applied sciences

MDPI

Article

Development of Semantic Web-Enabled BDI Multi-Agent Systems Using SEA_ML: An Electronic Bartering Case Study

Moharram Challenger [1,*], Baris Tekin Tezel [1,2], Omer Faruk Alaca [1], Bedir Tekinerdogan [3] and Geylani Kardas [1]

[1] International Computer Institute, Ege University, Bornova, Izmir 35100, Turkey; baris.tezel@deu.edu.tr (B.T.T.); omerfarukalaca@gmail.com (O.F.A.), geylani.kardas@ege.edu.tr (G.K.)
[2] Department of Computer Science, Dokuz Eylul University, Buca, Izmir 35390, Turkey
[3] Information Technology Group, Wageningen University & Research, 6706 KN Wageningen, The Netherlands; bedir.tekinerdogan@wur.nl
* Correspondence: m.challenger@gmail.com or moharram.challenger@ege.edu.tr; Tel.: +90-541-918-8836

Received: 4 March 2018; Accepted: 25 April 2018; Published: 28 April 2018

Abstract: In agent-oriented software engineering (AOSE), the application of model-driven development (MDD) and the use of domain-specific modeling languages (DSMLs) for Multi-Agent System (MAS) development are quite popular since the implementation of MAS is naturally complex, error-prone, and costly due to the autonomous and proactive properties of the agents. The internal agent behavior and the interaction within the agent organizations become even more complex and hard to implement when the requirements and interactions for the other agent environments such as the Semantic Web are considered. Hence, in this study, we propose a model-driven MAS development methodology which is based on a domain-specific modeling language (called SEA_ML) and covers the whole process of analysis, modeling, code generation and implementation of a MAS working in the Semantic Web according to the well-known Belief-Desire-Intention (BDI) agent principles. The use of new SEA_ML-based MAS development methodology is exemplified with the development of a semantic web-enabled MAS for electronic bartering (E-barter). Achieved results validated the generation and the development-time performance of applying this new MAS development methodology. More than half of the all agents and artifacts needed for fully implementing the E-barter MAS were automatically obtained by just using the generation features of the proposed methodology.

Keywords: multi-agent system; BDI agents; model-driven development; agent development methodology; semantic web service; ontology; SEA_ML; electronic bartering system

1. Introduction

Autonomous, reactive, and proactive agents have social ability and can interact with other agents and humans to solve their problems. To perform their tasks and interact with each other, intelligent agents constitute systems called Multi-Agent Systems (MASs) [1]. In addition, autonomous agents can evaluate semantic data and collaborate with semantically defined entities of the Semantic Web, such as semantic web services (SWS), by using content languages [2]. The implementation of agent systems is naturally a complex task when considering their characteristics. The internal agent behavior model and any interaction within the agent organizations become even more complex and hard to implement when the requirements and the interactions for other agent environments such as the Semantic Web [3,4] are considered.

Therefore, it is natural that methodologies are being applied to master the problem of defining such complex systems. One of the possible alternatives is represented by domain-specific modeling languages (DSMLs) [5,6] that have notations and constructs tailored towards an application domain (e.g., MAS). DSMLs raise the abstraction level, expressiveness, and ease of use.

The application of model-driven development (MDD) and use of DSMLs for MAS development emerged in agent-oriented software engineering (AOSE) research field especially for the last decade [7]. Researchers developed various metamodels (e.g., [8–10]) and DSMLs (e.g., [11–15]) to cope with the challenges encountered on design and implementation of MASs. Moreover, some fully fledged DSMLs (e.g., [16,17]) exist for developing software agents especially working in semantic web environments where agents can handle the Semantic Web content on behalf of their human users and interact with other semantic web environment entities, such as SWS. One of these MAS DSMLs is Semantic Web Enabled Agent Modeling Language (SEA_ML) [17] which has a built-in support for the modeling interactions of agent and semantic web services by including several specialized viewpoints. SEA_ML aims to enable domain experts to model their own MASs on the Semantic Web without considering the limitations of using existing MAS development frameworks (e.g., JADE [18], JADEX [19] or JACK [20]). The evaluations [21], conducted for the assessment of SEA_ML, show promising results considering the generation performance and the development time reduction during MAS design and implementation. According to the experiences gained from the multi-case study [21] conducted by using SEA_ML, the developers can benefit more from this DSML when they use different viewpoints of SEA_ML in a proper way in the development of MAS. Therefore, in this study, we propose a model-driven MAS development methodology which is based on an extended version of SEA_ML and covers the whole process of analysis, modeling, code generation and fully implementation of a MAS working in the Semantic Web according to the well-known Belief-Desire-Intention (BDI) [22] agent principles. The use of the new SEA_ML-based MAS development methodology is exemplified with the development of a semantic web-enabled MAS for electronic bartering (E-barter).

An agent-based E-barter system consists of agents that exchange goods or services of owners according to their preferences without using any currency. Although there are some studies developing agent-based E-barter systems such as [23–28], none of them use BDI agents and their internal reasoning mechanism which can bring extra intelligence in the procedure of matchmaking for the E-bartering. Also, these studies do not use SWSs as the automatic selection mechanism for the categories. Finally, while the methodologies applied in above studies are mostly code-centric and do not consider MDD, our study benefits from MDD and uses a DSML and its tool for the rapid implementation of the MAS. As discussed in this paper, this new model-driven MAS development methodology based on SEA_ML makes the design and development of the MAS system easier and less-costly since the agent developers work with the domain concepts and utilize generative capability of the tool.

The rest of the paper is organized as follows: The next section presents the proposed MAS development methodology based on SEA_ML modeling language. The analysis, design, and implementation of the E-barter system, using the proposed methodology are discussed in Section 3. Section 4 gives a demonstration of the implemented system. In Section 5, the related work is reported and compared with our study. Finally, the paper is concluded in Section 6.

2. SEA_ML-Based MAS Development Methodology

In this study, a model driven approach is adopted, and a model-based methodology is proposed for design and implementation of semantic web-enabled MASs. To this end, the proposed methodology covers the use of a DSML. In this way, the complex systems including SWSs and MAS components are modeled at a higher level of abstraction. In addition, these languages can model the interaction between SWSs and Agents. As a result, the system can be analyzed, and the required elements can be designed using the terms and notations very close to the domain. These domain-specific elements and their relations to each other creates the domain-specific instance models which pave the way to implement the system. As these models are persisted in a structural and formal way, they can be transformed to other proper paradigms, such as mathematical logics. In this way, they can be formally analyzed and validated based on formal methods. This can decrease the number of semantic errors later in the developed system. Furthermore, these models can be used to automatically generate the architectural codes for agents and artifacts of the complex systems which can end up with less syntactical errors and speed up the development procedure. According to the definition of artifacts in Agents & Artifacts (A&A) metamodel [8], artifacts in our study are environmental components and entities providing services, such as OWL-S documents (including process, grounding, and interface documents) for SWS. Faster development requires less efforts and it brings cost reduction in the projects. Moreover, fewer syntactical and semantical errors mean less iterations in the development phase and less testing phase which also reduce the cost and effort. Therefore, the system can be checked, and the errors can be partially found in the early phases of development, namely analysis and design phases, instead of finding them in the implementation and testing phases.

In the scope of this study, SEA_ML [17] is used as a DSML for the construction of semantic web-enabled MASs. SEA_ML 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. 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, such as SWSs. Within this context, it includes new viewpoints which specifically pave the way for the development of software agents working on the Semantic Web environment. Modeling agents, agent knowledge-bases, 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. 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. 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 describes 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. Table A lists the important SEA_ML concepts (meta-entities) and their brief descriptions for the comprehension of the corresponding visual notations used in the diagrams throughout this paper.

Based on SEA_ML, the analysis and the design of the software system can be realized using the application domain's terms and notations. This helps the end users to work in a higher level of abstraction (independent of target platform) and close to expert domain. Also, generative feature of SEA_ML paves the way to produce the configured templates from the designed models for the software system in the underlying languages and technologies. Currently, SEA_ML can generate architectural code for JADE [18], JADEX [19], and JACK [20] agent programming languages and OWL-S [29] and WSMO [30] SWS documents. This is realized by model to model transformation of the designed platform independent instance models to the instance models of the target MAS languages and SWS technologies. Then, these platform specific models are transformed to the platform specific codes by model to code transformations. This generation capability of SEA_ML can increase the development performance of the software system considerably. Finally, by constraints checking provided in SEA_ML, the instance models are controlled considering domain-specific syntactic and semantic rules. These rules are applied in the abstract and the concrete syntaxes of the language. This feature helps to reduce the number of errors during the analysis and design of the software system and avoid postponing them to the development and the testing phases.

In this section, the SEA_ML-based MAS development approach is discussed. Although this new development methodology also considers the adoption of SEA_ML, it differentiates from the previous development approach [17] as being a complete development methodology covering the analysis, design, and implementation of the MAS. Analysis phase, which does not exist previously, is now included in the methodology and both analysis and design phases are improved with two types of iterations. Such an iterative development process is not considered in the previous methodology. In addition to the modification of models, new methodology also supports the changes in auto-generated codes if required. The proposed SEA_ML-based MAS development methodology includes several steps following each other (see Figure 1): MAS Analysis, MAS Modeling, Model-to-Model (M2M) and Model-to-Code (M2C) Transformations, and finally code generation for exact MAS implementation. Following subsections discuss the methodology's phases covering those steps.

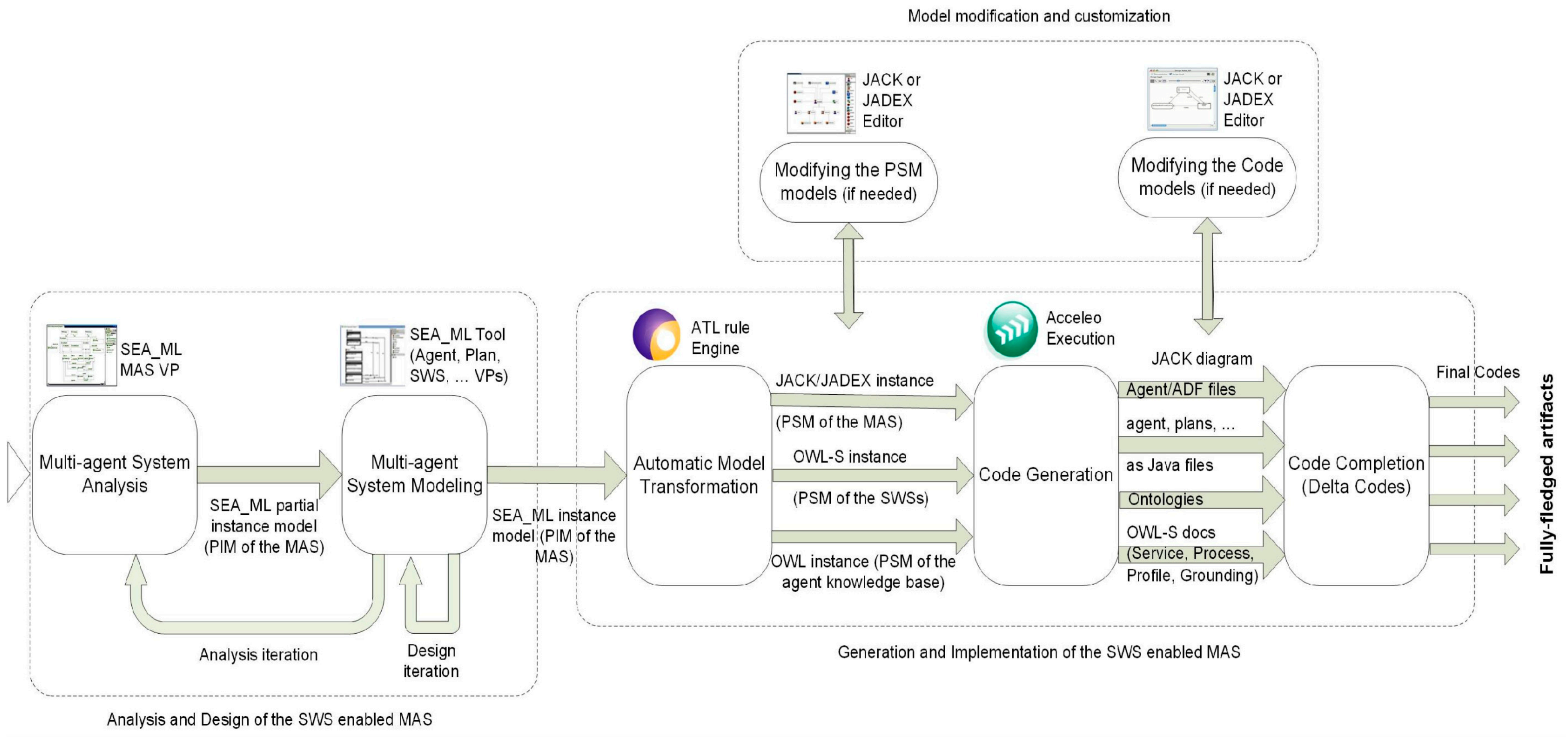

Model modification and customization
JACK or JADEX Editor
Modifying the PSM models (if needed)
JACK or JADEX Editor
Modifying the Code models (if needed)
SEA_ML MAS VP
SEA_ML Tool (Agent, Plan, SWS, ... VPs)
Multi-agent System Analysis
SEA_ML partial instance model (PIM of the MAS)
Multi-agent System Modeling
SEA_ML instance model (PIM of the MAS)
Analysis iteration
Design iteration
Analysis and Design of the SWS enabled MAS
ATL rule Engine
Automatic Model Transformation
JACK/JADEX instance
(PSM of the MAS)
OWL-S instance
(PSM of the SWSs)
OWL instance (PSM of the agent knowledge base)
Acceleo Execution
Code Generation
JACK diagram
Agent/ADF files
agent, plans, ...
as Java files
Ontologies
OWL-S docs (Service, Process, Profile, Grounding)
Code Completion (Delta Codes)
Final Codes
Fully-fledged artifacts
Generation and Implementation of the SWS enabled MAS

Figure 1. SEA_ML-based MAS development methodology. SEA_ML: semantic web enabled agent modeling language; MAS: Multi-Agent System; VP: View Point; SWS: semantic web services; PIM: platform independent model; PSM: platform-specific model; OWL-S: Web Ontology Language for Services; ADF: agent definition file.

