PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 31/05

An MDA-Based Approach for Developing Multi-Agent Systems

Beatriz Alvez de Maria
Viviane Torres da Silva
Carlos José Pereira de Lucena

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900
RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 31/05 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena Semptember, 2005

An MDA-Based Approach for Developing Multi-Agent Systems*

Beatriz Alvez de Maria Viviane Torres da Silva
Carlos José Pereira de Lucena

{beatriz, viviane, lucena}@inf.puc-rio.br

Abstract: In this paper, we propose an MDA-based approach for developing multi-agent systems (MAS). MDA specifies a structured software development process divided in modeling stages. In the PIM stage, where platform independent models are specified, we propose to use MAS-ML, a MAS modeling language that does not restrict or specify implementation platforms. In the PSM stage, where platform specific models are defined, we propose to transform MAS-ML models into UML models based on an object-oriented framework for implementing MAS. In the last stage, the application code is generated from UML models. The MDA stages and related transformations are detailed in the paper.

Keywords: MDA, Multi-Agent Systems, MAS-ML, UML, ASF.

Resumo: Neste artigo, nós propomos uma abordagem baseada em MDA para o desenvolvimento de sistemas multi-agentes (SMA). MDA especifica um processo de desenvolvimento de software estruturado dividido em etapas de modelagem. Na etapa PIM, onde modelos independentes de plataforma são especificados, nós propomos o uso de MAS-ML, um linguagem de modelagem para SMA que não restringe ou especifica as plataformas de implementação. Na etapa PSM, onde modelos específicos de plataforma são definidos, nós propomos transformar os modelos MAS-ML em modelos UML baseando-se em um framework orientado a objetos para implementação de SMA. Na última etapa, o código da aplicação é gerado a partir de modelos UML. As etapas de MDA e as transformações associadas são detalhadas no artigo.

Palavras-chave: MDA, Sistemas Multi-Agentes, MAS-ML, UML, ASF.

---

* This work has been sponsored by the Ministério de Ciência e Tecnologia da Presidência da República Federativa do Brasil

In charge for publications:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3114-1516 Fax: +55 21 3114-1530
E-mail: bib-di@inf.puc-rio.br

1 Introduction

The development of multi-agent systems (MAS) has rapidly increased in the last few years. Different applications' domains, like digital libraries, virtual markets and information systems in general, such as [Braubach et al., 2004; He et al., 2003; Kia et al., 2002; Wooldridge and Jennings, 1998], are being developed through the MAS approach. Modeling languages, programming languages, methodologies and some others MAS modeling and implementing techniques have been proposed with the purpose of helping the developers in building such systems.

In this paper, we propose a top-down MDA (Model Driven Architecture) [OMG, 2002] based approach for developing MAS. MDA is an architecture that provides a means for using platform independent viewpoints (high-level models) and platform specific viewpoints (low-level models) in systems development. MDA is composed of four main stages and a set of transformations that accomplish the tracking between the artifacts produced during each stage.

The MDA development process specifies three basic models: computational independent models (CIMs), platform independent models (PIMs) and platform specific models (PSMs). CIMs are application conceptual models that are independent of the computational characteristics of the solution. PIMs generated in the second stage of the development process are high-level models that are independent of implementation platforms. PSMs, which correspond to the third stage, are models created based on PIMs by including specific platform details. In the fourth (and final) stage of the MDA development process, PSMs are transformed into application code. Besides the specification of the models, MDA also defines a set of consecutive transformations that should be applied to the models in order to allow the transformation of high-level abstraction models into code.

Our proposal focuses on applying the top-down MDA architecture to build MAS by defining the MDA stages and the transitions between the stages according to the MAS characteristics. In this paper, we focus on the definition of MAS PIMs, on the transformations from these models into PSMs, on the specification of the PSMs and on the transformations from PSMs into code. The representation of CIMs as well as the generation of CIMs into PIMs is beyond the scope of this paper.

While defining the PIMs, we propose to use the MAS-ML modeling language [Silva et al, 2003; Silva et al., 2004c] to model MAS. MAS-ML is a modeling language that extends UML, including agent-related abstractions. Since MAS-ML does not restrict or specify implementation platforms, MAS-ML models are platform independent ones. Using MAS-ML, it is possible to model the MAS structural aspects (the abstractions, their properties and relationships) and the MAS dynamic aspects (interactions between the abstractions and their internal executions).

In order to define PSMs, we propose to transform the MAS-ML models into UML [OMG, 2005] models by using an object-oriented framework called ASF (Agent Society Framework] [Silva et al., 2004b]. The ASF framework defines a set of object-oriented modules that specifies abstract classes that should be extended in order to implement multi-agent applications. The MAS-ML models that describe a multi-agent application are transformed into UML models by instantiating the ASF framework. In this paper, we concentrate on the transformation of MAS-ML static models – created by using the three MAS-ML structural diagrams (organization diagram, role diagram and extended UML class diagram) – into UML structural models – created by using the UML class diagrams. The UML models generated according to ASF include several characteristics that depend on the framework used in the transformation. Therefore, the UML models are platform specific models.

According to the MDA top-down approach, the next stage corresponds to the transformation of PSMs into application code. In this context, the UML models that represent the MAS are automatically transformed into object-oriented code.
The use of MDA in the MAS development process offers several advantages. The MAS-ML models that describe an application are platform independent models that are portable to diverse systems and can be used to generate different computational models by applying various implementation platforms. Our approach proposes to use the ASF framework in order to generate platform specific models. However, platforms such as Jadex [Pokahr et al., 2003] could also be used.

Another advantage of the use of MDA is the generation of different MAS viewpoints through the specification of models that have different abstraction levels. The MAS-ML models are high-level models that focus on problem definition by describing the system in an agent-oriented level. The UML models are low-level models that focus on the solution of the problem by including implementation details and describing the system in an object-oriented level.

