Agüero, J.; Carrascosa Casamayor, C.; Rebollo Pedruelo, M.; Julian Inglada, VJ. (2013). Towards the development of agent-based organizations through MDD. International Journal on Artificial Intelligence Tools. 22(2):1-34. doi:10.1142/S0218213013500024.

International Journal on Artificial Intelligence Tools


# Towards the development of agent-based organizations through MDD

Jorge Agüero, Carlos Carrascosa, Miguel Rebollo, Vicente Julián
*Departamento de Sistemas Informáticos y Computación*
*Universitat Politècnica de València*
*Camino de Vera S/N, 46022, Valencia, Spain*
{*jaguero, carrasco, mrebollo, vinglada*}*@dsic.upv.es*

Received (15 Nov 2010)
Revised (28 Feb 2012)
Accepted (Day Month Year)

*Virtual Organizations* are a mechanism where agents can demonstrate their social skills since they can work in a cooperative and collaborative way. Nonetheless, the development of organizations using Multi-Agent Systems (MAS) requires extensive experience in different methodologies and platforms. Model-Driven Development (MDD) is a technique for generating application code that is developed from basic models and meta-models using a variety of automatic transformations. This paper presents an approach to develop and deploy organization-oriented Multi-Agent Systems using a model-driven approach. Based on this idea, we introduce a relatively generic agent-based meta-model for a *Virtual Organization*, which was created by a comprehensive analysis of the organization-oriented methodologies used in MAS. Following the MDD approach, the concepts and relationships obtained were mapped into two different platforms available for MAS development, allowing the validation of our proposal. In this way, the resultant approach can generate *Virtual Organization* deployments from unified meta-models, facilitating the development process of agent-based software from the user point of view.

*Keywords*: Model-Driven Development, Virtual Organization, Multi-Agent Systems.

## 1. Introduction

Advances in new technologies that are mainly based on the Internet and the Web, such as electronic commerce, mobile/ubiquitous computing, or social networks demonstrate the need to develop distributed applications with some intelligent capabilities[51,49,54]. These advances have led to the development of a new paradigm: *service-oriented computing* (SOC), that is, computing based on the interaction between entities, where computing occurs through communication acts among computational entities thereby becoming an inherently *social activity*[36,58,57]. This implies that the computational capabilities are offered and requested by entities inside or outside of the computational system.

To fulfill these advances, this new paradigm requires the technology used to have many features of interaction among independent entities and also to be somewhat intelligent, with the ability to adapt, coordinate, and organize each other[37,62,50,48].

Therefore, the *Virtual Organization* (VO) approach is particularly promising as a support to this paradigm and can be used as a regulatory system (framework) for the coordination, communication, and interaction among different computational entities[13,56].

*Virtual Organizations* are formed by sets of individuals and institutions that need to coordinate resources and services across institutional boundaries[29,13]. Thus, they are open systems formed by the grouping and collaboration of heterogeneous entities, having separated the form and function that require defining how a behavior will take place. They have been employed as a paradigm for developing MAS, where the most relevant approaches include: SODA[55], Electronic Institutions[26], OperA[22], OMNI[23] and GORMAS[7]. Organizations allow that systems to be modelled at a high level of abstraction. They include the integration of organizational and individual perspectives and also the dynamic adaptation of models to organizational and environmental changes by forming groups with visibility boundaries[14,27]. The organization describes the main aspects of a society that is based on different viewpoints such as: *Structure*, *Functionality*, *Norms*, *Interactions*, and the *Environment*[7,21].

These societies (organizations) require high levels of interoperability to integrate diverse information systems in order to share knowledge and facilitate collaboration among organizations. Thus, the organization needs to employ basic software components that support the development of fast and easy composition of distributed applications, even in heterogeneous environments, where the components are easily and cooperatively integrated into other applications to create flexible and dynamic processes. These levels of flexibility and cooperation among different software components is achieved using what is called *Agent-Oriented Software Engineering* (AOSE)[42,34,53,47].

Software engineering based on Multi-Agent Systems is a powerful technology with very significant applications in Distributed Systems and Artificial Intelligence [41,49,29,50]. MAS, which support all of these developments, could require the creation of platforms of highly heterogeneous agents, where agents work together through different interactions to support complex tasks in a collaborative and dynamic way[34,47]. One of the alternatives for providing these complex tasks is to consider the notion of open systems, which are composed of groups of cooperative and heterogeneous agents, that work with local or individual goals to fulfill global goals.

However, existing MAS methodologies propose varying models that are suitable for different domains. Each MAS methodology and platform has their own abstractions for conceptual and computational modeling. Thus, the developers often require the necessary acquisition of new skills to understand and design with the MAS methodologies. As a consequence, the creation of applications is very hard and difficult for the MAS developer, because there is no agreement about a common group of components that can be used across different MAS methodologies and platforms. Therefore, a major challenge when designing MAS is to provide efficient

tools that can be used by non-expert users.

Synthesizing a unified set of components from existing agent-oriented methodologies is a challenge. However, the Model-Driven Development approach can facilitate and simplify the design process and the quality of agent-based software since it allows the reuse of software[8,24,60,12] and transformation between models. MDD basically proposes the automatic generation of code using transformations from models that have platform-independent components. These models are translated into more specific components (or code) that depend on the execution platform, which integrates specific details about the system.

In the MAS literature, researchers are beginning to strive to formulate a set of models that guide the MAS development process using the Model-Driver approach. Some works have concentrated their efforts on creating a very generic unified model for analyzing and modeling different methodologies. Some of the most significant proposals are: TAO[61], FAML[11], Agent UML(AUML)[9], and AML[19]. These proposals create only a conceptual framework to develop and design MAS, but they are not intended to get the MAS deployments to run on specific platforms. Other works, such as PIM4AGENT[33] and CAFnE[40], have a unified meta-model (a little less generic), but these works can generate the MAS deployment to run on specific platforms. Finally, other approaches use MDD as a modelling tool for some MAS methodologies, but they only generate MAS deployments for a single platform. Some of the most significant proposals are: PASSI[20], TROPOS[46], and INGENIAS[31]. However, despite some of the earlier proposals (MDD in MAS uses the concept of organization in their meta-model), none of them focus the organizational development as is proposed by the Virtual Organizations approach, where it is necessary to create different deployments: one for the organization level and another for the agent level.

Thus, our purpose is to use the MDD approach for the design of *Virtual Organizations*. This work proposes an approach for developing MAS that can be implemented in different organization-oriented platforms applying the ideas of MDD. This paper first presents a relatively generic *Virtual Organization* meta-model, which was created mainly using a bottom-up perspective iteratively over organization-oriented agent methodologies. This paper then proposes two transformation models for translating the unified model of the *Virtual Organization* to two different platforms. This process generates code templates automatically (specific target deployments) and then the developer can write any additional code in these templates if deemed necessary. This allows the MAS development to be an easy and fast process. These transformations are proposed as examples, and they allow the feasibility of the proposal to be verified. The organization-oriented target platforms used are: THOMAS[a] [18] and E-Institutions[b][26]. However, this transformation process is not limited exclusively to these agent platforms but is open to other platforms, simply by defining new transformation rules.

[a]http://users.dsic.upv.es/grupos/ia/sma/tools/Thomas
[b]http://e-institutions.iiia.csic.es

The rest of the paper is structured as follows: Section 2 briefly describes the main concepts used in this work. It reviews the different technologies and platforms used to cope with organization development and MDD in the area of MAS. Section 3 details the different meta-models as orthogonal views that describe the complete system to be modelled at a high level of abstraction. Sections 4 and 5 explain how the proposed models can be used to design and develop a complete system. The former details the steps that developer must follow. The latter shows how transformation rules can be defined to generate automatic transformations between models. THOMAS and E-Institutions have been chosen to illustrate the process, and a usage scenario is described. Finally, conclusions of this work are presented in Section 6.

## 2. Background

This section presents a description of the topics and concepts that are the most relevant to the areas of *Virtual Organizations* and MAS development models. It also describes some related contributions with respect to organization modeling in agent-based systems and discusses some open problems. Finally, this section also explains how these problems can be addressed by using the MDD approach.

### 2.1. *Virtual Organizations*

In the area of Multi-Agent Systems, the term *Virtual Organization* (VO) has been primarily used to describe a set of agents that are coordinated with each other through interaction patterns in order to achieve the overall objectives of the system[13]. Therefore, we discuss the main characteristics of *Virtual Organizations* (VOs) and which factors or dimensions are needed for analysis and modeling in order to facilitate the development of Open MAS.

The first methodologies used in the MAS design were the *agent-oriented* ones[38,66,65]. They assume an individualistic perspective, where the principal entity is the agent, which follows its own individual targets based on its own beliefs and abilities. They also consider that agents are benevolent, all have common goals, and cooperate in order to achieve those goals. Therefore, they are only suitable for closed systems. Furthermore, social structures are not modeled specifically but are supposed to emerge as a result of the interaction of agents.

In recent years, some works on agent-based systems have focused on providing procedures and methods of designing open MAS, where agents may have self-interested behavior or be selfish. The open MAS should also permit the participation of heterogeneous agents with different architectures and even different languages[22]. Thus, in order to support open MAS, there is an emerging trend in developers to focus on the organizational aspects of the society of agents, to lead the system development process using the concepts of organization, norms, roles, etc. This has led to a new approach called *organization-oriented* methodologies.

In organization-oriented methodologies, the MAS designer focuses on the organization of the system, taking into account its main objectives, structure, and social norms. Two different trends can be observed when comparing several approaches. On the one hand, methods such as PASSI[20], MOISE[32], TROPOS[46], MESSAGE[17], and INGENIAS[59] detail system roles, groups, and relationships, but they do not explicitly consider social norms. On the other hand, methods and frameworks such as SODA[55], GAIAExOA[67], Electronic Institutions[26], OperA[22], OMNI[23], and GORMAS[7] are focused on the social norms and explicitly define control policies to establish and reinforce them. The main aim of methods of this kind is the design of open Multi-Agent Systems, in which agents with self-interested behavior can participate. These agents can be controlled by means of social norms and a proper organizational structure.

