Computer Science and Information Systems 14(3):875–912 https://doi.org/10.2298/CSIS170113034K

Supporting the platform extensibility for the model-driven development of agent systems by the interoperability between domain-specific modeling languages of multi-agent systems

Geylani Kardas[1], Emine Bircan[2], and Moharram Challenger[3]

International Computer Institute, Ege University, 35100, Bornova, Izmir, Turkey
[1]geylani.kardas@ege.edu.tr, [2]eminebircanbircan@gmail.com,
[3]moharram.challenger@ege.edu.tr

Abstract. The conventional approach currently followed in the development of domain-specific modeling languages (DSMLs) for multi-agent systems (MASs) requires the definition and implementation of new model-to-model and model-to-text transformations from scratch in order to make the DSMLs functional for each different agent execution platforms. In this paper, we present an alternative approach which considers the construction of the interoperability between MAS DSMLs for a more efficient way of platform support extension. The feasibility of using this new interoperability approach instead of the conventional approach is exhibited by discussing and evaluating the model-driven engineering required for the application of both approaches. Use of the approaches is also exemplified with a case study which covers the model-driven development of an agent-based stock exchange system. In comparison to the conventional approach, evaluation results show that the interoperability approach requires both less development time and effort considering design and implementation of all required transformations.

Keywords: Metamodel, Model transformation, Model-driven development, Domain-specific Modeling Language, Multi-agent System, Interoperability.

1. Introduction

Software agents in a Multi-agent system (MAS) interact with each other to solve problems in a competitive or collaborative manner within an environment. In a MAS, 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 [1-3]. 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 [4]. 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 [5].

Agent-oriented software engineering (AOSE) [6] researchers define various agent metamodels (e.g. [7-11]), which include fundamental entities and relations of agent systems in order to master the abovementioned problems of developing MASs. In addition, many model-driven agent development approaches are provided such as [12-15] and researchers also propose domain-specific languages (DSLs) / domain-specific modeling languages (DSMLs) (e.g. [16-24]) for facilitating the development of MASs by enriching MAS metamodels with some defined syntax and semantics (usually translational semantics [25]). DSLs / DSMLs [26-28] 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 [29], 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 [5].

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 [17, 21, 23] 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 [30], JADE [31] and JADEX [32]. 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 (PSMMs) according to the Object Management Group (OMG)’s well-known Model-driven Architecture (MDA) [33] as also indicated in [9] and [14].

Above described approach (which we can refer as the conventional or classical approach) applied in the current MAS DSML development studies, 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 [34] 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 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. [17, 20, 21, 23, 24]), 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 feasibility of using this new interoperability approach instead of the conventional approach of platform support for current MAS DSMLs by first discussing the MDE required for the application of both approaches, and then conducting a comparative evaluation of two approaches according to an evaluation framework [35] which is specific for the assessment of MAS DSMLs. For this purpose, application of the proposed interoperability approach is demonstrated by constructing horizontal transformations between two full-fledged agent DSMLs called SEA_ML [23] and DSML4MAS [9] respectively. In order to provide the related comparison, we also discuss the application of the classical approach on SEA_ML instance models. Use of both approaches is exemplified with a case study which covers the MDD of an agent-based stock exchange system. Finally, development costs of two approaches are evaluated.

This paper is an extended version of the paper [36]. It differs from the latter by including: 1) a completely new discussion on design and implementation of M2M and M2T transformations required for extending the platform support of a MAS DSML according to the conventional approach 2) an improved case study in which MDD of an agent-based stock exchange is realized by using both the proposed interoperability approach and the conventional approach currently in-use, and finally 3) another new section which covers the comparative evaluation of applying two approaches.

The rest of the paper is organized as follows: In Sect. 2, the approach considering the MAS DSML interoperability is presented. Two agent DSMLs used in this study are briefly discussed in Sect. 3. Sect. 4 discusses how the interoperability can be built between MAS DSMLs while Sect. 5 demonstrates the application of the conventional way of platform support for MAS DSMLs. In Sect. 6, a case study on the development of an agent-based stock exchange system with using both the proposed approach and following the conventional way is given. Sect. 7 includes the comparative evaluation of the two approaches. Related work is given in Sect. 8. Finally, Sect. 9 concludes the paper and states the future work.

2. Proposed approach for the interoperability of MAS DSMLs

As indicated in [36], 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 (such as JADE [31, 37], JACK [30, 38], JADEX [32, 39]) and almost all of them cannot 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 mostly support the execution of modeled agents on just one agent platform (e.g. [17-19, 21, 23]). Very few of them enable the execution of models on two different agent platforms (e.g. [9, 14]) and, as far as we know, there is no any MAS DSML which provides the execution of modeled agents on more than two different agent 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. Taking into account the MDE of software systems in general, our approach also fits into the theory of modeling spaces [40] where model transformations are proposed to bridge two conceptual spaces. In here, the metamodels of different MAS DSMLs can be considered as representing different conceptual spaces and our aim is to bridge these metamodels to support agent platform extensibility for the related DSMLs.

The construction of model transformations between MAS DSMLs and hence re-use of already existing transformations between those DSMLs and agent platforms are depicted in Fig. 1. 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 while each vertical line between a DSML and the MM of an agent platform represents the M2M transformations between this DSML and the agent platform. 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, developers 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 will be more feasible and easier than the vertical transformations since the DSMLs are in the same abstraction level according to MDA [33].

In this paper, we discuss the applicability of the above proposed approach by taking into account the construction of the interoperability between two MAS DSMLs called SEA_ML [23] and DSML4MAS [17]. Both DSMLs enable the 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. Before discussing the details of how the approach is applied over these DSMLs, SEA_ML and DSML4MAS are briefly introduced in the following section.

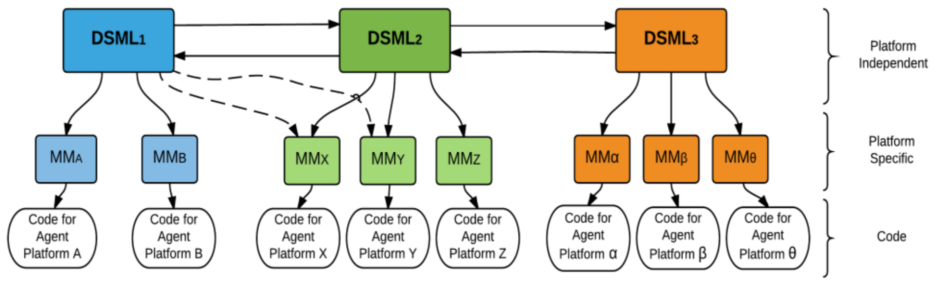

DSML1
DSML2
DSML3
Platform Independent
MMA
MMB
MMX
MMY
MMZ
MMα
MMβ
MMθ
Platform Specific
Code for Agent Platform A
Code for Agent Platform B
Code for Agent Platform X
Code for Agent Platform Y
Code for Agent Platform Z
Code for Agent Platform α
Code for Agent Platform β
Code for Agent Platform θ
Code

Fig. 1. Interoperability of MAS DSMLs via horizontal model transformations

3. Two agent DSMLs: SEA_ML and DSML4MAS

In the following subsections, main language features and metamodels of SEA_ML and DSML4MAS are briefly discussed before constructing the MAS DSML interoperability between them as proposed in this study.

3.1. SEA_ML

SEA_ML [23] is a MAS modeling language which enables the developers to model the agent systems in a platform independent level and then automatically achieve codes and related documents required for the execution of the modeled MAS on target MAS implementation platforms. It 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 [41], like Semantic Web Services (SWS) [42]. 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 [41]. 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 [23, 43]. *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) [44] agent architectures. *Interaction Viewpoint* expresses the interactions and the 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 is 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 [4].

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 [32] agent platform and semantic web service description documents conforming to *Web Ontology Language for Services (OWL-S)* ontology [45]. It is also possible to automatically check the integrity and validity of SEA_ML models [46]. A complete discussion on SEA_ML can be found in [23]. The language and its supporting tool are available in [47].

3.2. DSML4MAS

DSML4MAS [9, 17] 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. *Multi-agent 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 the 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.

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 [9]. For instance, SWS integration into the system models conforming to DSML4MAS is provided via introducing the SOAEnvironment entity [48] 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 [49] based on the aforementioned PIMM (PIM4Agents) and an operational semantics for the execution of modeled agent systems on JACK [30] or JADE [31] agent platforms. Extensions to the language introduced in [48] provide the description of the services inside an agent environment according to specifications such as *Web Services Modeling Language (WSML)* [50] or *Semantic Annotation of WSDL and XML Schema (SAWSDL)* [51]. Interested readers may refer to [48] and [9] for an extensive discussion on DSML4MAS. The language is available with its modeling tools in [52].

4. Building the interoperability between SEA_ML and DSML4MAS with horizontal model transformations

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 [32] and SWS generation according to the OWL-S ontology [45]. In order to extend its platform support capability, new M2M and M2T transformations should be prepared for each new implementation platform. Let us consider extending execution platforms for SEA_ML agents with another well-known and widely-used MAS execution and deployment platform called JADE [31]. In order to make SEA_ML instances also executable on the JADE platform, definition and implementation of M2M transformations are needed between the abstract syntax of SEA_ML and PSMM of JADE framework. 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 (as will be discussed in Sect. 5). The methodology described above is currently the dominating MAS DSML engineering approach and also the most preferred way of model-driven agent development in AOSE [4, 5, 53]. 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 [38] agent platforms and SAWSDL [51] and WSML semantic service ontologies [50] 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 WSML 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 [22] or [23] 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 (as will be discussed in Sect. 7 of this paper).

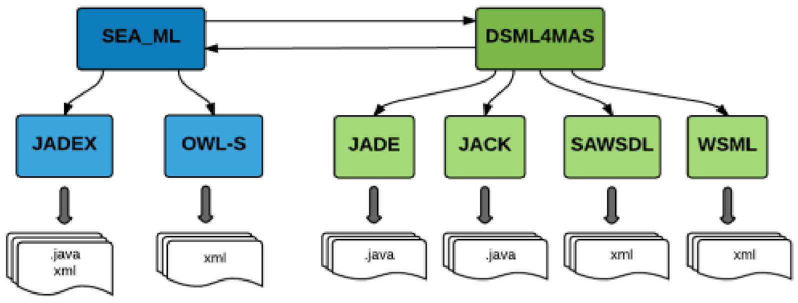


Fig. 2. Interoperability of SEA_ML and DSML4MAS

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.