The paper is organized as follows. In Section 2 there is a brief overview of the main characteristics of the MAS-ML modeling language and the ASF Framework. Section 3 describes our main contribution by showing the MAS development process using MDA. In Section 4 , we present the VisualAgent tool [DeMaria et al. 2005] that supports the development process proposed in this paper. The tool implements all stages presented in the process by allowing the modeling and implementation of MAS. Section 5 describes some related work and Section 6 presents some discussion about the advantages and disadvantages of using our proposed approach. Finally, Section 7 draws some conclusions and discusses future work.

2 Background

2.1 MAS-ML

MAS-ML is a modeling language focused on developing MAS. This language is an extension of UML by including agent-related concepts presented in the TAO (Taming Agents and Objects) conceptual framework [Silva et al., 2003]. TAO defines the core abstractions found in MAS, its properties, relationships and the way these entities execute and interact. Thus, by using MAS-ML it is possible to model not only objects, but also agents, environments, organizations and roles that are entities defined in TAO and usually found in MAS. Figure 1 illustrates part of the MAS-ML meta-model by depicting the meta-classes and stereotypes created to represent MAS entities (environment, agent, organization, object, agent role and object role) and their properties (goal, belief, axiom, plan, action, protocol, duty and right). More details about the MAS-ML meta-model can be found in [Silva and Lucena, 2004a].

The MAS-ML relationships (specialization, aggregation, association, dependency, inhabit, play and ownership), i.e., the relationships that can be used to link MAS entities, are represented in Figure 2. This figure is also part of the MAS-ML meta-model since it illustrates the meta-classes created to represent the relationships defined in MAS-ML that are not defined in UML.

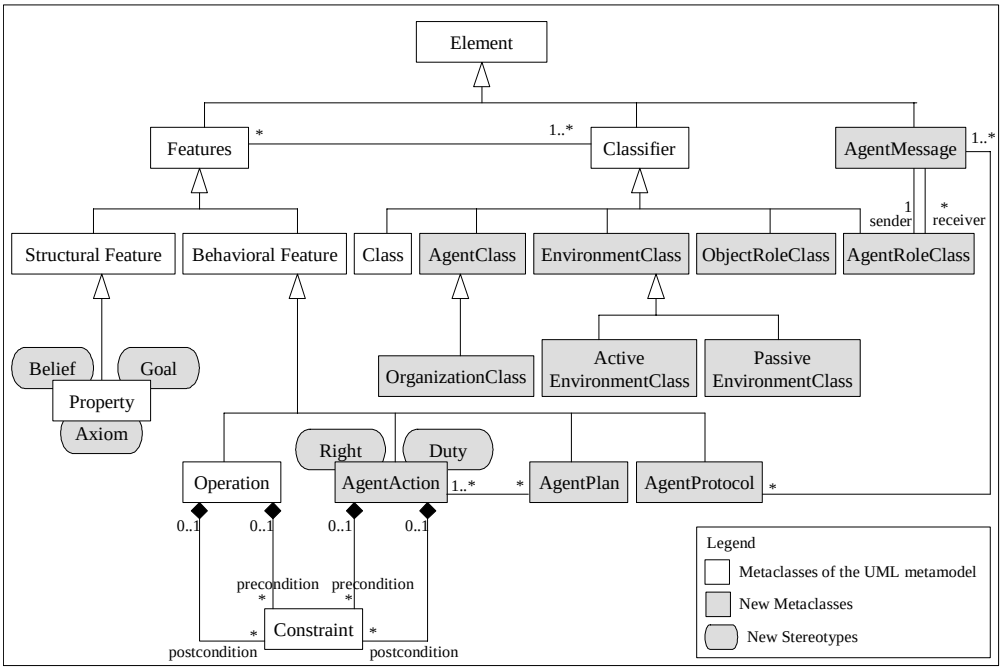

Figure 1. MAS-ML meta-model representing the MAS-ML entities and properties

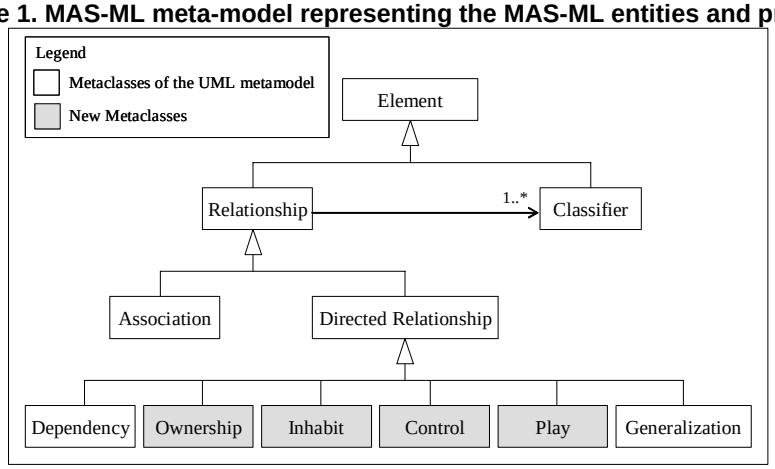

Figure 2. MAS-ML meta-model representing the MAS-ML relationships

The term organization is used to represent partitions and groups of entities such as departments, communities and societies. Similar to agents, organizations are autonomous, interactive and adaptive entities that have goals, beliefs, plans and actions. Organizations also define rules (norms or laws) that are expressed in terms of general organization constraints (or axioms). Agents, organizations and objects are immersed in environments where they execute and interact with each other. The inhabit relationship is used to represent the relationship between entities and the environments that they inhabit.

Organizations can define sub-organizations as soon as the complexity of the system increases. Both agents and sub-organizations play (agent) roles defined in organizations. The agent roles have associated goals, duties, rights and protocols that influence the execution of the entity that is playing the role.

An object is an interactive entity but it is not an autonomous one. The autonomy property is one of the most important differences between an agent and an object. Objects are reactive and passive entities since they need the assistance of another entity to do their job and since they respond to any request for assistance.

Similar to agents, objects can also play roles in organizations. An object role may restrict access to the attributes and methods of an object, but may also add other attributes and even methods to the object that plays the role. The ownership relationship is used to link organizations and the (agent or object) roles that they define. The play relationship is used to link the entities (agents, sub-organization and objects) with the roles (agent or object role) that they play. Other relationships such as dependency, association, aggregation and control are used to define the interaction between the entities.