*Virtual Organizations* provide a framework for the activity and interaction of agents through the definition of roles, expectations of behavior, and relations of authority such as control[67]. VOs exist in a new level that is independent of their constituent agents, which can be dynamically replaced. VOs provide a way to divide the system by separating it into groups or units (entities) that maintain certain relationships with each other (providing the context for interaction between agents and different entities) and by taking part in patterns of interaction with other roles in an institutionalized and systematic way.

A VO is represented in a way similar to human organizations, based on the Human Organization Theory[6,7]. This allows the description of the main aspects of an organization: its *structure*, *functionality*, *dynamism*, *environment*, and *norms*. These five elements describe those members (entities) that make up the organization, the topology of the organization, the services and features that the organization offers, the evolution of the organization over time, the environment where the organization is situated, and the rules about the conduct of members, respectively.

Most of the analyzed methodologies do not include all the phases necessary for developing the open MAS. They mainly exclude the latter phases, in which the *Virtual Organization* specification must be converted into executable code for specific agent-based platforms. The fundamental problem for obtaining executable code for VOs is the lack of agent platforms that give support to complex systems of this kind. Although there are currently different frameworks that support the execution of agents (such as JADE[c] or JACK[d]) and some of the platforms deal with organizational concepts, they cannot directly support the concepts that appear in the development process of open MAS, such as norms, roles, or organization topology.

Finally, we propose the use of this approach to create a basic organization-based meta-model that is aimed at modeling open societies in which heterogeneous and autonomous agents work together and that is focused on the integration of both

[c]http://jade.tilab.com/
[d]http://aosgrp.com/

Services and MAS technologies. Thus, entity functionality is described, published, and accessed by means of services.

### 2.2. *Model-Driven Development*

The MDD is a fairly new resource in the software engineering field. The objective of MDD is to build models that are readable by computers, that can be understood by automatic tools in order to generate templates, code, and test models, and that can integrate the code into multiple platforms and technologies[8,24,60,12].

Model-Driven approximation uses and creates different models at different abstraction levels to fuse and combine them when needed to implement the application. When the abstraction levels are too high, these models are known as meta-models (the term "meta" means a higher level of abstraction). A meta-model is simply a model of a modeling language that defines the structure, semantics, and restrictions for a family of models. In MDD, Meta Object Facility (MOF[e])[52] is the language that facilitates meta-model creation. MDD considers three kinds of models at different abstraction levels: the Computation Independent Model (CIM), which details the system's requirements in a model that is independent of the computation; the Platform Independent Model (PIM), which represents the system's functionalities without considering the final platform where it is going to be implemented; and the Platform Specific Model (PSM), which is obtained from combining the PIM model with the specific details of the selected platform.

One fundamental aspect of the MDD is the definition of the *transformation model*, which allows the models to be automatically converted. The transformations allow a model with a given abstraction level to become another one with a different level of abstraction[25,45]. Transformations can be applied to convert one specification from PIM to PSM. This is known as *vertical transformation* because it allows a more general model to be transformed into a more specific one. PIM-to-PIM or PSM-to-PSM transformations can also be applied. These are known as *horizontal transformations*. In general, all of these transformations are known as model-to-model transformations; however, since executable code can be generated from the PSM models, these transformations are known as *model-to-code* or *model-to-text* transformations.

From the viewpoint of the MAS design, different methodologies have identified a set of models to specify the different features of a system. These models can be fitted or reflected in different MDD meta-models by specifying the concepts that describe the MAS (roles, behaviors, tasks, interactions, protocols, etc). The models can be used to model a MAS without focusing on platform-specific details and requirements[63]. Then, it is possible to transform any agent model into agent implementations for different platforms.

Currently, the application of MDD in MAS has different approaches according

[e]MOF Core Specification, http://www.omg.org/docs/ptc/04-10-15.pdf

to the work goals. However, some trends can be observed when comparing these approaches. First, in works such as PASSI[20], TROPOS[16], INGENIAS[31], Sage[44], MetaDIMA[39], and others[15], the use of model-driven approach is proposed in order to wrap the natural complexity associated with the development of MAS. This wrapping must be done by collecting the differences of various methodologies for designing MAS in a specific and proprietary meta-model (rarely a unified meta-model) and then generating deployments that can run on a specific platform. Second, works such as TAO[61], FAML[11], Agent UML (AUML)[9], AML[19], and others[64] pursue the goal of creating a unified meta-model to design and model different MAS methodologies, but without worrying (pro tempora) about the MAS code generation, which can be executed in a platform. The main goal of those works is to provide a *Generic and Unified Conceptual Framework* to understand distinct abstractions, components, and their relationships in order to support the agent design of different MAS methodologies. Third, works such as PIM4AGENT[33] and CAFnE[40] are aimed at creating a unified meta-model (less generic than the previous) that allows agent design with some MAS methodologies and also allows the generation of agent code, so that these deployments can run on different platforms.

An analysis of these approaches indicates that only a few of them support the use of concept organization, for example, FAML, PIM4AGENT, and TAO, and none of them support the use of organizations as another framework different from the MAS framework. Other works propose their own model view with specific components, which creates added complexity for developers. Also, only a few of them achieve the implementation phase, and they only define high-level models. This enormously complicates the work of the developers when they try to obtain executable code.

Following the trend of previous work, we propose using Model-Driven Development in the organization-oriented MAS design. Therefore, we first create a set of VO-based meta-models that allow the deployment to be generated for different MAS platforms that support VOs: THOMAS and Electronic Institutions, as examples. The meta-model proposed is a little less generic than the FAML or TAO, which allow almost any MAS methodology to be analyzed by the framework. Nevertheless, the meta-models proposed are generic enough to support organization-oriented methodologies, as evidenced by the transformations presented in Section 5.

## 3. Modeling Virtual Organizations with MDD

The goal is to provide the user with a unified, intuitive, visual organizational model. Then, the user can use automatic transformations to allow flexible implementation (including deployment) on different agent platforms with support for organizations, to facilitate interoperability of the systems with minimal user intervention. Figure 1 shows a diagram that illustrates this process.

Our work is focused on the meta-model layer (PIM level), which defines different meta-models developed for the open MAS (application domain). This set of meta-models is called *Platform-Independent Virtual Organization Model* ($\pi$VOM). The

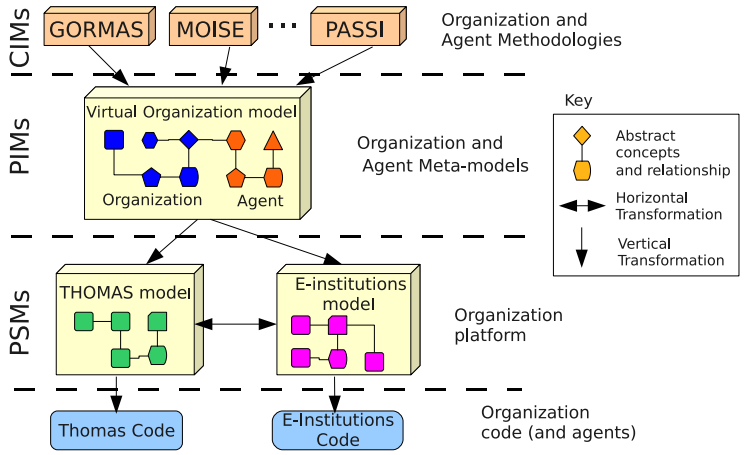


Fig. 1. Relationships among the different MDD models and automatic transformations

creation of this set of meta-models is realized by the detection of common concepts (bottom-up analysis) in existing agent and organizational methodologies (CIM level) complemented by the top-down evaluation of necessary agent and organizational concepts. After that, $\pi$VOM can be converted to a new model that is oriented to the implementation platform of the MAS (PSM level). This is done through a model-to-model transformation (PIM-to-PSM). Finally, the deployment of open MAS is obtained by a model-to-text transformation, which corresponds to the code generated by the model-driven methodology.

One fundamental challenge (when defining a platform-independent meta-model of an open MAS) is to select the concepts or components that should be included in order to model the organization. This is not a trivial task since existing methodologies propose very distinct and varied sets of abstractions that are suitable for different domains. Each methodology has its own abstractions incorporated for conceptual and computational modeling, and there is no agreement about a common group of abstractions that can be used across different methodologies. Also, certain concepts in one meta-model may be contradictory to concepts used in another MAS meta-model.

These problems are addressed in $\pi$VOM in two ways. First, due to the growth capability of the meta-model. $\pi$VOM can be tailored to different domains since it employs common concepts that can be extended to accommodate new abstractions for new domains, in a way similar to the TAO approach. Second, due to the ambiguity of natural language terms (different terms represent the same concept), the semantics of the concept used in the meta-models can be interpreted very broadly (in a way similar to the FAML approach). The developer may interpret the concepts in the most convenient way; this concepts are represented by natural language.

### 3.1. *Integration of Meta-model Concepts*

In *organization-oriented* methodologies, a VO is considered to be a social entity that consists of a specific number of members that carry out different tasks or functions. As discussed in Section 2.1, the main aspects of an organization are Structure, Functionality, Dynamic, Normative, and Environment. Therefore, to model the characteristics of these components in our approach, five key concepts are used: *Organizational Unit*, *Service*, *Environment*, *Norm*, and *Agent*[7]. These concepts make it possible to represent[21]:

- how the entities are grouped with each other in order to define the relationship between the elements and their environment.
- what functionality they offer, including services for the dynamic entry and exit of agents in the organization.
- what restrictions exist regarding the behaviors of system entities.