Table 1. Entity mappings between the metamodels of SEA_ML and DSML4MAS.

SEA_ML MM Entity	DSML MM Entity
SemanticWebAgent (SWA)	Agent
SSMatchmakerAgent	Agent
Role	Actor
SemanticWebService (SWS)	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
SemanticWebOrganization (SWO)	Organization

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 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 [23]. DSML4MAS services are described with Blackbox and Glassbox entities [48]. 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. Therefore, 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.

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 ([34, 54]). In this study, we preferred to use ATL Transformation Language (ATL) [55] 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 [56] 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 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 listing include an excerpt from the written ATL rules in order to give some flavor of M2M transformations provided in this study. To this end, the 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 the Multi-agent viewpoint of DSML4MAS. In line 1, the 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. More examples of the ATL rules written for the required transformations can be found in [36].

```
rule SWSInteractionVP2MultiagentSystem {
  from   swsinteractionvp:
        SWSInteraction!SWSInteractionViewpoint
     to         pim4agent: PIM4Agents!MultiagentSystem (
        agent <- Set
        {SWSInteraction!SemanticWebAgent.allInstances()},
        agent <- Set {SWSInteraction!SSMatchmakerAgent.allInstances()},
        role <- Set {SWSInteraction!Role.allInstances()},
        role <- Set {SWSInteraction!RegistrationRole.allInstances()},
        behavior <- Set {SWSInteraction!SS_AgreementPlan.allInstances()},
        behavior <- Set {SWSInteraction!SS_ExecutorPlan.allInstances()},
        behavior <- Set {SWSInteraction!SS_FinderPlan.allInstances()},
        behavior <- Set {SWSInteraction!SS_RegisterPlan.allInstances()},
        environment <-Set {SWSInteraction!SWS.allInstances()},
        environment <-Set {SWSInteraction!Grounding.allInstances()} )
   }
```

Listing 1. An excerpt from the SWSInteractionVP2MultiagentSystem rule

As it will be demonstrated in Sect. 6, the application of these horizontal model transformations on SEA_ML model instances automatically produces the counterparts of these MAS models according to DSML4MAS specifications. The ATL engine uses the Ecore representation of a SEA_ML MAS model as the input, executes the transformation rules and outputs the corresponding DSML4MAS model again in Ecore format. Produced MAS model is ready to be processed inside the IDE of DSML4MAS. The model can be opened and/or directly utilized in this IDE for the generation of executable codes for JADE or JACK agent platforms.

5. Following the conventional way: Execution of SEA_ML Models on JADE Platform via vertical M2M and M2T transformations

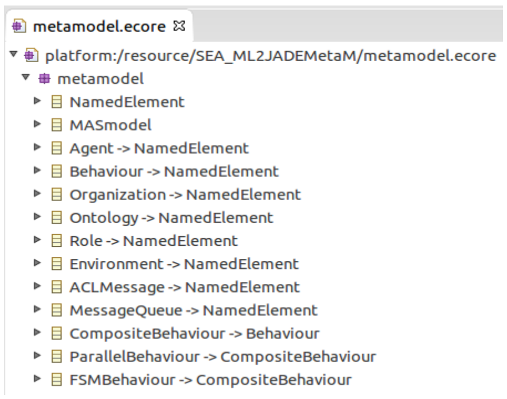
In order to provide a comparison between the new proposed approach and the classical way of platform support for MAS DSMLs, we also designed and implemented direct transformations from SEA_ML instances into the JADE counterparts and realized code generation from the output agent models. This section discusses how the new platform extensibility for SEA_ML can be enabled by a series of vertical M2M and M2T transformations according to the well-known MDA principles [33, 57] and hence it gives some flavor of applying MDD methodology which is currently followed by most of the agent developers to design and implement a DSML with including an operational semantics from scratch.

Execution of any MAS model conforming to an agent DSML requires first a M2M transformation to prepare the counterpart of the model in the targeted agent execution platform. Then a series of M2T transformations are applied on this platform specific model to generate executable software codes and/or files (e.g. [5, 9, 13, 23]). Hence, the first subsection describes how the transformations between SEA_ML and JADE are built while the second subsection discusses code generation from the output JADE model instances.

5.1. M2M Transformations between SEA_ML and JADE platform

Taking into consideration the MDA and its abstraction layers, SEA_ML resides on the platform independent model (PIM) layer and its abstract syntax (discussed in Sect. 3) can be utilized in this work as a PIMM while JADE platform locates on the platform specific model (PSM) layer and its metamodel represents a PSMM. JADE [31, 37] is one of the widely used agent development and execution platforms. It provides an open source Java API, currently distributed by Telecom Italia [58]. The API can be used to implement agents as Java objects. In JADE, agent internals including agent behaviours can be developed according to the IEEE Foundation for Intelligent Physical Agents (FIPA) standards [59]. Moreover, interactions between the software agents can be programmed based on the FIPA Agent Communication Language specifications [60] and MAS platform is supported with agent management and directory facilitator services which are all defined in FIPA standards to manage agents and provide yellow page services for agents to find and communicate with other agents.

After in-depth examination of the JADE API, a general metamodel of JADE platform has been derived and prepared in Ecore format which can be used as a PSMM. Fig. 3 gives an excerpt from this metamodel which reflects the main JADE entities for agent behaviours and messages. As its name denotes, *Agent* is any JADE entity which will be programmed as Java class for platform agents. In addition to one shot behaviours, the *Behaviour*s of agents can be in many types such as *CompositeBehaviour* (a series of behaviours bounded each other with input/output chains), *ParallelBehaviour* (hence concurrent actions of the agent can be modeled) and *FSMBehaviour* (tasks and actions of the agent can be modeled as a finite-state machine (FSM)). Each message between the platform agents can be modeled as *ACLMessage* instances which includes the information on the performative (e.g. INFORM, QUERY, PROPOSE), sender agent, receiver agent, applied conversation protocol, content language, used ontology, etc. All modeled agents, their behaviours and other related entities are covered in a *MASmodel* PIMM entity.



```
metamodel.ecore
▼ platform:/resource/SEA_ML2JADEMetaM/metamodel.ecore
  ▼ metamodel
    ▶ NamedElement
    ▶ MASmodel
    ▶ Agent -> NamedElement
    ▶ Behaviour -> NamedElement
    ▶ Organization -> NamedElement
    ▶ Ontology -> NamedElement
    ▶ Role -> NamedElement
    ▶ Environment -> NamedElement
    ▶ ACLMessage -> NamedElement
    ▶ MessageQueue -> NamedElement
    ▶ CompositeBehaviour -> Behaviour
    ▶ ParallelBehaviour -> CompositeBehaviour
    ▶ FSMBehaviour -> CompositeBehaviour
```

Fig. 3. An excerpt from the Ecore representation of the derived JADE metamodel

Following the derivation of the JADE metamodel as the PSMM in our study, we needed to construct the model transformation rules between SEA_ML PIMM and JADE PSMM. Similar to the method given in Sect. 4, entity mappings between these two metamodels are provided. Table 2 lists some of these entity mappings.

Table 2. Entity mappings between the metamodels of SEA_ML and JADE.

SEA_ML MM Entity	JADE MM Entity
SemanticWebAgent	Agent
SSMatchMakerAgent	DFAgent
SS_AgrementPlan	FSMBehaviour
SS_ExecutorPlan	ParallelBehaviour
SS_FinderPlan	OneShotBehaviour
SS_RegisterPlan	CyclicBehaviour
Role	SimpleBehaviour
OntologyMediatorRole	CompositeBehaviour
RegistrationRole	AgentManagementSystem (AMS)
RoleOntology	Ontology
OrganizationOntology	Ontology
ServiceOntology	Ontology
SemanticWebOrganization	Agent Platform
Environment	Environment

As seen from the table, SEA_ML SemanticWebAgent is mapped to JADE Agent as expected. SSMatchmakerAgent is mapped to DFAgent. SSMatchMaker agent is responsible for finding appropriate services which is needed by other agents in a MAS system, similar to DFAgent (Directory Facilitator Agent) on the JADE platform. Also SEA_ML SS_AgreementPlan, SS_ExecutorPlan, SS_FinderPlan and SS_RegisterPlan are mapped to FSMBehaviour, ParallelBehaviour, OneShotBehaviour and CyclicBehaviour respectively based on the characteristics of these plans on semantic service discovery, engagement and execution features of agents in SEA_ML and similarity of tasks defined in the corresponding JADE entities.

Moreover, Agents have roles and use ontologies to accomplish their duties in SEA_ML. A SEA_ML Role can be simply described as a JADE SimpleBehaviour while a specialization of role, SEA_ML OntologyMediatorRole is mapped to JADE CompositeBehaviour since it handles the ontologies with a more complex duty. SEA_ML RegistrationRole, which registers agents and their services, is mapped to JADE AgentManagementSystem (AMS) entity since AMS is responsible for managing a JADE agent platform with including the determination of agent statuses and registering/deregistering of agents. SEA_ML RoleOntology, ServiceOntology and OrganizationOntology are mapped to JADE Ontology entity as expected. In both sides, Ontology holds necessary information as a knowledgebase for the environment. SEA_ML Environment holds the non-agent resources for the agents and this can be mapped to a configuration file on JADE side, where access configurations for external sources are stated. Finally, SEA_ML SemanticWebOrganization entity models a MAS with including all SWAs, their goals and plans, so it matches well with JADE Agent Platform entity which possesses the similar features for platform agents living together.

```
rule SWSInteractionVP2MASmodel{
  from
    swsinteractionvp: SWSInteraction!SWSInteractionViewpoint
 to
    jademm: metamodel!MASmodel(
      hasAgent<- Set{SWSInteraction!SemanticWebAgent.allInstances()},
      hasDFAgent<- Set{SWSInteraction!SSMatchmakerAgent.allInstances()},
      hasFSMBehaviour <- Set{SWSInteraction!SS_AgreementPlan.allInstances()},
      hasParallelBehaviour <- Set{SWSInteraction!SS_ExecutorPlan.allInstances()},
      hasOneShotBehaviour <- Set{SWSInteraction!SS_FinderPlan.allInstances()},
      hasCyclicBehaviour <- Set{SWSInteraction!SS_RegisterPlan.allInstances()},
      hasSimpleBehaviour <- Set{Ontology!Role.allInstances()},
      hasCompositeBehaviour<-Set{Ontology!OntologyMediatorRole.allInstances()},
      hasOntology <- Set{Ontology!RoleOntology.allInstances()},
      hasOntology <- Set{Ontology!OrganizationOntology.allInstances()},
      hasOntology <- Set{Ontology!ServiceOntology.allInstances()},
      hasAgentPlatform <-Set{MASandOrg!SemanticWebOrganization.allInstances()},
      hasEnvironment <- Set{MASandOrg!Environment.allInstances()},
      hasAgentManagementSystem<-Set{SWSInteraction!RegistrationRole.allInstances()}
     )
 }
```