In order to model the structural and dynamic aspects of the entities, MAS-ML defines three structural diagrams and two dynamic diagrams. The structural diagrams allow the modeling of all entity classes, their properties and their relationships. The structural diagrams proposed by MAS-ML are the extended UML class diagram, the organization diagram and the role diagram. Both organization and role diagrams are not defined in the UML metamodel [OMG 2005] but were specifically defined in MAS-ML. The extended UML class diagram represents agents, organizations, environments and the relationship between these entities and classes. This diagram was extended to represent such MAS entities and their relationships that could not be modeled in the original UML class diagram. The MAS-ML organization diagrams model the system organizations, their properties, the roles that they define and the agents that play these roles. The MAS-ML role diagrams focus on modeling the relationship between the roles identified in the organization diagrams and between the roles and the resources (or objects) available in the organizations.

The dynamic diagrams defined by MAS-ML are the extended UML sequence and activity diagrams. Both diagrams were extended to model the MAS entities' behavior, focusing on their internal executions and interactions. Both interaction and internal executions of MAS entities could not be modeled by using the original UML dynamic diagrams.

In order to exemplify our approach, we will use the MAS-ML organization diagram. The objective of the organization diagrams is to model all the organizations of a system. Each organization diagram is responsible for modeling one organization, i.e., for modeling the properties of the organization (goals, beliefs, plans, actions and axioms), the roles defined by the organization, the entities (agents,classes and sub-organizations) that play these roles, and the environment that the organization inhabit. In organization diagrams it is also important to describe the properties of the roles (goals, beliefs, duties, rights and protocols) defined in the organization and the entities that play each role. More details about other MAS-ML diagrams can be found in [Silva, 2004a].

2.2 ASF Framework

The ASF Framework allows the implementation of MAS by using the object-oriented (OO) technology. By using this framework, it is possible to implement the agents, roles, organizations and environments modeled in MAS-ML. All entities defined in the MAS-ML meta-model together with their properties (Figure 1) were mapped into OO classes defined in the ASF frameworks. Moreover, all MAS-ML relationships (Figure 2) that link MAS-ML entities were mapped into UML relationships to link the OO classes defined in ASF. Table 1 represents the map between the MAS-ML abstractions and the ASF classes and relationships.

Table 1 - Map between the abstractions of the MAS-ML meta-model and the ASF classes and relationships

MAS-ML	ASF
Entities' properties	
Belief	Class called *Belief*
Goal	Class called *Goal*
Action	Abstract class called *Action*
Plan	Abstract class called *Plan*
Axiom	Class called *Axiom*
Duty	Class called *Duty*
Right	Class called *Right*
Protocol	Abstract class called *Protocol*
2.2.1 Entities	
Object	Class
Agent	Abstract class called *Agent* linked to *Belief*, *Goal*, *Action* and *Plan* classes
Organization	Abstract class called *Organization* linked to *Belief*, *Goal*, *Action*, *Plan* and *Axiom* classes
Environment	Abstract class called *Environment*
Agent role	Abstract class called *AgentRole* class linked to *Belief*, *Goal*, *Duty*, *Right* and *Protocol* classes
Object role	Abstract class called *ObjectRole*
Relationships	
Association	Association between the classes that represent the MAS-ML entities
Aggregation	Aggregation between the classes that represent the MAS-ML entities
Specialization	Specialization between the classes that represent the MAS-ML entities
Dependency	Dependency between the classes that represent the MAS-ML entities
Inhabit	Association between the *Agent* class and the *Environment* class, between the *Organization* class and the *Environment* class and between the classes created to represent objects and the *Environment* class
Play	Association between the *Agent* class and the *AgentRole* class, between *Organization* class and *AgentRole* class and between *Object* class and *ObjectRole* class
Ownership	Association between *Organization* class and *AgentRole* class and between *Organization* class and *ObjectRole* class
Control	Association between two *AgentRole* classes

As the entities modeled in MAS-ML are not OO abstractions, the framework maps these entities together with their properties into a set of OO modules. Each module that is part of the framework represents a MAS entity. A set of classes and relationships defined in each module make possible the implementation of the structural aspects of an entity (its properties and relationships with others entities) and their dynamic aspect (their behavior). Figure 3 shows all classes of the framework mentioned in Table 1. The modules that represents the agents (delimited by a continuous rectangle), organizations (defined by the dotted enlace) and agent roles (marked by the hatched rectangle) are complex modules since they group several classes to represent all the properties of the entities. The modules that represent the environments (hatched circle) and object roles

(continuous circle) are simple represented by one class because the properties of these entities can be directly represented as attributes and methods.