The meta-model creation was an iterative process. Using a bottom-up perspective, iterations were made between the different MAS methodologies, and, finally, the common subset identified was evaluated with a top-down perspective. This work identifies commonly used concepts that developers often use in organization-oriented methodologies. $\pi$VOM aims to combine several organization modeling language proposals, especially AML[19], AGRE[28], MOISE+[35], INGENIAS[59], GORMAS[7], and OMNI[23].

The *Structural Dimension* of $\pi$VOM takes into account the agent-group-role concepts employed in AGRE; the group, role and link notions employed in MOISE+ and GORMAS; and also the organizational unit concept of AML and its related usage in the Human Organization Theory. In AML, an *organizational unit* is seen both as a global atomic entity and as an association of internal entities, which are related to each other according to their roles, functionality, resources, and environment. Therefore, the structural dimension allows the specification of a system at a high level of abstraction by means of role and organizational unit concepts.

The *Functional Dimension* is normally represented by means of tasks and goals that are pursued by agents. For example, in MOISE+, global goals are defined and decomposed into missions performed by agents. $\pi$VOM Functional description extends previous proposals in three ways:

- Global functionalities are described as a *ComposedService* (complex services) that is composed of several *SingleService* (atomic services), so a complex service specification describes how agent behaviors are orchestrated.
- functionality is detailed in two ways: services that entities perform and services that entities need.
- functionality in $\pi$VOM is described employing the OWL-S standard, which allows the semantic description of services, enhancing their expressibility (for example, representing service preconditions and effects).

Therefore, our proposal focuses on expressing the functionality of a system and its components by means of service descriptions. Thus, Service-Oriented Computing (SOC) concepts such as ontologies, process models, choreography, facilitators, service level agreements, and quality of service measures can be applied to MAS.

The *Normative Dimension* contains a set of mechanisms for ensuring social order and preventing self-interested behaviors. Our proposal makes use of the normative approach of GORMAS, MOISE+, and OMNI. The norms define rules as the description of expected behavior. However, no deviation from the desired behavior is possible. In this sense, they assume the existence of a middleware that controls all agent interactions. Our proposal is not based on a centralized norm enforcer. Thus, agents are free to decide to respect norms. The $\pi$VOM normative dimension defines sanctions and rewards as a persuasive method for norm fulfillment.

The *Environmental Dimension*, which focuses on describing the elements of the environment, has been mainly considered in works such as: AGRE, AML, GORMAS, and INGENIAS. $\pi$VOM *Environmental Dimension* describes the environment components in a standard way, integrating the main abstraction of these approaches. The *Resource* concept has been adopted from the INGENIAS framework and the GORMAS methodology. This concept is similar to the *Body* abstraction of AGRE models, which indicates how agents perform actions on resources. Moreover, the *Port* concept of AML and GORMAS is also integrated in $\pi$VOM, which represents an abstraction for accessing both system resources and published functionality. Therefore, $\pi$VOM Environmental Dimension allows heterogeneous agents to access to external functionalities and resources.

### 3.2. *Meta-model Description: $\pi$VOM*

The intention is that $\pi$VOM will provide a set of generic concepts and components that are useful to a modeling language, while not necessarily providing all the details required by every specific agent-oriented platform. $\pi$VOM is structured in different meta-models or views. The different meta-models used in our approach are described below.

#### 3.2.1. *Structural Meta-model*

This meta-model (see Figure 2) describes the elements of the system (*agents* and *organizational units*) and how they are related. The proposed $\pi$VOM defines an *Organizational Unit* (OU) as a basic social entity that represents the minimum set of agents that carry out some specific and differentiated activities or tasks, following a predefined pattern of cooperation and communication[6,32,67]. This association can also be seen as a single entity at the analysis and design phases, since it pursues goals, offers and requests services, and plays a specific *Role* inside other units. An OU is formed by different *entities* (*has_member* relationship) throughout its life cycle, which can be both single agents and other OUs. The *Organizational Units* present different topologies and communication relationships depending on their

environment, the type of activities that they perform, and their purpose. The basic topologies are[34]: (i) *Simple Hierarchy*, in which a supervisor agent has control over other members; (ii) *Team*, which are groups of agents that share a common goal, collaborating and cooperating with each other; and (iii) *Flat*, in which there is no agent with control over other members. Any other structure can be defined in forms of these three basic topologies.

An OU includes a set of roles that can be acquired by its members (*has_role*) and the sort of relationships with each other (*has_relationship*). The *Role* concept is defined by three attributes: *Visibility*, *Accessibility*, and *Position*. The *Relationship* concept, which is based on [7,61,11], represents social connections between Roles. This relationship connects agents that are entitled to know each other and communicate relevant information. This relationship also implies a *monitoring*, *supervision*, and *controlling* process of agent activity. Table 1 summarizes the main concepts used in the Structural meta-model.

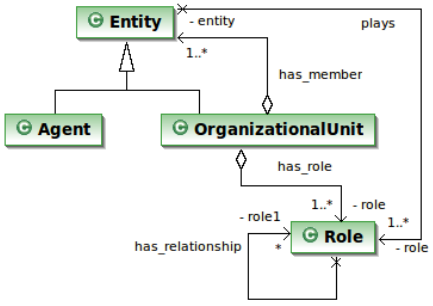


Fig. 2. Concepts used in the Structural meta-model

### 3.2.2. *Functional Meta-model*

This meta-model (see Figure 3) is focused on the integration of both Services and MAS technologies. *Services* represent the functionality that agents or OUs offer to other *entities*, independently of the concrete agent that makes use of it. *Services* can be atomic (simple task) or formed by several tasks. These tasks can be performed by the agent that offers the service, or they can be delegated to other agents by means of service invocation, composition, and orchestration.

An *Entity* is described by an identifier and a membership relation inside a unit in which it *plays* a specific *Role*. It is also capable of offering some specific functionality to other entities. Its behavior is motivated by their pursued *Goals*. Moreover, an *Entity* can also publish its requirements of services (*requires* relation), so then external agents can decide whether to participate inside, thus providing those services. Any service has one or more roles that are in charge of its provision (*provides*) and others that consume it. Furthermore, any service obviously has influence over system goals

(*affects* relation). The *Service* can also be composed of several sub-services, and a "workflow" can be defined using the *RelationType*. Table 1 summarizes the main concepts used in the Functional meta-model.

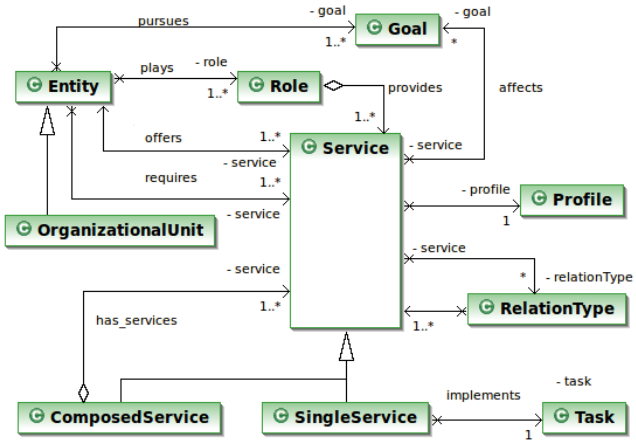


Fig. 3. Concepts used in the Functional meta-model

Table 1. Main concepts employed in the Structural and Functional meta-models

| $\pi VOM$ concepts | Description |
|---|---|
| Entity | Specification of something that has definite and individual existence inside of the organization. |
| Agent | The entity *agent* as usually is represented in MAS methodology. A rational and autonomous entity. |
| Organizational Unit | Specification of a collection or group of cooperative entities (Agents and OUs) to achieve organizational goals. |
| Role | Specification of a behavioural pattern expected from some members in a given organization. |
| Service | A single activity (or complex block of activities) that represents a functionality of agent/organization. |
| ComposedService | A collection of sub-services that make up a Service. |
| SingleService | A single Service that represents a functionality. |
| Goal | A specification of a state that the organization and agents are trying to achieve. |
| Task | Fundamental unit that represents the action performed by an agent. |
| Profile | Specification of a Service, including any preconditions and post-conditions. |
| RelationType | Specification workflow services or sub-services. |

### 3.2.3. *Environment Meta-model*

This meta-model is describes the environmental components, perceptions, and acts on these elements and defines permissions for accessing them. The proposed Environment meta-model (see Figure 4(a)) defines each element of the environment as a *Resource*, which represents an environmental component. It belongs to an entity (*has_resource*), which can be a single agent or an organizational unit. In this last case, an entity in charge of managing the access permissions to this element is needed (has_port). The resource is accessed and perceived through an *EnvironmentPort*. On the other hand, the *ServicePort* concept details the registration of a service in a service directory (registers) or its consumption (serves or requests). Each port is controlled by an entity, and it is employed by one or more roles (*use_port*). A *Port* represents a point of interaction between the entity and other elements of the model and serves as an interface to the real world. Table 2 summarizes the main concepts used in the Environment meta-model.