Listing 2. An excerpt from the SWSInteractionVP2MASmodel rule

After determination of the entity mappings, M2M transformation rules according to these mappings are written in ATL. This time the source metamodel which is used by the ATL engine will be SEA_ML metamodel while the target metamodel is JADE metamodel. Rules are executed on SEA_ML model instances to generate PSMs conforming to JADE specifications. In the following, some examples from the prepared model transformation rules are given. The first example is an excerpt which demonstrates a union rule (see Listing 2). Since SEA_ML is designed with multiple viewpoints, all necessary elements are united under a MAS viewpoint on target JADE model. The rule transforms all SemanticWebAgents instances encountered in a SEA_ML model into JADE Agents with including all its plans and ontologies.

An excerpt from the ATL rule which provides the transformation of SEA_ML agent-SWS agreement plan into a JADE FSM behavior is shown in Listing 3. Following this rule, a helper rule which is used by this rule is also given in Listing 4. setAgreementPlanName helper rule is called by the AgreementPlan2FSMBehaviour ATL rule to control “name” attribute of the SEA_ML SSAgreementPlan instance and set the default name for this attribute in case of it is not specified in the source model.

```
rule AgreementPlan2FSMBehaviour {
  from
     Agreementplan: SWSInteraction!SS_AgreementPlan
  to
     jBehaviour: metamodel!FSMBehaviour (
         name <- Agreementplan.setAgreementPlanName()
     )
}
```

Listing 3. An excerpt from the AgreementPlan2FSMBehaviour

```
helper context SWSInteraction!SS_AgreementPlan def:
         setAgreementPlanName():  String =
  if (self.name = thisModule.controlString or self.name.oclIsUndefined() ) then
     'SS_AGREEMENT_PLAN_NAME_IS_EMPTY'
  else
     self.name
  endif;
```

Listing 4. An excerpt from the setAgreementPlanName helper rule

5.2. M2T Transformations for code generation from JADE PSMs

In the interoperability approach we followed in Sect. 4, we did not need to worry about constructing the way of producing executables of the SEA_ML model instances on JADE platform since we benefited from ready-to-use code generation features already provided inside the DSML4MAS environment. However, this time, we should design and implement all M2T transformations for JADE model instances to generate artifacts executable inside the JADE platform. For this purpose, we prepared a series of M2T transformations by using Xpand language [61] which enables code generation from EMF models. Since the metamodel we derived for JADE is already encoded in Ecore (see Sect. 5.1), we can apply our Xpand rules on model instances conforming this metamodel to generate JADE Java classes. That completes the MDA we designed for the execution of SEA_ML models on JADE platform: SEA_ML instance models can be transformed into their JADE counterparts by executing the ATL rules discussed in Sect. 5.1 and then output of this transformation, which is a JADE model instance, can be processed with the prepared Xpand M2T rules to generate Java classes for this JADE model instance. Following Xpand snippets give some flavor of the implemented M2T rules. As seen in Listing 5, the reference model is imported first. Here, it is indicated which element from the JADE instance model is going to be used to create which target element in the text part. For example, if instance model has *hasAgent* element than it is going to be defined by counter element Agent Java class. In Listing 6, an excerpt from the template for definition of Java class for each Jade Agent element is given. Attribute values of the corresponding class are set and some parts of the methods are generated as the result of executing the related Xpand rule on the proper JADE model instance.

```
«IMPORT metamodel»
«DEFINE main FOR MASmodel»
«EXPAND agent FOREACH hasAgent»
«EXPAND dfagent FOREACH hasDFAgent»
«EXPAND parallelbehaviour FOREACH hasParallelBehaviour»
«EXPAND oneshotbehaviour FOREACH hasOneShotBehaviour»
«EXPAND cyclicbehaviour FOREACH hasCyclicBehaviour»
«EXPAND compositebehaviour FOREACH hasCompositeBehaviour»
«EXPAND ontology FOREACH hasOntology»
«EXPAND ams FOREACH hasAgentManagementSystem»
«EXPAND agentplatform FOREACH hasAgentPlatform»
«ENDDEFILE»
```

Listing 5. Xpand code snippet to parse a JADE model instance to set target elements in the text

```
«DEFINE agent FOR Agent»
«FILE name + ".java"»
 import jade.core.*;
 public class «name» extends Agent {
     /*constructor definiton*/
     public «name»(DataStore ds) {
         super ();
         this.ds=ds;
     }
     /*constructor definition*/
     public «name» () {
         super();
         this.ds=new DataStore ();
     }
     /* Agent initializations */
     protected void setup () {  }
}
```

Listing 6. An excerpt from the Java class template for Jade Agent elements

6. Case Study

In this section, the use of the proposed MAS DSML interoperability approach is demonstrated for the development of an agent-based stock exchange system which will be deployed on the JADE platform. The system-to-be-used is modeled in SEA_ML and transformed to a DSML4MAS instance by applying the method described in Sect. 4 in order to use the generation power of DSML4MAS language. In this way, the implementation of this system's agents on JADE (or JACK) platform can be possible by using the operational semantics of DSML4MAS which is already provided for the execution of agents and the generation of semantic web services (see Fig. 2). In the second part of the case study, we also exemplified the use of the direct transformations constructed between SEA_ML and JADE within the scope of applying the conventional approach (discussed in Sect. 5). Hence, it will be possible to evaluate and compare the new interoperability approach with the traditional model-driven agent development. In the first subsection, the general architecture of the agent-based stock exchange systems and their modeling with SEA_ML are briefly introduced. Following subsections discuss the development of the MAS with the interoperability between SEA_ML and DSML4MAS and direct transformations from SEA_ML to JADE respectively.

6.1. Agent-based Stock Exchange Systems

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 Managers (SSMatchmaker agent instances) 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 [62].

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. 4 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 the limits and scope of this paper. Discussion on the whole model can be found in [23] and the sources of the model pertaining to the case study are all available at the SEA_ML's distribution website [47].

We can see from the instance model given in Fig. 4 that an investor agent (e.g. InvestorA) plays the Buying role and applies its StockFinder plan for finding an appropriate Trading service interface of one TradingService SWS in order to buy some stocks. This plan enables the discovery by interacting with the TradeManager SSMatchmakerAgent which registers the services by applying the StockRecorder plan. 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 is made through the Trade interface of the SWS. Finally, if the result of the negotiation is positive, the agent applies the StockOrder plan to call the TradingFloor of the SWS by executing its Exchange process and using its TradeAccess grounding with which the service is realized. In a similar way, Investor agents can 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.

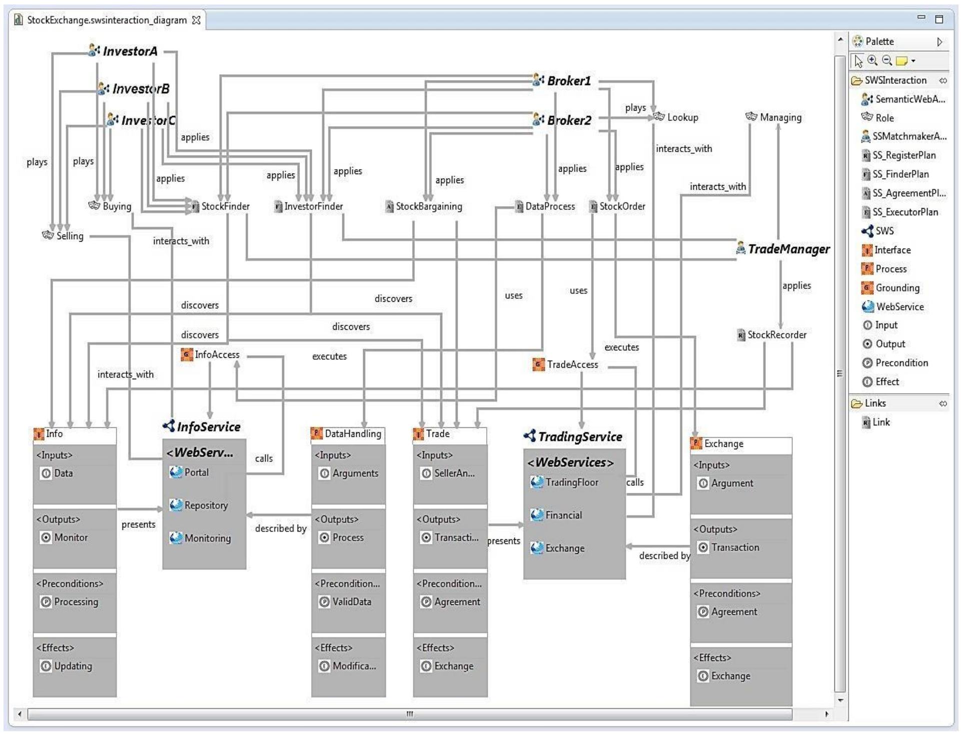


Fig. 4. Instance model of the multi-agent stock exchange system in SEA_ML with including the agents, semantic web services and their relations

6.2. Use of the interoperability between SEA_ML to DSML4MAS

The designed instance MAS model described in the previous subsection is controlled based on the provided constraint rules in SEA_ML tool to check its validity. Now, we can benefit from the interoperability provided between SEA_ML and DSML4MAS to enable the modeled MAS executable on the JADE platform. The horizontal model transformations discussed in Sect. 4 are executed on this SEA_ML instance model and as result; we succeed 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.

