Proceedings of the Federated Conference on Computer Science and Information Systems pp. 1555–1564
DOI: 10.15439/2016F196
ACSIS, Vol. 8. ISSN 2300-5963

Interoperability of MAS DSMLs via horizontal model transformations

Emine Bircan
International Computer Institute,
Ege University, 35100, Bornova,
Izmir, Turkey
eminebircanbircan@gmail.com

Moharram Challenger
International Computer Institute,
Ege University, 35100, Bornova,
Izmir, Turkey
moharram.challenger@mail.ege.edu.tr

Geylani Kardas
International Computer Institute,
Ege University, 35100, Bornova,
Izmir, Turkey
geylani.kardas@ege.edu.tr

***Abstract*—In this paper, we present our approach which aims at improving the mechanism of constructing language semantics over the interoperability of domain-specific modeling languages (DSMLs) developed for Multi-agent Systems (MAS) and hence providing a more efficient way of extension for the executability of modeled agent systems on various underlying agent platforms. Differentiating from the existing MAS DSML studies, our proposal is based on determining entity mappings and building horizontal model transformations between the metamodels of MAS DSMLs which are in the same abstraction level. The applicability of the approach is demonstrated in the paper by constructing horizontal transformations between two full-fledged agent DSMLs, called SEA_ML and DSML4MAS. Use of these transformations has enabled SEA_ML instance models now to be executable on new agent platforms and that feature has been provided with less effort comparing with the implementation of needed transformations between SEA_ML and those new agent platforms from scratch.**

***Keywords*—Metamodel; Model transformation; Domain-specific Modeling Language; Multi-agent System**

I. Introduction

Multi-agent systems (MASs) are those systems having software agents within an environment where agents interact to solve problems in a competitive or collaborative manner. In MASs, software agents are expected to be autonomous, mostly through a set of reactive/proactive behaviors designed for addressing situations likely to happen in particular domains [1]. Both internal agent behavior model and interactions within a MAS become even more complex and hard to implement when taking into account the varying requirements of different agent environments [2]. Hence, working 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 [3].

In order to master the abovementioned problems of developing MASs, agent-oriented software engineering (AOSE) researchers define various agent metamodels (e.g. [4-7]), which include fundamental entities and relations of agent systems. In addition, many model-driven agent development approaches are provided such as [8-10] and by enriching MAS metamodels with some defined syntax and semantics (usually translational semantics [11]), researchers also propose domain-specific languages (DSLs) / domain-specific modeling languages (DSMLs) (e.g. [12-20]) for facilitating the development of MASs. DSLs / DSMLs [21-23] have notations and constructs tailored toward a particular application domain (e.g. MAS) and help to the model-driven development (MDD) of MASs. MDD aims to change the focus of software development from code to models [24], and hence many AOSE researchers believe that this paradigm shift introduced by MDD may also provide the desired abstraction level and simplify the development of complex MAS software [3].

In AOSE, perhaps the most popular way of applying model-driven engineering (MDE) for MASs is based on providing DSMLs specific to agent domain with including appropriate integrated development environments (IDEs) in which both modelling and code generation for system-to-be-developed can be performed properly. Proposed MAS DSMLs such as [13], [17], [19] usually support modelling both the static and the dynamic aspects of agent software from different MAS viewpoints including agent internal behaviour model, interaction with other agents, use of other environment entities, etc. Within this context, abstract syntaxes of the languages are represented with metamodels covering those aspects and required viewpoints to some extent. Following the construction of abstract and concrete syntaxes based on the MAS metamodels, the operational semantics of the languages are provided in the current MAS DSML proposals by defining and implementing entity mappings and model-to-model (M2M) transformations between the related DSML's metamodel and the metamodel(s) of popular agent implementation and execution platform(s) such as JACK[1], JADE[2] and JADEX[3]. Finally, a series of model-to-text (M2T) transformations are implemented and applied on the outputs of the previous M2M transformations which are the MAS models conforming to the related agent execution platforms. Hence, agent software codes, MAS configuration files, etc. pertaining to the implementation and deployment of the modeled agent systems on the target MAS platform are generated automatically.

When we take into account the different abstractions covered by the metamodels of MAS DSMLs and the underlying agent execution platforms, DSML metamodels can be accepted as the platform-independent metamodels (PIMMs) of agent systems while metamodels of the agent execution platforms are platform-specific metamodels

This study is funded by the Scientific Research Projects Directorate of Ege University under grant 16-UBE-001.

[1] http://aosgrp.com/products/jack/ (last access: June, 2016)
[2] http://jade.tilab.com/ (last access: June, 2016)
[3] https://www.activecomponents.org/ (last access: June 2016)

(PSMMs) according to the OMG's well-known Model-driven Architecture (MDA)[4] as also indicated in [5] and [9].

Above described methodology applied in the current MAS DSML development approaches for the derivation of operational semantics unfortunately requires the definition and implementation of new M2M and M2T transformations from scratch in order to make the DSMLs functional for different agent execution platforms. In other words, for each new target agent execution platform, MAS DSML designers should repeat all the time-consuming and mostly troublesome steps of preparing the vertical transformations between the related DSML and this new agent platform.

Motivated by the similarity encountered in the abstract syntaxes of the available MAS DSMLs, we are quite convinced that both the definition and the implementation of M2M transformations between the PIMMs of MAS DSMLs would be more convenient and less laborious comparing with the transformations required between MAS PIMMs and PSMMs in the way of enriching the support of MAS DSMLs for various agent execution platforms. Hence, in this paper, we present our approach which aims at improving the mechanism of constructing language semantics over the interoperability of MAS DSMLs and hence providing a more efficient way of extension for the executability of modeled agent systems on various underlying agent platforms. Differentiating from the existing MAS DSML studies (e.g. [13], [16], [17], [19], [20]), our proposal is based on determining entity mappings and building horizontal M2M transformations between the metamodels of MAS DSMLs which are in the same abstraction level. In this paper, we also investigate the applicability of the proposed DSML interoperability approach by constructing horizontal transformations between two full-fledged agent DSMLs called SEA_ML [19] and DSML4MAS [5] respectively.

The rest of the paper is organized as follows: In Sect. 2, the approach for the MAS DSML interoperability is presented. Applicability of the approach is discussed in Sect. 3 by taking into consideration two MAS DSMLs. In Sect. 4, a case study on the development of an agent-based stock exchange system with using the proposed approach is given. Related work is given in Sect. 5. Finally, Sect. 6 concludes the paper and states the future work.