2.1. MAS Analysis and Design

Based on the proposed methodology, the development of a semantic web-enabled MAS starts with the analysis of the system by considering the MAS viewpoint of SEA_ML (see Figure 1). This viewpoint includes MAS elements such as organizations, environments, agents, and their roles. The viewpoint provides the eagle-view of the system and shapes the high-level structure of the system. The result is a partial platform independent instance model of the system covering the analysis phase of the system development and providing a preliminary sketch of the system.

In the system modeling step the agent developer can use the fully functional graphical editors of SEA_ML to elaborate the design of the system, which includes 7 viewpoints of the SEA_ML's syntax, in addition to the MAS viewpoint used in the analysis phase. These viewpoints cover both multi-agent part of the system (using Agent Internal, Plan, Role, Interaction, and Environment viewpoints) and semantic web aspect of the system (using Agent-SWS Interaction and Ontology viewpoints). Each viewpoint has its own palette which provides various controls leading the designers to provide more accurate models. By designing each of these models for viewpoints, additional details are added to the initial system model provided in the analysis phase. These modifications immediately are updated in the diagrams of all other viewpoints. As the other viewpoints may have some constraint checks to control some properties related to the newly added element, the developer will be directed to complete those other viewpoints to cover the errors and warnings (coming from the constraint checks). This can lead to several iterations in the design phase. The result of this phase is the development of a complete and accurate platform independent model for the designed MAS.

2.2. Transformation and Implementation

The next step in the MAS development methodology based on SEA_ML is the automatic model transformations. The models created in the previous step need to be transformed from platform independent level into the platform-specific level, e.g., to the JACK and OWL-S models as in the case of this study. These transformations are called M2M transformations.

According to OMG's well-known Model-driven Architecture (MDA) [31], SEA_ML metamodel can be considered as a Platform Independent Metamodel (PIMM) and JACK and OWL-S metamodels can be considered as Platform-specific Metamodels (PSMM). The model transformations between these PIMMs and PSMMs pave the way for the MDD of the Semantic Web-enabled MASs. These transformations are implemented using ATL Language [32] to produce the intermediate models which enable the generation of architecture code for the agents and SWS documents. An agent developer does not need to know both the details of these transformations written in ATL and the underlying model transformation mechanism. Following the creation of models in the previous modeling steps, the only thing requested from a developer is to initiate the execution of these transformations via the interface provided by SEA_ML's Graphical User Interface (GUI).

Upon completion of model transformations, the developers have two options at this stage: (1) They may directly continue the development process with code generation for the achieved platform independent MAS models or (2) if they need, they can visually modify the achieved target models to elaborate or customize them, which can lead to gain more accurate software codes in the next step, code generation. In either case, the achieved result of this step are platform-specific system models for JACK platform, OWL-S and OWL instances.

The next step in the proposed methodology is the software code generation for the MAS. To this end, the developers' platform-specific models (conforming to PSMMs) are transformed into the code in the target languages. The M2T transformation rules are automatically executed on the target models and the codes are obtained for the implementation of the MAS. In SEA_ML, it is possible to generate code for BDI agent languages such as JACK from SEA_ML models. In addition, semantic web components of the system can be obtained through other transformations to generate OWL-S documents. Based on the initial models of the developer, the generated files and codes are also interlinked during the transformations where it is required. To support the interpretation of SEA_ML models, the M2T transformation rules are written in Acceleo [33]. Acceleo is a language to convert models into text files and uses metamodel definitions (Ecore Files) and instance files (in XMI format) as its inputs. More details on how mappings and model transformation rules between SEA_ML and the target PSMMs are realized as well as how codes are generated from PSMs can be found in [17].

As the last step, the developer needs to add his/her complementary codes, aka delta codes, to the generated architectural code to have fully functional system. However, some agent development languages, such as JACK, have their graphical editor in which the developer can edit the structure of MAS code. The generated codes achieved from the previous step can be edited and customized to add more platform specific details which helps to reach more detailed agents and artifacts. Then the delta code can be added to gain the final code.

It is important to note that, although all above mentioned steps are supported by SEA_ML to be done automatically, at any stage the developer may intervene in this development process if he/she wishes to elaborate or customize the achieved agents and artifacts.

3. E-barter Case Study

In this paper, the design and the implementation of SWS-enabled agent-based E-barter system were realized using JACK agent language [20] and OWL-S SWS technology [29].

JACK is a BDI oriented MAS development language providing a framework in Java. It is a third-generation agent platform building on the experiences of the Procedural Reasoning System (PRS) [34] and Distributed Multi-Agent Reasoning System (dMARS) [35]. JACK is one of the MAS platforms that uses the BDI software model and provides its own Java-based plan language and graphical planning tools.

OWL-S (Semantic Markup for Web Services) enables the discovery, invocation, interoperation, composition, and verification of services. It builds on the formerly developed DAML-S [36] and was the first submission for describing SWS submitted to the W3C. Each SWS in OWL-S consists of a service profile, a service model, and a grounding. The service profile describes what the service does and is used to advertise the service. The service model answers the question "how it is used?" and describes how the service works internally. Finally, the service grounding specifies how to access the service. OWL-S is based on the Web Ontology Language OWL [37] and supplies web service providers with a core set of markup language constructs for describing the properties and capabilities of their web services in an unambiguous, computer-interpretable form.

The following subsections discuss the details of analysis, design, generation, and implementation of E-barter case study using the detailed methodology proposed in this study and benefiting from SEA_ML platform.

3.1. System Analysis and Design with SEA_ML

In this subsection, we discuss the analysis and the design of the agent-based E-barter system. System analysis is realized by specifying bartering elements using agents and their components in SEA_ML, while system design is realized by providing diagrams of different viewpoints of the system using SEA_ML.

3.1.1. System Analysis with MAS and Organization Viewpoint

An agent-based E-barter system consists of agents that exchange goods or services of owners according to their preferences without using any currency. The system analysis phase is performed by considering the MAS viewpoint of SEA_ML language. In fact, this viewpoint provides an overview of the system which is shown in Figure 2. When considering the structure of the system, the EbarterSystem constitutes of two semantic web organizations called Customers organization which include Customer agents, and Management organization where the Matchmaker agent and ServiceAgent agents reside. It is worth indicating that entities given in this overview can be considered as stereotypes and in the real system implementation, there can be many instances of these entities. For example, there can be many agents of type ServiceAgent working in this system.

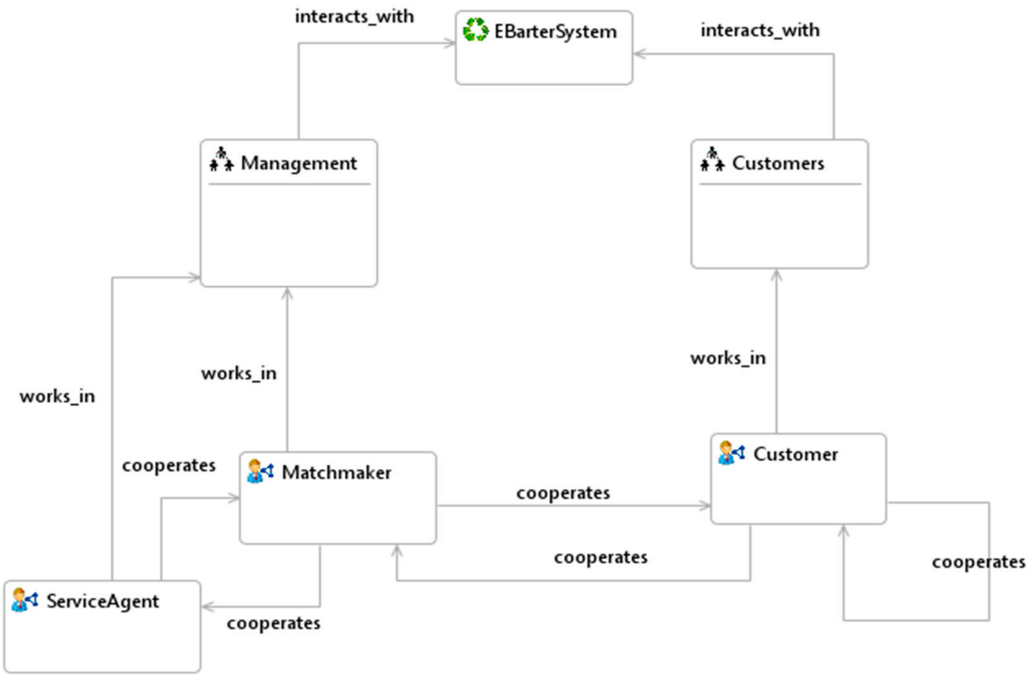


Figure 2. Overview of E-Barter MAS.

In this MAS, a Matchmaker agent, which is defined as a SWA, handles the interaction between Customer agents and ServiceAgents. This agent is responsible for registering SWS provided by each ServiceAgent in the system and matching proper services with customers. To infer about semantic closeness between offered and purchased items based on some defined ontologies, Matchmaker may use SWS. Conforming to its matchmaking definition, Matchmaker needs to discover the proper SWS, interact with the candidate service and realize the execution of SWS after an agreement.

Customer agents represent the end users in the E-barter system. This agent receives the user's offer and purchase items and interacts with Matchmaker and other Customers to realize bartering. At the first stage, this agent interacts with Matchmaker to find out if there is a proper service containing candidate customers. In case of success, it receives the service addresses and interacts with those services to get the list of suitable customers. These services contain ontologically close customers with our customer needs. In case of failure, the Matchmaker simply registers the customer into the proper service. A Customer agent, having the list of candidate customers for bartering in hand, starts to negotiate with them one by one to make an agreement and realize bartering.

The ServiceAgent agents represent the E-bartering SWSs in the system. They interact with the Matchmaker agent to register, update, and un-register the SWSs used in the system.

3.1.2. MAS Design by Modeling in SEA_ML

In accordance with the SEA_ML-based MDD methodology, we start by creating system models based on different viewpoints. The information needed for designing these models is gathered with the appropriate requirements engineering within the domain of the E-barter System.

In this study, we present three viewpoints of the E-barter system in SEA_ML namely MAS, Agent Internal and AgentSWS interaction viewpoints which represent both the MAS and SWS aspect of the system. The diagrams representing the models in these viewpoints are shown in Figures 2–4 respectively.

An agent's general interaction and bartering processes are modeled with using SEA_ML based on the performed analysis discussed above. In this step, we evaluate the SWA agent instances in the MAS and then model the internal structure of each agent using Agent Internal Viewpoint. It is worth noting that, the instance models and the instances of specific elements (such as SWA) are related to example models conforming to the SEA_ML meta-model within the model-driven approach. After that, we model the interactions of these agents with services, using the internal components of the semantic web services of Agent-SWS interaction viewpoint.

According to the system analysis realized in the previous phases (discussed in Section 3.1.1), the system constitutes of two semantic web organizations: Management and Customers. Each semantic web organization is a composition of SWAs having similar goals or duties. These organizations need access to some resources in the EBarterSystem environment. For this reason, there are interactions with the EBarterSystem environment to get access permission. In addition, the interactions of agents with each other are modeled. In this case, study, the MAS-to-be-implemented consists of 3 types of semantic web agents: Matchmaker, Customer and ServiceAgent. All Customers and ServiceAgents cooperate with Matchmaker to access the E-Barter system. In addition, customer agents interact with each other to negotiate for bartering. For instance, a Customer agent cooperates with a Matchmaker agent to get the list of Customer agents who have the requested product(s).

When the system's agents are determined in MAS viewpoint, the internal structure of each semantic web agent is modelled. The instance model should cover all the required roles, behaviors, plans, beliefs, and goals of an agent. Figure 3, illustrates an instance model of the agent internal viewpoint for a Customer agent.