The generated model conforms to the specification of DSML4MAS's abstract syntax, so it can be handled with DSML4MAS's graphical editor [49]. 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 the 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 since DSML4MAS already owns a chain of M2M and M2T transformations for these agent execution platforms and service ontologies as discussed in Sect. 3.2.

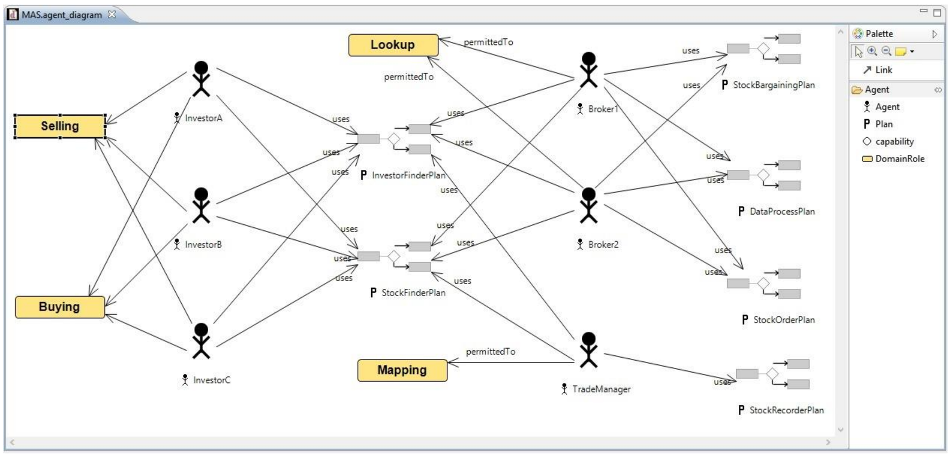


Fig. 5. Partial instance model of the agent-based stock exchange system in DSML4MAS achieved after application of the defined M2M transformations

6.3. Use of direct transformations between SEA_ML and JADE

The second way of implementing the same modeled agent-based stock exchange system on the JADE platform is to employ the vertical M2M and M2T transformations introduced in Sect. 5. At first, we need to automatically produce the corresponding JADE model of the same MAS currently modeled in SEA_ML. For this purpose, Ecore representations of SEA_ML, JADE and the instance MAS model of the stock exchange system are all given to the ATL engine and M2M transformation rules are executed on the SEA_ML instance model to achieve the counterpart instance model conforming to the JADE metamodel. Hence, the JADE model of all investor and broker agents in the stock exchange system with including their knowledgebases and other attributes can be automatically produced. An excerpt from the output XMI model is given in Listing 7.

The output of the above M2M transformation will be the input of the next vertical transformation which is a M2T transformation providing the automatic code generation for the JADE platform. As discussed in Sect. 5.2, M2T rules, we defined with using Xpand, can be executed on a JADE instance model to generate executable artifacts. When we apply those transformations on the JADE model of our stock exchange system, template codes for the Java classes for each agent in the system are automatically generated by parsing the instance model, determining each JADE model entity instance and producing Java codes described in the appropriate Xpand rule. For example, Listing

8 includes a code snippet from the Java class generated for the investor agent, called Investor A.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<metamodel:MASmodel xmi:version="2.0"
            xmlns:xmi="http://www.omg.org/XMI"
   xmlns:metamodel="http://JADEmetamodel">
       <hasAgent name="InvestorC"/>
       <hasAgent name="Broker2"/>
  <hasAgent name="InvestorB"/>
  <hasAgent name="InvestorA"/>
  <hasAgent name="Broker1"/>
  <hasOntology name="SellingAndBuyingRolesOnto"/>
  <hasOntology name="StockUserOrgOnt"/>
  <hasOntology name="SearchServiceOnt"/>
   …
```

Listing 7. An excerpt from the target JADE instance model