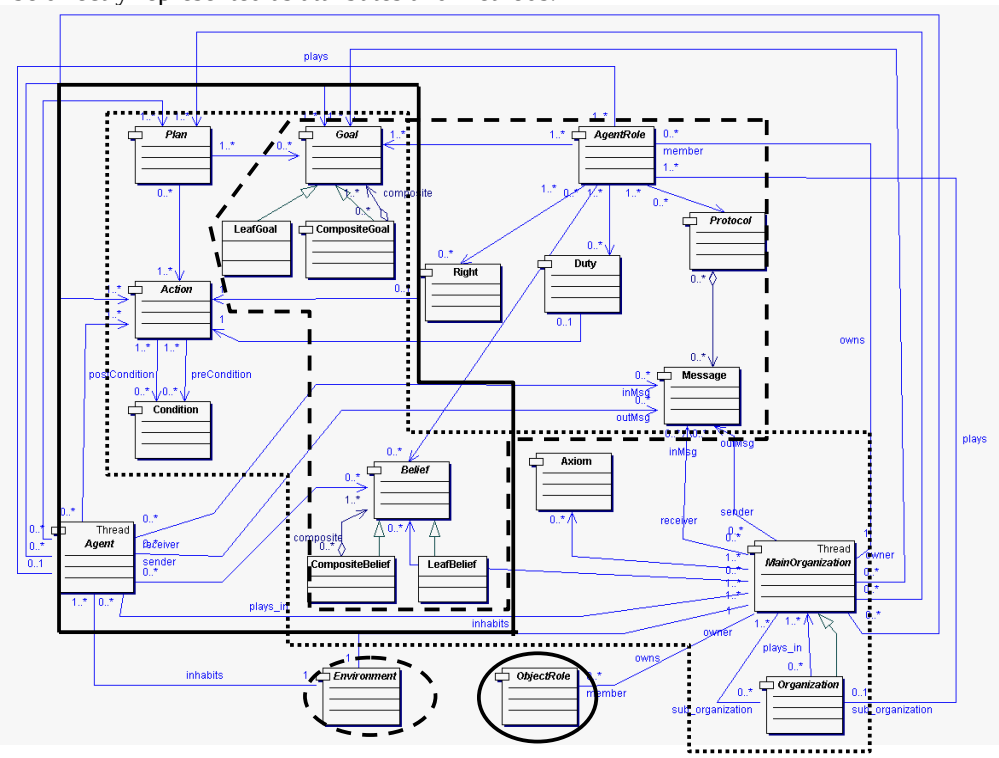

Figure 3. The ASF classes and relationships

3 Multi-Agent Systems and MDA

The development process of complex and large-scale systems, such as MAS, involves the construction of different models based on a variety of requirements. The transformation of a system specification into models and of these models into code is usually accomplished in a non-organized way that is not easily adaptable to technology changes.

Since MAS are gaining widespread acceptance, it is necessary to create a MAS development process that clearly specifies the different process stages and the transitions between these stages. The proposed process aims to specify the implementation of agent-oriented systems by applying the top-down MDA approach. It is important to clarify that MDA is not limited to the top-down approach used in this paper.

The MAS development process proposed in this paper is illustrated in Figure 4. In the CIM stage, the system can be described by using domain-specific ontologies or any other computational independent technology. For instance, ontologies define the concepts and relationships related to the problem domain that can be used to specify a system. The definition of such concepts and relationships do not depend on the computational abstractions used to solve the system problem. It is not within the scope of this paper either to exemplify the CIM models of MAS or to demonstrate the transformation from CIMs into PIMs.

Next, the CIM models should be transformed into PIMs by using any platform independent modeling language, such as AUML [Odell et al, 2000] and MAS-ML. In our approach, we propose to use the MAS-ML modeling language. The main difference between MAS-ML and other MAS modeling languages is that MAS-ML

defines organizations, roles, agents and environment as first-class entities. By defining these abstractions as first-class entities it is possible to define their properties, relationships, internal executions and interactions.

After the creation of PIMs, PSMs should be generated by choosing an implementation platform. In order to implement MAS, several architectures, frameworks and platforms, such as ASF, Jadex [Pokahr et al, 2003] and RETISINA [Sycara et al., 2003], can be used. In our approach, the MAS-ML models are transformed into UML models by using the ASF framework. The ASF framework was chosen since it is possible to implement organizations, roles and environments by using the framework. Other implementation platforms do not deal with the implementation of such abstractions. For instance, to apply Jadex to implement systems modeled by using MAS-ML is not a simple task since Jadex and MAS-ML do not deal with the same set of abstractions. New transformation rules needed to be created to convert MAS-ML abstractions into Jadex abstractions.

Finally, PSMs are transformed into code. The programming language will depend on the platform used to implement the system. In the proposed approach, MAS will be implemented in Java since this is the language used by ASF.

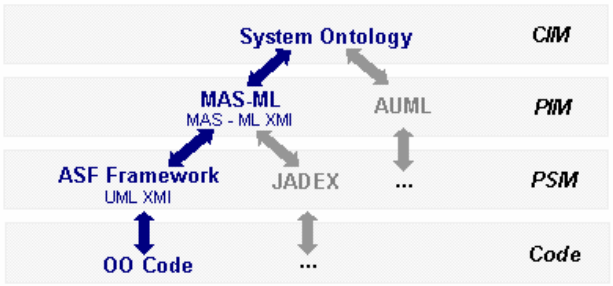

Figure 4. MAS development process based on MDA

In order to exemplify the MAS development process, an e-commerce application called *Virtual Marketplace* will be used. This application is a MAS benchmark application [He et al., 2003; Lomuscio et al., 2003; Wooldridge and Jennings, 1998]. The *Virtual marketplace* application defines markets located in the Web where users buy items sold in the market. The users locate items they desire to buy and negotiate with the sellers that offer the items. The sellers represent the markets during the negotiations.

3.1 PIM

MAS-ML was chosen to model MAS PIMs due to three main factors. First, this language is not a platform dependent modeling language. Although MAS-ML uses agent and object-oriented abstractions, MAS-ML does not restrict the implementation of its models to a specific implementation platform. Second, MAS-ML is a modeling language that is MOF [OMG, 2003a] compliant [Silva, 2004a]. This characteristic facilitates the MAS development process since XMI [OMG, 2003b] can be used. XMI provides a mapping from MOF to XML and, therefore, can be used to describe MOF compliant models. XMI is used in our approach to represent the MAS-ML models, to assist in the transformation from MAS-ML models into UML models, to represent the UML models and to help the transformation from UML models into code. Finally, as stated before, MAS-ML models abstractions such as organizations, environments and roles that are not adequately modeled in other MAS modeling languages [DeLoach, 2001; Odell et al, 2000; Wagner, 2000]. Without modeling these abstractions it is not possible to model, for instance, agents playing different roles in different organizations.

As stated before, in this paper we focus on modeling and implementing the structural characteristics of MAS. Therefore, instead of demonstrating the modeling and transformation of all MAS-ML diagrams into code we concentrate on presenting the MAS-ML organization diagram, that is one of the MAS-ML structural diagrams. This diagram is used to illustrate a MAS platform independent model, to show the transformation of such PIM models into PSM (Section 3.2 and 3.3), and then into object-oriented code (Section 3.4).