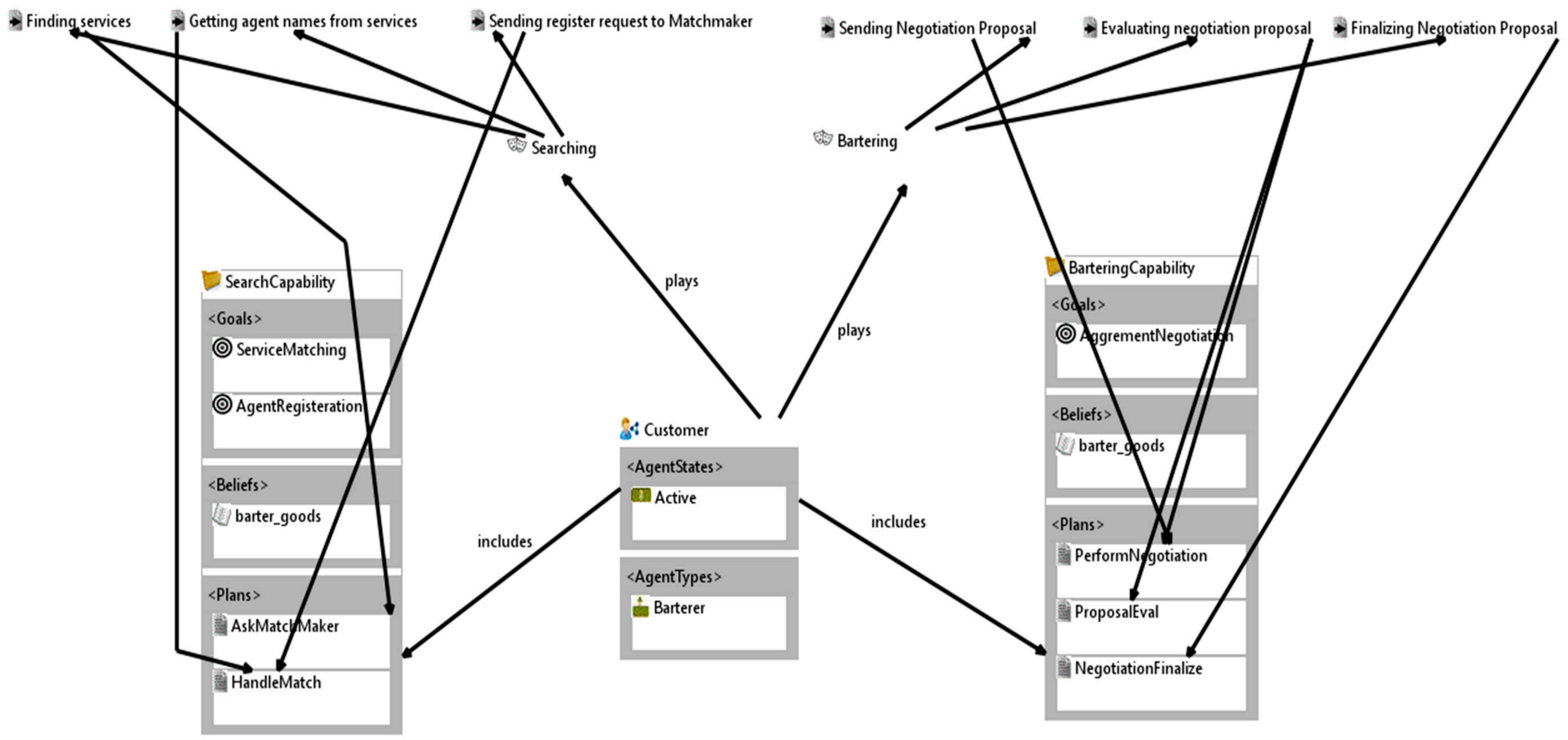


Figure 3. Agent Internal Diagram.

Customer agent has two Capabilities called SearchCapability and BarteringCapability. The SearchCapability includes its Goals (“ServiceMatching” and “AgentRegisteration”), Belief (“barter_goods”), and Plans (“AskMatchmaker” and “HandleMatch”). The BarteringCapability includes its own Goal (“AggrementNegotiation”), Belief (“barter_goods”), and Plans (“PerformNegotiation”, “ProposalEval” and “NegatiationFinalize”). When considering the Beliefs, the agent uses them to know which goods it has, and which goods it needs. Therefore, the agent decides what to offer and what to require for in the bartering process. Also, the agent could play Searching and Bartering roles. The Searching role could realize its task over “Finding Services” behavior by calling the AskMatchmaker plan. If this plan is executed successfully, “Getting Agent Names from Services” behavior is performed with the “HandleMatch” plan. Otherwise, the agent performs “Sending Register Request to Matchmaker” behavior. The Bartering role realizes all behaviors associated with the bartering transaction. The bartering transaction is carried out among the Customer agents and the “Bartering Role” covers all these process behaviors which are realized by relevant plans.

Figure 4 shows the instance model which includes semantic services and the required plan instances of the Agent-SWS interaction viewpoint. The instance model contains all the plans for discovering, negotiating with and executing the candidate services. Customer and Matchmaker agents are modeled with relevant plan instances to find, make the agreement with, and execute the services which are the instances of the SS_FinderPlan, SS_AgreementPlan, and SS_ExecutorPlan, respectively. The services could also be modeled for interaction between the SWS’s internal components (such as Process, Grounding, and Interface), and the SWA’s plans.

Therefore, when considering Customer agent request for bartering Foods, the agent should play the FoodBartering role. While playing this role, the agent applies AskMatchmaker plan for finding a suitable service interface of a Food SWS. This plan is realized by interacting with the Matchmaker agent which applies RegisterRequest plan to register services. Therefore, Customer agent cooperates with Matchmaker to receive some services for getting name of Customer agents who are candidates to barter. Finally, the Customer agent applies its FoodServiceCall plan to collect candidate customer agents with whom it can negotiate.

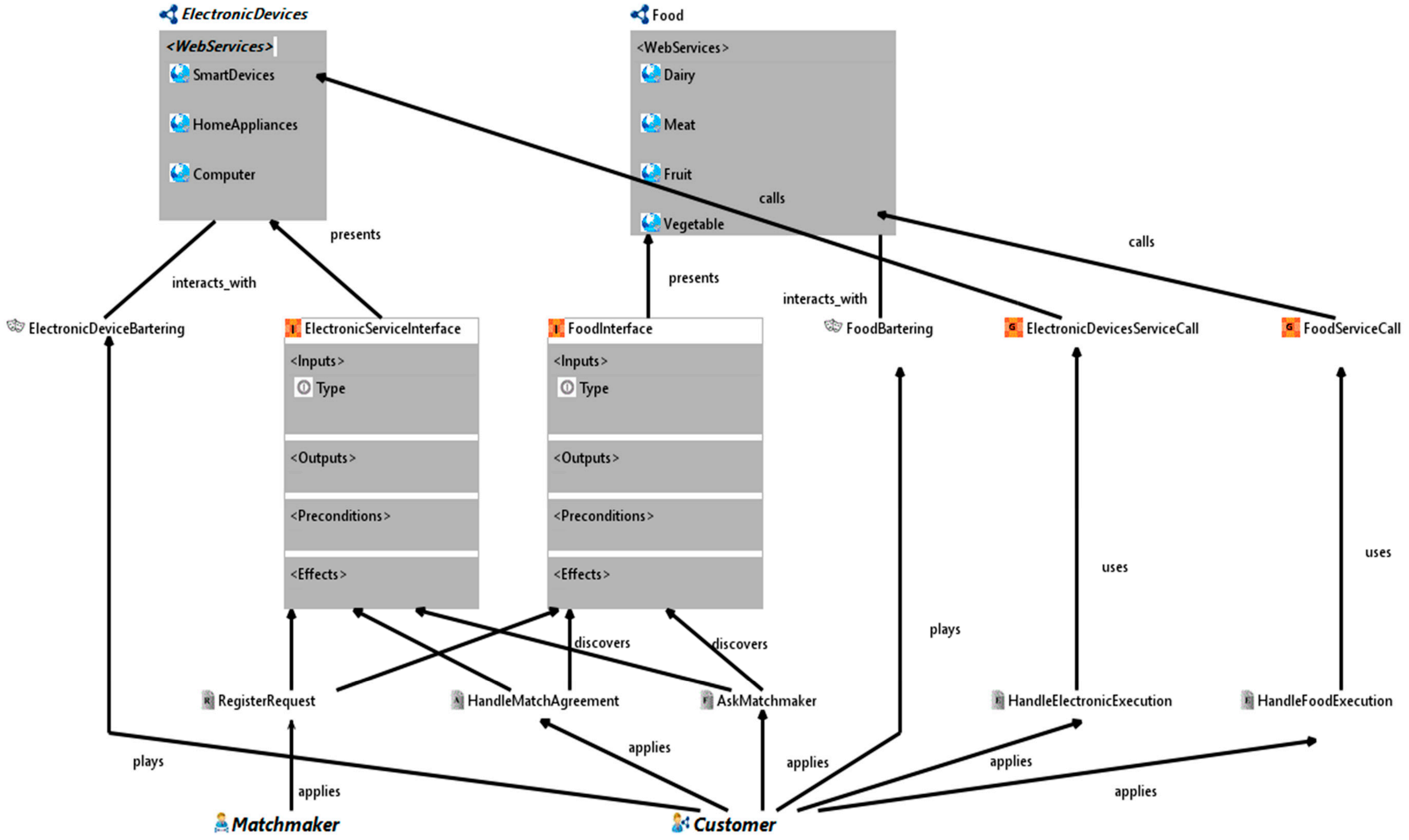


Figure 4. Agent—Semantic Web Service Interaction Diagram.

3.2. System Implementation with Model Transformations and Delta Code Development

In this study, the proposed multi-agent E-barter system is implemented using the JACK platform [20,38]. JACK is selected as it is one of the widely accepted Java-based BDI MAS development platforms. Also, it is as a mature and robust commercial product and meets the appropriate needs for industry adoption, such as scalability and integration.

OWL, the standard language of the W3C for the definition and development of ontologies, is employed in the realization of ontological concepts of SEA_ML. OWL is built on RDF and RDF Schema [39] and adds more vocabulary for describing the properties and classes such as relationship between classes, cardinality, equality, richer typing of properties, and the characteristics of properties and enumerated classes. SEA_ML adopts the Ontology Definition Metamodel (ODM) of OMG [40] as the metamodel of OWL and that metamodel is used as target PSMM during the transformation.

```
<?xml version=“1.0” encoding=“ISO-8859-1”?>
<owls:OWLSplatform
xmi:version=“2.0”
xmlns:xmi=“http://www.omg.org/XMI”
xmlns:xsi=“http://www.w3.org/2001/XMLSchema-instance”
xmlns:owls=“http://owls.com”>
<containsService name=“Food”>
<presentedBy name=“FoodServiceCall”>
<containsInput name=“Type”/>
<containsOutput name=“ “/>
<containsCondition name=“ “/>
<containsEffect name=“ “/>
</presentedBy>
…
<supportedBy name=“FoodServiceCall”/>
</containsService>
<containsService name=“ElectronicDevices”>
<presentedBy name=“ElectronicServiceInterface”/>
<containsInput name=“Type”/>
<containsOutput name=“ “/>
<containsCondition name=“ “/>
<containsEffect name=“ “/>
</presentedBy>
…
<supportedBy name=“ElelctronicDevicesServiceCall”/>
</containsService>
…
</owls:OWLSplatform>
```

Listing 1. Part of the generated OWL-S model for the E-barter system.

The semantic web services modeled in SEA_ML are transformed into OWL-S services to enable the implementation of these services. OWL-S offers a high-level service ontology that can store three basic information about a service. The Service Profile tells you what the service is doing and provides information to discover a service. The Service Model describes how the service can be used and the composition of the service. Finally, Service Grounding provides information on how to interact with the service. Therefore, in our study, each SWS modeled in SEA_ML is transformed into an OWL-S Service element and the appropriate Service Profile, Service Model and Service Grounding documents are created for the related SWS.

3.2.1. Model Transformations

Based on the proposed methodology, M2M transformations are applied for transforming the models designed in SEA_ML as platform independent models and JACK BDI agent and OWL-S models as platform-specific models. A part of generated OWL-S instance model for E-barter system is depicted in Listing 1. In this model the Food service is defined in Lines 7–16, and Electronic Devices service in Lines 17–26 with their interfaces and groundings.

```
01  <?xml version="1.0" encoding="ISO-8859-1" ?>
02  <!DOCTYPE uridef [
03  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns">
04  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema">
05  <!ENTITY owl "http://www.w3.org/2002/07/owl">
06  <!ENTITY service "http://www.daml.org/services/OWL-S/1.0/Food.owl">
07  <!ENTITY congo_profile "http://www.daml.org/services/OWL-S/1.0/FoodProfile.owl">
08  <!ENTITY congo_grounding "http://www.daml.org/services/OWL-S/1.0/FoodGrounding.owl">
09  <!ENTITY DEFAULT "http://www.daml.org/services/OWL-S/1.0/FoodService.owl"> ]>
10  <rdf:RDF
11  xmlns:rdf = "&rdf;#"
12  xmlns:rdfs ="&rdfs;#"
13  xmlns:owl = "&owl;#"
14  xmlns:service= "&service;#"
15  xmlns:profile= "&profile;#"
16  xmlns:process= "&process;#"
17  xmlns:grounding= "&grounding;#"
18  xmlns:tradingServiceProfile=&profile;#
19  xmlns:tradingServiceModel=&process;#
20  xmlns:tradingServiceGrounding=&grounding;#
21  xmlns ="&DEFAULT;#"
22  xml:base="&DEFAULT;">
23  <owl:Ontology rdf:about="">
24  <owl:versionInfo>   $Id:Service.owl generated at: 20/02/2018 10:12:13   </owl:versionInfo>
25  <rdfs:comment>
26  This ontology represents the OWL-S service description for the FoodService web service.
27  </rdfs:comment>
28  <owl:imports rdf:resource= "&FoodService_service;" />
29  <owl:imports rdf:resource= "&FoodService_profile;" />
30  <owl:imports rdf:resource= "&FoodService_process;" />
31  <owl:imports rdf:resource= "&FoodService_grounding;" />
32  </owl:Ontology>
33  <service:Service rdf:ID= "FoodService">
34  <service:presents rdf:resource="&Food_profile; #ServiceProfile /> <!-- Reference to the Profile -->
35  <!-- Reference to the Process Model -->
36  <service:describedBy rdf:resource=&food_process; #ServiceModel/>
37  <!-- Reference to the Grounding -->
38  <service:supports rdf:resource= &FoodServiceCall_grounding; #ServiceGrounding'/>
39  </service:Service>
40  …
41  </rdf:RDF>
```

Listing 2. An excerpt of the generated OWL-S service file.

