeOf owner;
  reference depend[1-*]: ServiceOntology
     oppositeOf dependedBy;
  reference advertisedBy[0-1]: RegistryRole
     oppositeOf advertiseService;
}
```

Listing 1. An excerpt from the abstract syntax of the SEA_ML

3. THE CONCRETE SYNTAX

While specification of abstract syntax includes the concepts that are represented in the language and the relationships between those concepts, concrete syntax definition provides a mapping between meta-elements and their textual or graphical representations. SEA_ML concrete syntax enables the agent programmers to specify their programs textually. The main objective is the concise and precise specifications and comprehensible programs even for non-programmers. SEA_ML concrete syntax consists of syntactic constructs that represent the agent concepts. These constructs enable to translate textual agent programs to their equivalent instance models. Textual Concrete Syntax (TCS) [14] is used to develop SEA_ML's concrete syntax. Listing 2 shows an excerpt from the TCS specification of SEA_ML. A domain user can specify a Semantic Web agent with the role it plays. Furthermore, he can define agent plans (e.g., SemanticServiceFinder and SemanticServiceExecutor) and OWL input/output classes for service interaction within a SemanticWebAgent specification. Keywords, provided as a part of SEA_ML concrete syntax, enable the domain user to specify all these details in a comprehensible way

```
template SemanticWebAgent main context:
 "SemanticWebAgent" name "plays" plays {
  seperator=";"}
  "knows OWLOntology"  knows { seperator=";"}
  ["SemanticServiceFinderPlan" |
  "SemanticServiceExecutorPlan"] applies
  { seperator=";"}  ;
  template Plan abstract;
  template RoleOntology addToContext: name;
  template SemanticServiceFinderPlan addToContext:
    name "discovers" interface
   "{"
      OWLClass" input  "OWLClass" output
     …
   "}"
```

Listing 2. An excerpt from the TCS of the SEA_ML

4. THE INTERPRETER

It is not sufficient to complete the DSML definition only by specifying the notions and their representations. The complete definition requires that one provide semantics of language concepts in terms of other concepts whose meaning is already established. Therefore, the abstract syntax of the SEA_ML is mapped into the metamodels of existing agent platforms (such as NUIN [15]) that have well-defined and understood semantics. The mapping is achieved through model transformations. One of the target agent platforms we use is the NUIN platform which is a Java framework for building belief-desire-intention (BDI) agents [16], with a particular emphasis on Semantic Web agents. We provided the model transformation rules based on the mappings between the SEA_ML and NUIN concepts. Transformation rules are written by using the AtlanMod Transformation Language (ATL) [14]. Listing 3 presents an example ATL rule for transforming Semantic Web Agents into the Agent concept of the NUIN platform. This ATL rule, called *SemanticWebAgent2NUINAgent*, maps name, contain, and believe attributes of a NUIN Agent construct with the equivalent concepts in the SEA_ML language. Determination of the Semantic Web Agent instance and its plan components in the source pattern are realized by three helper rules called *partofPatternforWebAgents*, *executorPlans* and *finderPlans*. The context of these helpers is not given here due to the space limitations.

```
create OUT : NUIN from IN : Agent;
rule SemanticWebAgent2NUINAgent {
  from
    ag: Agent!SemanticWebAgent (
      ag.partofPatternforWebAgent )
     to
       na: NUIN!NUINAgent (
            name <- ag.name,
           contain <- Sequence{ag.executorPlans,
                               ag.finderPlans},
            believe <- ksdec),
       ksdec: NUIN!KnowledgestoreDeclaration
}
```

Listing 3. An example ATL rule for transforming Semantic Web Agents into NUIN Agents

Moreover, considering the NUIN framework, we also provided a model to text transformation in order to generate Nuinscripts [15] (program codes for NUIN Agents) from the MAS models conforming to NUIN metamodel. Agent programs written in Nuinscripts are parsed into the Java objects used by the NUIN platform. Hence, generated Nuinscripts are the executable artifacts of the SEA_ML language.

5. CASE STUDY

In order to illustrate the use of the introduced language, consider the modeling of a simple multi-agent based e-barter system. A barter system is an alternative commerce approach where customers meet at a marketplace in order to exchange their goods or services without currency. In barter marketplaces, the amount of purchased goods or services are paid by manufactured goods or offered services. An agent-based e-barter system consists of agents that exchange goods or services of owners corresponding to their preferences. The *Customer* agents are responsible for adding and evaluating barter proposals. The *Barter Manager* agent manages all trades in the system. This agent is responsible for collecting barter proposals, matching proper barter proposals and tracking the bargaining process between Customer agents. In such an e-barter system, suppose that a Barter Manager agent needs to interact with semantic web services to match bidden and demanded goods and determine the value of the exchange. For instance, two customer agents (one from the automotive industry and other from the healthcare sector) may need to exchange their offered goods and services such that: A car manufacturer offers to sell car spare parts to a health insurance company (e.g., for company's service cars) and wants to procure health insurance for its employees. Consider that the intention of the health insurance company is vice versa.