The organization diagram illustrated in Figure 5 models the *Virtual Marketplace* previously described. Together with the market (or organization), the diagram also models the roles defined by the market, the agents and objects that can play the roles and the environment the market inhabits. The market *General Store* defines the agent roles *Buyer* and *Seller* that are played by the agents *User Agent* (that represents the users in the system) and *Store Agent* (that represents the sellers of the market), respectively. Furthermore, the market also defines the *item* negotiated by the agents and the object roles (*Desire* and *Offer*) played by the item. The model illustrated in Figure 5 is uses a simplified representation that suppresses the middle and the bottom compartments of the diagram elements where the structural and dynamic aspects of the entities are detailed.

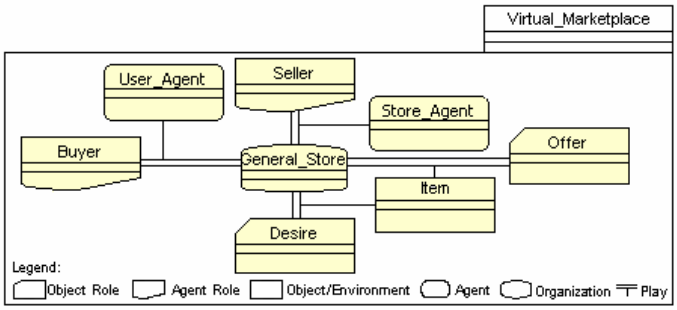

Figure 5. The organization diagram of the e-commerce application

In order to exemplify how the properties of an entity are defined, some properties of the organization class *General Store* are detailed in Figure 6. As stated in MAS-ML, the properties of an organization are goals, beliefs, axioms, plans and actions. The goal represented in Figure 6 is the *management of sellers* goal. To manage the sellers, this organization needs to know the sellers that are negotiating. The *list of sellers* is one of the organization beliefs. To guarantee that all sellers will inform the organization about the sales, the organization defines the axiom *send information about sale*. To achieve the goal *management of sellers*, the store defines a plan to create the sellers to negotiate with buyers (*creating sellers*). The plan *creating sellers*, the actions that compose the plan and the goal related to the plan are modeled in Figure 6. Both Figure 5 and Figure 6 are used in this Section to represent one of the PIM models that specify the structural characteristics of the e-commerce application.

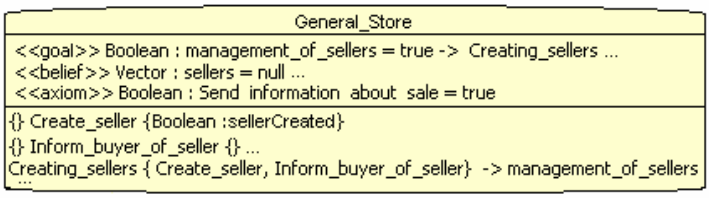

Figure 6. The organization General Store

3.2 Transforming PIMs into PSMs

After using MAS-ML to model PIMs, the models should be transformed into PSMs. The transformation of MAS-ML models into UML models occurs in two phases. The first phase consists of describing the MAS-ML models in a textual description by using the XMI format. In order to do so, it was necessary to create a MAS-ML DTD that specifies the MAS-ML models in XMI. The MAS-ML DTD extends the UML DTD according to the extensions proposed by MAS-ML to the UML meta-model. The XMI file created by using the MAS-ML DTD represents the MAS-ML models. Such XMI is called MAS-ML XMI.

The second stage of the transformation consists of refining the MAS-ML XMI into UML XMI, according to the ASF framework. The MAS-ML XMI generated in the previous phase is converted into a UML XMI through the instantiation of the ASF Framework. The transformation uses a UML XMI file that contains the framework specifications. This file describes all classes and relationships among classes that are defined in the framework (Figure 4).

During the transformation, the UML XMI file that represents the framework is extended with the application details that are described in MAS-ML XMI. The UML XMI extended file represents the application modeled in MAS-ML and implemented through the instantiation of the ASF framework. The rules used to extend the UML XMI and to instantiate the ASF framework are represented in Table 2. Such rules describe how the entities represented in the MAS-ML XMI are implemented using ASF. For instance, the transformation of the *General Store* organization class modeled in Figure 5 and Figure 6 by using MAS-ML consists of three steps: (1) the creation of an object-oriented class to represent the *General Store* organization, (2) the creation of classes to implement the actions and plans of the organization by extending the abstract classes *Action* and *Plan*, respectively, and (3) the implementation of the constructor method of the *General Store* organization class created in (1).

Table 2. Rules used to map MAS-ML XMI into UML XMI.

Entity	Transformation Rules
Agent	1. Create a concrete class that will represent the agent and will extend the abstract class *Agent*. 2. Create concrete classes that extend *Plan* and *Action* classes to implement the behavior defined by plans and actions. 3. Implement the constructor method of the concrete class created in 1 to instantiate goals, beliefs, plans and actions according to the agent properties. The creation of beliefs and goals is realized by the instantiation of *Belief* and *Goal* classes, respectively.
Organization	1. Create a concrete class that will represent the organization. This class will extend the abstract class *Organization* or *MainOrganization*, according to its definition. 2. Create concrete classes that extend *Plan* and *Action* classes to implement the behavior defined by plans and actions. 3. Implement the constructor method of the concrete class created in 1 to instantiate goals, beliefs, axioms, plans and actions according to the agent properties. The creation of beliefs, goals and axioms is realized by the instantiation of *Belief*, *Goal* and *Axiom* classes, respectively.
Agent Role	1. Create a concrete class that will represent the agent role. This class will extend the abstract class *AgentRole*. 2. Create concrete classes to represent protocols and that extend the abstract class *Protocol.*

	3. Implement the constructor method of the concrete class created in 1 to instantiate rights and duties according to the agent role properties. The creation of rights and duties is realized by the instantiation of *Right* and *Duty* classes, respectively.
Object Role	1. Create a concrete class that will represent the object role. This class will extend the abstract class *ObjectRole*.
Environment	1. Create a concrete class that will represent the object role. This class will extend the abstract class *Environment*.