The M2T rules are applied on platform-specific models (JACK and OWL-S models) for the generation of JACK BDI agent codes and OWL-S documents (including Service, Profile, Process and Grounding documents) corresponding to semantic web agents and semantic web services designed in the system. As an example of the generated code, an excerpt of OWL-S Service file ("Service.owl") is shown in Listing 2. This file consists of the definition of the other documents for the service.

Although the codes generated for the MAS can be executed directly in the JACK environment, additional codes should be added into these generated codes, called delta code, by the developer to have the fully functional system.

The Generated Codes in the Target Language Environment

According to the proposed MAS development methodology, the generated code can be modified in the target language environment, JACK editor in the case of our study. JACK environment has a built-in graphical user interface that represents classes and their relations. After model-to-text transformations, JACK Java classes are produced for the customer agent from intermediate models. Apart from creating a Java class for the customer agent, separate Java classes are generated for this agent's capabilities, plans, events, and beliefs. Part of the generated codes demonstrated in the JACK editor is depicted in Figures 5 and 6. In the generated codes, the customer agent has two capabilities, namely Bartering and Searching. There are separate Java classes produced for these capabilities which are interlinked to the generated architecture code. Also, the Java classes that are generated for each capability of the agent, are linked to the plan, event, and belief classes that this capability requires.

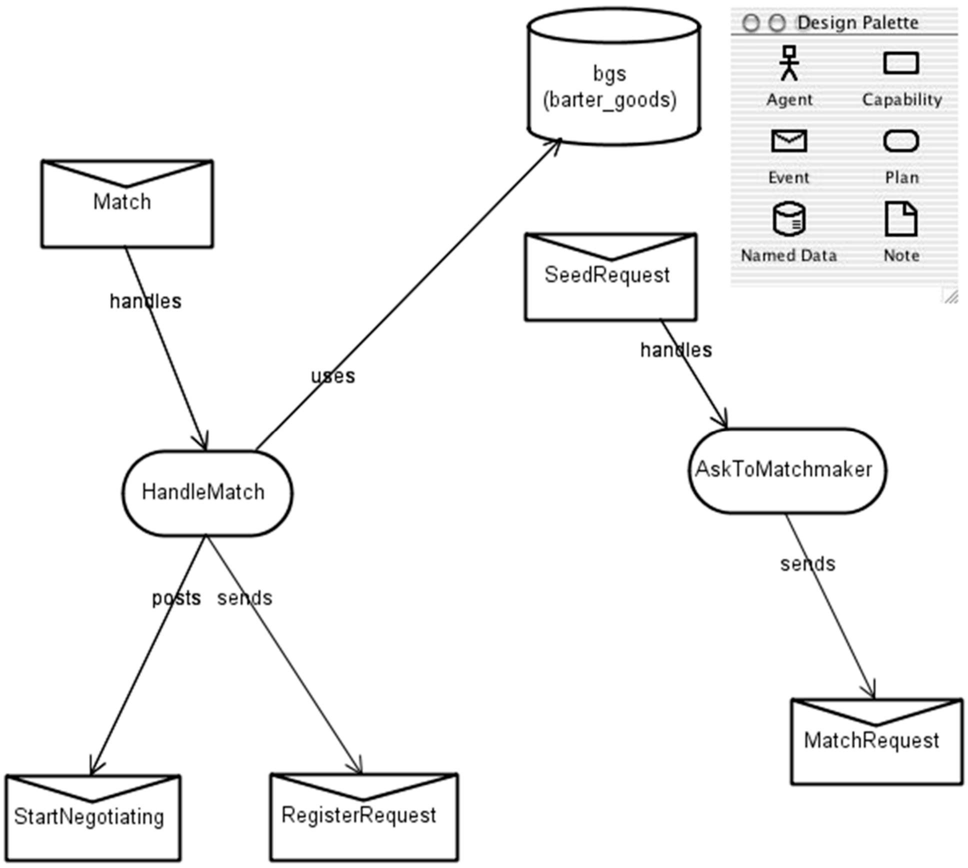


Figure 5. Searching capability in JACK editor.

As it is shown in Figure 5, the searching capability has two plans. One of them is AskToMatchmaker plan. This plan handles SeedRequest event and sends MatchRequest event to the Matchmaker agent for getting suitable services list. The HandleMatch plan handles Match event which is sent from the Matchmaker agent. The Match event encapsulates a service list. If the service list is null, it means that there is no suitable service for MatchRequest sent from Customer agent. Otherwise, the HandleMatch plan try to get appropriate agents from the services for bartering. If the

Customer agent finds suitable agents using the HandleMatch plan, it will start negotiation with these agents, if not, it will send a request to Matchmaker agent to register itself in a suitable service.

Bartering capability (see Figure 6) is responsible for the negotiations between the agents. When StartNegotiating event is posted, the PerformNegotiation plan is executed to handle it. NegotiationProposal event is created and sent to the relevant Customer agent. ProposalEval plan is responsible for evaluating incoming proposals and responding them. If the answer of the proposal is positive, the NegotiationFinalize plan is used to finalize the negotiation between two agents.

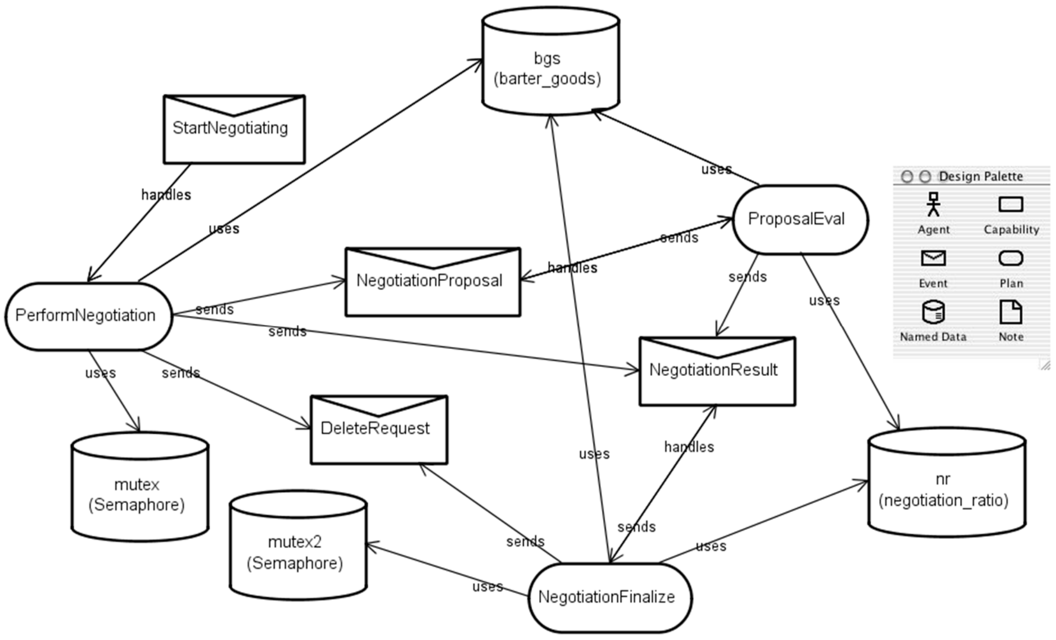


Figure 6. Bartering capability in JACK editor.

3.2.2. Delta Code Development

The codes generated by SEA_ML are architectural codes and the relations are established by the language considering the model which are controlled by the language at the semantic control stage that prevents most of the semantic errors in the code. Also, the codes do not have any syntactical errors at this level. The delta codes should be added manually to establish behavioral logic. Such code completion is need for both MAS and SWS parts of the system.

3.2.2.1. Delta Code for MAS Part of the E-barter System

The codes containing the negotiation logic of the Customer agents are critical to the system. These codes are located mostly in the plans of the agents, such as ProposalEval, NegotiationProposal, and PerformNegotiation, where most of the delta codes are added. In Listing 3, the delta code for the reasoning method of ProposalEval plan that allows an agent to evaluate proposals, is shown.

In this Listing, the templates of communicating messages in Lines 1, 9, 13, and 18–20 are generated and the other lines are added by the developer as delta code. In ProposalEval plan, there is a lower limit and an upper limit for the ratio between offered and needed products of each agent desiring a deal. In Lines 7 and 11, if the incoming bid is below the lower limit or there is an epsilon (0.0001) difference with the previous incoming bid, then the proposal is accepted by the agent. On the other hand, if the offer is above the upper limit, the agent will refuse the offer, as shown in Lines 12–14. If the incoming bid remains between the lower and upper limits, the Customer agent sends a new proposal to the agent which the initial proposal came from (Lines 15 to 20).

```
body(){
double actR=nr.getActRatio(ev1.from,$seed.getValue(),$offer.getValue());
double l=$lratio.getValue();
double u=$uratio.getValue();
double r=1.0/ev1.ratio;
self.guiMessage("incoming r:"+r+" from "+ ev1.from+" actR:"+actR);
if(r<=l || Math.abs(r-actR)<0.0001){
nr.add(ev1.from,$seed.getValue(),$offer.getValue(),r);
@reply(ev1,ev2.result(0,$seed.getValue(),$offer.getValue(),r));
self.guiMessage("The proposal came from "+ev1.from+" was accepted");
}
else if(r>u) {
@reply(ev1,ev2.result(1,$seed.getValue(),$offer.getValue(),r));
self.guiMessage("The proposal came from "+ev1.from+" was refused");
} else if(actR>0) r=(r+actR)/2.0;
else if(r<=(l+u)/2) r=(r+(l+u)/2)/2;
self.guiMessage("Sended r: "+r+ " to "+ev1.from );
nr.add(ev1.from,$seed.getValue(),$offer.getValue(),r);
@reply(ev1,ev2.result(2,$seed.getValue(),$offer.getValue(),r));
}
```

Listing 3. The generated and delta code for ProposalEval plan of Customer agent.

3.2.2.2. Delta Code for SWS Part of the E-barter System

In this case, study, each service has an ontology to help matchmaking. As there are 2 semantic web services developed for the E-barter system, we have two ontologies called electronic devices and food. These ontologies are used to demonstrate affinity relations in directing appropriate customer agents to appropriate services for barter processing. In the process of matching between the customer needs with services, each semantic web service uses its own ontology.

In the generation procedure, a structure is generated from the system model for each ontology. These structures are extended to develop the complete ontologies for the services. Part of the ontology developed for the food service is depicted in Figure 7. In this ontology, there are 4 product categories under the basic food node. These are the fruit, diary, vegetable and meat categories. These categories are divided into subcategories within themselves to obtain a tree structure in which the closeness relation can be established semantically.