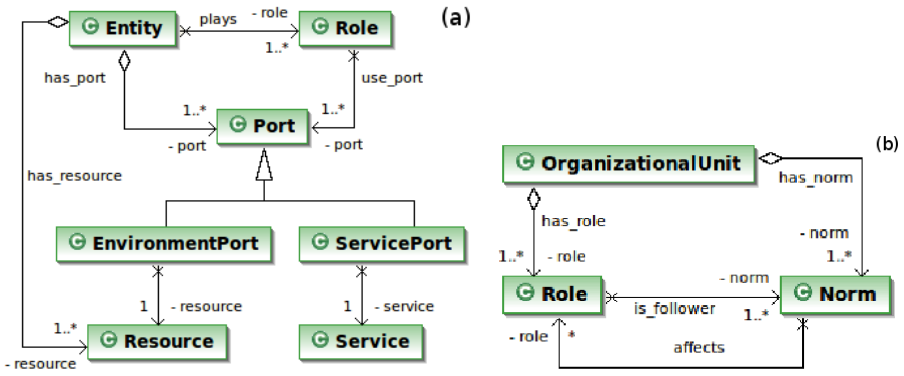


Fig. 4. Concepts used in the Environment(a) and Normative(b) meta-models

### 3.2.4. *Normative Meta-model*

This meta-model assumes that the coordination between agents is achieved through the use of *social norms*. These describe the expected behavior of the members, i.e., what actions are permitted, required, or necessary and which to avoid. They also include penalties to be applied in the case of undesirable actions and the rewards or recognition to be offered for those actions carried out as established by the norm(or rule). *Norms* are used as mechanisms to limit the autonomy of agents in large systems and to solve complex coordination problems. This meta-model (see Figure 4(b)) specifies the set of rules and actions defined to control the behavior of members of the organization, specifically the *Roles* of the organization.

Each OU has a set of *norms* that restricts its member behaviors (*has_norm*). A norm affects a role directly (*affects*), which it is obliged, forbidden, or permitted to perform the specified action. Valid actions are service requesting, registering, or providing. Sanctions and rewards are expressed by means of norms. There are roles that are responsible for controlling norm fulfillment (*is_follower*), whereas defender and promoter roles are responsible for carrying out sanctions and rewards, respectively. Table 2 summarizes the main concepts used in the Normative meta-model.

Table 2. Main concepts employed in the Environment and Normative meta-models

| $\pi VOM$ concepts | Description |
|---|---|
| Port | This abstraction is a facility for receiving and/or transferring information. Access point to a component that allows the input/output of data. |
| EnvironmentPort | Access point to interact with the environment (the communication with the world where the agents are located). |
| ServicePort | Access point to use a service. |
| Resource | Specification of something that has reasonable representation in the environment, that can be perceived and shared. |
| Service | A single activity (or complex block of activities) that represents a functionality of the agents/organization. |
| Norm | A set of rules that are used as mechanisms to limit the autonomy of the organization members. |

### 3.2.5. *Agent Meta-model*

An *Agent* is the basic entity of MAS that is within the organization and uses a series of interaction protocols. The Agent meta-model is a set of interrelated components, each serving a specific function for the agent definition. The main components are: *Behaviours*, *Capabilities*, and *Tasks* (shown in Figure 5).

- *Tasks* represent the *know-how* of the *Agent* and are the components where action or activity is implemented.
- *Capabilities* represent the different situations of the agent and control where *Tasks* are applied. *Capabilities* follow a pattern of *event-condition-action*.
- *Behaviours* are roles that encompass/group these capabilities.

The main reason for splitting the whole problem-solving method is to provide an abstraction that organizes the problem-solving knowledge in a modular and gradual way. The *Task* concept is the concept that incorporates the needed know-how that allows the agent to try to solve a problem. This concept is encapsulated in the meta-model in a *Capability*, which is an event-oriented component to express the circumstances under which a *Task* must be launched to execution.

Moreover, a set of *Capabilities* can be encapsulated into a *Behaviour* that models the response of the agent to different situations. An agent state defines a situation

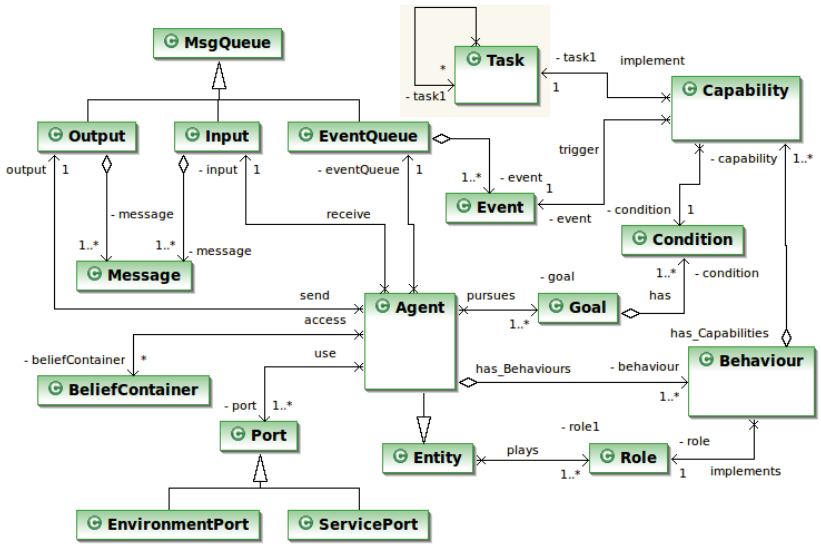


Fig. 5. Concepts used in the Agent meta-model

(which is represented by the current *Beliefs* and *Goals*) that activates a *Behaviour* or allows it to go on being activated. Table 3 summarizes the main components and concepts employed in the Agent meta-model.

Table 3. Main concepts employed in the Agent meta-model

| $\pi VOM$ concepts | Description |
|---|---|
| Agent | The entity *agent* as usually is represented in MAS methodology. A rational and autonomous entity. |
| Behaviour | It encapsulates a set of capabilities activated in specific circumstances; it represents the abstract concept of role. |
| Capability | It represents an event-driven approach to solve a specific problem. |
| Task | The know-how related to a specific problem. |
| Event | It is employed to activate capabilities inside the agent. Occurrence of something that changes the environment and/or agents. |
| BeliefContainer | An abstraction employed to represent the agent knowledge. |
| Goal | A specification of a state that the organization and agents are trying to achieve. |
| Condition | A specification of a set of constraints. |
| MsgQueue | Specification of a collection of different messages (Input, Output, Events). |
| Message | The typical mechanism employed for intercommunication among agents. |

## 4. Development Process

Once the set of models that characterize our proposal of *Platform-Independent Virtual Organization* Model has been presented, the process for transforming the VO into different platforms must be defined. The design process begins by selecting how abstract concepts (which are part of the unified organization model) are mapped onto the target platforms. In this paper, we focus on the study of transformations on two platforms that support agent organizations: THOMAS[18] and E-Institutions[26].

The transformation defines a set of mapping rules. The first set of mapping rules defines which concepts of the source meta-model ($\pi$VOM) are transformed to which concepts of the target meta-model. This process is a model-to-model transformation (PIM-to-PSM), which is illustrated by dotted lines in Figure 6. The second transformation translates the models into the code templates of the organization, which can be optionally combined with code that is written manually by the user. This process is a model-to-text transformation (PSM-to-code).

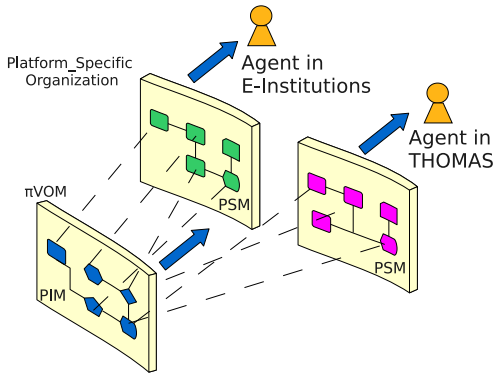


Fig. 6. Transformation from $\pi$VOM to different platforms

The *Development Process* constitutes a set of steps or phases that result in the executable code; however, a set of tools that support the whole process is also needed. The steps employed at each design stage and their required tools are explained in the following subsections.

### 4.1. *Model Creation*

The developer creates diagrams (through graphical tools) that model the different units, roles, tasks, etc. of the developed system. To perform this step, the Eclipse IDE[f] with a set of *plug-ins* is used. These plug-ins are mainly *EMF*, *Ecore*, *GMF*, and *GEF*, which allow the user to draw the models that represent the VO. Obviously, the

[f]http://www.eclipse.org/

meta-models needed ($\pi$VOM, see Section 3) must be loaded into the development environment (CASE tool) in order to generate the appropriate VO models.

To illustrate this phase, a case scenario for making flight and hotel arrangements is used (see Section 5.3 for more detail). The programmer must draw (UML-like) the VO that represents the *Travel Agency*. This scenario is modeled as an organization (*TravelAgency*) inside of which there are two *Organizational Units* (HotelUnit and FlightUnit). Each unit is dedicated to hotels or flights, respectively. Two kinds of *Roles* can interact in the Travel Agency example: the Client role and the Provider role. Figure 7 shows the *TravelAgency* structure, with its units, roles, and their relationships with each other, using GORMAS notation. Similar diagrams must be created in this phase according to the different models that are part of $\pi$VOM.

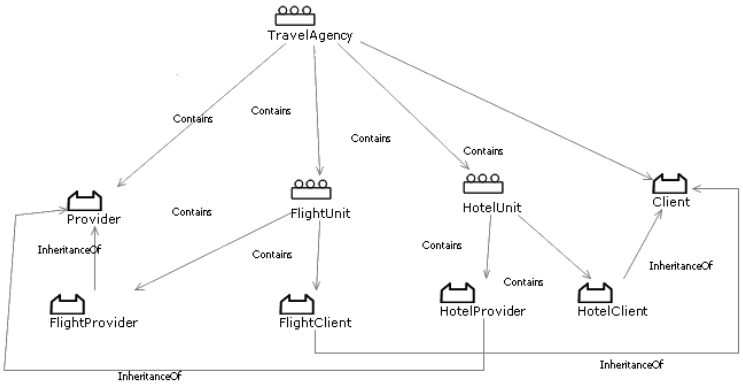


Fig. 7. Structural model of TravelAgency using $\pi$VOM

### 4.2. *Platform Selection and Model Requirement*

Once the PIM is complete by using the different views (structural, functional, norms, environmental, agent), the developer must select the platforms that will be used to execute the different components. The developer must select the platform on which the user wants to execute the different agents that make up the VO. In this step, the agents can be executed on different platforms according to the system modeling (scenario). For example, a possible scenario is one where different ubiquitous agents run on various embedded platforms (PDAs or cellular phones) that interact with *Virtual Organizations* to request different services.