II. Proposed Approach for the Interoperability of MAS DSMLs

As indicated in the introduction, support of current MAS DSMLs for each agent execution platform is enabled by repetitively defining and implementing a chain of vertical M2M and M2T transformations. Available M2M and M2T transformations are specific for each different agent platform and almost all of them can not be re-used while extending the executability of the MAS models for a new agent platform. Due to the difficulty encountered on repeating those vertical model transformation steps, current MAS DSML proposals (e.g. [13-15], [17], [19]) mostly support the execution of modeled agents on just one agent platform. Rarely, two different platforms are supported (e.g. [5], [9]) and as far as we know, there is no any MAS DSML which provides an operational semantics for more than two different agent execution platforms. In order to increase the platform variety, we propose benefiting from the vertical transformations already existing between the syntax of a MAS DSML (let us call $DSML_1$) and metamodels of various agent platforms for enabling model instances of another MAS DSML (let us call $DSML_2$) executable on the same agent platforms by just constructing horizontal transformations between the PIMMs of the MAS DSMLs in question. Therefore, instead of defining and implementing N different M2M and M2T transformations for N different agent platforms, creation of only one single set of M2M transformations between $DSML_1$ and $DSML_2$ can be enough for the execution of $DSML_2$'s model instances on these N different agent platforms.

Fig. 1 depicts the construction of model transformations between MAS DSMLs and hence re-use of already existing transformations between those DSMLs and agent platforms. Let the abstract syntaxes of $DSML_1$, $DSML_2$ and $DSML_3$ be the metamodels MM_1, MM_2 and MM_3 respectively. Horizontal lines between these MAS DSMLs represent the M2M transformations between these metamodels. According to the figure, agent systems modeled in $DSML_1$ are already executable on the agent platforms A and B (due to the existing vertical transformations for these platforms), while $DSML_2$ model instances are executable on the agent platforms X, Y and Z. Similarly M2M and M2T transformations were already provided for the execution of $DSML_3$ model instances on the agent platforms α, β, θ respectively. If $DSML_1$ is required to support X and Y agent platforms, designers should prepare new model transformations separately for those agent platforms (shown with dotted arrows in Fig. 1) in case of the absence of horizontal transformations between MM_1 and MM_2. Hence, construction of only one set of horizontal M2M transformations between $DSML_1$ and $DSML_2$ enables $DSML_1$'s automatic support on agent platforms X, Y (and also Z). Conversely, same is also valid for extending the $DSML_2$'s support for agent execution platforms. Interoperability between $DSML_1$ and $DSML_2$ over these newly defined horizontal transformations also makes transformation and code generation of $DSML_2$ model instances for the agent platforms A and B. In addition to the important decrease in the number of transformations, construction of horizontal model transformations between the PIMMs of MAS DSMLs is more feasible and easier than the vertical transformations since the DSMLs are in the same abstraction level according to MDA.

III. Interoperability between SEA_ML and DSML4MAS

In this section, we discuss the applicability of the proposed approach by taking into account the construction of the interoperability between two MAS DSMLs called SEA_ML and DSML4MAS. Both DSMLs enable the

[4] http://www.omg.org/mda/ (last access: June 2016)

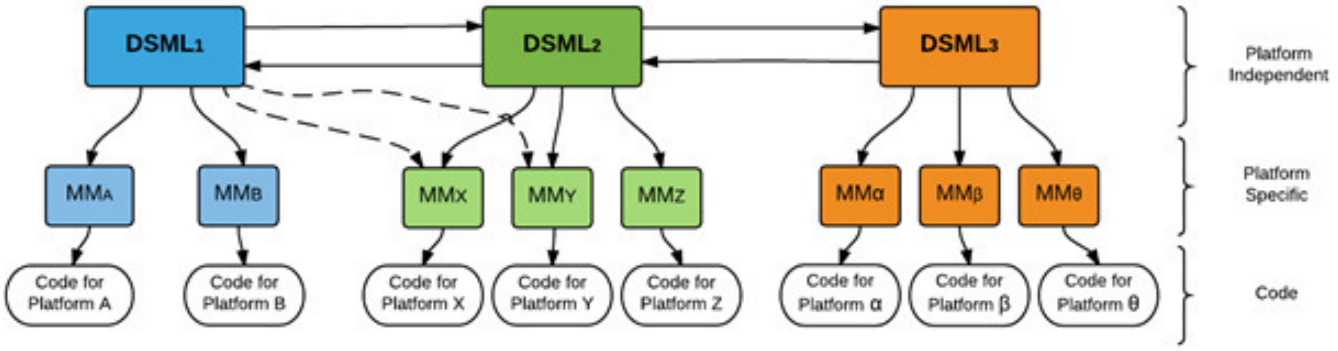

Fig. 1 Interoperability of MAS DSMLs via horizontal model transformations

modeling of agent systems according to various agent internal and MAS organizational viewpoints. They provide a clear visual syntax for MAS modeling and code generation for agent implementation and execution platforms. Moreover, both languages are equipped with Eclipse-based IDEs in which modeling and automatic generation of MAS components are possible. These features of the languages led us to choose them in this study. In the following subsections, brief introduction of these DSMLs and implemented model transformations between these languages are given.

A. SEA_ML

SEA_ML [19] provides a convenient and handy environment for agent developers to construct and implement software agent systems working on various application domains. In order to support MAS experts when programming their own systems, and to be able to fine-tune them visually, SEA_ML covers all aspects of an agent system from the internal view of a single agent to the complex MAS organization. In addition to these capabilities, SEA_ML also supports the model-driven design and implementation of autonomous agents who can evaluate semantic data and collaborate with semantically-defined entities of the Semantic Web, like Semantic Web Services (SWS) [25]. That feature exactly differentiates SEA_ML and makes unique regarding any other MAS DSML currently available. Within this context, it includes new viewpoints which specifically pave the way for the development of software agents working on the Semantic Web environment. Modeling agents, agent knowledgebases, platform ontologies, SWS and interactions between agents and SWS are all possible in SEA_ML.

SEA_ML's metamodel is divided into eight viewpoints, each of which represents a different aspect for developing Semantic Web enabled MASs [19]. *Agent's Internal Viewpoint* is related to the internal structures of semantic web agents (SWAs) and defines entities and their relations required for the construction of agents. It covers both reactive and Belief-Desire-Intention (BDI) agent architectures. *Interaction Viewpoint* expresses the interactions and communications in a MAS by taking messages and message sequences into account. *MAS Viewpoint* solely deals with the construction of a MAS as a whole. It includes the main blocks which compose the complex system as an organization. *Role Viewpoint* delves into the complex controlling structure of the agents and addresses role types. *Environmental Viewpoint* addresses the use of resources and interaction between agents with their surroundings. *Plan Viewpoint* deals with an agent Plan's internal structure, which are composed of Tasks and atomic elements such as Actions. *Ontology Viewpoint* addresses the ontological concepts which constitute agent's knowledgebase (such as belief and fact). *Agent - SWS Interaction Viewpoint* defines the interaction of agents with SWS including the definition of entities and relations for service discovery, agreement and execution. A SWA executes the semantic service finder Plan (SS_FinderPlan) to discover the appropriate services with the help of a special type of agent called SSMatchMakerAgent who executes the service registration plan (SS_RegisterPlan) for registering the new SWS for the agents. After finding the necessary service, one SWA executes an agreement plan (SS_AgreementPlan) to negotiate with the service. After negotiation, a plan for service execution (SS_ExecutorPlan) is applied for invoking the service.