```
import jade.core.*;
public class InvestorA extends
            Agent {
    /*constructor definiton*/
    public InvestorA(DataStore ds){
        super ();
         this.ds=ds;
    }
    /*constructor definition*/
    public InvestorA(){
        super ();
        this.ds=new DataStore();
    }
    /* Agent initializations */
    protected void setup () { }
}
```

Listing 8. A snippet from the template JADE code generated for a system agent

7. Evaluation

An evaluation of the proposed MAS DSML interoperability approach was performed in this study by taking into account the language developers' perspective. Although the discussion given in the previous sections shows the applicability and effectiveness of the interoperability approach in the way of extending the execution platform support of

MAS DSMLs, we believe that some kind of comparative evaluation may help clarifying the feasibility of choosing the interoperability approach instead of the conventional one, i.e. design and implementation of separate M2M and M2T transformations for each new agent execution platform. For this purpose, we adopted the evaluation framework proposed in [35] which provides the systematic assessment of both the language constructs and the use of agent DSMLs according to various dimensions and criteria. To the best of our knowledge, the work in [35] presents the current unique evaluation framework which is specific to the MAS DSMLs and guides the assessment of model-driven agent development methodologies in general. However, since the scope of our evaluation in this study is limited mainly with model transformations in different abstractions levels and does not cover the evaluation of a full-fledged MAS DSML, only the dimensions called *model transformations* and *development* and the evaluation criteria pertaining to these dimensions called *M2M*, *M2T*, *Overall (output) performance ratio* and *development time* defined in [35] are taken into consideration. Furthermore, we revisited these dimensions and criteria in order to make them more meaningful and appropriate for our quantitative evaluation; and we separated our evaluation into two parts in which the related dimensions and metrics are included, namely Time Evaluation and Development Effort Evaluation.

A group of four software developers was employed during this evaluation. All of the evaluators were graduate students in computer science: one of them was a PhD candidate while remaining three evaluators were M.Sc. students. All group members had experience on software modeling and development of agent systems ranging from 2 to 4 years. In addition, two of the evaluators were also working as software engineer in industry for 2 years on average at the time of this study was realized. All group members passed related graduate courses in their master or PhD program, including Agent-oriented Software Development, Multi-agent Systems and Model-driven Software Engineering which are taught in Computer Engineering Department and International Computer Institute of Ege University, Turkey. All evaluators were familiar with Eclipse environment and skilled on Java programming language. They also had knowledge and practical experience on using MDD technologies like ATL, MOFScript, Xpand, Acceleo earned from above listed courses and previous projects.

The evaluation was performed both for 1) the interoperability provided between SEA_ML and DSML4MAS via horizontal model transformations and 2) definition and implementation of vertical M2M and M2T transformations directly between SEA_ML and JADE. In the remaining of the discussion, the former approach is shortly referred as the interoperability approach while the latter is referred as the conventional approach. It is worth stating that the evaluators worked individually for the application of these two approaches and both elapsed times and development throughputs (e.g. number of written M2M rules) were recorded for each evaluator and for each approach separately. Average times and throughputs were calculated for each phase of the development required during the application of interoperability or conventional approach by considering all evaluators' development processes. These average results gained from abovementioned time and development effort evaluations are reported in Sect. 7.1 and 7.2 respectively. Discussion on these results of the conducted evaluation is given in Sect. 7.3.

7.1. Time Evaluation

Time evaluation consists of measuring, analyzing and comparing the time elapsed for the design and the implementation of transformations required for each approach.

As it is discussed in Sect. 4, building the interoperability between two MAS DSMLs, SEA_ML and DSML4MAS includes the determination of entity mappings between two DSMLs' metamodels (which are in the same abstraction level) and writing the horizontal model transformations according to these mappings. Hence, the horizontal transformations between SEA_ML PIMM and DSML4MAS PIMM are realized by using ATL transformation language in four steps:

1. *Analyzing the source PIMM, SEA_ML metamodel*: This analysis is performed by the developer of the transformations to comprehend and infer on the source metamodel by considering the MAS features and elements.
2. *Analyzing the target PIMM, DSML4MAS metamodel*: Similar to step 1, the language developer also needs to analyze the metamodel of the target language to be able to determine main language entities and their relations.
3. *Mapping*: In this step, the language developer maps the meta elements in the source PIMM (SEA_ML) with the meta elements of the target PIMM (DSML4MAS) in a way that the semantics of mapped elements are representing similar concepts and associations in the domain (see Table 1). Mappings can be in m:n manner.
4. *Implementation*: Based on the defined entity mappings, M2M transformation rules and supporting helper rules are written by using ATL. Hence source models conforming to SEA_ML PIMM can be transformed into target DSML4MAS instance models by executing these transformations on ATL engine. This step also contains the test procedure of all written rules.

Average times spent by the evaluators for each of the abovementioned steps are calculated for time evaluation of the interoperability approach as shown in Table 3.

Table 3. Cost of building horizontal transformations for the interoperability approach

Step	Analysis for source PIMM (SEA_ML)	Analysis for target PIMM (DSML4MAS)	Determination of entity mappings	Implementation of M2M transformations	Total
Average Elapsed Time (in hours)	8	7	2	4	21

On the other hand, as discussed in Sect. 5, each developer (evaluator) needs to design and implement two types of vertical transformations for extending the execution support of SEA_ML on JADE platform in case of following the conventional approach:

A) PIM to PSM transformation between SEA_ML PIMM and JADE PSMM.

B) PSM to Code transformation for code generation from instance MAS models conforming to JADE PSMM.

The steps of preparing the vertical M2M transformations for type A are similar to the steps of providing horizontal transformations of the interoperability approach:

A) PIM to PSM transformation between SEA_ML PIMM and JADE PSMM

1. *Analyzing the source PIMM, SEA_ML metamodel*: This analysis is performed by the developer of the transformations to comprehend and infer on the source metamodel by considering the MAS features and elements.
2. *Analyzing JADE platform and derivation of JADE metamodel as a PSMM*: Unlike SEA_ML and DSML4MAS, the JADE MM needs to be prepared from scratch.
3. *Mapping*: In this step, the language developer maps the meta elements of the source PIMM (SEA_ML) with the meta elements of the target PSMM (JADE) in a way that the semantics of mapped elements are representing similar concepts and associations in the domain (see Table 2). Mappings can be in m:n manner.
4. *Implementation*: Based on the defined entity mappings, M2M transformation rules and some supporting helper rules are written by using ATL. Hence source models conforming to SEA_ML PIMM can be transformed into target JADE instance models by executing these transformations on ATL engine. This step also contains the test procedure of all written rules.

Average times spent by the evaluators for each of the abovementioned steps are shown in Table 4.

Table 4. Cost of building vertical transformations between SEA_ML PIMM and JADE PSMM

Step	Analysis for source PIMM (SEA_ML)	Derivation of target PSMM (JADE)	Determination of entity mappings	Implementation of M2M transformations	Total
Average Elapsed Time (hours)	8	16	3	6	33

For the conventional approach, the evaluators should also provide the M2T transformations for code generation from JADE instance models. Followings are the steps required for this transformation.

B) PSM to Code transformation for code generation from instance MAS models conforming to JADE PSMM

1. Analyzing JADE API for required Java class structures
2. Design of code templates for JADE PSMM meta-entities
3. *Implementation*: A series of M2T transformations are written by using Xpand (see Sect. 5.2).

Average times spent for each of the abovementioned steps are shown in Table 5.

Table 5. Cost of building vertical transformations for code generation from instance MAS models conforming to JADE PSMM

Step	Analysis for the agent platform API	Design of code templates	Implementation of M2T transformations	Total
Average Elapsed Time (in hours)	2	3	5	10

The figures presented in the abovementioned tables will be used in Sect. 7.3 to compare the interoperability approach with the conventional one.

7.2. Development Effort Evaluation

In this part, the development required both for the interoperability approach and the conventional approach is evaluated by comparing the number of rules, helper rules and templates as the main building blocks of the transformations including PIM to PIM, PIM to PSM and PSM to Code.

In the interoperability approach, each evaluator only needed to write horizontal M2M transformation rules in ATL which are required for the transformation between SEA_ML and DSML4MAS. The related figures for average numbers of rules and helper rules and average total number of line of codes (LoC) pertaining to these transformation rules are given in Table 6.

Table 6. Specification of the horizontal M2M transformations for the interoperability approach

Item	M2M Rules	M2M Helper Rules	Total LoC
Average Quantity	8	11	200

In the conventional approach, each evaluator needed to write vertical M2M transformation rules in ATL which are required for the transformation between SEA_ML and JADE. Moreover, templates for the generation of codes from JADE instance models were also needed to be written in Xpand. The related figures showing average quantities of rules, helper rules, templates and LoC are shown in Table 7.

Table 7. Specification of the vertical M2M and M2T transformations for conventional approach

Phase	PIM to PSM (SEA_ML to JADE)			Code Generation	
Item	M2M Rules	M2M Helper Rules	Total LoC	Templates	Total LoC
Average Quantity	15	17	240	12	316

The analysis and comparison of these figures are discussed in the next section.

7.3. Discussion

In this section, average results gained from time and development effort evaluations (Sect. 7.1 and 7.2 respectively) are discussed. To ease analysis, time evaluation results are shown in a bar chart (see Fig. 6) and development effort evaluation results are shown in another bar chart (see Fig. 7).

Based on the result figures given in Fig. 6, the followings can be deduced:

- Average times elapsed for the analysis of source PIMMs in both approaches are equal. This is expected since this step consists of the efforts for analyzing the same PIMM (SEA_ML MM).