To do this, a model-to-model transformation (PIM-to-PSM) must be applied using the Eclipse IDE and the *ATL plug-in*[5] that incorporate the appropriate set of transformation rules. It is important to note that the same general VO model can be transformed into different specific VO platforms. The rules for the component transformations between two VO meta-models (from $\pi$VOM to E-Institutions and

THOMAS) are explained in detail in Section 5. In this way, VO concepts are mapped from source models to target models, and VO components are transferred, moved, or changed from one model to another. This step is illustrated in Figure 8.

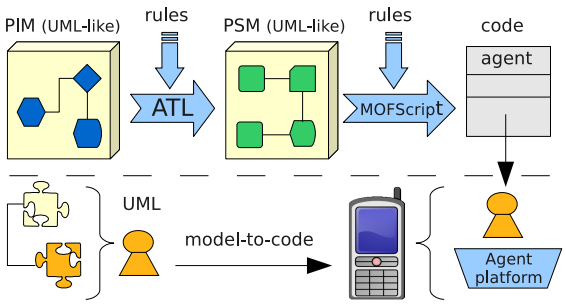


Fig. 8. Development Process for VO design using two stages of transformation

Transformation rules are hidden to the developer, and the programmer only uses them when the execution platform (PSM) is selected. By applying transformation rules, the developer obtains a specific model for the chosen platform. Different platforms can be chosen for different parts of the system. After that, the developer can refine the model to add the details that correspond to the new abstraction level.

To illustrate how the rules are defined in the *ATL* language, Figure 9 shows Rule 9 (see Section 5, to view the definition of this rule). This rule generates all the *Scenes* (in E-Institutions) from the *OUs* (in $\pi$VOM) using the same Class Name. This code also shows the function `getAllRoles()`, which examines all the *Roles* associated with each *OU*. This function will be mapped to the *Roles* Agent that is used in the different *Scenes* (the function `getAllRoles()` will be used by the *Roles* Rule).

```
helper context PIVOM!OrganizationalUnit
 def : getAllRoles() : OrderedSet(PIVOM!OrganizationalUnit) =
 self.children->iterate(child; hasRole: OrderedSet(PIVOM!OrganizationalUnit)=
    if child.oclIsTypeOf(PIVOM!Roles) then
          hasRole.append(child)
    endif
 );
rule OrganizationalUnit2Scene {
  from
       PIM : PIVOM!OrganizationalUnit(PIM.isOrganizationalUnit Root())
  to
       PSM : EInstitution!Scene (
          name <- PIM.name
        ...
```

Fig. 9. Rule 9 (Organizational Unit to Scene) in ATL language

### 4.3. *Code Generation*

In the last step, the developer applies a model transformation to convert the designed models into code. To do this, the developer must use a PSM-to-code transformation. In this case, *MOFScript*[5], which is an Eclipse *plug-in* that uses templates to do the translation process, is used. From a practical viewpoint, the transformation/generation of code consists of going through an XML file that describes the components and relationships of the source meta-model and then generating another XML file that contains the specification of the E-Institution or THOMAS platforms that will be the application launcher. This step is illustrated in Figure 8, assuming that the agent is running on a cellular phone.

Figure 10 illustrates how the transformation rule is implemented using *MOFScript*. This rule corresponds to **Rule 2** (see Section 5). This rule generates code for the *Agent* concept in the THOMAS platform. These templates have been developed specifically for E-Institution and THOMAS. Additional transformations for other execution platforms can be defined. Only the rules that map the concepts to the target platform must be defined.

```
texttransformation UMLAGENT2THOMAS (in myAgentModel:uml2)
...
//Rule1: Agent transformation
uml.Package::mapPackage () {
  self.ownedMember->forEach(c:uml.Class)
     if (c.name != null) if (c.name = Agent) c.outputGeneralization()
}
uml.Class::outputGeneralization(){
   file (package_dir + self.name + ext)
   self.classPackage()
   self.standardClassImport ()
   self.standardClassHeaderComment ()
   <% public class %> self.name <% extends Agent { %>
      self.classConstructor()
      <% // Attributes  %>
      self.ownedAttribute->forEach(p : uml.Property) {
            p.classPrivateAttribute()
      }
      newline(2)
 <%}%>
...
}
```

Fig. 10. Example of transformation of the *Agent* concept using MOFScript.

Finally, after completing these steps, the designer has a method for developing agents in a fast and easy way by means of a design tool. First, the user creates platform-independent models, drawing the agent organizations using a UML-based approach. Second, the user selects the specific platform where the models are executed, in order to do this the appropriate transformation process is applied (only by selecting the appropriate option in the CASE tool), and thereby obtain the corresponding deployments.

This facilitates the development process, as the rules and the transformation

process are hidden from the user point of view (the developer). CASE tools internally load the transformation rules of the specific platform and execute the transformation process of model-to-model and model-to-code automatically. The transformation from the user point of view is to select the target platform and run the translation. This process generates code templates automatically and then the user can write any additional code in these templates if deemed necessary.

## 5. Transformation Rules

Once the *Virtual Organization* meta-model ($\pi$VOM) and the *Development Process* have been presented, the transformation rules from a PIM to different PSMs must be described. To do this, a model-to-model transformation (PIM-to-PSM) must be applied. The components and concepts are transferred, or changed, from one model to another. These transformations are performed at two levels: the organizational level (organization framework) and the agent level (organization members).

### 5.1. *Organizational Level Transformation*

This section explains how to translate the model that represents the organization framework (PIM) into two target platform models (PSMs). The chosen PSMs are: *THOMAS* and *E-Institutions*. The same process has to be done for any other platform. The only limitation is that the platform must include organizational concepts.

#### 5.1.1. *THOMAS Architecture and Execution Framework*

THOMAS (MeTHods, Techniques, and Tools for Open Multi-Agent Systems) is a recent open Multi-Agent System architecture that consists of a related set of modules that are suitable for the development of systems applied in environments that work as a "society"[43]. Due to the technological advances of recent years, the term "society" needs to meet several requirements:

- Distribution, constant evolution, and flexibility to allow members to enter or exit the society.
- Appropriate management of the organizational structure that defines the society.
- Multi-device agent execution including devices with limited resources, and so on.

The THOMAS platform uses EMFGormas[30g], which is CASE tool to support it. This is an organization-based CASE tool that allows agent and *Virtual Organizations* to be modeled.

The $\pi$VOM meta-model presented in this paper is very similar to the model of the organization programmed in THOMAS since both works are partially based on

[g]http://users.dsic.upv.es/grupos/ia/sma/tools/EMFGormas

the methodology and artifacts proposed by GORMAS (these meta-models can be found at [7]). For this reason, the automatic transformations are relatively easy to describe. Almost all of the abstract concepts of $\pi$VOM are represented in THOMAS, so the model-to-model transformation rules are expressed almost as one-to-one relationships. It is convenient to note that some concepts in THOMAS have a more detailed feature than $\pi$VOM because THOMAS is a platform-specific model. The main transformation rules that must perform the translation between different models are shown in Table 4 (from **Rule 1** to **Rule 8**). Since there is a 1 to 1 mapping between both models (PIM and PSM), the transformation rules are not described.

Table 4. Rules from $\pi$ *VOM* to THOMAS platform

| Rule | Concept | Transformation to THOMAS |
|---|---|---|
| 1 | Organizational Unit | $\pi$VOM.OU ⇒ THOMAS.OU |
| 2 | Agent | $\pi$VOM.Agent ⇒ THOMAS.Agent |
| 3 | Role | $\pi$VOM.Role ⇒ THOMAS.Role |
| 4 | Service | $\pi$VOM.Service ⇒ THOMAS.Service |
| 5 | Norm | $\pi$VOM.Norm ⇒ THOMAS.Norm |
| 6 | RelationType | $\pi$VOM.RelationType ⇒ THOMAS.Process |
| 7 | Resource | $\pi$VOM.Resource ⇒ THOMAS.Resource |
| 8 | Goal | $\pi$VOM.Goal ⇒ THOMAS.Goal |

#### 5.1.2. *E-Institutions Platform*

E-institutions provide a set of tools that is widely used with agents in order to model organizations. E-institutions can be effectively designed and implemented as *electronic institutions* composed of a vast number of heterogeneous (human and software) agents that play different roles and interact by means of speech acts[26]. This platform is based on traditional human institutions and offers a general agent-mediated computational model that serves to create an *agent-mediated electronic institution* platform. Table 5, shows the main components of E-Institutions.

The relationships between these components are shown in Figure 11. This is the target meta-model used in the transformation process (model-to-model) from $\pi$VOM to E-Institutions.

The main transformation rules from $\pi$VOM to E-Institutions are shown in Table 6 (from **Rule 9** to **Rule 16**).

Some details of the main transformation rules are described below:

- **Rule 9**. This rule indicates that each OU is a Scene. These are the entities where agents collaborate to perform the actions of the organization.
- **Rule 10** and **Rule 11**. These rules have a 1 to 1 mapping, since both the *Agent* and *Role* concepts are represented on both platforms (i.e., *Agents* are part of the organization on both platforms and each is assigned a *Role* to *Play*).