The collection of SEA_ML viewpoints constitutes an extensive and all-embracing model of the MAS domain. SEA_ML's abstract syntax combines the generally accepted aspects of MAS (such as MAS, Agent Internal, Role and Environment) and introduces two new viewpoints (Agent-SWS Interaction and Ontology) for supporting the development of software agents working within the Semantic Web environment [2].

SEA_ML can be used for both modeling MASs and generation of code from the defined models. SEA_ML instances are given as inputs to a series of M2M and M2T transformations to achieve executable artifacts of the system-to-be-built for JADEX agent platform and semantic web service description documents conforming to *Web Ontology Language for Services (OWL-S)* ontology[5]. It is also possible to automatically check the integrity and validity of SEA_ML models [26]. Complete discussion on SEA_ML can be found in [19].

[5] https://www.w3.org/Submission/OWL-S/ (last access: June 2016)

B. DSML4MAS

DSML4MAS [5], [13] is perhaps one of the first complete MAS DSMLs in which a PIMM, called PIM4Agents, provides an abstract syntax for different aspects of agent systems. Similar to SEA_ML's viewpoints, both internal behavior model of agents and agent interactions in a MAS are covered by PIM4Agents views / aspects. *Multiagent view* contains all the main concepts in a MAS such as Agent, Cooperation, Capability, Interaction, Role and Environment. *Agent view* focuses on the single autonomous entity (agent), the roles it plays within the MAS and the capabilities it has to solve tasks and to reach the environment resources. *Behavioural view* describes how plans are composed by complex control structures and simple atomic tasks like sending a message and how information flows between those constructs. In here, a plan is a specialized version of behavior composed of activities and flows. Activities and tasks are minimized parts of the work and flows provide the communication between these parts. *Organization view* describes how single autonomous entities cooperate within the MAS and how complex organizational structures can be defined. Social structure in the system is defined with cooperation entity where agents and organizations take part in. The structure has its own protocol defining how the entities interact in a cooperation. Agents have "domainRoles" for the interaction and these roles are attached to actors by "actorBinding" entities where actors are representative entities within the corresponding interaction protocol. *Role view* examines the behaviour of an agent entity in an organization or cooperation. An agent's role covers the capabilities and information to have access to a set of resources. *Interaction view* describes how the interaction in the form of interaction protocols takes place between autonomous entities or organizations. Agents communicate over the PIM4Agents Protocol which refers to actors and "messageFlows" between these actors. Finally, *Environment view* contains the resources accessed and shared by agents and organizations. Agents can communicate with the environment indirectly via using resources. Resources can store knowledge from BDI agents for changing beliefs by using Messages and Information flows.

As indicated in [5], grouping modelling concepts in DSML4MAS allows the metamodel evolution by adding new modelling concepts in the existing aspects, extending existing modelling concepts in the defined aspects, or defining new modelling concepts for describing additional aspects of agent systems. For instance, SWS integration into the system models conforming to DSML4MAS is provided via introducing the SOAEnvironment entity [27] which extends the Environment entity and contains service descriptions. Agents use service descriptions to specify the Services they are searching for and then service interaction is realized by InvokeWS and ReceiveWS tasks which are inherited from Send and Receive task entities described in PIM4Agents.

Similar to SEA_ML, DSML4MAS also enables the MDD of MAS including a concrete graphical syntax [28] based on the abovementioned PIMM (PIM4Agents) and an operational semantics for the execution of modeled agent systems on JACK or JADE agent platforms. Extensions to the language introduced in [27] provide the description of the services inside an agent environment according to specifications such as *Web Services Modeling Language (WSML)*[6] or *Semantic Annotation of WSDL and XML Schema (SAWSDL)*[7]. Interested readers may refer to [5], [13] and [27] for an extensive discussion on DSML4MAS.

C. Horizontal Model Transformations between SEA_ML and DSML4MAS

We have applied the horizontal transformability approach described in Sect. 2 for establishing the interoperability between SEA_ML and DSML4MAS. As shown in Fig. 2, SEA_ML currently supports the MAS implementation for JADEX BDI architecture and SWS generation according to the OWL-S ontology. In order to extend its platform support capability, new M2M and M2T transformations should be prepared for each new implementation platform. For instance, M2M transformations are needed between the abstract syntax of SEA_ML and PSMM of JADE framework to make SEA_ML instances also executable on the JADE platform. It is worth indicating that definition and application of M2T transformations are also required for the code generation from the outputs of the previous SEA_ML to JADE transformations. Instead, we can follow the approach introduced in Sect. 2 by just writing the horizontal transformation rules between the metamodels of SEA_ML and DSML4MAS and running those transformations on SEA_ML instances for the same purpose: making SEA_ML models executable also on JADE platform. That is possible since DSML4MAS has already support on JADE and JACK agent platforms and SAWSDL and WSDL semantic service ontologies via vertical transformations between its metamodel and metamodels of the corresponding system implementation platforms. Realization of horizontal transformations between SEA_ML and DSML4MAS has extra benefits such as the execution of SEA_ML instances also on JACK platform and/or implementation of the modeled SWS according to SAWSDL or WSDL specifications (Fig. 2).

Before deriving the rules of transformations, we should determine the entity mappings between both languages since the transformations are definitely based on these entity mappings. Comparing with the mappings we previously provided in [15] or [19] for the transformability of SEA_ML instances to MAS execution platforms, we have experienced that the determination of the entity mappings in this study was easier and took less time. We believe that the reason of this efficiency originates from the fact that metamodels of SEA_ML and DSML4MAS are in the same abstraction level and provide close entities and relations in similar viewpoints for MAS modeling.

[6] http://www.wsmo.org/wsml/ (last access: June, 2016)
[7] https://www.w3.org/TR/sawsdl/ (last access: June, 2016)

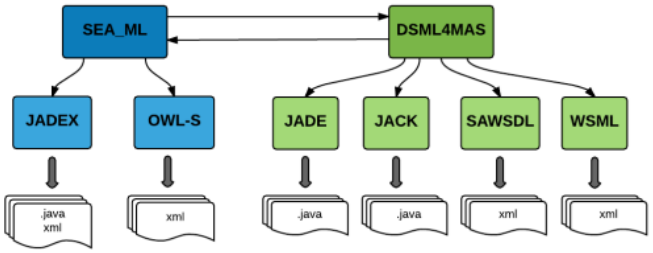

Fig. 2 Interoperability of SEA_ML and DSML4MAS

Table 1 lists some of the important mappings constructed between first-class entities of these two languages. For instance, two agent types (SWA and SSMatchmakerAgent) defined in SEA_ML are mapped onto the autonomous entity Agent defined in DSML4MAS. Likewise, meta-entities pertaining to agent plan types (SS_RegisterPlan, SS_FinderPlan, SS_AgreementPlan and SS_ExecutorPlan) required for the interaction between the semantic services are mapped with the Plan concept of DSML4MAS. Since Actor entity in DSML4MAS has access to resources and owns capabilities needed for agent interactions, SEA_ML's Role entity is mapped onto Actor entity.