3.3 PSM

The UML XMI generated in the previous stage represents a PSM of the application. By using graphical tools, such as Rational Rose1 and Poseidon2, which import XMI files, it is possible to generate UML models of the application.

The UML model created in this stage is a UML class diagram that contains the ASF framework classes and the classes related to the application that instantiate the framework. Since this paper does not concern the MAS-ML dynamic diagrams during the transformation, UML dynamic diagrams are not part of the PSM models. All application entities, properties and relationships modeled on the three MAS-ML structural diagrams during the PIM phase are represented in a UML class diagram of the PSM phase.

In order to illustrate the mapping between MAS-ML models and UML models, i.e., in order to demonstrate the PSM model that represent the MAS-ML models shown in Figure 5 and Figure 6, we will describe how the organization *General Store* was implemented using ASF. As stated before in Table 2 (Section 3.2), such implementation consists of three steps. In the first step the *General Store* class was created extending the *Main Organization* class represented in the framework since there is not any other organization in the system. In the second step, concrete classes that extend the abstract classes *Plan* and *Action* defined in the framework are created in order to represent the plans and actions of the organization. Each concrete class implement the method *execute()* that represent the execution of the plan or action. In the third step, the constructor method of the *General Store* class is implemented according to the organization properties defined in Figure 6. As mentioned before, beliefs, goals and axioms are created by instantiating the *Belief*, *Goal* and *Axiom* classes, respectively. *Plans* and *actions* are created by instantiated the concrete classes that were created in the second step. Figure 7 illustrates the classes of the framework associated with the organization module (shown in Figure 3 by a dotted enlace) and the classes created while instantiating the organization *General Store*. The classes created during the instantiation are highlighted by circles.

1 Rational Rose, http://www-306.ibm.com/software/rational/

2 Poseidon, http://www.gentleware.com

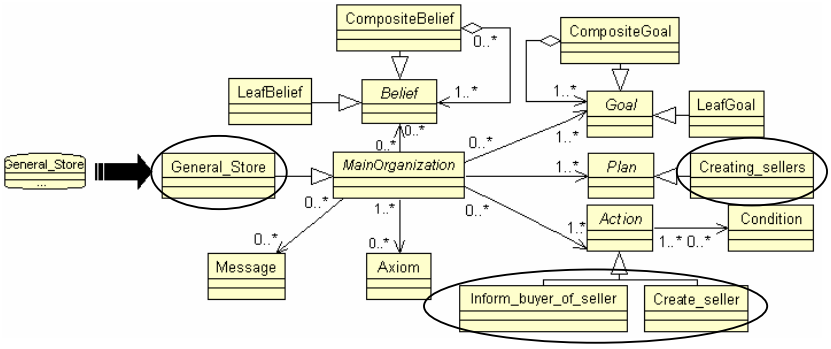

Figure 7. Transformation of MAS-ML into UML by using ASF

3.4 Transforming PSMs into code

In the final stage of the MAS development process, the UML models are transformed into code. This transformation corresponds to the last stage of the MDA approach that transforms PSMs into code. Almost all graphic tools that import UML XMI, such as Rational Rose and Poseidon, are capable of automatically generating code from a UML XMI. Therefore, we omit in this paper the rules that transform UML XMI into Java code. In order to exemplify the code generation process, Figure 8 illustrates the source code of the *General Store* class. Such code illustrates part of the constructor method of the class that instantiate the organization's beliefs, goals, actions, plans and axioms.

```
public class General_Store extends MainOrganization{
    public General_Store (Environment theEnvironment){ ...
        Belief objectBelief = new LeafBelief ("Vector", "sellers", null);
        this.beliefs.add (objectBelief);
        Goal objectGoal = new LeafGoal("boolean","management_of_sellers","true");
        this.goals.add (objectGoal);
        Action objectAction = new Create_seller ();
        this.actions.add (objectAction);
     Condition objCondition = new Condition("boolean","sellerCreated","");
         objectAction.setPostCondition (objectCondition);
        Plan objectPlan = new Creating_sellers ();
        objectPlan.setGoal (objectGoal);
        objectPlan.setAction (objectAction);
        objectGoal.setPlan (objectPlan);
        objectAction = new Inform_buyer_of_seller ();
        this.actions.add (objectAction);
        objectPlan.setAction (objectAction);
        Axiom objectAxiom = new Axiom("boolean","Send_information_about_sale","true");
        this.axioms.add(objectAxiom);...
}...}
```

Figure 8. Java code of the General Store class

4 Tool to Support the Proposed Development Process

In order to support the development process proposed in Section 3 , the VisualAgent tool was created. The tool provides support for the implementation of all the three stages presented in the proposed process. This tool, similar to other UML CASE tools, permits the user to graphically model the system by using the MAS-ML modeling language and to generate Java code based on the ASF framework.

The user can model their MAS application by creating new MAS-ML models. The current version of the tool only supports the modeling of the three structural MAS-ML diagrams. Once the MAS has been modeled, i.e., once PIM models have been created, the designer can use the tool to automatically transform such models into PSMs. The tool generates a MAS-ML XMI file in order to save the MAS-ML models and creates the UML XMI file that represents the application.

To create the UML XMI the user has to choose (i) the UML XMI version (1.4 or 1.5), (ii) the XMI version (1.1 or 1.2) and (iii) to include or not the implementation of the methods. These options make it possible to export artifacts that could be used by different CASE tools. CASE tools use different versions of UML and XMI, for instance, Together[3] and Poseidon use different versions of XMI. In addition, some of them, such as Together, do not recognize the method implementation described by the UML XMI.
The VisualAgent tool also offers Java code generation. This generation is based on the UML XMI generated by the tool. This functionality was developed by using the AndroMDA tool[4]. The artifacts that can be generated by the tool are: the MAS-ML XMI, the UML XMI or the Java code. The tool also imports MAS-ML XMI files.