- Analyzing the target PIMM (DSML4MAS) took a bit less time than the first step in the interoperability approach since abstract syntax of SEA_ML is more complicated with including a detailed Agent-SWS interaction viewpoint comparing with DSML4MAS. However, analyzing the target PSMM (JADE MM) in the conventional approach took more than two times (16 hours) comparing with the corresponding PIMM analysis in the interoperability approach. Main reason of this extra cost encountered in the conventional approach is each evaluator’s need for examining the whole JADE platform first and then derive its metamodel in Ecore format to be used during the model transformations whereas each evaluator just needed to analyze an already available PIMM (metamodel of DSML4MAS) in the interoperability approach. Another reason of this cost in the conventional approach is the necessity to work in different abstraction levels according to MDA while working in the platform independent level is sufficient in the proposed interoperability approach.
- Mapping and Implementation steps of the interoperability approach took also less time than the corresponding steps in the conventional way on average. Average time elapsed for the MAS entity mappings between SEA_ML and DSML4MAS in the interoperability approach (2 hours) is less than the average time needed for setting the mappings between SEA_ML and JADE (3 hours). That difference is also another result of working in different abstraction levels in the conventional approach. Since they are in the same abstraction level, concepts and relations defined in SEA_ML and DSML4MAS are closer to each other and it is relatively easy for evaluators to set mapping between these concepts. However, each evaluator should deal with setting mappings between SEA_ML and JADE which are in different abstraction levels.

Moreover, there is an additional cost of the conventional approach comparing with the interoperability approach: allocating time for building vertical transformations for code generation from instance MAS models conforming to JADE PSMM. That process includes the analysis for JADE API, design of code templates and the implementation of M2T rules.

Considering the average total cost of vertical transformations constructed in the conventional approach, about 76.7% of the cost comes from PIMM to PSMM transformations (33 hours) and the rest, about 23.3% comes from PSMM to Code transformations (10 hours) which is not required in the interoperability approach.

When we compare the time cost of developing the horizontal transformations in the interoperability approach with the vertical ones in the conventional approach to provide the platform extensibility of a MAS DSML, we can see that the average grand total of time needed for the interoperability (21 hours) is approximately half (about 48.8%) of the average grand total time needed for the conventional approach (43 hours). That is because the proposed interoperability approach benefits from the already provided M2T transformations and only needs the construction of M2M transformations between two MAS DSMLs (in our case, SEA_ML and DSML4MAS) while in the conventional approach, it is required to prepare both 1) M2M transformations between a MAS PIMM and agent platform PSMM (in our case SEA_ML and JADE) and 2) M2T transformations for code generation.

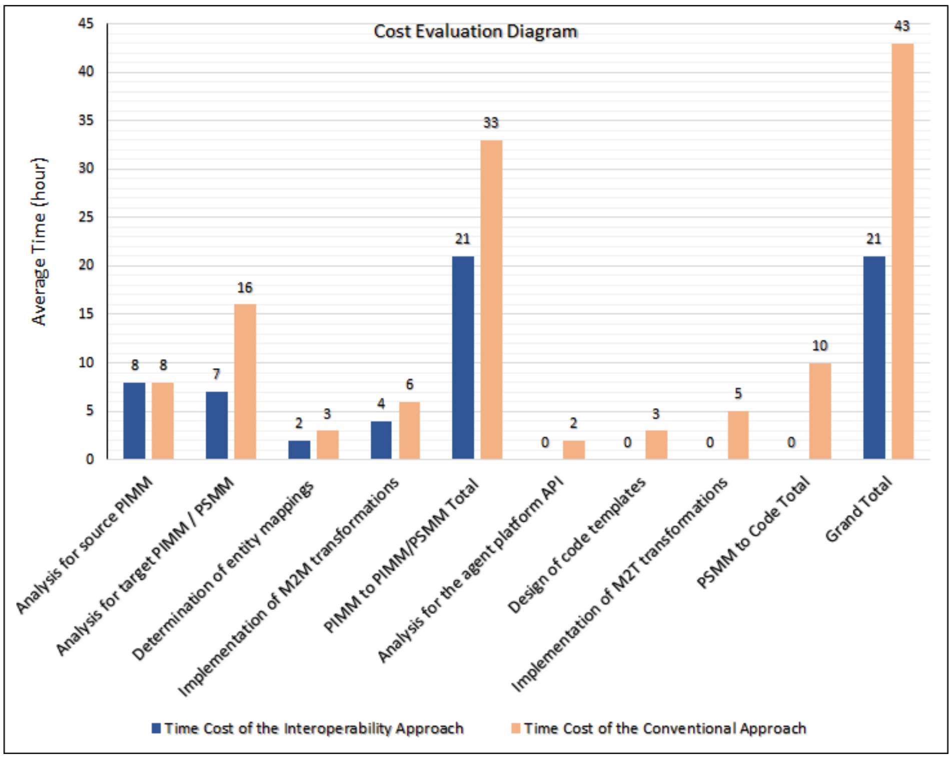


Fig. 6. Demonstration of average time evaluation results for both the interoperability and the conventional approaches

When we consider comparing only the M2M transformations required for both approaches instead of comparing the grand total efforts, we can see that horizontal M2M transformations in the interoperability needs 21 hours on average which is about 36.4% less than the time required for providing the vertical M2M transformations of the conventional approach (33 hours). Therefore, even in the case that the conventional approach has no need for M2T transformations to be developed, the interoperability approach would be still advantageous. The reason of this difference is clear: working in the same abstraction level for MAS modeling (in the interoperability approach) takes less development time for M2M transformations comparing with the overhead of preparing M2M between agent models residing at the different abstraction levels (as in the case of the conventional approach).

Considering the evaluation of the development effort given for the application of each approach, the average cost figures for both the interoperability and the conventional approaches are shown inside a bar chart (Fig. 7).

According to the figures given in Fig. 7, the average number of M2M rules and helper rules for the interoperability approach are much less than those of the conventional approach. The reason is that the horizontal transformations written by the evaluators for the interoperability are only between the metamodels of two MAS DSMLs while the vertical transformations of the conventional approach consist of both PIM to PSM and PSM to Code transformations.

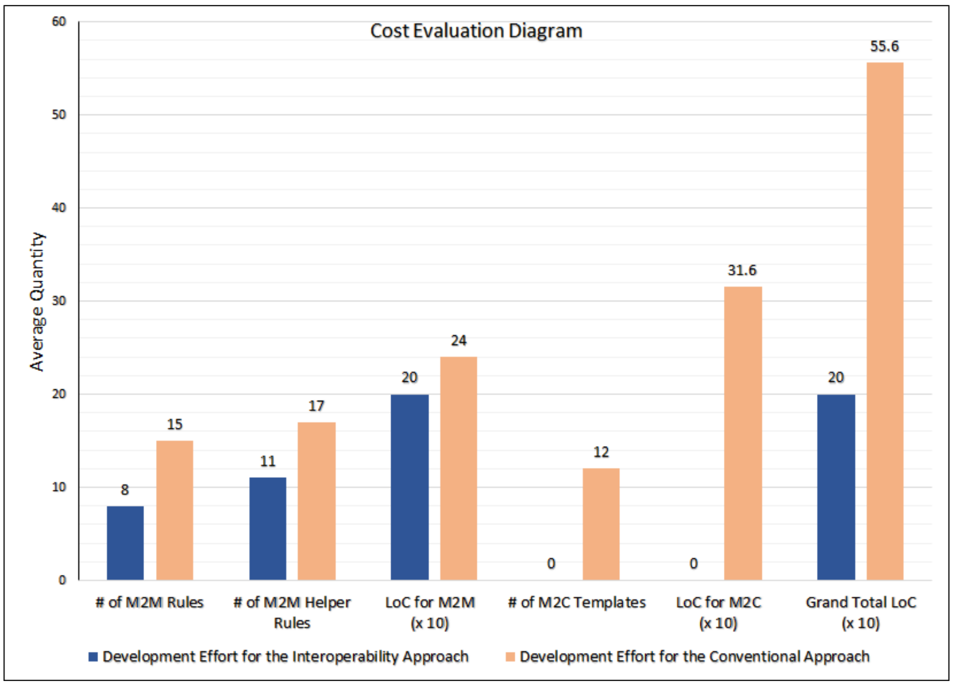


Fig. 7. Demonstration of the development effort evaluation results on average for both the interoperability and the conventional approaches

As we discussed earlier in this section, the conventional approach has an additional transformation phase of M2T. Comparing the total LoC developed for both approaches, we can see that the application of interoperability approach with 200 LoC on average needs about 64% less coding effort for transformations against the conventional approach where 556 LoC are written on average. The main reason is that M2T transformation for code generation is very close to the target agent platform, so it has lots of domain details that need to be coded in the templates. However, the interoperability approach does not need this part of transformation since it is already available for the PIMM of the target MAS DSML.

Taking into consideration the threats to validity of this study, the first threat can be the number of the evaluators which may have an effect upon the generalization of the findings. However, software developers especially with practical experience on the application of both agents and MDD comparing with most of the other domains are very rare. In addition, the evaluators need to have knowledge and experience of languages, tools and frameworks for constructing transformation rules both for the interoperability and the conventional approaches. Although these limitations caused us to have a relatively small size of evaluators, we think that our sample was still sufficient for measuring the variability in such a dedicated field. Moreover, the conducted study aimed at performing evaluation according to the perspective of MAS DSML developers instead of MAS DSML users (agent developers) since the work herein mostly includes language implementation rather than language use. Within this context, the number of the evaluators employed in this study is again satisfactory since the number of MAS DSML developers constitutes a small amount inside the total number of MAS DSML users currently available.

In the execution phase of the evaluation, application of both approaches (interoperability and conventional) can be realized with a combination of single group/two groups and one problem domain/two different domains. In this study, we preferred to use a single group for the evaluation of both approaches instead of employing two groups with different evaluator profiles, in which, e.g. the first group experiences the application of the interoperability approach while the second group deals with applying the conventional approach. Choosing one single evaluator group may cause another threat to the validity. Using two different evaluator groups can be a good option where methods and implementation technologies required for each assessment case vary for the groups and comparison of these variations present the key point of the conducted evaluation. One such example of utilizing two separate groups for the assessment of a MAS DSML can be found in our previous work [35]. However, implementation methods and used language frameworks / technologies are so similar for the comparative evaluation of interoperability and conventional approaches in this study. For instance, the way of developing model transformations is almost same for both approaches in the conducted evaluation: evaluators implemented horizontal transformations in the interoperability approach while vertical transformations were constructed in the conventional approach with using the same Eclipse framework and ATL. Hence, we benefited from using a single group of evaluators having the same level of knowledge and experience in fair comparison of two approaches.

Finally, one may find the demonstration of applying the proposed horizontal approach with a single case study (discussed in Sect. 6) as an additional threat to validity since generation throughputs of employing both interoperability and conventional approaches probably differ in developing real agent systems for various business domains. However, the evaluation performed in this study mainly considers the language implementation and we investigate how an interoperability between MAS DSMLs facilitates the construction of both a model-driven MAS development process and its supporting tools for DSML developers. The evaluators in our work play a MAS DSML developer role more than a MAS DSML user role. Hence, we believe that size, complexity or type of the case study selected for exemplifying the use of the proposed approach is not critical and does not directly affect the achieved results of evaluating interoperability and conventional approaches within the scope of implementing DSML-based development processes as given in this study.

8. Related Work

In the last decade, AOSE researchers have significant efforts on using model-driven approaches for agent development and the derivation of DSLs / DSMLs for MAS. For instance, Agent-DSL [16] was used to specify the agency properties that an agent needs to accomplish its tasks. However, the proposed DSL was presented only with its metamodel and provided just a visual modeling of the agent systems according to agent features, like knowledge, interaction, adaptation, autonomy and collaboration. Likewise, in [63], the authors introduced two dedicated modeling languages and called these languages as DSMLs. These languages were described by metamodels which can be seen as the representations of main concepts and relationships identified for each of the particular domains again introduced in [63]. The study included only the abstract syntaxes of the related DSMLs and did not give the concrete syntaxes or semantics. In fact, the study only defined generic agent metamodels for MDD of MASs. The work in [14] presented a methodology based on OMG's MDA [33] for modeling and implementing agent and service interactions on the Semantic Web. A PIMM for MAS and model transformations from instances of this PIMM to two different MAS deployment platforms were discussed in this study. But neither a DSML approach nor semantics of service execution was covered in the study.

As previously discussed in this paper, Hahn [17] introduced a DSML for MAS called DSML4MAS. The abstract syntax of the DSML was derived from a platform independent metamodel [9] 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 [49]. Furthermore, DSML4MAS supports the deployment of modeled MASs both in JACK and JADE agent platforms by providing an operational semantics over model transformations. Combination of these studies [9, 17, 49] are important because they provided the construction of probably the first complete DSML for agents with all of its specifications and guided MDD of agent applications. For instance, Ayala et al. [64] used DSML4MAS for the development of agent-based ambient intelligence systems. The metamodel of DSML4MAS was employed as a source metamodel to support the modeling of context aware systems and conforming models were transformed into target models which are instances of an aspect-oriented agent metamodel called Malaca. Code generation enabled the implementation of Malaca models to run in the ambient intelligence devices.

Another DSML was provided for MASs in [21]. The abstract syntax was presented using the Meta-object Facility (MOF) [65], the concrete syntax and its tool was provided with Eclipse Graphical Modeling Framework (GMF) [66], and finally the code generation for the JACK agent platform was realized with model transformations using Eclipse JET [67]. 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 [68]. A similar study was performed in [18] 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 offered a related MDD tool based on a specific MAS development methodology called INGENIAS [69].