One interesting mapping is encountered between SEA_ML's SWS entity and DSML4MAS's SOAEnvironment since it enables the representation of SEA_ML semantic services in DSML4MAS model instances. On the DSML4MAS side, SOAEnvironment entity, which is extended from Environment entity, includes services in general. Hence, SEA_ML SWS entity is mapped onto SOAEnvironment entity and SEA_ML WebService entities are mapped onto Service entities. In SEA_ML WebService definition, every service has Interface, Process and Grounding. Interface entity represents the information about service inputs, outputs and any other necessary information. Process entity has internal information about the service and finally Grounding entity defines the invocation protocol of the web service [19]. DSML4MAS services are described with Blackbox and Glassbox entities [27]. BlackBox is used to define a service's functional and non-functional parameters while Glassbox includes the description of the internal service process. The Functionals are described in terms of service signature that are input and output parameters, and specifications that are preconditions and effects. The NonFunctionals are defined in terms of price, service name and developer. Hence, Interface and Process entities of services defined in SEA_ML are mapped onto DSML4MAS Functionals which have input and output definitions. On DSML4MAS side, agent interactions with services are provided by InvokeWS and ReceiveWS tasks. So, SEA_ML Grounding, which represents the physical structure of the underlying web service executed for the corresponding SWS, is mapped to InvokeWS. Remaining mappings listed in Table 1 (e.g. SEA_ML SWO to DSML4MAS Organization, SEA_ML Environment to DSML4MAS Environment) are very simple to determine since the related entities on both sides have similar or almost same functionality within the syntaxes of the languages.

TABLE I.
ENTITY MAPPINGS BETWEEN THE METAMODELS OF SEA_ML AND DSML4MAS

SEA_ML MM	DSML MM
SWA (Semantic Web Agent)	Agent
SSMatchmakerAgent	Agent
Role	Actor
SWS (Semantic Web Service)	SOAEnvironment
Environment	Environment
WebService	Service
Interface	Functionals
Process	Functionals
Grounding	InvokeWS
Input	Input
Output	Output
Precondition	Precondition
SS_RegisterPlan	Plan
SS_FinderPlan	Plan
SS_AgreementPlan	Plan
SS_ExcecutorPlan	Plan
SWO (Semantic Web Organization)	Organization

After determining the entity mappings between SEA_ML and DSML4MAS, it is necessary to provide model transformation rules which are applied at runtime on SEA_ML instances to generate DSML4MAS counterparts of these instances. For that purpose, transformation rules should be formally defined and written according to a model transformation language. In this study, we preferred to use ATL Transformation Language (ATL)[8] to define the model transformations between SEA_ML and DSML4MAS. ATL is one of the well-known model transformation languages, specified as both metamodel and textual concrete syntax. An ATL transformation program is composed of rules that define how the source model elements are matched and navigated to create and initialize the elements of the target models. In addition, ATL can define an additional model querying facility which enables specifying the requests onto models. ATL also allows code factorization through the definition of ATL libraries. Finally, ATL has a transformation engine and an IDE that can be used as a plug-in on an Eclipse platform. These features of ATL caused us to prefer it as the implementation language for the horizontal transformations from SEA_ML to DSML4MAS.

ATL is composed of four fundamental elements. The first one is the header section defining attributes relative to the transformation module. The next element is the import section which is optional and enables the importing of some existing ATL libraries. The third element is a set of helpers

[8] https://eclipse.org/atl/(last access: June, 2016)

that can be viewed as the ATL equivalents to the Java methods. The last element is a set of rules that defines the way target models are generated from source models.

Following listings include some excerpts from the written ATL rules in order to give some flavor of M2M transformations provided in this study. To this end, rule in Listing 1 enables the transformation of the elements covered by the Agent-SWS Interaction viewpoint of SEA_ML to their counterparts included in DSML4MAS's Multiagent system viewpoint. In line 1, rule is named uniquely. In line 2, the source metamodel is chosen and renamed as swsinteractionvp with "from" keyword. The target metamodel is indicated and renamed as pim4agents with "to" keyword (Line 3). In the following lines (between 4 and 14), instances of SEA_ML SWA and SSMatchmakerAgent entities are selected and transformed to DSML4MAS Agent instances. Transformation of agent roles and plans are also realized by using "Set" and "allInstances" functions. It is worth indicating that types of Plan instances seem to be transformed to DSML4MAS behavior in the given listing although all SEA_ML Plan types are semantically mapped to DSML4MAS Plan as listed in Table 1. That is because some of the DSML4MAS meta-entities are collected with tag definitions in Ecore representations which take the same name with the related viewpoint. For instance, plans are not defined solely with their names; instead they are collected in behavior definitions. Hence, in order to provide the full transformations of the plans with all their attributes, ATL rule is written here as mapping SEA_ML plan instances to the DSML4MAS behaviors. Inside another helper rule, those behaviors are separated into the corresponding plans and so exact transformation of SEA_ML plan instances to DSML4MAS plans are realized.

```
01 rule SWSInteractionVP2MultiagentSystem {
02  from   swsinteractionvp: SWSInteraction!SWSInteractionViewpoint
03  to     pim4agent: PIM4Agents!MultiagentSystem(
04       agent <- Set{SWSInteraction!SemanticWebAgent.allInstances()},
05       agent <- Set{SWSInteraction!SSMatchmakerAgent.allInstances()},
06       role <- Set{SWSInteraction!Role.allInstances()},
07       role <- Set{SWSInteraction!RegistrationRole.allInstances()},
08       behavior <- Set{SWSInteraction!SS_AgreementPlan.allInstances()},
09       behavior <- Set{SWSInteraction!SS_ExecutorPlan.allInstances()},
10       behavior <- Set{SWSInteraction!SS_FinderPlan.allInstances()},
11       behavior <- Set{SWSInteraction!SS_RegisterPlan.allInstances()},
12       environment<-Set{SWSInteraction!SWS.allInstances()},
13       environment<-Set{SWSInteraction!Grounding.allInstances()} )
14 }
```