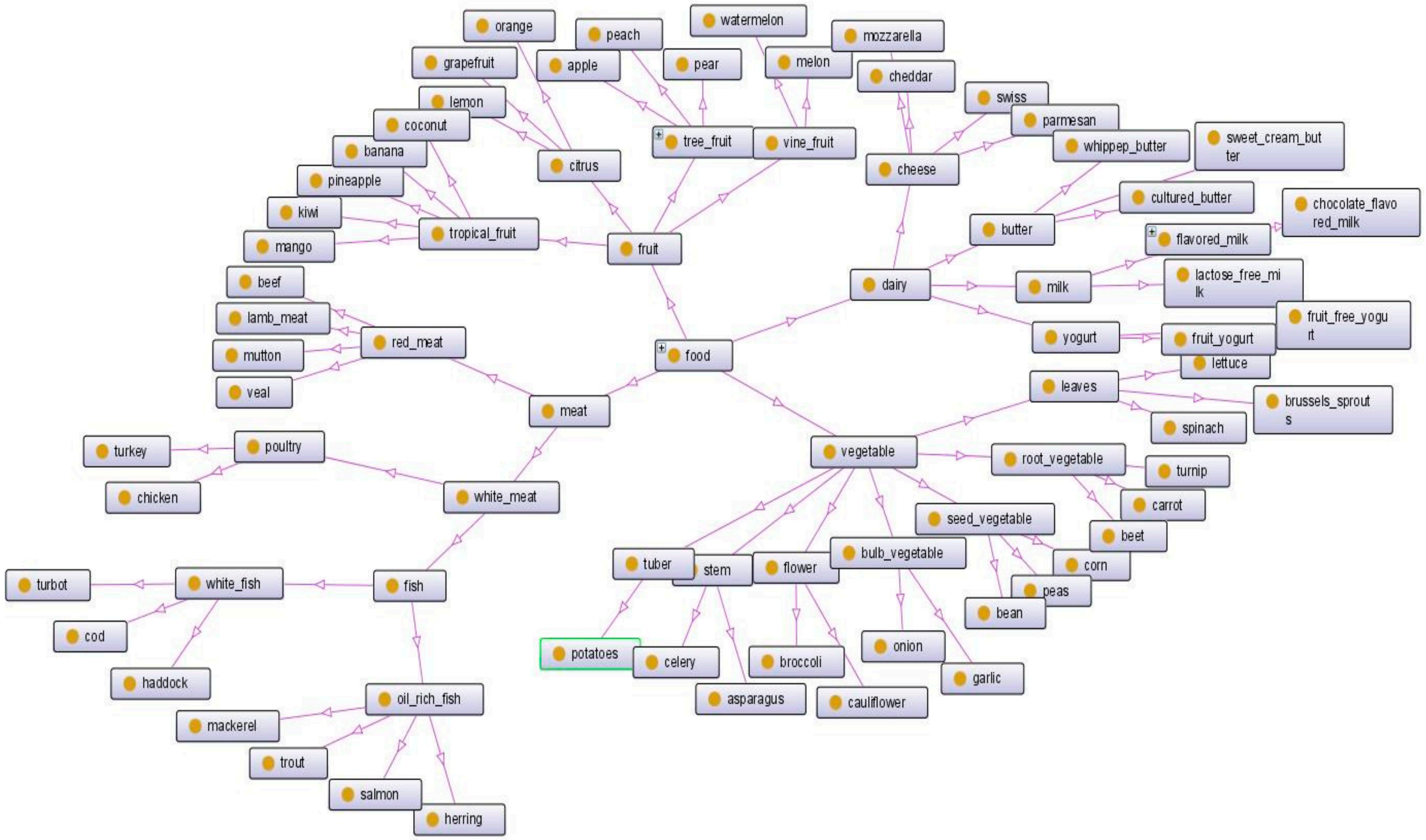


Figure 7. The ontology for food semantics web service.

The services used in E-barter system also need to have semantic web service documents to provide semantic web service functionality. For this purpose, the draft documents produced through SEA_ML have been used. Listing 4 shows the draft Profile.owl document produced by SEA_ML for the food service. The document generated by SEA_ML basically provides a draft of the resources imported with the owl: ontology tag in the rdf:RDF tag (Listing 4—Lines 16–25).

```
1   <?xml version="1.0"?>
2
3   <rdf:RDF xmlns:rdf= "&rdf;#"
4       xmlns:rdfs= "&rdfs;#"
5       xmlns:owl = "&owl;#"
6       xmlns:actor= "&actor;#"
7       xmlns:service= "&service;#"
8       xmlns:process= "&tradeFlyer;#"
9       xmlns:profile= "&tradeFlyer;#"
10      xmlns:profileHierarchy= "&profileHierarchy;#"
11      xmlns:xsd= "&xsd;#"
12      xmlns= "&DEFAULT;#"
13      xml:base= "&DEFAULT;">
14
15      <owl:Ontology rdf:about="">
16          <owl:imports rdf:resource="&service;" />
17          <owl:imports rdf:resource="&process;" />
18          <owl:imports rdf:resource="&profile;" />
19          <owl:imports rdf:resource="&foodService_process;" />
20          <owl:imports rdf:resource="&foodService_profile;" />
21
22          <owl:imports rdf:resource= "&time; />
23          <owl:imports rdf:resource= "&profileHierarchy; />
24      </owl:Ontology>
25  </rdf:RDF>
26
27    <!-- ################################################################### -->
28    <!-- #         Instance Definitions of the Serivce goes here           # -->
29    <!-- ################################################################### -->
30
```

Listing 4. Generated Profile for the food service.

This draft document was modified by the system developer in accordance with the ontology and hence the complete Profile.owl document (see Listing 5) was obtained. In this document, the profile tag, which is the label we used to determine the semantic proximity, was added to the draft to obtain the document in Listing 5 (Lines 18–39). Within the profile:Profile tag, the methods provided by that service are addressed which are based on the top-level concepts of the food ontology.

For the semantic proximity detection in the E-barter system, the methods that provide the main categories (the top-level concepts in the food ontology) are mapped to the profile documents. Thus, the Matchmaker agent can determine the appropriate method for the Customer agent through the profile document to find the appropriate service. Within this case study, only the Profile.owl document was used for SWS operations. The Matchmaker agent can propose the appropriate service to the Customer agent with the help of the profile:hasOutput tag in this document. The profile document contains categories at the method level, and these services correspond to the top-level concepts on the ontology. If the product category searched by the Matchmaker cannot be found in the profile, the service's ontology is traversed to find out if there is an upper category containing this product and the search in the profile document is repeated.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xml:base="http://localhost:8080/FoodService/FoodWebService/FoodWebService_Profile.owl#"
  xmlns:owl="http://jamsci.servehttp.com/owlsedit/owl.rdf#"
  xmlns:process="http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlseditFYP/owls11/Process.owl#"
  xmlns:profile="http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlseditFYP/owls11/Profile.owl#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://jamsci.servehttp.com/owlsedit/rdf-schema.rdf#"
  xmlns:service="http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlseditFYP/owls11/Service.owl#">
  <owl:Ontology rdf:about="">
    <owl:versionInfo>Version 1.0</owl:versionInfo>
    <rdfs:comment>Add Ontology Comment</rdfs:comment>
    <owl:imports rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns"/>
    <owl:imports rdf:resource="http://jamsci.servehttp.com/owlsedit/owl.rdf"/>
    <owl:imports rdf:resource="http://jamsci.servehttp.com/owlsedit/rdf-schema.rdf"/>
    <owl:imports rdf:resource="http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlseditFYP/owls11/Profile.owl"/>
    <owl:imports rdf:resource="http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlseditFYP/owls11/Service.owl"/>
  </owl:Ontology>
  <profile:Profile rdf:ID="FoodWebService_Profile">
    <service:presentedBy rdf:resource="FoodWebService_Service"/>
    <profile:serviceName>FoodWebService</profile:serviceName>
    <profile:textDescription/>
    <profile:hasInput rdf:resource="/FoodWebService/FoodWebService_ProcessModel#FoodWebService_register_agent_IN"/>
    <profile:hasInput rdf:resource="/FoodWebService/FoodWebService_ProcessModel#FoodWebService_register_need_IN"/>
    <profile:hasInput rdf:resource="/FoodWebService/FoodWebService_ProcessModel#FoodWebService_register_offer_IN"/>
    <profile:hasOutput rdf:resource="/FoodWebService/FoodWebService_ProcessModel#FoodWebService_register_return_OUT"/>
    <profile:hasInput rdf:resource="/FoodWebService/FoodWebService_ProcessModel#FoodWebService_delete_agent_IN"/>
    <profile:hasInput rdf:resource="/FoodWebService/FoodWebService_ProcessModel#FoodWebService_delete_need_IN"/>
    <profile:hasInput rdf:resource="/FoodWebService/FoodWebService_ProcessModel#FoodWebService_delete_offer_IN"/>
    <profile:hasOutput rdf:resource="/FoodWebService/FoodWebService_ProcessModel#FoodWebService_delete_return_OUT"/>
    <profile:hasInput rdf:resource="/FoodWebService/FoodWebService_ProcessModel#FoodWebService_meat_need_IN"/>
    <profile:hasOutput rdf:resource="/FoodWebService/FoodWebService_ProcessModel#FoodWebService_meat_return_OUT"/>
    <profile:hasInput rdf:resource="/FoodWebService/FoodWebService_ProcessModel#FoodWebService_dairy_need_IN"/>
    <profile:hasOutput rdf:resource="/FoodWebService/FoodWebService_ProcessModel#FoodWebService_dairy_return_OUT"/>
    <profile:hasInput rdf:resource="/FoodWebService/FoodWebService_ProcessModel#FoodWebService_fruit_need_IN"/>
    <profile:hasOutput rdf:resource="/FoodWebService/FoodWebService_ProcessModel#FoodWebService_fruit_return_OUT"/>
    <profile:hasInput rdf:resource="/FoodWebService/FoodWebService_ProcessModel#FoodWebService_vegetable_need_IN"/>
    <profile:hasOutput rdf:resource="/FoodWebService/FoodWebService_ProcessModel#FoodWebService_vegetable_return_OUT"/>
    <profile:qualityRating/>
  </profile:Profile>
</rdf:RDF>
```

Listing 5. Completed Profile for the food service.

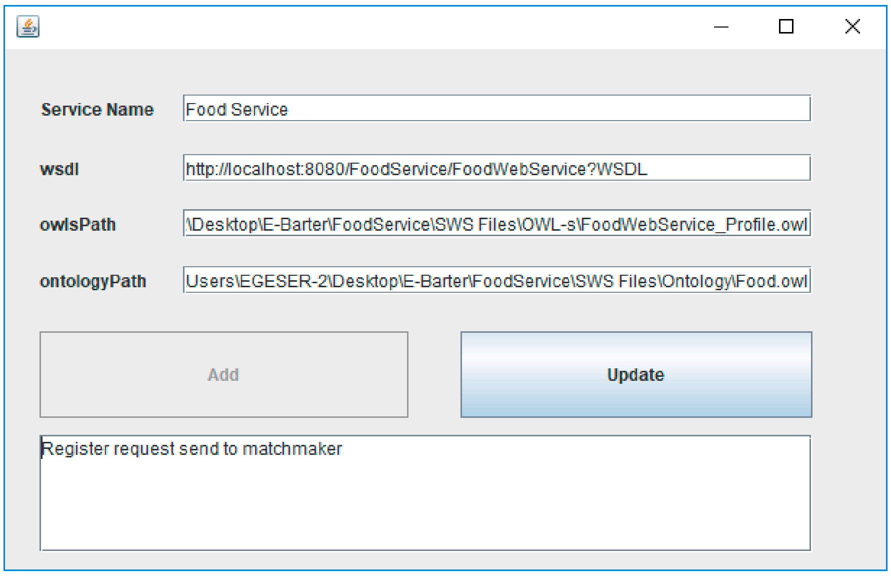
The Matchmaker returns the WSDL addresses of the appropriate services to the Customer agent after determining the appropriate services. When the Customer agent sends a request to a service which is just found, its repository is used which is an XML file containing the names, requests, and offers of agents eligible for bartering. The service searches for those agents which can match with the Customer agent on this XML file. If some agents are found, the service will return the names of these agents to the Customer agent.

4. Demonstration Scenario

For a detailed demonstration of the implemented system, this section illustrates a system execution consisting of three Customer agents and two semantic web services.

When the system is started to run on the client side, the initial interface is shown. On this interface, the user can add a service agent that represents a semantic web service or a Customer agent which represents himself/herself to propose a bargaining.

First, service agents, that represent semantic web services, should be added to the system. When the "Add Service Agent" in the initial user interface is selected, the interface for adding a service agent appears (see Figure 8) to get the information necessary for registering the semantic web service. Then an agent is run to represent the relevant semantic web service and sends a message to Matchmaker agent to register the semantic web service.



Service Name: Food Service

wsdl: http://localhost:8080/FoodService/FoodWebService?WSDL

owlsPath: \Desktop\E-Barter\FoodService\SWS Files\OWL-s\FoodWebService_Profile.owl

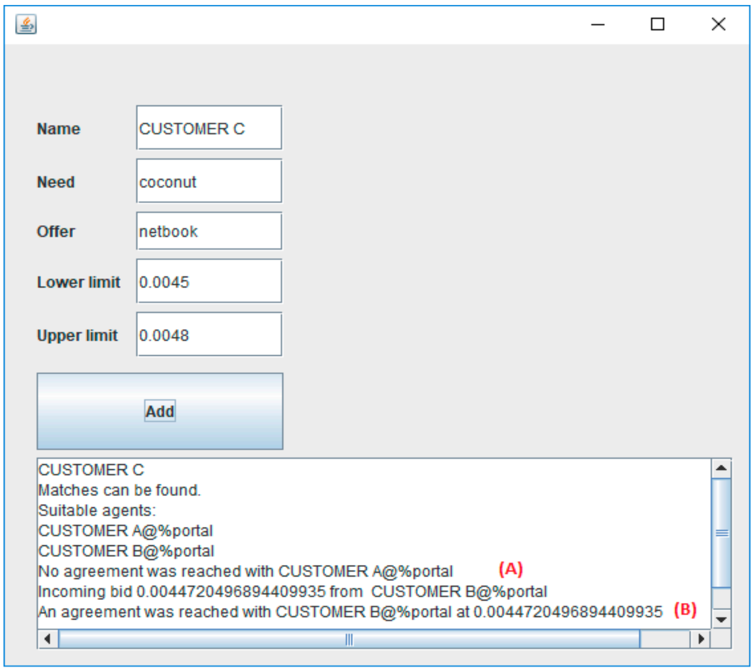
ontologyPath: Users\EGESER-2\Desktop\E-Barter\FoodService\SWS Files\Ontology\Food.owl