Table 5. Core concepts used in E-Institutions

| E-Institution concepts | Description |
|---|---|
| Agent | A rational and autonomous entity inside of the E-Institutions. |
| Role | Specification of a behavioural pattern expected from the E-Institution agents. |
| Scene | A scene is a pattern of multi-agent interaction. |
| State | Represent a node of a finite state oriented graph, which describes Scene protocol. |
| Transitions | Specification of the workflow among the Scenes. |
| Illocutions | Valid expressions of the agent communication language, which are the arcs between States. |
| Ontology | The knowledge of the agent and the E-Institutions. |
| World | Access point to interact with the environment. |
| Norm | A set of rules that are used as mechanisms to limit the autonomy of the E-Institution agents. |

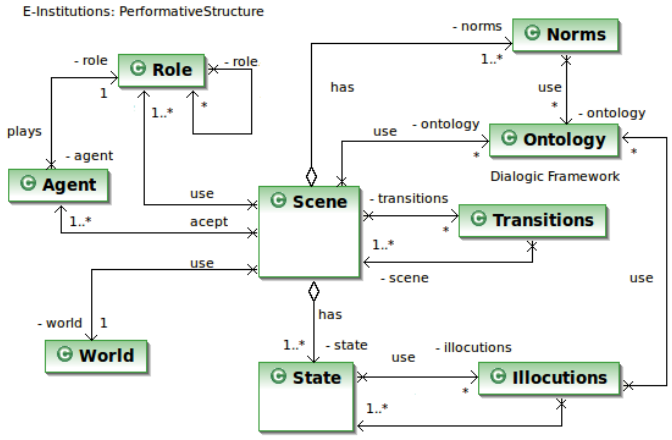


Fig. 11. Core concepts used in E-Institutions (target meta-model)

Table 6. Rules from $\pi VOM$ to E-Institutions

| Rule | Concept | Transformation to E-Institutions |
|---|---|---|
| 9 | Organizational Unit | $\pi$VOM.OU $\Rightarrow$ EI.Scene |
| 10 | Agent | $\pi$VOM.Agent $\Rightarrow$ EI.Agent |
| 11 | Role | $\pi$VOM.Role $\Rightarrow$ EI.Role |
| 12 | SingleService | $\pi$VOM.Service $\in$ OU $\Rightarrow$ EI.State $\in$ Scene |
| 13 | RelationType | $\pi$VOM.RelationType $\Rightarrow$ EI.Transition OR EI.Illocutions |
| 14 | Norm | $\pi$VOM.Norm $\Rightarrow$ EI.Norm |
| 15 | Goal | $\pi$VOM.Goal $\Rightarrow$ EI.Norm |
| 16 | Resource | $\pi$VOM.Resource $\Rightarrow$ EI.World |

- **Rule 12**. The *Service* represents the functionality of the OU and similarly a set of *States* provides functionality in a *Scene*. Therefore, when a functionality of an OU is modeled with a *ComposedService*, this *Composed-*

*Service* must be transformed to a set of *States* (i.e., a *SingleService* should be translated as a *State*).

- **Rule 13**. The *RelationType* describes the level of *Services* workflow. This rule is closely related to *Rule 12* because when the composition between the services corresponds to *SingleService*, the *RelationType* must be transformed to *Illocution*. Now, when the *RelationType* represents the flow between *ComposedService*, the mapping should be to *Transition*.
- **Rule 14** and **Rule 15**. The *Norms* describe what an agent can do within each Scene and State (in E-Institutions). For this reason, the *Norms* and *Goals* pursued in $\pi$VOM are translated into E-Institutions *Norms*. They specify what the agent is allowed to do, and what the agent must do to achieve specific norms (*Goals*).
- **Rule 16**. The *Resources* describes the objects or artifacts (in the environment) that are accessible by agents or entities. Thereby, these *Resources* are transferred to the abstract concept of environment in E-Institutions, to the *world* concept.

Summarizing, the transformation rules show that our meta-model has enough generality to transfer the MAS design to two organization-oriented platforms. However, the transformation process is not limited exclusively to these organization-oriented agent platforms (E-Institutions and THOMAS), but it is also open to other platforms. Thus, our meta-model is relatively generic to define new transformation rules to new platforms, simply by defining new rules for the specified target platform.

Finally, the above transformation rules are incorporated or loaded into the CASE tools to make the process of transformation (automatic or semi-automatic translation). These rules are hidden from the developer and will be used when the developer wants to translate the model into a deployment (platform model) for its final execution.

### 5.2. *Agent Level Transformation*

As stated above each agent identified in the system must be modeled according to the proposed agent meta-model. Then, each agent modeled in the system can be transformed into code according to the specific agent platform where the agent will be executed. The agent model analyzed in this paper is the JADE[h] agent model.

#### 5.2.1. *JADE Platform*

**JADE**[10], which is one of the most popular platforms that support agent execution, is widely used because it provides programming concepts that simplify the MAS

[h]http://jade.tilab.com/

implementation. JADE is FIPA compliant in the communication infrastructure between agents. This agent platform is supported by THOMAS and E-Institutions.

One of the main concepts used in the implementation of JADE agents is *Behaviour*[i]. A *Behaviour* represents a specific task that the agent executes. There are different types of behaviours that the agent can execute. To support this, JADE offers different *Behaviour* classes, which are specializations of a simple *Behaviour* such as: temporal, sequential, and parallel, etc. Table 7 summarizes the main *Behaviours* used by the JADE agent.

The communication paradigm that is adopted is asynchronous message passing. Agents must share the same language, vocabulary, and protocols. This is done by defining an *ontology* that permits semantics to be used in the content of the messages that are exchanged among the agents. Another important concept in JADE is the schema. A schema is a structured framework that represents the structure of the concepts that make up an ontology. JADE schemas are concepts that provide a kind of data structure that directly maps the structure of an ontology. Table 7 summarizes the main concepts used by JADE.

Table 7. The JADE components model

| Concept | Use | Descriptions |
|---|---|---|
| Agent | Behaviour | The task that an agent can carry out. |
| | Ontology | The agent's Knowledge. |
| Ontology | Schema | Data structure of Messages. |
| ACL Communications | Performative | FIPA compliant Messages. |
| **Type Behaviours** | **Specialization** | **Descriptions.** |
| SimpleBehaviour | OneShot | Executes an action only once. |
| | Ticker | Executes an action periodically. |
| | Weaker | Waits for a period of time to execute an action. |
| | Cyclic | Executes an action cyclically. |
| CompositeBehaviour | Sequential | Executes several actions sequentially. |
| | FSM | The actions are executed in a Finite State Machine. |
| | Parallel | Executes several actions in parallel. |

Table 8 shows the transformations rules needed to transfer a $\pi$VOM agent model to a JADE agent model; as a convention the JADE Model (PSM model) is called JADEM.

Some details of the main transformation rules are described below:

- **Rule 17**. The conversion is direct because our agent model matches the JADE agent model. After the transformation, the methods have to be reviewed to check that the JADE agent works properly. One of the most important methods to be derived is `init()` because this method contains the code executed by the agent. Then, the `init()` method of the Agent is moved into the `setup()` method of JADEM, i.e., `init()` $\rightarrow$

[i]which has a different meaning than in the Agent Meta-model

Table 8. Transformation rules from *Agent* meta-model to JADE

| Rule | Concept | Transformation to JADE |
|---|---|---|
| 17 | Agent | $\pi$VOM.Agent ⇒ JADEM.Agent |
| 18 | Behaviour | $\pi$VOM.Behaviour ⇒ JADEM.ParallelBehaviour |
| 19 | Capability | $\pi$VOM.Capability ⇒ JADEM.OneShotBehaviour |
| 20 | Task | $\pi$VOM.Task ⇒ JADEM.Behaviour |
| 21 | Events | $\pi$VOM.Event ⇒ JADEM.ACLMessage |
| 22 | Beliefs | $\pi$VOM.BeliefContainer ⇒ JADEM.Schema |
| 23 | Goal | $\pi$VOM.Goal ⇒ JADEM.Ontology |
| 24 | Message | $\pi$VOM.Message ⇒ JADEM.ACLMessage |

`setup()`. Other methods are also derived: the method to destroy the agent `destroy()` → `takeDown()` and the method to add behaviors `addBeh()` → `addBehaviour()`.

- **Rule 18**. A *Behaviour* in this agent model is a set of actions that can be executed. To make it possible to launch several actions, a **Behaviour** corresponds with a CompositeBehaviour in JADEM. Specifically, for each *Behaviour* referenced in Agent, a ParallelBehaviour must be added in JADEM. This ParallelBehaviour will be empty at first, but a new *Behaviour* will be added for each task in the model when the *Capability* and *Task* of Agent are transformed.
- **Rule 19**. A *Capability* is a component that may or may not launch an activity depending on the arrival of the corresponding event, that is, the *Capability* to launch a *Task* if its trigger event has arrived (event-driven). To emulate this behaviour, each *Capability* corresponds with a JADE simpleBehaviour, whose goal is to verify the arrival of an event. If the event is the correct one, then the activity will be launched.
- **Rule 20**. A *Task* in our agent model can be a simple or a complex action. The type of *Task* establishes a specific transformation to a `SimpleBehaviour` or a `CompositeBehaviour`. For example, if there is a cyclic task in Agent, a `CyclicBehaviour()` must to be added in JADE. For each *Task* in Agent, a type of *Behaviour* must be added in JADEM. A *Task* is the place where users write their code. Therefore, it is important to define how to do this in JADEM. This can be done by translating the `doing()` method of Agent to the `action()` method in JADE.
- **Rule 21**. The transformation of an *Event* is not direct, but it can be done easily, since each event type corresponds to an ACL message type with a concrete performative in JADE.
- **Rule 22**. A *BeliefContainer* stores the agent knowledge (which corresponds to the schema concept in JADE) that is used in the ontology definition (*Schema*).
- **Rule 23**. A *Goal* is mapped to the components used in JADE to represent knowledge, which is the ontology.
- **Rule 24**. The *message* transformation is simple: the message in our model

corresponds to an ACL message in JADE-Leap with a specific performative.