During the bargain between the agents of the car manufacturer and the health insurance company, our Barter Manager agent may use a semantic web service called *Barter Service*. In order to invoke that service, Barter Manager first needs to discover the proper semantic web service. Then, Barter Manager interacts with the candidate service(s) and after an agreement, the exact execution of the semantic web service is realized. Figure 2 portrays the instance MAS model conforming to the metamodel discussed in Section 2.

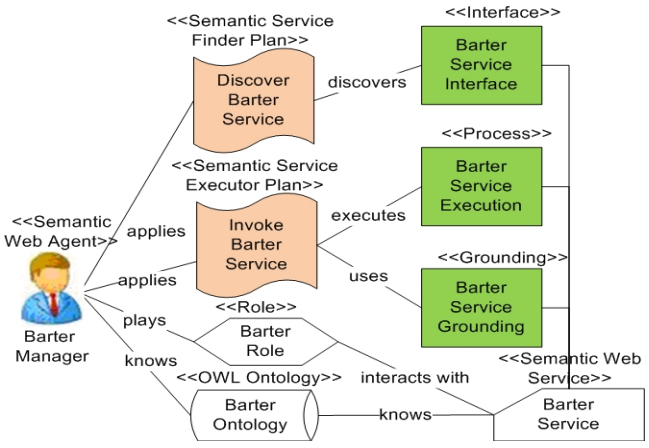

Figure 2. An instance agent model for the e-barter system conforming to the metamodel of SEA_ML

The program code of the Barter Manager agent can be provided according to the concrete syntax defined in TCS of the SEA_ML. Barter Manager plays the Barter Role and applies two plans called *discoverBarterService* and *invokeBarterService* which compose tasks for the interaction with a semantic web service. The agent first searches for a semantic web service which can match a "*Car_Spare*" OWL concept with a "*Health_Insurance*" OWL concept and then execute the service to find the counterpart of a bargained car spare part: an OWL individual for BMW 520 Tyre. Listing 4 shows an excerpt from the program code of the Barter Manager agent written according to the concrete syntax of SEA_ML discussed in Section 3. The code also includes the appropriate comments.

```
SemanticWebAgent BarterManager plays BarterRole
 //references for the environment members
 ref: SemanticServiceMatchmakerAgent BarterServiceMatchmaker;
 ref: SemanticWebService BarterService;
 //definitions for the semantic web service access
 interacts_with BarterService {
          //service interaction definitions
          Interface BarterServiceInterface;
          Process BarterServiceExecution;
          Grounding BarterServiceGrounding;
 }
 //knowledgebase definitions
 knows OWLOntology BarterOntology;
 //plans to achieve agent goals
 SemanticServiceFinderPlan discoverBarterService discovers
    BarterServiceInterface {
   ...
 }
 SemanticServiceExecutorPlan invokeBarterService executes
    BarterServiceExecution uses BarterServiceGrounding {
  //access execution mechanism and invocation protocol of the service
  BarterServiceExecution ←
                  BarterServiceInterface->presentedBy->describedBy;
  BarterServiceGrounding ←
                   BarterServiceExecution->describes->supportedBy;
  //set execution parameters
  input->value ←BarterOntology->getOWLIndividual("BMW520Tyre");
  //invoke the service
  output->value ← BarterServiceGrounding->callOperation(input);
 }
```

Listing 4. An excerpt from the program code of the Barter Manager agent in SEA_ML concrete syntax

Now consider the implementation of the related e-barter system on the NUIN platform. The interpreter of SEA_ML first applies an ATL transformation onto the e-barter instance model (pictured in Figure 2) and outputs the NUIN counterpart of that source model (target model). Then agent program codes (Nuinscripts for this case) are automatically generated by applying a model-to-text transformation on the NUIN target model. Interested readers may refer to [11] for this transformation and auto-generated sample NUIN codes.

6. RELATED WORK

Studies on DSMLs for agents are recently emerging and those very few studies are in their preliminary states. For instance, a domain specific language called Agent-DSL is introduced in [17]. Agent-DSL is used to specify the agency properties that an agent could have to accomplish its tasks. However, the proposed DSL is presented only with its metamodel and provides just the visual

modelling of the agent systems. Likewise in [18], Rougemaille et al. introduce two dedicated modeling languages and call those languages as DSMLs. The languages are described by metamodels which can be seen as representations of the main concepts and relationships identified for each of the particular domains introduced in [18]. In fact, the study only defines generic agent metamodels for MDE of MASs. Hahn [19] introduces a DSML for MAS. The abstract syntax of the DSML is derived from a platform independent metamodel which is structured into several aspects each focusing on a specific viewpoint of a MAS. In order to provide the concrete syntax, the appropriate graphical notations for the concepts and relations are defined. The semantics of the language is also given. This study is noteworthy because it seems to be the first complete DSML for agents with all of its specifications. However, it supports neither the agents on the Semantic Web nor the interaction of Semantic Web enabled agents with other environment members such as semantic web services. Our study contributes to aforementioned efforts by specializing on the Semantic Web support of the MASs.