```
Register request send to matchmaker
```

Figure 8. The user interface for adding a service agent to the system.

In our demonstration scenario, there are two semantic web services called "Food Service" and "Electronic Device". For each one, a separate service agent must be established.

After the services are registered in the system, Customer agents can be included in the system. For this purpose, "Adding Customer Agent" interface (an example shown in Figure 9) needs to be launched from the initial user interface. Using this interface, the agent is created after the information for the Customer agent is entered. To instantiate a Customer agent, the product which is needed and offered as well as the lower and upper limits must be prescribed by the user for the bartering. In this scenario, three agents named CUSTOMER A, CUSTOMER B, and CUSTOMER C, are included to the system with the details (such as Name, Need, Offer, Lower and Upper limits) provided in Table 1. Figure 9 shows, as an example, the instantiation of "CUSTOMER C" agent in the system.



Name: CUSTOMER C

Need: coconut

Offer: netbook

Lower limit: 0.0045

Upper limit: 0.0048

```
CUSTOMER C
Matches can be found.
Suitable agents:
CUSTOMER A@%portal
CUSTOMER B@%portal
No agreement was reached with CUSTOMER A@%portal        (A)
Incoming bid 0.0044720496894409935 from  CUSTOMER B@%portal
An agreement was reached with CUSTOMER B@%portal at 0.0044720496894409935  (B)
```

Figure 9. Instantiation of "CUSTOMER C" agent.

Table 1. Details of the Customer Agents in the E-barter System.

Name	Need	Offer	Lower Limit	Upper Limit
CUSTOMER A	netbook	Coconut	150	200
CUSTOMER B	netbook	Coconut	200	250
CUSTOMER C	coconut	Netbook	0.0045	0.0048

CUSTOMER A agent would like to barter with some other agents who offer netbooks. Meanwhile, this agent offers coconuts for netbooks. To achieve this, CUSTOMER A creates a barter request. This barter request is sent to the Matchmaker agent which uses each service's own ontology to send the customer a list of the most appropriate semantic web services. In this way, the agent gets semantically close services that accommodate the agents providing the needed product. However, till this point of the scenario, no agent has been introduced to the system before, so if CUSTOMER A agent finds appropriate services, these services will not return any agents that CUSTOMER A can barter. Therefore, CUSTOMER A will be registered to an appropriate service and wait for the new agents to be added to the system.

In this scenario, after CUSTOMER A, CUSTOMER B and CUSTOMER C agents are involved in the system. The requirements of the CUSTOMER B are the same as the CUSTOMER A, except for the lower and upper limits that the CUSTOMER A determines for bartering. In this case, CUSTOMER B would register to a suitable service because it will not find a suitable agent for bartering. The CUSTOMER C agent offers "netbook" and requests "coconut", unlike CUSTOMER A and CUSTOMER B. Therefore, a semantic web service, which is found by the Matchmaker agent, sends suitable agents for bartering to CUSTOMER C. In this case, CUSTOMER A and CUSTOMER B are the suitable agents for CUSTOMER C.

At this point, the negotiation between agents starts. First, CUSTOMER C sends a proposal to CUSTOMER A. However, as seen in Line (A) of Figure 9, the CUSTOMER A rejects this offer because the offer sent by the CUSTOMER C is higher than the upper limit of the CUSTOMER A. Then, CUSTOMER C sends the same proposal to the CUSTOMER B. Since the proposal is in the acceptable range of CUSTOMER B, the negotiation between them begins and eventually, an agreement was reached by these two agents as shown in Line (B) of Figure 9.

5. Related Work

The work conducted in this study is mainly related with two research fields: MAS development methodologies and e-barter systems. Hence, in the following, we first discuss the efforts given on deriving MAS development methodologies as similar to our proposal and then give some noteworthy studies on developing e-barter systems which especially consider employing agents.

There are various AOSE methodologies which can be used for the development of MAS. Methodologies such as ADELFE [41], Gaia [42], INGENIAS [43], PASSI [44], Prometheus [45] and Tropos [46] provide systematic processes including analysis and design of agent systems. It is also possible to integrate the outcomes of these methodologies with various agent execution platforms/frameworks such as JADE [18], JACK [20], Jason [47], CArtAgO [48], MOISE [49] and JaCaMo [50] to implement designed agents. However, neither of these methodologies nor platforms directly support model-driven MAS development. In fact, re-engineered and improved versions of some of these methodologies (e.g., [51] for ADELFE, [52] for INGENIAS, [53] for Prometheus, [54] for Tropos) enable MAS development according to MDD paradigm as indicated in [55]. Although MAS modeling from different viewpoints and code generation for various agent platforms are also covered in these updated methodologies, model-driven development of semantic web services and interactions between agents and these semantic web services are not included. Moreover, an iterative process supporting modeling and implementation are not considered and it is too difficult to modify intermediate models and update auto-generated codes by using most of these methodologies. The only exception is INGENIAS, which deems supporting the iterative development and modification of models and codes as similar to the MDD methodology proposed in this study.

In addition to abovementioned AOSE methodologies, the researchers have significant efforts on using model-driven approaches for agent development and the derivation of DSMLs for MAS. For instance, Agent-DSL [56] is used to specify the agency properties that an agent needs to accomplish its tasks. However, the proposed language is presented only with its metamodel and provided just a visual modeling of the agent systems according to agent features, such as knowledge, interaction, adaptation, autonomy and collaboration. The A&A metamodel introduced in [8] considers the notion of artifacts for agents. In the A&A metamodel, agents are modeled as proactive entities for the systems' goals and tasks while the artifacts represent the reactive entities providing the services and the functions and, hence, constitute the environment for MAS. The FAML metamodel, introduced in [9], is a synthesis of various existing metamodels for agent systems. Design time and runtime concepts for MASs are given and validation of these concepts is provided by their use at various MAS development methodologies.

Hahn [11] introduces a DSML for MAS called DSML4MAS. The abstract syntax of the DSML is derived from a platform independent metamodel structured into several aspects, each focusing on a specific viewpoint of a MAS. To provide a concrete syntax, the appropriate graphical notations for the concepts and relations are defined. Furthermore, DSML4MAS supports the deployment of modeled MASs both in JACK and JADE agent platforms by providing an operational semantics over model transformations [57]. DSML4MAS also guides MDD of different agent applications. For instance, Ayala et al. [58] use DSML4MAS for the development of agent-based ambient intelligence systems. The metamodel of DSML4MAS is employed as a source metamodel to support the modeling of context aware systems and conforming models are transformed into target models which are instances of an aspect-oriented agent metamodel called Malaca. Code generation enables the implementation of Malaca models to run in the ambient intelligence devices.

Another model driven MAS development approach is provided in [12] with introducing a new DSML. The abstract syntax of the DSML is presented using the Meta-object Facility (MOF), the concrete syntax and its tool is provided with Eclipse Graphical Modeling Framework (GMF), and finally the code generation for the JACK agent platform is realized with model transformations using Eclipse JET. The language supports modeling of agents according to Prometheus methodology [45]. A similar study is performed in [59] which proposes a technique for the definition of agent-oriented engineering process models and can be used to define processes for creating both hardware and software agents. This study also offers a related MDD tool based on INGENIAS methodology [52].

The work conducted in [13] aims at creating a UML-based agent modeling language, called MAS-ML, which can model the well-known types of agent internal architectures, namely simple reflex agent, model-based agent, reflex agent, goal-based agent and utility-based agent. Representation and exemplification of all supported agent architectures in the concrete syntax of the introduced language are given. MAS-ML is also accompanied with a graphical tool which enables agent modeling. However, the current version of MAS-ML does not support any code generation for MAS frameworks which prevents the execution of the modeled agent systems.

Wautelet and Kolp [60] investigate 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 [61] 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. However, implementation of the required transformations and code generation are not included in the study.

Bergenti et al. [15] propose a language, called JADEL, for the MDD of agents on JADE platform. Instead of covering all features of JADE, JADEL only provides high-level agent-oriented abstractions, namely agents, behaviors, communication ontologies, and interaction protocols. JADEL is supported with a compiler which enables source code generation for implementing agents on JADE platform. However, the related code generation feature of JADEL is not currently functional enough to fully implement JADE agents as also indicated in [15].

The new MAS development methodology, introduced in this paper, differentiates from many of the above MAS development approaches with presenting a complete development process including analysis, design and implementation phases according to MDD principles. In most of these studies (e.g., [8,9,13,15,56]) only the derivation of metamodels and/or DSMLs is considered without a guide for how those metamodels/DSMLs can be utilized within a structural development process. Only the remaining works in [11,12,59] and can be said to describe some sort of MDD processes along with the proposed DSMLs. Benefiting from the features of SEA_ML, the development process, discussed in this paper, enables both modeling and automatic generation of semantic constructs required for the discovery and execution of semantic web services by the agents. Such a development opportunity for agent-semantic web services interactions is not considered in those MDD processes.

On the other hand, there are some studies in literature addressing the development of E-barter systems with different approaches. Generally, these studies aim at formalizing the domain and increasing its effectiveness.

For example, Lopez et al., performed two consecutive studies to create a formal framework for E-barter systems [62,63]. In their first study [62], they propose a formal framework in which customers are grouped in local markets according to their location, so that a global market takes a tree-like shape. While all these processes are identified, algebraic notation and some microeconomic theory concepts have been used. In addition, a utility function has been defined to indicate the valuation of customers in the exchange of goods. The use of algebra and micro-economy help to eliminate ambiguity and get the scheme of the system. The second study [63] focuses on transactions and shipping costs which are not considered in the first study. The early framework has been extended to include these concepts.

Another study to formalize E-barter systems is the study of Nunez et al. [64]. This study presents a classical algebra-based language to identify and analyze E-barter systems. This framework also suggests a hierarchical market structure. Product exchanges are made using the agents representing the customers. It is shown that the barter balance of the goods provides a Pareto optimum.

These studies focus on the formal representation of E-barter systems while our study mostly focuses on the efficient development of these systems. The market structure in our system is adopted from [64]. However, any other structure and formalism can be integrated into our study.

Cavalli and Maag [65] have developed an approach and its supporting tool that generates test scenarios suitable for the E-Barter system. These scenarios are intended to test the compatibility of the system with the intended functions. System specs were implemented using the Specification and Description Language (SDL) [66]. With this method, design mistakes are prevented in the early stages of development. However, test case generation is not the aim of our study.

In the study by Bravetti et al. [23], an E-barter system have been designed using multi-agent architecture with web services. In this design, BPEL4WS [67] web services are used. The focus of this study is filling the gap between formal representation and design of the system. This study is based on the formal representations provided in [62,63]. However, in [23], the authors focus on the problems in the use of these formal representations in design time. Bravetti et al. also designed the E-barter system using WS-BPEL [67] web services [68]. The studies of [23,68] uses web services as the base element for the development of the E-barter system, however, our study benefits from semantic web services and provides semantically matching capability for bartering.

Ragone et al. [69] focused on E-barter systems with a new knowledge-based approach in their study. The goal is to ensure that multiple barter situations are performed with optimal matching. In this study, a logical language was introduced that provides more complex specifications of agents' requests. It is also intended to simulate the semantic similarity between proposals that will be presented in a logic-based utility function [69]. On the other hand, in [70], a game concept was defined to describe the interactions in the barter system. According to this concept, there are several agents, and these agents have vectors with parameters specifying their requests and bids. Bartering is performed according to the matching of these vectors.

The studies presented in [69,70] focuses on improving the matching mechanism for bartering using more specific parameters in the definition of requirements and logic-based utility functions.