The PSM model must be transformed into code (PSM-to-code), translating each concept that is included in the meta-model into a code template. Figure 12 shows the code template generated by the JADE agent model.

Summarizing, this section explains the transformation rules that allow the MAS design to be transferred to a JADE-based deployment. However, our meta-model is generic and flexible enough to allow the transformation process to be extended to other agent platforms, simply by defining new rules for the specified target platform (i.e. JACK[j] or MAGENTIX2[k]). In fact, this process has been successfully tested in other agent platforms[4,2], in which our meta-agent model was transferred to two light-weight embedded agent platforms: ANDROMEDA[l] and JADE-Leap.

Similar to the rules at the organization level, these rules are incorporated into the CASE tools to perform the process of transformation (automatic or semi-automatic translation). These rules are hidden from the developer and will be used when the developer wants to translate the MAS model into a MAS deployment (platform model) for its final execution.

```
public class my_Customer extends Agent {
   ...
  protected void setup(){                                    Behaviour
     ParallelBehaviour my_FlightClient =
        new ParallelBehaviour( ParallelBehaviour.WHEN_ALL );

     my_FlightClient.addSubBehaviour( new my_Service(this) );
     ...
     addBehaviour(my_FlightClient);
   }  // ----------- end setup()
                                                            Capability
     class my_Service extends OneShotBehaviour {
         public my_Service(Agent a) {
           super(a); }

         public void action() {
           ... //check condition == true
           addBehaviour(new my_TaskService(this) );
         }
     } // ----------- end OneShotBehaviour

     ...                                                        Task
     class my_TaskService extends Behaviour {
         public my_TaskService(Agent a) {
            super(a); }

         public void action()  {
             //...this is where the code Task goes !!
         }
     } // ----------- end Behaviour

} // ----------- end class Agent
```

Fig. 12. Code template in JADE platform

[j]http://aosgrp.com/
[k]http://www.gti-ia.dsic.upv.es/sma/tools/magentix2/
[l]http://www.gti-ia.upv.es/sma/tools/Andromeda/

### 5.3. *Usage Scenario*

To illustrate the automatization process produced by the usage of the rules, a case scenario for making flight and hotel arrangements is presented. This is a well-known example that has been modeled by means of electronic institutions in previous works (Dignum[22]; Argente et al[7]). The *Travel Agency* example is an application that facilitates the interconnection between clients (individuals, companies, travel agencies) and providers (hotel chains, airlines) delimiting services that each one can request or offer. The system controls which services must be provided by each entity. Provider entities are responsible for the internal functionality of these services. However, the system imposes some restrictions on service profiles, service request orders, and service results.

In this system, agents can search for and make hotel and flight reservations and pay in advance for bookings. This case study is modeled as an organization (*TravelAgency*) inside which there are two organizational units (HotelUnit and FlightUnit) that represent a group of agents. One of these units is dedicated to hotels and one is dedicated to flights. Three kinds of roles can interact in the Travel Agency example: the Customer, Provider, and Payee roles. The Customer role requests system services. More specifically, it can request hotel or flight search services, booking services for hotel rooms or flight seats, and payment services. The Provider role is in charge of performing the service. The Payee role provides the advanced payment service. Figure 7 shows the *TravelAgency* structure, with its units, roles, and relationships with each other.

Even though each *Organizational Unit* can provide different services, to simplify this usage scenario, we assume that there is just one service. Therefore, the *TravelAgency Unit* offers the service of travel *search*, *FlightUnit* offers *Seats* reservations on airline flights, and the *HotelUnit* offers *Rooms* reservations in Hotels (see Figure 13). To provide the *Search* service, *TravelAgency Unit* requires the use of the *Seats* and *Rooms* services offered by other *Organizational Units*. This generates a workflow among different services through *RelationType*(RT) (see Figure 13). The ComposedService of *TravelAgency* can be composed into single services (as Figure 13 shows), that the Service *Rooms* is divided into sub-services: *CheckDestination*, *CheckAvailability*, and *SelectOffers*. The *Search* and *Seats* services are also divided into sub-services; that for reasons of simplicity these sub-services in the usage scenario are not shown.

The process begins by modeling the Travel Agency (structural and functional models (see Figures 7 and 13)), and applying the rules in order to obtain the organizations in the THOMAS and E-Institutions platforms. In the case of the THOMAS platform, since the models are very similar and their transformations are almost direct, they have not been analyzed here.

In contrast, to obtain the organization in the E-Institution platform and to create the components of *PerformativeStructure* (see Figure 14), the application of different rules is required, for instance: **Rule 9**, **Rule 11**, **Rule 12**, and **Rule 13**

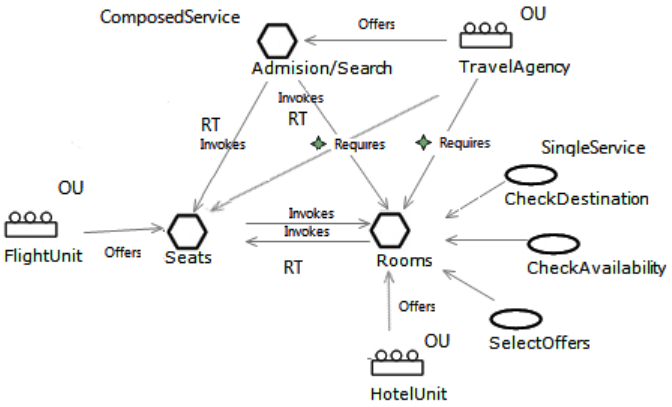


Fig. 13. Functional model (partial view) of TravelAgency

(Section 5).

**Rule 9** allows all of the *Scenes* in E-Institutions that correspond to the *OU* of $\pi$VOM (3 in total) to be obtained. It is then possible to obtain the *Roles* allowed in each *Scene* by applying **Rule 11**. It is important to note that two *Scenes* for entrance and exit must be added (root and output) in the PerformativeStructure.

After applying **Rule 13** (*RelationType*) and analyzing the existing workflow in *ComposedService*, we can specify each type of transition among the different *Scenes* in E-Institutions, which, in this case, correspond to *Transitions*.

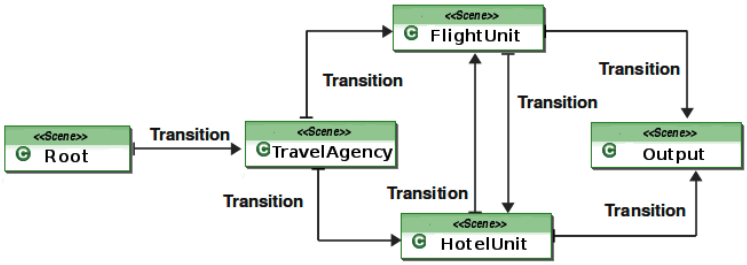


Fig. 14. E-Institution Concepts used in TravelAgency

As stated above, this mapping generates the basic template of the components/concepts used in the PerformativeStructure of E-Institutions (Figure 14). With the application of the remaining rules, a more detailed description of the PerformativeStructure is obtained. The transformation process can still be developed further. Figure 13 shows that the *Rooms* Service is composed of three sub-services: *CheckDestination*, *CheckAvailability*, and *SelectOffers*. If we know the workflow among these three *SingleServices*, the *States* in the *Scene* can be obtained, after **Rule 12** and **Rule 13** are applied. Figure 15 shows the workflow among the

*SingleServices*.

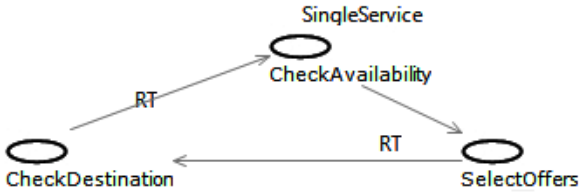


Fig. 15. Workflow among the SingleServices (Functional model)

This mapping generates the basic template of the *States* used in the *Scene* (Figure 16). After applying **Rule 13** (*RelationType*) and analyzing existing workflow among *SingleServices*, we can specify each type of transition among the different *States*, which in this case correspond to *Illocutions*.

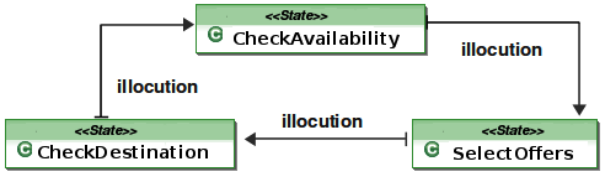


Fig. 16. State machine in E-Institutions

This usage scenario demonstrates the feasibility of the proposed meta-model and its transformations to develop organizational-oriented MAS.

## 6. Discussion and Conclusions

This paper has presented a MDD approach to develop *agent-based open organizations*. MDD can be considered as a new paradigm to develop software systems in which the different stages in the software development process can be automatically connected by defining mappings. Thus, in the context of MDD in AOSE, we have identified the following advantages that our MDD approach offers:

- The employment of an abstract meta-model to design and model agent systems based on Virtual Organizations.
- The generation of a transformation process from PIM to PSM, which could provide a simple interface to implement the models created by $\pi$VOM (abstract meta-model). Therefore, the approach reduces: (i) the gap between design and implementation; (ii) the knowledge required for the implementation of MAS with respect to the deployment MAS platforms.

Similar approaches can be observed in works such as: TROPOS, INGENIAS, Sage, and MetaDIMA. However, these works use proprietary meta-models and typically generate deployments that can only run on a specific platform.

In works such as TAO, FAML, Agent UML (AUML), and AML, the main goal is to provide a *Generic and Unified Conceptual Framework* to understand distinct abstractions in order to support the agent design of different MAS methodologies. These approaches are mainly focused on the analysis phase, whereas the implementation phase is missing. Instead, our approach is a relatively generic meta-model, that it allows to analyze some MAS methodology, and additionally seeks to obtain the MAS deployments.