Originating from a well-formalized syntax and semantics, Ciobanu and Juravle defined and implemented a language for mobile agents in [20]. 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. The work conducted in [24] 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 were 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.

Wautelet and Kolp [70] investigated how a model-driven framework can be constructed to develop agent-oriented software by proposing strategic, tactical and operational views. Within this context, they introduced a Strategic Services Model in which strategic agent services can be modeled and then transformed into the dependencies modeled according to the well-known i* early phase system modeling language [71] for a problem domain. In addition, generated i* dependencies can be converted to BDI agents to be executable on appropriate agent platforms such as JACK [30] and JADEX [32]. However, implementation of the required transformations and code generation were not included in this study. Another work for model-driven development of BDI agents [72] introduced a metamodel for the definition of entities and relations pertaining to Jason BDI architecture [73]. The work only consisted of a metamodel and a graphical concrete syntax for this metamodel. Generation of executable artifacts was not included in the study.

In a recent work [74], a metamodel, describing some modelling units and constraints, was introduced in order to identify the real time requirements of a MAS during the analysis phase of the development. Hence, the requirement analysis was supported with a model-driven approach to determine real-time tasks. Bergenti et al. [75] proposed a DSL, called JADEL, for the MDD of agents on JADE platform. Instead of covering all features of JADE, JADEL only provided high-level agent-oriented abstractions, namely agents, behaviours, communication ontologies, and interaction protocols. JADEL was supported with a compiler which enabled source code generation for implementing agents on JADE platform. However, the related code generation feature of JADEL is not functional enough to fully implement JADE agents as also indicated by the authors in [75].

Finally, by considering our previous studies, in [19] and [22], 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 [23]. 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 [46] and discussed how the applied methodology can pave the way of evolutionary language development for MAS DSLs [4]. Moreover, qualitative evaluation and quantitative analysis of SEA_ML have been recently performed over a multi-case study protocol [35].

The work presented in this paper contributes to the abovementioned 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 the 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 only the work conducted in [15] considers the application of horizontal transformations for agent domain apart from our proposal. However, that study just provides the transformation between the metamodels of two specific AOSE methodologies (Prometheus [68] and INGENIAS [69]) to realize MAS implementation on exactly one agent deployment platform and does not support MAS DSML interoperability or language extensibility on various agent platforms.

Taking into account the interoperability of software systems within the context of MDE, various noteworthy studies also exist for enabling these systems to work together [76]. For instance, an MDE platform was used in [77] both for representing various software bug tracing tools and executing transformations among their conceptual models to enhance the interoperability between these tools. Likewise, Sun et al. [78] benefited from MDE to address tool interoperability for supporting different data formats among similar tools. Kern [79] introduced an interoperability interface for the exchange of metamodels and models between MetaEdit+ and Eclipse EMF tools based on the mappings specified at meta-metamodel level. That bridging approach was extended to construct the interoperability between modeling tools such as ARIS, EMF, MetaEdit+ and Microsoft Visio in [80]. BPM-X-Change tool, introduced in [81], provided the interchange of data models and their visual diagrams between different enterprise management tools and repositories for the interoperability of them. In [82], a comprehensive analysis of modeling tools was performed by considering both the degree of supported interoperability and the variety of approaches for realizing the interoperability. Horizontal transformations, defined and implemented between MAS DSMLs in our study, directly support the interoperability of agent modeling tools owned by these DSMLs. Hence, the work herein can also be considered inside above software tool interoperability studies with emphasizing MDD of agent systems.

9. Conclusion

We presented an approach for extending the execution platform support of MAS DSMLs over language interoperability 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. Extending the platform support with applying the conventional way which is widely in-use for MAS DSMLs was also demonstrated in the paper to provide a comparison for the new interoperability approach. In comparison to the conventional approach, evaluation results showed that the interoperability approach requires both less development time and effort considering the quantity of transformation rules, code generation templates and total LoC required for all transformations. Due to being at the same abstraction level, both mapping the model entities and implementing the model transformations were more convenient and less laborious comparing with M2M and M2T transformation chain required in the way of enriching the support of DSMLs for various agent execution platforms in the conventional approach.

As the future work, we plan to extend the applicability of this interoperability approach for some other MAS DSMLs. For instance, modeling SEA_ML agents can be improved by constructing a similar interoperability with another MAS DSML, called MAS-ML [24]. MAS-ML owns a built-in modeling for agent architectures such as reflex agent, model-based agent, or utility-based agent which are not currently supported in SEA_ML. Hence, instead of constructing all required components for such agent architecture support in SEA_ML from scratch, an interoperability with MAS-ML can automatically improve the features of SEA_ML within this context.

Acknowledgment. This study is funded by the Scientific and Technological Research Council of Turkey (TUBITAK) Electric, Electronic and Informatics Research Group (EEEAG) under grant 115E591 and the Scientific Research Projects Directorate of Ege University under grant 16-UBE-001.

References

1. Wooldridge, M., Jennings, N. R.: Intelligent Agents - Theory and Practice. Knowledge Engineering Review, Vol. 10, No. 2, 115-152. (1995)
2. Badica, C., Budimac, Z., Burkhard, H. D., Ivanovic, M.: Software agents: Languages, tools, platforms. Computer Science and Information Systems, Vol. 8, No. 2, 255-298. (2011)
3. Paprzycki, M.: Editorial: Agent-oriented computing for distributed systems and networks. Journal of Network and Computer Applications, Vol. 37, 45–46. (2014)
4. Challenger, M., Mernik, M., Kardas, G., Kosar, T.: Declarative specifications for the development of multi-agent systems. Computer Standards & Interfaces, Vol. 43, 91-115. (2016)
5. Kardas, G.: Model-driven development of multiagent systems: a survey and evaluation. The Knowledge Engineering Review, Vol. 28, No. 4, 479-503. (2013)
6. Shehory, O., Sturm, A.: Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks. Springer-Verlag Berlin Heidelberg. (2014)
7. Bernon, C., Cossentino, M., Gleizes, M.-P., Turci, P., Zambonelli, F.: A study of some multi-agent meta-models. Agent-Oriented Software Engineering V, Lecture Notes in Computer Science, Vol. 3382, 62-77. (2005)
8. Omicini, A., Ricci, A., Viroli, M.: Artifacts in the A&A meta-model for multi-agent systems. Autonomous Agents and Multi-Agent Systems, Vol. 17, No. 3, 432-456. (2008)
9. Hahn, C., Madrigal-Mora, C., Fischer, K.: A Platform-Independent Metamodel for Multiagent Systems. Autonomous Agents and Multi-Agent Systems, Vol. 18, No. 2, 239-266. (2009)

10. Beydoun, G., Low, G., Henderson-Sellers, B., Mouratidis, H., Gomez-Sanz, J. J., Pavon, J., Gonzalez-Perez, C.: FAML: A Generic Metamodel for MAS Development. IEEE Transactions on Software Engineering, Vol. 35, No. 6, 841-863. (2009)
11. Garcia-Magarino, I.: Towards the integration of the agent-oriented modeling diversity with a powertype-based language. Computer Standards & Interfaces, Vol. 36, 941–952. (2014)
12. Bauer, B., Odell, J.: UML 2.0 and agents: how to build agent-based systems with the new UML standard. Engineering Applications of Artificial Intelligence, Vol. 18, No. 2, 141-157. (2005)
13. Pavon, J., Gomez-Sanz, J., Fuentes, R.: Model driven development of multi-agent systems. Lecture Notes in Computer. Science, Vol. 4066, 284–298. (2006)
14. Kardas, G., Goknil, A., Dikenelli, O., Topaloglu, N. Y.: Model driven development of semantic web enabled multi-agent systems. International Journal of Cooperative Information Systems, Vol. 18, No. 2, 261-308. (2009)
15. Gascuena, J. M., Navarro, E., Fernandez-Caballero, A., Martínez-Tomas, R.: Model-to-model and model-to-text: looking for the automation of VigilAgent. Expert Systems, Vol. 31, No. 3, 199-212. (2014)
16. Kulesza, U., Garcia, A., Lucena, C., Alencar, P.: A generative approach for multi-agent system development. Lecture Notes in Computer Science, Vol. 3390, 52–69. (2005)
17. Hahn, C.: A Domain Specific Modeling Language for Multiagent Systems. In Proceedings of 7th International Conference on Autonomous Agents and Multi-Agent Systems, Estoril, Portugal, 233-240. (2008)
18. Fuentes-Fernandez, R., Garcia-Magarino, L., Gomez-Rodriguez, A. M., Gonzalez-Moreno, J. C.: A technique for defining agent-oriented engineering processes with tool support. Engineering Applications of Artificial Intelligence, Vol. 23, No. 3, 432-444. (2010)
19. Demirkol, S., Challenger, M., Getir, S., Kosar, T., Kardas, G., Mernik, M.: SEA_L: A Domain-specific Language for Semantic Web enabled Multi-agent Systems. In Proceedings of the 2nd Workshop on Model Driven Approaches in System Development, held in conjunction with 2012 Federated Conference on Computer Science and Information Systems, IEEE Conference Publications, Wrocław, Poland, 1373-1380. (2012)
20. Ciobanu, G., Juravle, C.: Flexible Software Architecture and Language for Mobile Agents. Concurrency and Computation-Practice & Experience, Vol. 24, No. 6, 559-571. (2012)
21. Gascuena, J. M., Navarro, E., Fernandez-Caballero, A.: Model-Driven Engineering Techniques for the Development of Multi-agent Systems. Engineering Applications of Artificial Intelligence, Vol. 25, No. 1, 159-173. (2012)
22. Demirkol, S., Challenger, M., Getir, S., Kosar, T., Kardas, G., Mernik, M.: A DSL for the development of software agents working within a semantic web environment. Computer Science and Information Systems, Vol. 10, No. 4, 1525-1556. (2013)
23. Challenger, M, Demirkol, S., Getir, S., Mernik, M., Kardas, G., Kosar, T.: On the use of a domain-specific modeling language in the development of multiagent systems. Engineering Applications of Artificial Intelligence, vol. 28, 111-141. (2014)
24. Goncalves, E. J. T., Cortes, M. I., Campos, G. A. L., Lopes, Y. S., Freire, E. S. S., da Silva, V. T., de Oliveira, K. S. F., de Oliveira, M. A.: MAS-ML2.0: Supporting the modelling of multi-agent systems with different agent architectures. Journal of Systems and Software, Vol. 108, 77-109. (2015)
25. Bryant, B.R., Gray, J., Mernik, M., Clarke, P. J., France, R. B., Karsai, G.: Challenges and Directions in Formalizing the Semantics of Modeling Languages. Computer Science and Information Systems, Vol. 8, No. 2, 225-253. (2011)
26. Mernik, M., Heering, J., Sloane, A.: When and how to develop domain-specific languages. ACM Computing Surveys, Vol. 37, No. 4, 316-344. (2015)