However, our study tackles semantic matching in two levels: one in finding the closest semantic web service and another in the level of items to be bartered using ontologies.

Abdalla et al. [25] have designed an agent-based application called Bartecell. In this application, software agents can work on wireless networks and reach mutually beneficial barter agreements. New negotiation algorithms have also been introduced for transactions between agents. An E-barter architecture compatible with mobile devices has been introduced that provides location-based services [25]. This study focusses on the use of agents and benefiting from wireless networks in the negotiation of those agents for E-bartering propose. However, unlike our study, Bartecell does not utilize semantic web services and neither considers the semantic discovery of these services nor semantic matching of the bartering items.

Dhaouadi et al. [26] have designed and developed a MAS for supply chain automation. The system automatically recommends suitable suppliers for handicraft women (HDWs). The recommendation procedure is based on two supplier selection levels and then a negotiation phase. The first level is the process of selecting vendors that sell the necessary products. On the second level, it only specifies vendor profiles that can successfully match HDW. During the negotiation phase, the relevant actors will conduct discussions on the required quantity, quality, cost and delivery processes. Ontologies have also been utilized in the operation of these processes. Although this study addresses a different domain than ours, the general approach is close when considering the two selection levels. However, they use the ontologies only in the second level where they specify vendor profiles which match HDWs. In our study, selection of the categories in the first level is also done with the help of semantic matching of services. Moreover, the selection mechanism in the first level is automatized by using semantic web services.

In [24], a MAS was developed for the E-barter systems. Unlike other studies, an architecture has been designed and implemented that uses ontology-based comparisons in bid mapping. This architecture introduces a type of agent named Barter Manager Agent in addition to the E-barter agents. This agent determines the barter partners according to semantic proximity. With this approach, it is aimed to find the best match, not just based on the price and quantity of goods but also considering the relation between supply and demand. There are two groups of agents in this design. The Service Management Agent group including the Barter manager agent, SWS agent, and Cargo agent, which are responsible for managing the barter operations. The barter manager Agent is responsible for the management of the agent barter operations and matching. The SWS agent is responsible for the mapping based on the ontological proximity. The cargo agent plays a role in the exchange of products in the next stage after the barter operation is completed. The User Agents group is the group that contains the Customer agents. In this group, there are agents that request barter. In this study, the system was implemented using the JADE language [18].

In [71], an E-barter system was designed using MAS-CommonKADS methodology [72]. The focus of this system is in the negotiation phase. A bargaining protocol between two matched agents was presented.

In [27], an E-barter system was designed and developed by using Prometheus methodology [44]. In this system, BDI agents are used which is not the case in the previous studies in the literature. System development is realized using JACK intelligent agent platform [20,38]. Ontologies have been used for the matching purpose. There are two basic agents in the system. The Matchmaker agent is responsible for performing appropriate matching between customers. It benefits from ontologies for these mappings. The customer agents request bartering and negotiate with each other to realize the barter transaction. This study proposes a new rationale for the negotiations which takes the ratio of offer and need into consideration.

We believe that our work on the development of E-barter systems contributes to the abovementioned efforts by first utilizing the semantic web services and capabilities of BDI agents for E-bartering instead of reactive agents which is preferred in most of the previous work. In addition, the remaining studies (e.g., [27]) which utilizing BDI model for E-bartering do not benefit from internal reasoning mechanism of BDI agents which can bring extra intelligence in the procedure of matchmaking for the E-bartering. Also, these studies do not use SWSs as the automatic selection mechanism for the categories. Finally, while the methodologies of the other studies are generic for agent development, our study uses SEA_ML and its tool inside a domain-specific methodology. This makes the design and development of the MAS system easier and less-costly as the developer works with the domain concepts and benefits from generative capability of the provided tool.

6. Discussion and Conclusions

In this paper, a development methodology is proposed for development of MASs working in semantic web environments. This methodology is based on a DSML, called SEA_ML. The study is demonstrated using a case study for E-barter. To this end, the BDI agents and the SWSs for the E-barter system are analyzed, designed and developed using different viewpoints and features of SEA_ML. Also, a demonstration scenario is provided for the implemented system.

In the traditional E-barter systems [62–64], customers and their products' information are stored in databases in a monolithic way. This approach has two major disadvantages. First, the system is not scalable. By adding different product categories, the maintenance effort and cost of the system will be increased. The second disadvantage is that a semantic approach cannot be achieved with the traditional methods. In this study, these two disadvantages have been overcome by using SWS. The use of SWS primarily ensures that the customer and the product information are stored categorically in the external services. This leads to a more scalable system. In addition, these services with the semantic structure allow a semantic logic to be implemented in the matching and gives the opportunity to the customers to communicate only with the appropriate services. This increases the likelihood that the barter process has successful result in a limited time.

In this study, we also experienced that the proactive behavior of the BDI agents may help the fruitful application of E-barter systems by especially preventing bartering the goods ineffectively with undesired exchange ratios. Implemented BDI agents in here aim at choosing the most appropriate plan to achieve the maximum gain out of the bargaining on behalf of their users. It is possible to develop a similar MAS for the same purpose with agent models other than BDI which probably leads to provide desired efficiency in bargaining. However, in addition to the achieved fruitfulness, we also found modeling and implementation of the MAS for e-bartering convenient by utilizing BDI constructs and their relations.

On the other hand, using the proposed methodology, an efficient implementation of the system is possible through an accurate design with few errors in the analysis and the design phases [21]. In our work, semantic errors can be detected during the analysis and design phase using a SEA_ML-based methodology and the implementation process is completed in a shorter time with fewer problems. Achieved results validated the generation and the development-time performance of SEA_ML discussed in [21]. For the E-barter case study discussed in this paper, application of the SEA_ML-based MAS development methodology enabled the generation of approximately more than half of the whole agents and artifacts.

Despite the abovementioned advantages, the proposed approach in this work can be improved in several ways. Our future work consists of the followings: The semantic web services in the system are currently added to the system in a static way to facilitate the development These services can be expected to be added to the system automatically by communicating with the E-barter system. Moreover, the negotiation logic between the customers is presented at a basic level. Successful barter transactions may require a more complex negotiation rationale [73]. In addition, the business logic in the agent plans are mostly developed as the delta codes and hence our aim is to leverage the comprehensiveness of the models. In this way, it is possible to reduce the delta code and gain more generation performance using the proposed methodology. We also plan to empower the current methodology by using the formal methods discussed in [74,75] for SEA_ML to validate the models and check some domain properties in the design level for the MAS.

As another future study, our aim is to provide the execution of SEA_ML agents on Jason platform [47]. Similar to operational semantics of SEA_ML currently provided for JACK, MAS models prepared in SEA_ML can be transformed into Jason model instances and code generation can also be possible for this platform. Recently, we derived a metamodel [76] for Jason agents and we plan to use this metamodel as another PSMM and prepare a series of M2M transformations between SEA_ML metamodel and this metamodel which will lead to generation of Jason agents inside the MAS development methodology proposed in this paper.

Author Contributions: Moharram Challenger and Geylani Kardas conceived the proposed MAS development methodology. Moharram Challenger also leaded the writing of the paper. Baris Tekin Tezel and Omer Faruk Alaca implemented the E-barter system and performed the experiments. Bedir Tekinerdogan contributed to the analysis and assessment of the results.

Acknowledgements: This work is partially funded by the Scientific and Technological Research Council of Turkey (TUBITAK) under grant 115E591.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Descriptions of Selected SEA_ML Concepts

Table A1. SEA_ML concepts, their notations and descriptions.

Icon.	Concept	Description
👤	Semantic Web Agent (SWA)	Semantic web agent in the SEA_ML stands for each agent which is a member of semantic web-enabled MAS. It is an autonomous entity which can interact with both the other agents and the semantic web services, within the environment.
👤	Semantic service matchmaker agent (SSMatchmakerAgent)	It is a SWA extension. This meta-element represents matchmaker agents which store the SWS' capabilities list in a MAS and compare it with the service capabilities required by the other agents, in order to match them.
📖	Belief	Beliefs represent the informational state of the agent, in other words its knowledge about the world (including itself and other agents).
◎	Goal	A goal is a desire that has been adopted for active pursuit by the agent.
🎭	Role	An agent plays different roles to realize different behaviors in various situations, such as organizations, or domains.
📁	Capability	Taking BDI agents into consideration, there is an entity called Capability which includes each agent's Goals, Plans and Beliefs about the surroundings.
📄	Fact	The statement about the agent's environment which can be true. Agents can decide based on these facts.
📄	Plan	Plans are sequences of actions that an agent can perform to achieve one or more of its intentions.
R	Semantic service register plan (SS_RegisterPlan)	The Semantic Service Register Plan (SS_RegisterPlan) is the plan used to register a new SWS by SSMatchmakerAgent.
F	Semantic service finder plan (SS_FinderPlan)	Semantic Service Finder Plan (SS_FinderPlan) is a Plan in which automatic discovery of the candidate semantic web services take place with the help of the SSMatchmakerAgent.
A	Semantic service agreement plan (SS_AgreementPlan)	Semantic Service Agreement Plan (SS_AgreementPlan) is a concept that deals with negotiations on quality of service (QoS) metrics (e.g., service execution cost, duration and position) and contract negotiation.
E	Semantic service executor plan (SS_ExecutorPlan)	After service discovery and negotiation, the agent applies the Semantic Service Executor Plan (SS_ExecutorPlan) to invoke appropriate semantic web services.
✉	Send	An action to transmit a message from an agent to another. This can be based on some standard such as FIPA_Contract_Net
✉	Receive	An action to collect a message from an agent. This can be based on some standard such as FIPA_Contract_Net

	Task	Tasks are groups of actions which are constructing a plan in an agent.
	Action	An action is an atomic instruction which constitutes a task.
	Message	A package of information to be send from an agent to another; possibly to deliver some information or instructions. Two special types of actions, namely Send and Receive, are used to handle these messages.
	Agent state	This concept refers to certain conditions in which agents are present at certain times. An agent can only have one state (Agent State) at a time, e.g., waiting state in which the agent is passive and waiting for another agent or resource.
	Resource	It refers to the system resources that the MAS is interacting with. For example, the database.
	Service	Any computer-based service presented to the users.
	Web Service	Type of service which is presented via web.
	Semantic Web Service	Semantically defined web services which can be interpreted by machines.
	Process	It describes how the SWS is used by defining a process model. Instances of the SWS use the process via described_by to refer to the service's ServiceModel.
	Interface	This document describes what the service provides for prospective clients. This is used to advertise the service, and to capture this perspective, each instance of the class Service presents a Service Interface.
	Grounding	In this document, it is described how an agent interact with the SWS. A grounding provides the needed details about transport protocols. Instances of the class Service have a supports property referring to a Service Grounding.
	Input	Defines the inputs for processes and interfaces of a SWS.
	Output	Defines the output for processes and interfaces of a SWS.
	Precondition	Defines the pre-conditions for processes and interfaces of a SWS.
	Effect	Defines the post-conditions or effects for processes and interfaces of a SWS.
	Semantic web organization	Refers to an organized group of semantic web agents (SWAs).
	Interaction	For communication and collaboration of agents, they can use series of messages via a message sequence which results to an agent interaction.
	Environment	The agent's surroundings including digitized resources, fact, and services.