Listing 1 An excerpt from the SWSInteractionVP2MultiagentSystem rule

Another example can be given for the transformation of SEA_ML SWS instances to DSML4MAS SOAEnvironment instances. In Listing 2, the first rule provides the related transformation. After controlling the name of the SWS by applying the helper rule "nameControl", its attributes are converted to their counterparts in the abstract syntax of DSML4MAS. Again with using helper rules, the web services composed by this semantic service are determined and transformed to DSML4MAS Service instances. Only a fragment of SWS2SOAEnvironment can be given in here due to space limitations. The remaining part of this rule provides the transformation of each SEA_ML semantic service's discovery, engagement and execution components (e.g. Interface, Process and Grounding) to their counterparts in DSML4MAS models according to the mappings given in Table 1.

```
01  rule SWS2SOAEnvironment{
02    from  envIN: SWSInteraction!SWS(envIN.nameControl)
03    to envOUT: PIM4Agents!SOAEnvironment (
04          name <- envIN.setName(),
05          service<-envIN.setWebServices()
06    )
07  }

08  rule WebService2Service{
09    from  webService: SWSInteraction!WebService
10    to service: PIM4SWS!Service (
11          ID<- webService.setName()
12    )
13  }
```

Listing 2 Excerpts from the SWS2SOAEnvironment and WebService2Service rules

In Listing 3, the helper rules used in above transformation rules are given. "nameControl" helper is executed on the SEA_ML SWS instances. It controls whether a SWS has a name attribute. Based on this attribute's existence, the rules returns true (line 4) or false (line 5). That Boolean result is evaluated by the caller rule given in Listing 2. In the second helper rule, the name attribute of a semantic service is controlled. In case of the name is empty, the string given in line 10 is assigned as the value for the transformed service's (SOAEnvironment) name attribute. Otherwise, the name of the source SWS is returned back to the called rule (SWS2SOAEnvironment) to be assigned as the name of the transformed SOAEnvironment instance which will be included in the target DSML4MAS model.

```
01  helper context SWSInteraction!SWS
02    def: nameControl: Boolean =
03      if not (self.name.oclIsUndefined())
04        then   true
05        else  false
06      endif;

07  helper context SWSInteraction!SWS
08    def: setName(): String =
09     if (self.name = '')
10       then  'SERVICE_NAME_IS_EMPTY'
11       else  self.name
12     endif;
```

Listing 3 Helper rules used by the SWS2SOAEnvironment rule

IV. Case Study: Agent-Based Stock Exchange System

In this section, the interoperability of SEA_ML and DSML4MAS is demonstrated by applying the proposed horizontal model transformations for the development of an agent-based stock exchange system. The system is modeled in SEA_ML and transformed to a DSML4MAS instance to use the generation power of DSML4MAS language. In this way, the implementation of this system's agents on JADE (or JACK) platform and services as SAWSDL or WSML ontology instances can be possible by using the operational semantics of DSML4MAS which is already provided for the

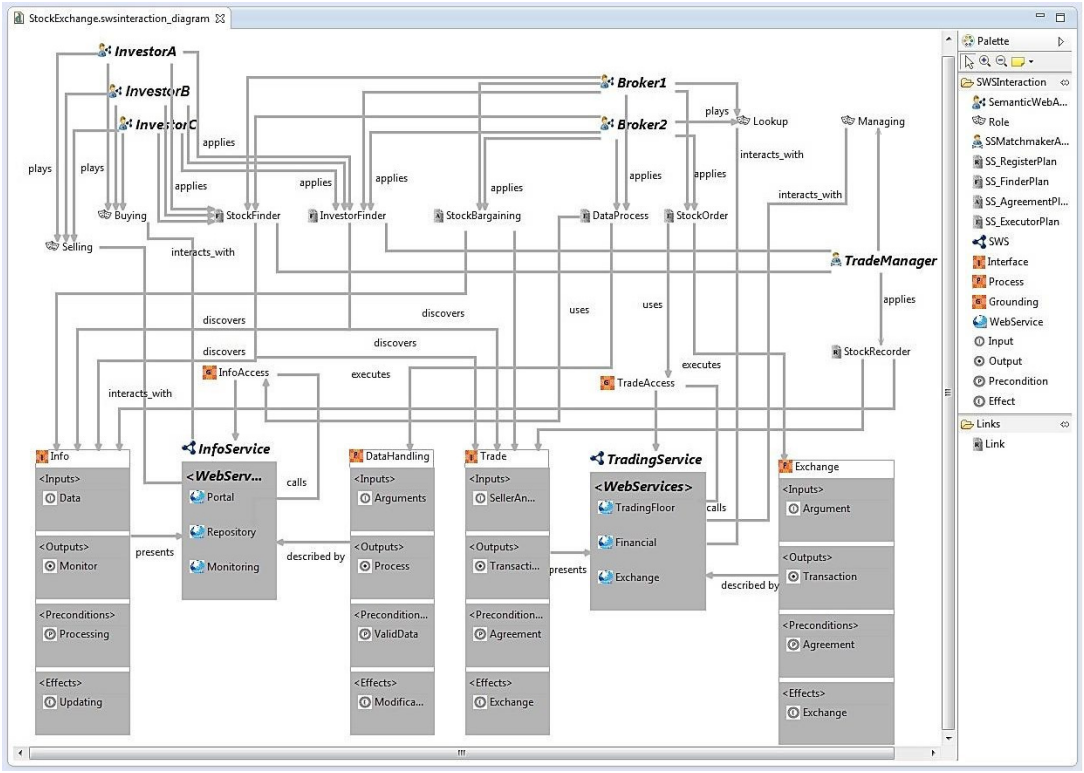

Fig. 3 Instance model of the multi-agent stock exchange system in SEA_ML with including the agents, semantic web services and their relations

execution of agents and the generation of semantic web services (see Fig. 2).

Stock trading is one of the key items in economy and estimating its behavior and taking the best decision in it are among the most challenging issues. Agents in a MAS can share a common goal or they can pursue their own interests. That nature of MASs exactly fits to the requirements of free market economy. Moreover, Stock Exchange Market has lots of services which are offered for Investors (Buyer or Seller), Brokers, and Stock Managers. These services can be represented with semantic web services to achieve more accurate service finding and service matching.

When considering the structure of the system, the semantic web agents work within a semantic web organization for Stock System including sub-organizations for Stock Users where the Investor and Broker agents reside, and the Stock Market where the system's internal agents, e.g. Trade Manager (a SSMatchmaker agent instance) work. The Stock Market organization also has two sub-organizations, the Trading Floor and the Stock Information System. These organizations and sub-organizations have their own organizational roles. These organizations also need to access some resources in other environments. Therefore, they have interactions with the required environments to gain access permissions. For example, agents in the Stock Market sub-organization need to access bank accounts and some security features, so that they can interact with the Banking & Security environment. All of the user agents including Investors and Brokers cooperate with Trade Manager to access the Stock Market. Also, the user agents interact with each other. For instance, Investor Agents can cooperate with Brokers to exchange stock for which Brokers are expert. More information on developing such stock trading agents can be found in [29].