5 Related Work

Although, some MAS methodologies such as Prometheus [Padgham and Winikoff 2002], Tropos [Bresciani et al., 2004] and MaSE [DeLoach 1999] have not used an MDA approach, they have already proposed the mapping between the design models into implementation code and have also provided some tools for supporting both the design and the implementation of MAS. However, they do not clearly demonstrate the mapping from design models into code by presenting the rules used in the transformation. Therefore, it is extremely difficult to use the design models created by using the methodologies to generate code to another platform or framework that has not been addressed by them.

In addition, they do not separate the models into platform independent models and platform specific models. By using some of these methodologies, it is possible to describe platform specific details during the design of the application. In such cases, the high-level design models are platform dependent and, consequently, are not easily portable to any other platform.

Other authors have already used the MDA approach in other to define a MAS development process. Vallecillo et al [2004] demonstrate the use of MDA to derive MAS low-level models from MAS high-level models. The authors propose to use the Tropos methodology and the Malaca models in the MDA approach. The high-level models created while using the Tropos methodology are transformed into low-level Malaca models. However, the transformation from the Tropos models into Malaca models is not completely automated. It requires manual intervention. Moreover, such an approach does not deal with the transformation from Malaca models into code.

Novikay [2004] analyzes how GR [Corradini et al., 1997] based on the Tropos visual model can be related to MDA. The author interprets the MDA approach as a visual modeling activity where more abstract models are refined in more detailed models, using transformation techniques. This work covers only the requirement stage existent in Tropos. The difference between our approach and this approach is that ours contemplates the PIM, PSM and code stages.

In Kazakov et al. [2002], the authors recommended a methodology based on a model-driven approach for the development of distributed mobile agent systems. They define a mobile agent conceptual model for distributed environments and describe a set of components, represented by a collection of intelligent mobile agents. While such an approach focuses on a specific application domain, our approach is a domain-independent development process.

---

3 Together, http://www.borland.com/together

4 AndroMDA, http://www.andromda.org

6 Discussions

Besides the simple example of an e-commerce application presented in Section 3 , we have developed other two complex and real applications by using the VisualAgent tool and, consequently, the proposed development process. A supply chain management system [Huget, 2002b; Fox et al., 2000] as well as a web-based paper submission and reviewing system [DeLoach, 2002; Zambonelli et al., 2001] were developed.

In order to compare the use of our approach with a haddock approach, two different groups have developed the same web-based paper submission and reviewing system using the different approaches. By using the haddock approach, the developers do not have any pre-defined process or guideline that helps them transforming the high-level agent-oriented models into object-oriented code. The developers could use any modeling language or methodology to provide the design models and any platform or framework for implementing the systems.

In this context, the developers that used the haddock approach have an important advantage to the others. They could use the techniques they were familiar with and do not need to learn the modeling language (MAS-ML) and the framework (ASF) provided by the tool and by our proposed approach. On the other hand, they needed to define the mapping between the high-lever models produced by the modeling language or methodology being used into code in order to implement the system. They needed to map their PIM models into PSM, and the PSM models into code without using any provided technique. Since different agent-oriented modeling languages, methodologies and platforms (or frameworks) use different agent abstractions, it is not easy to define those maps. Consequently, although the group that used our approach needed to learn MAS-ML and ASF, the system could be quickly and easily developed since the mapping between PIM models into code was already defined b y our MDA approach.

7 Conclusion and Future Work

The MAS development process presented in this paper intends to provide an approach for modeling and implementing MAS by using MDA. The proposed process is composed of three stages. In the first stage, our approach proposes the use of MAS-ML to model MAS by creating platform independent models. MAS-ML was chosen since it models several MAS models that are not (appropriately) modeled in other MAS modeling languages. In the second stage, the MAS-ML models are transformed into UML models by using the ASF framework. The ASF framework was chosen since it defines a set of object-oriented models that implement the MAS entities modeled in MAS-ML. Finally, the UML models are automatically transformed into code in the third stage proposed by the MAS development process.

The proposed MAS development process takes advantage of the MDA approach used to define the process. Three main characteristics can be enumerated: (i) portability and reusability: Since nowadays there is still no commonly used platform for implementing MAS, it is fundamental to separate implementation details from design models. In our approach be use MAS-ML to define the design models since MAS-ML models are portable to different implementing platforms because such models describe MAS without including implementation details. Thus, MAS-ML models can be reused by several developers to implement the system by using different platforms; (ii) interoperability: The use of XMI to describe models created during the process provides interoperable PIM and PSM models. MAS-ML models as well as UML models are interoperable due to the use of MAS-ML XMI and UML XMI;

(iii) low coupling: Due to the intrinsic characteristics of MDA, PIMs (or conceptual models) and PSMs (computation models) are low coupling. In our approach, MAS-ML models are PIMs that do not include details related to a specific platform. Such details are present just in UML models, PSMs.

The model-to-model transformation approach presented in Section 3.3 is classified as a graph-transformation-based approach [Czarnecki and Helsen, 2003]. The graph-transformation-based approach uses graph transformation rules (or graph transformation patterns) rendered in the concrete syntax of the respective source or target languages. The transformation presented in Section 3.5 can be classified as template-based approach [Czarnecki and Helsen, 2003]. Tools that generate source code from UML XMI documents use a template-based transformation approach.

To make the construction of MAS feasible by using our approach, a modeling tool was developed to support the design and implementation of MAS. The tool allows the designer to graphically model MAS systems by using MAS-ML and to implement them while generating Java code by using the ASF framework.

With the aim of enhancing the tool that gives support to the development of MAS by using MDA, several important improvements should be made. First, the transformer that generates code from MAS-ML models should also consider the MAS-ML dynamic diagrams. Second, the tool should make the visualization and also the modification of the UML models that represent the system implementation feasible. In addition, the tool should provide a model checker to analyze and verify the consistency of the different models (MAS-ML models and UML models).

Acknowledgements

This work is partially supported by CNPq/Brazil under the project "ESSMA", number 5520681/2002-0.

References

BAUER, B. MÜLLER, J.P. and ODELL, J. Agent UML: A Formalism for Specifying Multiagent Software Systems In: Ciancarini and Wooldridge (Eds) Agent-Oriented Software Engineering, Springer-Verlag, LNCS vol 1957, 2001.