27. Varanda Pereira, J. M., Mernik, M., da Cruz, D., Henriques, P. R.: Program Comprehension for Domain-specific Languages. Computer Science and Information Systems, Vol. 5, No. 2, 1-17. (2008)

28. Lukovic, I., Varanda Pereira, J. M., Oliveira, N., da Cruz, D., Henriques, P. R.: A DSL for PIM specifications: Design and attribute grammar based implementation, Computer Science and Information Systems, Vol. 8, No. 2, 379-403. (2011)
29. Selic, B.: The pragmatics of model-driven development. IEEE Software, Vol. 20, 19-25. (2003)
30. Howden, N., Rönnquist, R., Hodgson, A., Lucas, A.: JACK intelligent agents-summary of an agent infrastructure. In Proceedings of the 5th International Conference on Autonomous Agents, Montreal, Canada, 1-6. (2001)
31. Bellifemine, F. L., Caire, G., Greenwood, D.: Developing Multi-Agent Systems with JADE, Wiley & Sons. (2007)
32. Pokahr, A., Braubach, L., Walczak, A., Lamersdorf, W.: Jadex-engineering goal-oriented agents. Developing Multi-Agent Systems with JADE. Bellifemine et al.(Eds)., Wiley & Sons, 254-258. (2007)
33. Object Management Group (OMG). Model Driven Architecture (MDA) Specification. (2003). [Online]. Available: http://www.omg.org/mda/ (current June 2017)
34. Mens, T., Van Gorp, P.: A Taxonomy of Model Transformation. Electronic Notes in Theoretical Computer Science, Vol. 152, issue 27, 125-142. (2006)
35. Challenger, M., Kardas, G., Tekinerdogan, B.: A systematic approach to evaluating domain-specific modeling language environments for multi-agent systems. Software Quality Journal, vol. 24, no. 3, pp. 755-795. *(*2016)
36. Bircan, E., Challenger, M., Kardas, G.: Interoperability of MAS DSMLs via Horizontal Model Transformations. In Proceedings of the 4th Workshop on Model Driven Approaches in System Development, held in conjunction with 2016 Federated Conference on Computer Science and Information Systems, IEEE Conference Publications, Gdansk, Poland, 1555-1564. (2016)
37. Bellifemine, F., Poggi, A., Rimassa, G.: Developing multi-agent systems with a FIPA-compliant agent framework. Software-Practice & Experience, Vol. 31, No. 2, 103-128. (2001)
38. Agent Oriented Software (AOS). Agent Oriented Software Inc., JACK Intelligent Agents. (2001). [Online]. Available: http://www.aosgrp.com/products/jack/ (current June 2017).
39. Pokahr, A., Braubach, L., Lamersdorf, W.: Jadex: A BDI reasoning engine. Multi-agent programming, Springer, 149-174. (2005)
40. Djuric, D., Gasevic, D., Devedzic, V.: The Tao of Modeling Spaces. Journal of Object Technology, Vol. 5, No. 8, 125-147. (2006)
41. Shadbolt, N., Hall, W., Berners-Lee, T.: The Semantic Web revisited. IEEE Intelligent Systems, Vol. 21, No. 3, 96-101. (2006)
42. Sycara, K., Paolucci, M., Ankolekar, A., Srinivasan, N.: Automated discovery, interaction and composition of Semantic Web Services. Journal of Web Semantics, Vol. 1, No. 1, 27-46. (2003)
43. Challenger, M., Getir, S., Demirkol, S., Kardas, G.: A Domain Specific Metamodel for Semantic Web Enabled Multi-Agent Systems. Advanced Information Systems Engineering Workshops, Lecture Notes in Business Information Processing, Vol. 83, 177-186. (2011)
44. Rao, A. S., Georgeff, M. P.: BDI-agents: From Theory to Practice. In Proceedings of the 1st International Conference on Multiagent Systems, San Francisco, USA, 312-319. (1995)
45. World Wide Web Consortium (W3C). OWL-S: Semantic markup for web services. (2004). [Online]. Available: http://www.w3.org/Submission/OWL-S/ (current June 2017).
46. Getir, S., Challenger, M., Kardas, G.: The formal semantics of a domain-specific modeling language for semantic web enabled multi-agent systems. International Journal of Cooperative Information Systems, Vol. 23, No. 3, 1-53. (2014)
47. SEA_ML: A domain-specific modeling language for MAS. (2016). [Online]. Available: http://serlab.ube.ege.edu.tr/resources.html#SEA_ML (current June 2017)

48. Hahn, C., Nesbigall, S., Warwas, S., Zinnikus, I., Fischer, K., Klusch, M.: Integration of Multiagent Systems and Semantic Web Services on a Platform Independent Level. In Proceedings of 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, Sydney, Australia, 200-206. (2008)
49. Warwas, S., Hahn, C.: The concrete syntax of the platform independent modeling language for multiagent systems. In Proceedings of the 2nd Workshop on Agent-based Technologies and Applications for Enterprise Interoperability, Estoril, Portugal, 94-105. (2008)
50. Web Service Modeling Ontology (WSMO). (2005). [Online]. Available: https://www.w3.org/Submission/WSMO/ (current June 2017)
51. Kopecky, J., Vitvar, T., Bournez, C., Farrell, J.: Sawsdl: Semantic annotations for wsdl and xml schema. IEEE Internet Computing, Vol. 11, No. 6, 60-67. (2007)
52. DSML4MAS Development Environment. (2008). [Online]. Available: https://sourceforge.net/projects/dsml4mas/ (current June 2017)
53. Gomez-Sanz, J. J., Fuentes-Fernandez, R.: Understanding Agent-Oriented Software Engineering methodologies. The Knowledge Engineering Review, Vol. 30, Issue 4, 375–393. (2015)
54. Milanovic, M., Gasevic, D., Giurca, A., Wagner, G., Lukichev, S., Devedzic, V.: Model Transformations to Bridge Concrete and Abstract Syntax of Web Rule Languages. Computer Science and Information Systems, Vol. 6, No. 2, 47-85. (2009)
55. Jouault, F., Allilaire, F., Bezivin, J., Kurtev, I.: ATL: A model transformation tool. Science of Computer Programming, Vol. 72, No. 1-2, 31-39. (2008)
56. Eclipse. ATL Model Transformation Language and Toolkit. (2007). [Online]. Available: http://www.eclipse.org/atl/ (current June 2017)
57. Frankel, D.: Model Driven Architecture: Applying MDA to Enterprise Computing: The Complete Book. Wiley Publishing, US. (2008)
58. JAVA Agent DEvelopment Framework (JADE). (2015). [Online]. Available: http://jade.tilab.com/ (current June 2017)
59. Foundation for Intelligent Physical Agents (FIPA). (2002b). "IEEE Foundation for Intelligent Physical Agents (FIPA), FIPA Standards." http://www.fipa.org (current June 2017)
60. Foundation for Intelligent Physical Agents (FIPA).: The Foundation for Intelligent Physical Agents, Agent Communication Language Message Structure Specification 00061. (2005). [Online]. Available: http://www.fipa.org/specs/fipa00061/ (current June 2017)
61. Xpand. (2016). [Online]. Available: http://wiki.eclipse.org/Xpand (current June 2017)
62. Kardas, G., Challenger, M., Yildirim, S., Yamuc. A.: Design and implementation of a multiagent stock trading system. Software: Practice and Experience, Vol. 42, No. 10, 1247-1273. (2012)
63. Rougemaille, S., Migeon, F., Maurel, C., Gleizes, M-P.: Model Driven Engineering for Designing Adaptive Multi-agent Systems, Lecture Notes in Artificial Intelligence, Vol. 4995, 318-333. (2007)
64. Ayala, I., Amor, M., Fuentes, L.: A model driven engineering process of platform neutral agents for ambient intelligence devices. Autonomous Agents and Multi-agent Systems, Vol. 28, 214-255. (2014)
65. Object Management Group (OMG). Meta Object Facility (MOF). (2002). [Online]. Available: http://www.omg.org/spec/MOF (current June 2017)
66. Eclipse. Graphical Modeling Framework (GMF). (2006). [Online]. Available: http://www.eclipse.org/modeling/gmp/ (current June 2017)
67. Eclipse. The Eclipse Modeling project, Model to Text (M2T) transformation, JET code generator (2007). [Online]. Available: http://www.eclipse.org/modeling/m2t/?project=jet#jet (current June 2017)
68. Padgham, L., Winikoff, M.: Prometheus: A practical agent-oriented methodology. Agent-oriented methodologies, Henderson-Sellers and Giorgini (Eds), 107-135. (2005)

69. Pavón, J., Gómez-Sanz, J. J., Fuentes, R.: The INGENIAS methodology and tools. Agent-oriented methodologies, Henderson-Sellers and Giorgini (Eds), Vol. 9, 236-276. (2005)
70. Wautelet, Y., Kolp, M.: Business and model-driven development of BDI multi-agent systems. Neurocomputing, Vol. 182, 304–321. (2016)
71. Yu, E., Giorgini, P., Maiden, N., Mylopoulos, J.: Social Modeling for Requirements Engineering: The Complete Book. MIT Press, Cambridge, Massachusetts. (2011)
72. Tezel, B. T., Challenger, M., Kardas, G.: A Metamodel for Jason BDI Agents. In Proceedings of the 5th Symposium on Languages, Applications and Technologies, Maribor, Slovenia, OpenAccess Series in Informatics, Vol. 51, 8:1-8:9. (2016)
73. Bordini, R. H., Hübner, J. F., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak Using Jason, John Wiley & Sons. (2007)
74. Ashamalla, A., Beydoun, G., Low, G.: Model Driven Approach for Real-time Requirement Analysis of Multi-Agent Systems. Computer Languages, Systems & Structures, DOI: 10.1016/j.cl.2017.05.006. (2017)
75. Bergenti, F., Iotti, E., Monica, S., Poggi, A.: Agent-Oriented Model-Driven Development for JADE with the JADEL Programming Language. Computer Languages, Systems & Structures, DOI: 10.1016/j.cl.2017.06.001. (2017)
76. Chen, D.: Practices, principles and patterns for interoperability. Research report of INTEROP NoE, FP6 - Network of Excellence - Contract no: 508011, Deliverable 6.1 (2005)
77. Bezivin, J., Bruneliere, H., Jouault, F., Kurtev, I: Model Engineering Support for Tool Interoperability. In Proceedings of the 4th UML Workshop in Software Model Engineering, Montego Bay, Jamaica, 2-16. (2005)
78. Sun, Y., Demirezen, Z., Jouault, F., Tairas, R., Gray, J.: A Model Engineering Approach to Tool Interoperability. Lecture Notes in Computer Science, Vol. 5452, 178-187. (2009)
79. Kern, H.: The Interchange of (Meta)Models between MetaEdit+ and Eclipse EMF Using M3-Level-Based Bridges. In Proceedings of 8th OOPSLA Workshop on Domain-Specific Modeling, Birmingham, USA, 14-19. (2008)
80. Kern, H.: Model Interoperability between Meta-Modeling Environments by using M3-Level-Based Bridges. Ph.D Thesis, University of Leipzig, Leipzig (2016)
81. Kammermeier, F., Rautenberg, V., Scherer, H.-J.: Pattern-based model transformations: Software for the interoperability of enterprise management tools and model interchange. (2011). [Online]. Available: http://www.bpm-x.com/ (current June 2017)
82. Kern, H.: Study of Interoperability between Meta-Modeling Tools. In Proceedings of 2014 Federated Conference on Computer Science and Information Systems, Warsaw, Poland, 1629–1637 (2014)

Geylani Kardas received his B.Sc. in computer engineering and both M.Sc., and Ph.D. degrees in information technologies from Ege University in 2001, 2003 and 2008. He is currently an associate professor at Ege University, International Computer Institute (ICI) and the head of Software Engineering Research Laboratory (Ege-SERLab) at ICI. His research interests mainly include agent-oriented software engineering, model-driven software development, and domain-specific (modeling) languages. He has authored or co-authored over 60 peer-reviewed papers in these research areas. Dr. Kardas worked and is still working as the principle investigator, researcher or consultant in various R&D projects funded by governments, agencies and private corporations. He is a member of the ACM.

Emine Bircan received her B.Sc. in computer engineering from Izmir Institute of Technology in 2013 and M.Sc. in information technologies from Ege University, International Computer Institute (ICI) in 2017. During her graduate study, she is a student member of Software Engineering Research Laboratory (Ege-SERLab) at ICI. She is currently a researcher in the Scientific and Technological Research Council of Turkey (TUBITAK). Her interests include model-driven software engineering, multi-agent systems and recently data security & privacy.

Moharram Challenger received his B.Sc., and M.Sc. degrees in computer engineering from IAU-Shabestar and IAU-Arak Universities (Iran) in 2001 and 2005 respectively. He also received his Ph.D. in Information Technologies from Ege University (Turkey), in Feb 2016. After PhD, he has worked as an external Postdoc researcher in IT group, Wageningen University of Research (the Netherlands) about 1 year. Since Jan 2017, he is working as an assistant professor at International Computer Institute, Ege University. His research interests include domain-specific (modeling) languages, multi-agent systems, and Internet of Things.

Received: January 13, 2017; Accepted: June 24, 2017