To model the system in SEA_ML, Agent-SWS Interaction viewpoint is considered as the representative for SEA_ML viewpoints. This viewpoint is the most important aspect of MASs working in semantic web environments. Fig. 3 shows a screenshot from the SEA_ML's modeling environment in which instances of both the semantic services and the agent plans required for the stock exchange are modeled, including their relations according to Agent-SWS interaction viewpoint of SEA_ML. Investor and Broker agents can be modeled with appropriate plan instances in order to find, make the agreement with and execute the services. The services can also be modeled for the interaction between the semantic web service's internal components (such as Process, Grounding, and Interface), and the SWA's plans. It is important to indicate that the stock exchange system given in here was already modeled in the SEA_ML environment before this study and instead of re-modeling the whole system (e.g. in DSML4MAS), the existing model is intentionally adopted in here to examine the applicability of the proposed approach. In fact, the model in question is much more complicated and we can only consider the agent-SWS interaction aspect due to page limits of this paper. Discussion on the whole model can be found in [19] and the sources of the model are available at the SEA_ML's distribution website[9].

[9] SEA_ML, its modeling tool and the instance models for the case study are available at: http://serlab.ube.ege.edu.tr/resources.html (last access: June 2016)

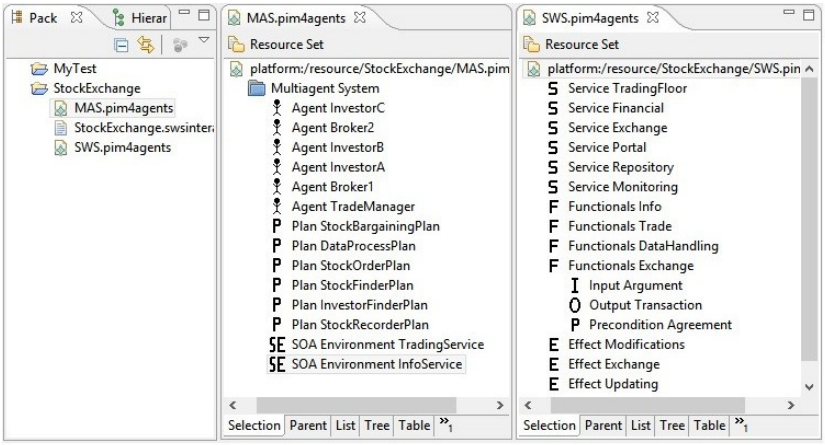

Fig. 4 An excerpt of the output in its Ecore tree view representation achieved after executing the M2M transformations on the SEA_ML instance

We can see from the instance model given in Fig. 3 that an investor agent (e.g. InvestorA) would play the Buying role and apply its StockFinder plan for finding an appropriate Trading service interface of one TradingService SWS in order to buy some stocks. This plan would realize the discovery by interacting with the TradeManager SSMatchmakerAgent which has registered the services by applying the StockRecorder plan. As InvestorA cooperates with Broker1 in order to receive some expert advice for its investment, at the next step, the Broker1 agent applies its StockBargaining plan for negotiating with the already discovered services. This negotiation would be made through the Trade interface of the SWS. Finally, if the result of the negotiation were positive, the agent would apply the StockOrder plan to call the TradingFloor of the SWS by executing its Exchange process and using its TradeAccess grounding with which the service would be realized. In a similar way, Investor agents could cooperate with Brokers and interact with the TradeManager in order to collect some information about the market, e.g. the rate of exchange for a currency or the fluctuation rate for a specific stock.

The designed instance model is controlled based on the provided constraint rules in SEA_ML tool to check its validity. Then, the horizontal model transformations discussed in the previous section are executed on this SEA_ML instance model and as result, we have succeeded to automatically achieve the counterpart models conforming to DSML4MAS. To realize the transformation, the SEA_ML metamodel, the SEA_ML instance models for this case study, and the DSML4MAS metamodel are given to the ATL engine as input and the instance models of the case study in DSML4MAS are generated by the engine with executing our transformation rules. An excerpt of the XMI file containing the output DSML4MAS instance model is given in Fig. 4 in which the output instance can be seen in its Ecore tree view representation.

The generated model conforms to the specification of DSML4MAS's abstract syntax, so it can be handled with DSML4MAS's graphical editor[10]. To visualize the instance model in DSML4MAS, the only thing needed is to add the related graphical notations to the generated instance model. The screenshot given in Fig. 5 shows the appearance of output instance model in the concrete syntax of DSML4MAS. We can examine from the figure that the agents and their relations we modeled in SEA_ML are exactly reflected to a DSML4MAS model after execution of the M2M transformations proposed in this study. From now on, it is straightforward to automatically achieve platform-specific executables and documents of this MAS model for JADE or JACK agent platforms and SAWSDL or WSML semantic service ontologies since DSML4MAS has already own a chain of M2M and M2T transformations for these agent execution platforms and service ontologies as discussed in Sect. 3.2.

V. Related Work

In the last decade, AOSE researchers have noteworthy efforts on the derivation and use of DSLs / DSMLs for MAS. For instance, the Agent-DSL [12] is used to specify the agency properties that an agent needs to accomplish its tasks. However, the proposed DSL is presented only with its metamodel and provides just a visual modeling of the agent systems according to agent features, like knowledge, interaction, adaptation, autonomy and collaboration. Likewise, in [30], the authors introduced two DSMLs. These languages are described by metamodels which can be seen as the representations of the main concepts and relationships identified for each of the particular domains again introduced in [30]. The study included only the abstract syntax of the related DSMLs and does not give the concrete syntax or semantics of the DSMLs.

As previously discussed in this paper, Hahn [13] introduced a DSML for MAS called DSML4MAS. The abstract syntax of the DSML was derived from a platform independent metamodel [5] which was structured into several aspects each focusing on a specific viewpoint of a MAS. In order to provide a concrete syntax, the appropriate graphical notations for the concepts and relations were defined [28]. Furthermore, DSML4MAS supports the

[10]The IDE of DSML4MAS is available at: https://sourceforge.net/projects/dsml4mas/ (last access: June 2016)