7. DISCUSSION AND FUTURE WORK

An abstract syntax, a concrete syntax and an interpreter mechanism for a DSML for MAS development is introduced. We examine that the defined syntax provides a clear and a formal definition of both inner plan and communication structures of software agents especially considering their interactions with semantic web services on the Semantic Web. The interpreter of the proposed DSML with its model transformation and code generation capabilities enables the implementation of the modeled MASs according to various agent development frameworks.

Due to the lack of formal semantics techniques for DSMLs, the real meaning of a modeling language is available only in associated model interpreters. In our DSML, part of the MAS semantics is hardcoded within ATL transformation rules. Agent programmers need to provide the rest of the semantics by inserting new codes within generated codes (e.g., Nuinscripts). All these scattered semantics specifications cause problems during system maintenance, testing and analyze stages. Moreover, specification of semantics by transformation rules is very demanding to correctly map the constructs of the DSML into the constructs of the target language. The underlying reason is the mappings which are not at the same level of abstraction.

We may conclude that the DSM environments should have a formal foundation:

- that can be used to define the dynamic semantics of a modeling language,
- that can enable to interoperate with analysis tools and verify the correctness of a model
- that can be used to automate the construction of modeling tools.

Although we provide the current semantics of SEA_ML by mapping its abstract syntax into the metamodels of existing agent platforms, our next work will be the formal representation of the semantics by using a formal notation language. This enables us to analyze and verify the SEA_ML programs.

8. ACKNOWLEDGMENTS

This study is partially funded by The Scientific and Technological Research Council of Turkey (TUBITAK) under grant 109E125.

9. REFERENCES

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

[2] Schmidt, D.C. 2006. Guest Editor's Introduction: Model-Driven Engineering. IEEE Computer. 39 (2), 25-31.

[3] Vallecillo, A. 2008. A Journey through the Secret Life of Models. In: Model Engineering of Complex Systems (MECS), Dagstuhl Seminar Proceedings.

[4] Sprinkle, J., Mernik, M., Tolvanen, J.-P., and Spinellis, D. 2009. Guest Editors' Introduction: What Kinds of Nails Need a Domain-Specific Hammer?. IEEE Software. 26(4), 15-18.

[5] Gray, J., Tolvanen, J-P., Kelly, S., Gokhale, A., Neema, S., and Sprinkle, J. 2007. Domain-Specific Modeling. In Handbook of Dynamic System Modeling, CRC Press, 1-7.

[6] Bernon, C., Cossentino, M., Gleizes, M-P., Turci, P., and Zambonelli, F. 2005. A Study of some Multi-Agent Meta-Models. Lect. Notes Comput. Sc. 3382, 62-77.

[7] Hahn, C., Madrigal-Mora, C., and Fischer, K. 2009. A platform-independent metamodel for multiagent systems. Auton. Agent. Multi-Ag. 18(2), 239-266.

[8] Gracanin, D., Singh, H.L., Bohner, S. A., and Hinchey, M. G. 2005. Model-Driven Architecture for Agent-Based Systems. Lect. Notes Artif. Int. 3228, 249-261.

[9] Pavon, J., Gomez-Sanz, J.J. and Fuentes, R. 2006. Model Driven Development of Multi-Agent Systems. Lect. Notes Comput. Sc. 4066, 284-298.

[10] Kardas, G., Goknil, A., Dikenelli, O., and Topaloglu, N.Y. 2007. Modeling the Interaction between Semantic Agents and Semantic Web Services using MDA Approach. Lect. Notes Artif. Int. 4457, 209-228.

[11] Kardas, G., Goknil, A., Dikenelli, O., and Topaloglu, N.Y. 2009. Model Driven Development of Semantic Web Enabled Multi-agent Systems. Int. J. Coop. Inf. Syst. 18(2), 261-308.

[12] Object Management Group. 2009. Ontology Definition Metamodel. http://www.omg.org/spec/ODM/1.0/

[13] OWL-S Coalition. 2004. OWL-S: Semantic Markup for Web Services. http://www.daml.org/services/owl-s/1.1/overview/

[14] Kurtev, I., Bezivin, J., Jouault, F., and Valduriez, P. 2006. Model-based DSL Frameworks. In Proceedings of the 21st symposium on Object-oriented Programming, Systems, Languages, and Applications. ACM Press, 602-615.

[15] NUIN Agent Framework. http://www.nuin.org/

[16] Rao, A., and Georgeff, M. 1995. BDI Agents: From Theory to Practice, In Proceedings of the 1st International Conference on Multi-Agent Systems (ICMAS-95), 312-319.

[17] Kulesza, U, Garcia, A., Lucena, C., and Alencar, P. 2005. A Generative Approach for Multi-agent System Development. Lect. Notes Comput. Sc. 3390, 52-69.

[18] Rougemaille, S., Migeon, F., Maurel, C., and Gleizes, M-P. 2007. Model Driven Engineering for Designing Adaptive Multi-Agent Systems. Lect. Notes Artif. Int . 4995, 318-33.

[19] Hahn, C. 2008. A Domain Specific Language for Multiagent Systems. In Proceedings of the 7th Autonomous Agents and Multiagent Systems Conf. AAMAS'08. ACM Press, 233-240