BRAUBACH, L., POKAHAR, A. and LAMERDORF, W. Jadex: A Short Overview. In: Main Conference Net.ObjectDays, AgentExpo, pp. 195-207, 2004.

BRESCIANI, P. Tropos: An Agent-Oriented Software Development Methodology. Int. Journal of Autonomous Agents and Multi-Agents Systems, 8(3):203-236, 2004

CORRADINI, A., MONTANARI, U., ROSSI, F., EHRIG, H., HECKEL, R. and LOWE, M. Algebraic Approaches to Graph Transformation 1: Basic Concepts and Double Pushout Approach. Handbook of Graph Grammars and Computing by Graph Transformation, 1997.

CZARNECKI, K. and HELSEN, S. Classification of Model Transformation Approaches. In Proceedings of 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture, 2003

DEMARIA, B. ; SILVA, V.; CHOREN, R.; LUCENA, C. VisualAgent: A Software Development Environment for Multi-Agent Systems, In Proceeding of the Tool Section

of the Brazilian Symposium on Software Engineering (SBES 2005), Uberlância, Brazil, 2005.

DELOACH, S. A. Multiagent Systems Engineering: a Methodology and Language for Designing Agent Systems. In: Wagner, G.; Yu, E. (Eds), Proceedings of Agent Oriented Information Systems, Agent-oriented Information System (AOIS99), Washington. 1999.

DELOACH, S. A. Analysis and Design using MaSE and agentTool. In Proceeding 12th Midwest Artificial Intelligence and Cognitive Science Conference, 2001.

HE, M.; JENNINGS, N.R.; LEUNG, H. On Agent-Mediated Electronic Commerce. IEEE Transaction on Knowledge and Data Engineering, 15(4):985-1003, 2003.

KAZAKOV,M.; ABDULRAB, H.; DEBARBOUILLE, G. A Model Driven Approach for Design of Mobile Agent Systems for Concurrent Engineering: MAD4CE Project. Rapport Interne 01-002, Université et INSA de Rouen, 2002.

KIA, A.; AIMEUR, E.; KROPF, P. aKIA Automobile: a Multi-Agent System in E-Commerce. In International Conference on Electronic Commerce (ICERC-5), vol. CD-ROM, 2002.

LOMUSCIO, A.R.; WOOLDRIDGE, M.; JENNINGS, N. A classification scheme for negotiation in electronic commerce. In International Journal of Group Decision and Negotiation, 12(1):31-56, 2003.

NOVIKAU, A. Model Driven Architecture approach in Tropos. Technical Report T04-06-03, Istituto Trentino di Cultura, 2004.

ODELL,J.; PARUNAK, H., BAUER,B. Extending UML for Agents. In Proceedings of Agent-Oriented Information System Workshop at AAAI, pp. 3-17, 2000.

OLIVEIRA, T.; MATHIAS, I.; Lucena, C.; COWAN, D.; ALENCAR, P. Software Process Representation and Analysis for Framework Instantiation. IEEE Transaction on Software Engineering, 2004.

OMG, OMG MDA Guide. Version 1.0.1. Available at: http://www.omg.org/docs/omg/03-06-01.pdf. Accessed on: 11/2004

OMG, Meta Object Facility (MOF) 2.0 Core Specification. Version 2.0. Available at: http://www.omg.org/docs/ptc/03-10-04.pdf. Accessed in: 11/2004.

OMG, OMG XML Metadata Interchange. Version 1.2. Available at: http://www.omg.org/docs/formal/02-01-01.pdf. Accessed on: 11/2004.

OMG, OMG 2.0 Superstructure Specification. Version 2.0. Available at:http://www.omg.org/docs/ptc/04-10-02.pdf. Accessed on: 07/2005.

PADGHAM, L; WINIKOFF, M. Prometheus: A Methodology for Developing Intelligent Agents, In: Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems, Italy. 2002.

POKAHR, A., BRAUBACH, L.; LAMERDORF W. Jadex: Implementing a BDI-Infrastructure for Jade Agents. Research of Innovation, 3(3):76-85, 2004.

SILVA,V.; GARCIA,A.; BRANDAO,A., CHAVEZ,C., LUCENA,C., ALENCAR, P. Taming Agents and Objects in Software Engineering. Software Engineering for Large-Scale Multi-Agent Systems, LNCS 2603, 2003

SILVA,V., LUCENA,C. From a Conceptual Framework for Agents and Objects to a Multi-Agent System Modeling Language. Journal of Autonomous Agents and Multi-Agent Systems, Kluwer Academic Publishers, ISSN 1387-2532, 9(1-2):145-189.

SILVA,V., CORTES,M., LUCENA,C. An Object-Oriented Framework for Implementing Agent Societies. Technical Report MCC32/04, PUC-Rio. Rio de Janeiro, Brazil, 2004.

SILVA, V., CHOREN, R., LUCENA, C. Using the MAS-ML to Model a Multi-Agent System. Lucena, C., Garcia, A.; Romanovky, A., Castro, J., Alencar, P. (Eds.) Software Engineering for Large-Scale Multi-Agent Systems II, LNCS 2940, Springer, 2004.

SYCARA,K., PAOLUCCI,M., VAN VELSEN, M., CRIAMPAPA,J. The RETSINA MAS Infrastructure. Special joint issue of Autonomous Agents and MAS, 7(1-2):29-48, 2003.

VALLECILLO,A., AMOR,M., FUENTES,L. Bridging the Gap Between Agent-Oriented Design and Implementation Using MDA. Autonomous Agents and Multi-Agent Systems Workshop, pp.93-108, 2004.

WAGNER, G. The Agent-Object-Relationship Metamodel. In: Second International Symposium: From Agent Theory to Agent Implementation, 2000.

WOOLDRIDGE, M.; JENNIGS, N. Applications of Intelligent Agents. Agent Technology: Foundations, Applications, and Markets, pp. 3-28, 1998.