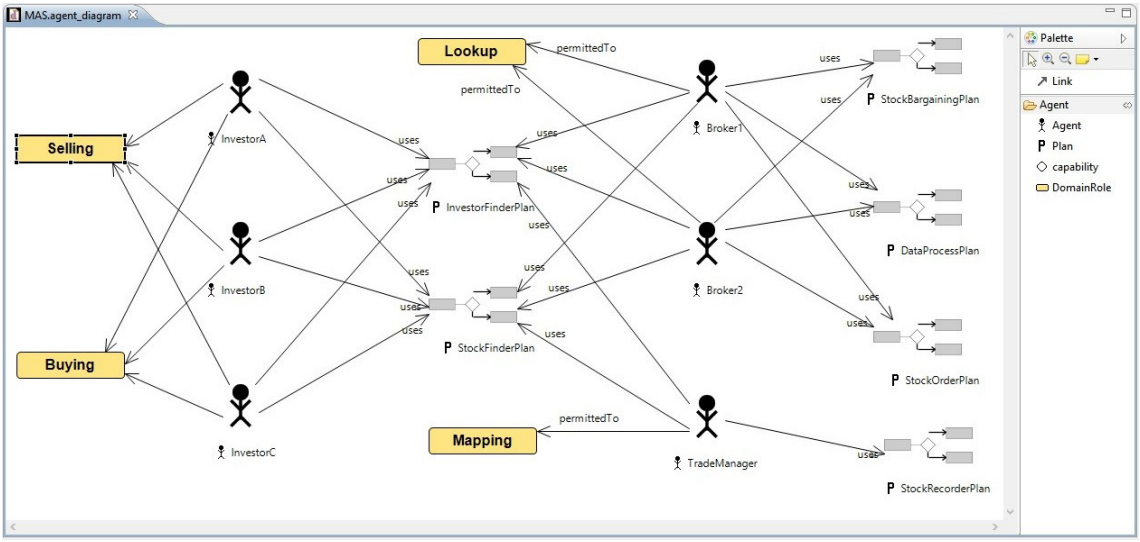

Fig. 5 Partial instance model of the agent-based stock exchange system in DSML4MAS achieved after application of the defined M2M transformations

deployment of modeled MASs both in JACK and JADE agent platforms by providing an operational semantics over model transformations.

Another DSML was provided for MASs in [17]. The abstract syntax was presented using the Meta-object Facility (MOF)[11] , the concrete syntax and its tool was provided with Eclipse Graphical Modeling Framework (GMF)[12], and finally the code generation for the JACK agent platform was realized with model transformations using Eclipse JET[13]. However, the developed modeling language was not generic since it was based on only the metamodel of one of the specific MAS methodologies called Prometheus. A similar study was performed in [14] which proposes a technique for the definition of agent-oriented engineering process models and can be used to define processes for creating both hardware and software agents. This study also offers a related MDD tool based on a specific MAS development methodology called INGENIAS.

Originating from a well-formalized syntax and semantics, Ciobanu and Juravle defined and implemented a language for mobile agents in [16]. They generated a text editor with auto-completion and error signaling features and presented a way of code generation for agent systems starting from their textual description. A recent work conducted in [20] aimed at creating a UML-based agent modeling language, called MAS-ML, which is able to model the well-known types of agent internal architectures, namely simple reflex agent, model-based agent, reflex agent, goal-based agent and utility-based agent. Representation and exemplification of all supported agent architectures in the concrete syntax of the introduced language are given. MAS-ML is also accompanied with a graphical tool which enables agent modeling. However, the current version of MAS-ML does not support any code generation for MAS frameworks which prevents the execution of the modeled agent systems.

Finally, by considering our previous studies, in [15] and [18], we showed the derivation of a DSL for the MDE of agent systems working on the Semantic Web. That initial version of the language was refined and enriched with a graphical concrete syntax in [19]. This new language, called SEA_ML, covered an enhanced version of agent-SWS interaction viewpoint in which modeling those interactions can be elaborated as much as possible for the exact implementation of agent's service discovery, agreement and execution dynamics. We also presented the formal semantics of the language [26] and discussed how the applied methodology can pave the way of evolutionary language development for MAS DSLs [2]. Moreover, qualitative evaluation and quantitative analysis of SEA_ML have been recently performed over a multi-case study protocol [31].

The work presented in this paper contributes to the above mentioned MAS DSL/DSML studies by introducing the interoperability of the languages and hence the proposed MDE technique helps to facilitate the platform support of the MAS DSMLs comparing with the existing agent platform extensibility approaches which deal with the definition and implementation of new M2M and M2T transformations for each execution platform. To the best of our knowledge, the work herein is the first effort on the interoperability of the MAS DSMLs and it is the first study in AOSE which employs horizontal model transformations to enable this interoperability. It is worth indicating that apart from our proposal, only the work conducted in [10] considers the application of horizontal transformations. However, that study just provides the transformation between the metamodels of two specific AOSE methodologies (Prometheus and INGENIAS) to realize MAS implementation on exactly one agent deployment platform and does not consider MAS DSML interoperability or language extensibility on various agent platforms.

[11] http://www.omg.org/mof/ (last access: June 2016)
[12] http://www.eclipse.org/modeling/gmp/ (last access: June, 2016)
[13] https://eclipse.org/modeling/m2t/?project=jet (last access: June 2016)

II. Conclusion

An approach for extending the execution platform support of MAS DSMLs over language interoperability is presented in this paper. The interoperability is provided by defining and implementing horizontal M2M transformations between the agent metamodels which constitute the syntaxes of MAS DSMLs. Due to being at the same abstraction level, both mapping the model entities and implementing the model transformations are more convenient and less laborious comparing with the M2M and M2T transformation chain required in the way of enriching the support of DSMLs for various agent execution platforms. The applicability of the approach is demonstrated by constructing transformations between two full-fledged agent DSMLs.

As the future work, we first plan to extend the applicability of this interoperability approach for some other MAS DSMLs such as MAS-ML [20]. Later, the assessment of the language interoperability will be performed e.g. by taking into consideration the amount and quality of the automatically generated artifacts for MAS software. For this purpose, an improved version of the evaluation framework described in [31] can be employed.

References

[1] C. Badica, Z. Budimac, H. D. Burkhard, M. Ivanovic. 2011. Software agents: Languages, tools, platforms, *Computer Science and Information Systems* 8(2): 255-298, DOI: 10.2298/CSIS110214013B

[2] M. Challenger, M. Mernik, G. Kardas, T. Kosar. 2016. Declarative specifications for the development of multi-agent systems, *Computer Standards & Interfaces* 43: 91-115, DOI: 10.1016/j.csi.2015.08.012

[3] G. Kardas. 2013. Model-driven development of multiagent systems: a survey and evaluation. *The Knowledge Engineering Review* 28(4): 479-503, DOI: 10.1017/S0269888913000088

[4] A. Omicini, A. Ricci, M. Viroli. 2008. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems* 17(3): 432-456, DOI: 10.1007/s10458-008-9053-x

[5] C. Hahn, C. Madrigal-Mora, K.Fischer. 2009. A Platform-Independent Metamodel for Multiagent Systems. *Autonomous Agents and Multi-Agent Systems* 18(2): 239-266, DOI: 10.1007/s10458-008-9042-0

[6] G. Beydoun, G. Low, B. Henderson-Sellers, H. Mouratidis, J. J. Gomez-Sanz, J. Pavon, C. Gonzalez-Perez. 2009. FAML: A Generic Metamodel for MAS Development. *IEEE Transactions on Software Engineering* 35(6): 841-863, DOI: 10.1109/TSE.2009.34