	Registration Role	A specialized type of architectural role which is used to register SWSs in the multi agent systems.
	Behavior	In re-active agents, a behavior is a re-action of an agent towards an external or internal stimulus.
	Agent type	The agents in a multi-agent system can have different types taking various responsivities and representing various stakeholders.

References

1. Wooldridge, M.; Jennings, N.R. Intelligent agents: Theory and practice. *Knowl. Eng. Rev.* **1995**, *10*, 115–152.
2. Kardas, G.; Goknil, A.; Dikenelli, O.; Topaloglu, N.Y. Model Driven Development of Semantic Web Enabled Multi-agent Systems. *Int. J. Coop. Inf. Syst.* **2009**, *18*, 261–308.
3. Berners-Lee, T.; Hendler, J.; Lassila, O. The Semantic Web. *Sci. Am.* **2001**, *284*, 34–43.
4. Shadbolt, N.; Hall, W.; Berners-Lee, T. The Semantic Web Revisited. *IEEE Intell. Syst.* **2006**, *21*, 96–101.
5. Mernik, M.; Heering, J.; Sloane, A. When and how to develop domain-specific languages. *ACM Comput. Surv.* **2005**, *37*, 316–344.
6. Fowler, M. *Domain-Specific Languages*; Addison-Wesley Professional: Boston, MA, USA, 2001; pp. 1–640.
7. Kardas, G.; Gomez-Sanz, J.J. Special issue on model-driven engineering of multi-agent systems in theory and practice. *Comput. Lang. Syst. Struct.* **2017**, *50*, 140–141.
8. Omicini, A.; Ricci, A.; Viroli, M. Artifacts in the A&A meta-model for multi-agent systems. *Autonom. Agents Multi-Agent Syst.* **2008**, *17*, 432–456.
9. Beydoun, G.; Low, G.C.; Henderson-Sellers, B.; Mouratidis, H.; Gomez-Sanz, J.J.; Pavon, J.; Gonzalez-Perez, C. FAML: A Generic Metamodel for MAS Development. *IEEE Trans. Softw. Eng.* **2009**, *35*, 841–863.
10. Garcia-Magarino, I. Towards the integration of the agent-oriented modeling diversity with a powertype-based language. *Comput. Stand. Interfaces* **2014**, *36*, 941–952.
11. Hahn, C. A Domain Specific Modeling Language for Multiagent Systems. In Proceedings of the 7th International Conference on Autonomous Agents and Multi-Agent Systems, Estoril, Portugal, 12–16 May 2008; pp. 233–240.
12. Gascuena, J.M.; Navarro, E.; Fernandez-Caballero, A. Model-Driven Engineering Techniques for the Development of Multi-agent Systems. *Eng. Appl. Artif. Intell.* **2012**, *25*, 159–173.
13. 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. *J. Syst. Softw.* **2015**, *108*, 77–109.
14. Faccin, J.; Nunes, I. A Tool-Supported Development Method for Improved BDI Plan Selection. *Eng. Appl. Artif. Intell.* **2017**, *62*, 195–213.
15. Bergenti, F.; Iotti, E.; Monica, S.; Poggi, A. Agent-oriented model-driven development for JADE with the JADEL programming language. *Comput. Lang. Syst. Struct.* **2017**, *50*, 142–158.
16. 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 the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT 2008), Sydney, Australia, 9–12 December 2008; pp. 200–206.
17. 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. *Eng. Appl. Artif. Intell.* **2014**, *28*, 111–141.
18. Bellifemine, F.L.; Caire, G.; Greenwood, D. *Developing Multi-Agent Systems with JADE*; John Wiley & Sons: Hoboken, NJ, USA, 2007; Volume 7.
19. Pokahr, A.; Braubach, L.; Lamersdorf, W. Jadex: A BDI Reasoning Engine. In *Multi-Agent Programming Languages, Platforms and Applications*; Bordini, R.H., Dastani, M., Dix, J., El Fallah Seghrouchni, A., Eds.; Springer: Berlin, Germany, 2005; pp. 149–174.
20. Howden, N.; Ronnquist, R.; Hodgson, A.; Lucas, A. Jack intelligent agents- summary of an agent infrastructure. In Proceedings of the 5th International Conference on Autonomous Agents (AGENTS’01), Montreal, QC, Canada, 28 May–1 June 2001.
21. Challenger, M.; Kardas, G.; Tekinerdogan, B. A systematic approach to evaluating domain-specific modeling language environments for multi-agent systems. *Softw. Qual. J.* **2016**, *24*, 755–795.
22. Rao, A.S.; Georgeff, M.P. Decision procedures for BDI logics. *J. Log. Comput.* **1998**, *8*, 293–343.
23. Bravetti, M.; Casalboni, A.; Nunez, M.; Rodriguez, I. From Theoretical e-barter Models to an Implementation Based on Web Services. *Electr. Notes Theor. Comput. Sci.* **2006**, *159*, 241–264, doi:10.1016/j.entcs.2005.09.034.
24. Demirkol, S.; Getir, S.; Challenger, M.; Kardas, G. Development of an agent-based E-barter system. In Proceedings of the International Symposium on INnovations in Intelligent SysTems and Applications, Istanbul, Turkey, 15–18 June 2011; pp. 193–198, doi:10.1109/INISTA.2011.5946060.

25. Abdalla, S.; Swords, D.; Sandygulova, A.; O'Hare, G.M.P.; Giorgini, P. BarterCell: An agent-based bartering service for users of pocket computing devices. In Proceedings of the International Conference on Industrial Applications of Holonic and Multi-Agent Systems, Prague, Czech Republic, 26–28 August 2013. doi:10.1007/978-3-642-40090-2_21.
26. Dhaouadi, R.; Salah, K.B.; Miled, A.B.; Ghédira, K. Ontology based multi-agent system for the handicraft domain e-bartering. In Proceedings of the 28th Bled eConference: Wellbeing, Bled, Slovenia, 7–10 July 2015; pp. 353–367.
27. Cakmaz, Y.E.; Alaca, O.F.; Durmaz, C.; Akdal, B.; Tezel, B.; Challenger, M.; Kardas, G. Engineering a BDI Agent-based Semantic e-Barter System. In Proceedings of the International Conference on Computer Science and Engineering (UBMK'17), Antalya, Turkey, 5–8 October 2017.
28. Miyashita, K. Incremental Design of Perishable Goods Markets through Multi-Agent Simulations. *Appl. Sci.* **2017**, *7*, 1300, doi:10.3390/app7121300.
29. Martin, D.; Burstein, M.; Hobbs, J.; Lassila, O.; McDermott, D.; McIlraith, S.; Narayanan, S.; Paolucci, M.; Parsia, B.; Payne, T.; et al. OWL-S: Semantic Markup for Web Services. W3C Member Submission. Available online: http://www.w3.org/Submission/OWL-S/ (accessed on 7 April 2018).
30. WSMO: Web Service Modeling Ontology. Available online: http://www.w3.org/Submission/WSMO/ (accesses on 27 April 2018).
31. Frankel, D.S. *Model Driven Architecture: Applying MDA to Enterprise Computing*; Wiley Publishing: Hoboken, NJ, USA, 2003.
32. Jouault, F.; Allilaire, F.; Bezivin, J.; Kurtev, I. ATL: A model transformation tool. *Sci. Comput. Programm.* **2008**, *72*, 31–39.
33. Acceleo Code Generator. Available online: https://www.eclipse.org/acceleo/ (accessed on 7 April 2018).
34. Myers, K.L. *User Guide for the Procedural Reasoning System*; SRI International AI Center Technical Report; SRI International: Menlo Park, CA, USA, 1997.
35. d'Inverno, M.; Luck, M.; Georgeff, M.; Kinny, D.; Wooldridge, M. The dMARS architecture: A specification of the distributed multi-agent reasoning system. *Autonom. Agents Multi-Agent Syst.* **2004**, *9*, 5–53.
36. Anupriya, A.; Mark, B.; Jerry, R. H.; Ora, L.; David, M.; Drew, M.; Sheila, A.M. DAML-S: Web service description for the semantic web. In *International Semantic Web Conference*; Springer: Berlin/Heidelberg, Germany, 2002; pp. 248–363.
37. World Wide Web Consortium, OWL 2 Web Ontology Language—Document Overview (Second Edition), W3C Recommendation, 2012. Available online: https://www.w3.org/TR/owl2-overview/ (accessed on 7 April 2018).
38. Agent Oriented Software Inc. JACK Intelligent Agents. Available online: http://www.aosgrp.com/products/jack/ (accessed on 7 April 2018).
39. Broekstra, J.; Kampman, A.; Van Harmelen, F. Sesame: A generic architecture for storing and querying rdf and rdf schema. In Proceedings of the International Semantic Web Conference, Sardinia, Italy, 9–12 June 2002; Springer: Berlin/Heidelberg, Germany, 2002; pp. 54–68.
40. Object Management Group, Ontology Definition Metamodel (ODM) Version 1.1, 2014. Available online: https://www.omg.org/spec/ODM/ (accessed on 7 April 2018).
41. Bernon, C.; Gleizes, M.-P.; Peyruqueou, S.; Picard, G. ADELFE: A methodology for adaptive multi-agent systems engineering. *Lect. Notes Artif. Intell.* **2003**, *2577*, 70–81.
42. Zambonelli, F.; Jennings, N.R.; Wooldridge, M. Developing multiagent systems: The Gaia methodology. *ACM Trans. Softw. Eng. Methodol.* **2003**, *12*, 317–370.
43. Pavón, J.; Gómez-Sanz, J.J. Agent oriented software engineering with INGENIAS. In *CEEMAS*; Mark, V., Müller, J.P., Pechoucek, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2003; pp. 394–403.
44. Cossentino, M. From requirements to code with the PASSI methodology. In *Agent-Oriented Methodologies*; Henderson-Sellers, B., Giorgini, P., Eds.; Idea Group Publishing: Hershey, PA, USA, 2005; pp. 79–106.
45. Padgham, L.; Winikoff, M. Prometheus: A methodology for developing intelligent agents. In Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1, Bologna, Italy, 15–19 July 2002; pp. 37–38.
46. Bresciani, P.; Perini, A.; Giorgini, P.; Giunchiglia, F.; Mylopoulos, J. Tropos: An agent-oriented software development methodology. *Autonom. Agents Multi-Agent Syst.* **2004**, *8*, 203–236.
47. Bordini, R.H.; Hübner, J.F.; Wooldridge, M. *Programming Multi-Agent Systems in AgentSpeak Using Jason*; John Wiley & Sons: Hoboken, NJ, USA, 2007; Volume 8.

48. Ricci, A.; Viroli, M.; Omicini, A. CArtAgO: A framework for prototyping artifact-based environments in MAS. In Proceedings of the International Workshop on Environments for Multi-Agent Systems, Hakodate, Japan, 8 May 2006; Springer: Berlin/Heidelberg, Germany, 2006.
49. Hannoun, M.; Boissier, O.; Sichman, J.S.; Sayettat, C. MOISE: An organizational model for multi-agent systems. In *Advances in Artificial Intelligence*; Springer: Berlin/Heidelberg, Germany, 2000; pp. 156–165.
50. Boissier, O.; Bordini, R.H.; Hübner, J.F.; Ricci, A.; Santi, A. Multi-agent oriented programming with JaCaMo. *Sci. Comput. Programm.* **2013**, *78*, 747–761.
51. Rougemaille, S.; Migeon, F.; Maurel, C.; Gleizes, M.-P. Model Driven Engineering for Designing Adaptive Multi-Agent Systems. In Proceedings of the 8th Annual International Workshop on Engineering Societies in the Agents World (ESAW 2007), Athens, Greece, 22–24 October 2007.
52. Pavon, J.; Gomez, J.; Fuentes, R. Model Driven Development of Multi-Agent Systems. *Lect. Notes Comput. Sci.* **2006**, *4066*, 284–298.
53. Thangarajah, J.; Padgham, L.; Winikoff, M. Prometheus design tool. In Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems, Utrecht, The Netherlands, 25–29 July 2005; pp. 127–128.
54. Penserini, L.; Perini, A.; Susi, A.; Mylopoulos, J. From Stakeholder Intentions to Software Agent Implementations. *Lect. Notes Comput. Sci.* **2006**, *4001*, 465–479.
55. Kardas, G. Model-driven development of multi-agent systems: A survey and evaluation. *Knowl. Eng. Rev.* **2013**, *28*, 479–503.
56. Kulesza, U.; Garcia, A.; Lucena, C.; Alencar, P. A generative approach for multi-agent system development. *Lect. Notes Comput. Sci.* **2005**, *3390*, 52–69.
57. Hahn, C.; Madrigal-Mora, C.; Fischer, K. A Platform-Independent Metamodel for Multiagent Systems. *Autonom. Agents Multi-Agent Syst.* **2009**, *18*, 239–266.
58. Ayala, I.; Amor, M.; Fuentes, L. A model driven engineering process of platform neutral agents for ambient intelligence devices. *Autonom. Agents Multi-Agent Syst.* **2014**, *28*, 214–255.
59. 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. *Eng. Appl. Artif. Intell.* **2010**, *23*, 432–444.
60. Wautelet, Y.; Kolp, M. Business and model-driven development of BDI multi-agent system. *Neurocomputing* **2016**, *182*, 304–321.
61. Yu, E.; Giorgini, P.; Maiden, N.; Mylopoulos, J. *Social Modeling for Requirements Engineering: The Complete Book*; MIT Press: Cambridge, MA, USA, 2011.
62. López, N.; Nunez M.; Rodriguez, I.; Rubio, F. A Formal Framework for E-Barter Based on Microeconomic Theory and Process Algebras. In *International Workshop on Innovative Internet Community Systems*; Springer: Berlin/Heidelberg, Germany, 2002; pp. 217–228.
63. López, N.; Núñez, M.; Rodríguez, I.; Rubio, F. A Multi-Agent System for e-barter including Transaction and Shipping Costs. In Proceedings of the 2003 ACM Symposium on Applied Computing, Melbourne, FL, USA, 9–12 March 2003; pp. 587–594.
64. Núñez, M.; Rodríguez, I.; Rubio, F. Formal specification of multi-agent e-barter systems. *Sci. Comput. Programm.* **2005**, *57*, 187–216, doi:10.1016/j.scico.2005.01.002.
65. Cavalli, A.; Maag, S. Automated test scenarios generation for an e-barter system. In Proceedings of the 2004 ACM Symposium on Applied Computing—SAC '04, New York, NY, USA, 14–17 March 2004; p. 795.
66. International Telecommunication Union (ITU). *Telecommunication Standardization, Specification and Description Language (SDL)*; Technical Report Z-100; ITU: Geneva, Switzerland, 2011; p. 206.
67. Andrews, T.; Curbera, F. Web Service Business Process Execution Language, Working Draft, Version 2.0, 1; 2004. Available online: https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.emisa/Downloads/emisaforum06.pdf (accessed on 7 April 2018).
68. Bravetti, M.; Casalboni, A.; Nunez, M.; Rodriguez, I. From Theoretical e-Barter Models to Two Alternative Implementations Based on Web Sevices. *J. UCS* **2007**, *13*, 2035–2075, doi:10.3217/jucs-013-13-2035.
69. Ragone, A.; Di Noia, T.; Di Sciascio, E.; Donini, F.M. Increasing bid expressiveness for effective and balanced e-barter trading. *Lect. Notes Comput.* **2009**, *5397 LNAI*, 128–142, doi:10.1007/978-3-540-93920-7_9.
70. Tagiew, R. Barter Double Auction as Model for Bilateral Social Cooperations. In Proceedings of the 1st Computer Science and Electronic Engineering Conference (CEEC'09), Colchester, UK, 19–21 September 2009; pp. 1–7.

71. Dagdeviren, Z.A.; Kardas, G. A Case Study on the Development of Electronic Barter Systems using Software Agents. In Proceedings of the 5th Turkish National Software Engineering Symposium (UYMS 2011), Ankara, Turkey, 26–28 September 2011; pp. 123–126.
72. Arenas, A.E.; Barrera-Sanabria, G. Applying the MAS-CommonKADS Methodology to the Flights Reservation Problem: Integrating Coordination and Expertise. In Proceedings of the 5th Joint Conference on Knowledge-Based Software Engineering, Maribor, Slovenia, 11–13 September 2002.
73. Ye, D.; Zhang, M.; Vasilakos, A.V. A survey of self-organization mechanisms in multiagent systems. *IEEE Trans. Syst. Man Cybern. Syst.* **2017**, *47*, 441–461.
74. Getir, S.; Challenger, M.; Kardas, G. The Formal Semantics of a Domain-specific Modeling Language for Semantic Web enabled Multi-agent Systems. *Int. J. Coop. Inf. Syst.* **2014**, *23*, 1–53, doi:10.1142/S0218843014500051.
75. Challenger, M.; Mernik, M.; Kardas, G.; Kosar, T. Declarative specifications for the Development of Multi-agent Systems. *Comput. Stand. Interfaces* **2016**, *43*, 91–115, doi:10.1016/j.csi.2015.08.012.
76. Kardas, G.; Tezel, B.T.; Challenger, M. A domain-specific modeling language for belief-desire-intention software agents. *IET Softw.* **2018**, doi:10.1049/iet-sen.2017.0094.