[7] I. Garcia-Magarino. 2014. Towards the integration of the agent-oriented modeling diversity with a powertype-based language. *Computer Standards & Interfaces* 36: 941–952, DOI: 10.1016/j.csi.2014.02.002

[8] J. Pavon, J. Gomez-Sanz, R. Fuentes. 2006. Model driven development of multi-agent systems, *Lecture Notes in Computer. Science* 4066: 284–298, DOI: 10.1007/11787044_22

[9] G. Kardas, A. Goknil, O. Dikenelli, N.Y. Topaloglu. 2009. Model driven development of semantic web enabled multi-agent systems, *International Journal of Cooperative Information Systems* 18(2): 261-308, DOI: 10.1142/S0218843009002014

[10] J. M. Gascuena, E. Navarro, A. Fernandez-Caballero, R. Martínez-Tomas. 2014. Model-to-model and model-to-text: looking for the automation of VigilAgent. *Expert Systems* 31(3): 199-212, DOI: 10.1111/exsy.12023

[11] B.R. Bryant, J. Gray, M. Mernik, P. J. Clarke, R. B. France, G. Karsai. 2011. Challenges and Directions in Formalizing the Semantics of Modeling Languages. *Computer Science and Information Systems* 8(2): 225-253, DOI: 10.2298/CSIS110114012B

[12] U. Kulesza, A. Garcia, C. Lucena, P. Alencar. 2005. A generative approach for multi-agent system development. *Lecture Notes in Computer Science* 3390: 52–69, DOI: 10.1007/978-3-540-31846-0_4

[13] C. Hahn. 2008. A Domain Specific Modeling Language for Multiagent Systems. 7th Int'l Conf. on Autonomous agents and Multi-agent systems (AAMAS 2008), pp. 233-240

[14] R. Fuentes-Fernandez, L. Garcia-Magarino, A. Maria Gomez-Rodriguez, J. Carlos Gonzalez-Moreno. 2010. A technique for defining agent-oriented engineering processes with tool support. *Engineering Applications of Artificial Intelligence* 23(3): 432-444, DOI: 10.1016/j.engappai.2009.08.004

[15] S. Demirkol, M. Challenger, S. Getir, T. Kosar, G. Kardas, M. Mernik. 2012. SEA_L: A Domain-specific Language for Semantic Web enabled Multi-agent Systems. 2nd Workshop on Model Driven Approaches in System Development at FedCSIS 2012, pp. 1373-1380

[16] G. Ciobanu, C. Juravle. 2012. Flexible Software Architecture and Language for Mobile Agents. *Concurrency and Computation-Practice & Experience* 24(6): 559-571, DOI: 10.1002/cpe.1854

[17] J. M. Gascuena, E. Navarro, A. Fernandez-Caballero. 2012. Model-Driven Engineering Techniques for the Development of Multi-agent Systems. *Engineering Applications of Artificial Intelligence* 25(1): 159-173, DOI: 10.1016/j.engappai.2011.08.008

[18] S. Demirkol, M. Challenger, S. Getir, T. Kosar, G. Kardas, M. Mernik. 2013. A DSL for the development of software agents working within a semantic web environment. *Computer Science and Information Systems* 10(4): 1525-1556, DOI: 10.2298/CSIS121105044D

[19] M. Challenger, S. Demirkol, S. Getir, M. Mernik, G. Kardas, T. Kosar. 2014. On the use of a domain-specific modeling language in the development of multiagent systems. *Engineering Applications of Artificial Intelligence* 28: 111-141, DOI: 10.1016/j.engappai.2013.11.012

[20] E. J. T. Goncalves, M. I. Cortes, G. A. L. Campos, Y. S. Lopes, E. S. S. Freire, V. T. da Silva, K. S. F. de Oliveira, M. A. de Oliveira. 2015. MAS-ML2.0: Supporting the modelling of multi-agent systems with different agent architectures. *Journal of Systems and Software* 108: 77-109. DOI: 10.1016/j.jss.2015.06.008

[21] M. Mernik, J. Heering, A. Sloane. 2015. When and how to develop domain-specific languages. *ACM Computing Surveys* 37(4): 316-344, DOI: 10.1145/1118890.1118892

[22] M. Joao Varanda Pereira, M. Mernik, D. Da Cruz, P. R. Henriques. 2008. Program Comprehension for Domain-specific Languages. *Computer Science and Information Systems* 5(2): 1-17, DOI: 10.2298/CSIS0802001P

[23] I. Lukovic, M. Joao Varanda Pereira, N. Oliveira, D. Carneiro da Cruz, P. R. Henriques. 2011. A DSL for PIM specifications: Design and attribute grammar based implementation, *Computer Science and Information Systems* 8(2): 379-403, DOI: 10.2298/CSIS101229018L

[24] B. Selic. 2003. The pragmatics of model-driven development. *IEEE Software* 20: 19-25, DOI: 10.1109/MS.2003.1231146

[25] K. Sycara, M. Paolucci, A. Ankolekar, N. Srinivasan. 2003. Automated discovery, interaction and composition of Semantic Web Services. *Journal of Web Semantics*, 1(1): 27-46, DOI: 10.1016/j.websem.2003.07.002

[26] S. Getir, M. Challenger, G. Kardas. 2014. The formal semantics of a domain-specific modeling language for semantic web enabled multi-agent systems. *International Journal of Cooperative Information Systems* 23(3): 1-53, DOI: 10.1142/S0218843014500051

[27] C. Hahn, S. Nesbigall, S. Warwas, I. Zinnikus, K. Fischer, M. Klusch. 2008. Integration of Multiagent Systems and Semantic Web Services on a Platform Independent Level. IEEE/WIC/ACM Int'l Conf. on Web Intelligence and Intelligent Agent Technology, pp. 200-206

[28] S. Warwas, C. Hahn. 2008. The concrete syntax of the platform independent modeling language for multiagent systems. Agent-based Technologies and applications for enterprise interoperability

[29] G. Kardas, M. Challenger, S. Yildirim, A. Yamuc. 2012. Design and implementation of a multiagent stock trading system. *Software: Practice and Experience* 42(10): 1247-1273, DOI: 10.1002/spe.1137

[30] S. Rougemaille, F. Migeon, C. Maurel, M-P. Gleizes. 2007. Model Driven Engineering for Designing Adaptive Multi-agent Systems, *Lecture Notes in Artificial Intelligence* 4995: 318-333, DOI: 10.1007/978-3-540-87654-0_18

[31] M. Challenger, G. Kardas, B. Tekinerdogan. 2016. A systematic approach to evaluating domain-specific modeling language environments for multi-agent systems. *Software Quality Journal*, DOI: 10.1007/s11219-015-9291-5