Therefore, works such as PIM4AGENT and CAFnE are aimed at creating a unified meta-model that allows agent design with some MAS methodologies as well as the generation of agent code, and these deployments can run on different platforms. However, these approaches have a limited view of the agent organization (as well as FAML and TAO), and none of them view or support organizations (*Virtual Organizations*) as another framework different from the MAS frameworks.

Finally, this work presents how to develop *Agent-Based Virtual Organizations* using MDD. This work introduces a *Virtual Organization* meta-model (called $\pi$VOM), which allows organizations in MAS to be modeled using abstract components that are independent of the implementation platform following a MDD approach. This meta-model is divided into five views that focus on the most important aspects of *Virtual Organizations*. These views can easily be extended to a specific domain if required.

The meta-model proposed can be used to create code templates for specific platforms for organizations. This work has discussed the use of transformations on the THOMAS and E-Institutions platforms. These transformations show that the meta-model can be considered to be platform-independent. This work has been tested at different levels of abstractions. In Agüero et al[3] platform level transformations were evaluated, while agent level transformations were checked in Agüero et al[1,4].

The above target platforms that have been used and discussed in this work (E-Institutions and THOMAS) allow the feasibility of our proposal to be verified, defining the transformation rules for each platform. These transformation rules are presented to show that our meta-model has enough generality to translate the MAS design to two organization-oriented platforms. However, this transformation process is not limited exclusively to these agent platforms; it is also open to other platforms, simply by defining new specific transformation rules for each different platform. Our proposal is relatively generic for defining new transformation rules to new platforms.

Our approach allows the MAS to be developed in a fast and easy way. The developer creates the platform-independent models, drawing the agent organizations using an UML-based approach. Then, the developer selects the specific platform where agents (and organization models) are executed using the transformation process, and thereby obtains the corresponding deployments automatically. This facilitates the development process and the rules and the transformation process are hidden from

the user's point of view (the developer). The MAS software tools internally load the transformation rules of the target platforms and execute the transformations of model-to-model and model-to-code automatically. This process generates code templates automatically and then the developer can write any additional code in these templates if deemed necessary.

As future work, we plan to propose new transformations in order to obtain the agent instances and to generate the agent code in other frameworks. We also plan to introduce specific components/views so that the *Virtual Organization* can provide the framework and components necessary to model agreements among autonomous entities using the MDD approach to design them.

## References

1. Agüero, J., Rebollo, M., Carrascosa, C., Julián, V.: MDD-based agent-oriented software engineering for ubiquitous deployment. In: The Sixth Annual International Conference on Mobile and Ubiquitous Systems (MobiQuitous 2009), vol. ICST/IEEE press, pp. 1–2. (2009).
2. Agüero, J., Rebollo, M., Carrascosa, C., Julián, V.: Developing Pervasive Systems as Service-oriented Multi-Agent Systems. In: 7th International ICST Conference on Mobile and Ubiquitous Systems: (MobiQuitous 2010), vol. CD press, pp. 1–12 (2010).
3. Agüero, J., Rebollo, M., Carrascosa, C., Julián, V.: MDD for Virtual Organization design. In: International conference on Practical Applications of agents and multiagent systems(PAAMS2010), vol. 71, pp. 9–17 (2010).
4. Agüero, J., Rebollo, M., Carrascosa, C., Julián, V.: Model-driven development for ubiquitous MAS. In: International Symposium on Ambient Intelligence (ISAmI 2010), *Advances in Soft Computing*, vol. 72, pp. 87–95 (2010).
5. Allilaire, F., Bézivin, J., Jouault, F., Kurtev, I.: ATL: Eclipse Support for Model Transformation. In: European Conference on Object-Oriented Programming (ECOOP2006) (2006).
6. Argente, E., Julian, V., Botti, V.: Multi-Agent System Development based on Organizations. Electronic Notes in Theoretical Computer Science **150**, 55–71 (2006)
7. Argente, E., Julian, V., Botti, V.: MAS Modelling based on Organizations. In: 9th Int. Workshop on Agent Oriented Software Engineering (AOSE08), pp. 1–12 (2008)
8. Atkinson, C., Kuhne, T.: Model-driven development: a metamodeling foundation. IEEE software **20**(5), 36–41 (2003)
9. Bauer, B.: UML Class Diagrams Revisited in the Context of Agent-Based Systems. Proceedings Agent-Oriented Software Engineering pp. 101 – 118 (2002).
10. Bellifemine, F., Poggi, A., Rimassa, G.: JADE - A FIPA -compliant agent framework. In: Proceedings of the Practical Applications of Intelligent Agents (1999).
11. Beydoun, G., Low, G., Henderson-Sellers, B., Mouratidis, H., Gomez-Sanz, J., Pavón, J., Gonzalez-Perez, C.: FAML: A Generic Metamodel for MAS Development. IEEE Transactions on Software Engineering pp. 841–863 (2009)
12. Bézivin, J.: On the unification power of models. Software and Systems Modeling **4**(2), 171–188 (2005)
13. Boella, G., Hulstijn, J., van der Torre, L.: Virtual Organizations as Normative Multiagent Systems. Hawaii International Conference on System Sciences (HICSS) **7**, 192–201 (2005).
14. Boissier, O., Hübner, J., Sichman, J.: Organization oriented programming: From closed

to open organizations. Engineering Societies in the Agents World VII pp. 86–105 (2007)
15. Brando, A., Silva, V., Lucena, C.: A model driven approach to develop multi-agent systems. Tech. rep., Technical Report, Departmento de Informatica-Pontificia Universidade Catolica do Rio de JaneiroPUCRio (2005). ftp://ftp.inf.puc-rio.br/pub/docs/techreports/05_09_brandao.pdf
16. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems **8**(3), 203–236 (2004)
17. Caire, G., Coulier, W., Garijo, F., Gomez, J., Pavon, J., Leal, F., Chainho, P., Kearney, P., Stark, J., Evans, R., et al.: Agent oriented analysis using MESSAGE/UML. Agent-Oriented Software Engineering II pp. 119–135 (2001)
18. Carrascosa, C., Giret, A., Julian, V., Rebollo, M., Argente, E., Botti, V.: Service Oriented Multi-agent Systems: An open architecture. In: Autonomous Agents and Multiagent Systems (AAMAS), pp. 1–2 (2009)
19. Cervenka, R., Trencansky, I.: The Agent Modeling Language – AML, vol. ISBN: 978-3-7643-8395-4. Whitestein Series in Software Agent Technologies and Autonomic Computing (2007)
20. Cossentino, M., Potts, C.: PASSI: A process for specifying and implementing multi-agent systems using UML. Tech. rep., Technical report, University of Palermo (2001)
21. Criado, N., Argente, E., Julián, V., Botti, V.: Designing Virtual Organizations. In: 7th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS2009), vol. 55, pp. 440–449, (2009)
22. Dignum, V.: A model for organizational interaction: based on agents, founded in logic. Phd dissertation, Utrecht University (2003)
23. Dignum, V., Vázquez-Salceda, J., Dignum, F.: Omni: Introducing social structure, norms and ontologies into agent organizations. Lecture Notes Artificial Intelligent **3346**, 181–198 (2005)
24. D'Souza, D.: Model-driven architecture and integration opportunities and challenges. In: Version 1.1, Kineticum (2001)
25. Elvesæter, B., Hahn, A., Berre, A., Neple, T.: Towards an interoperability framework for model-driven development of software systems. Interoperability of Enterprise Software and Applications pp. 409–420 (2006)
26. Esteva, M., Rodríguez-Aguilar, J.A., Sierra, C., Arcos, J.L.: On the formal specifications of electronic institutions. Agent mediated electronic commerce. Lecture Notes in Computer Science **1991**, 126–147 (2001)
27. Ferber, J., Gutknecht, O., Michel, F.: From agents to organizations: an organizational view of multi-agent systems. Proceedings AOSE, Lecture Notes in Computer Science **2935**, 214–230 (2003)
28. Ferber, J., Michel, F., Baez, J.: AGRE: Integrating environments with organizations. Environments for multi-agent systems: first international workshop, E4MAS pp. 48–56 (2005)
29. Foster, I., Kesselman, C., Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal Supercomputer Applications **15**(3), 200–222 (2001)
30. Garcia, E., Argente, E., Giret, A.: A modeling tool for service-oriented Open Multiagent Systems. In: The 12th International Conference on Principles of Practice in Multi-Agent Systems. PRIMA 2009, *LNAI*, vol. 5925, pp. 345–360. Springer-Verlag (2009)
31. Garca-Magario, I., Gómez-Sanz, J., Fuentes, R.: INGENIAS Development Assisted

with Model Transformation By-Example: A Practical Case. In: 7th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2009), pp. 40 – 49 (2009)
32. Gateau, B., Boissier, O., Khadraoui, D., Dubois, E.: Moiseinst: An organizational model for specifying rights and duties of autonomous agents. In: Proceedings of Workshop Coordination and Organisation (CoORG 2005), pp. 484–485 (2005)
33. Hahn, C., Madrigal-Mora, C., Fischer, K.: A platform-independent metamodel for multiagent systems. Autonomous Agents and Multi-Agent Systems **18**(2), 239–266 (2008)
34. Horling, B., Lesser, V.: A survey of multi-agent organizational paradigms. The Knowledge Engineering Review **19**(04), 281–316 (2005)
35. Hübner, J., Simao Sichman, J., Boissier, O.: S-Moise+: A middleware for developing organised multi-agent systems. Lecture Notes in Computer Science **3913**, 64–77 (2006)
36. Huhns, M., Singh, M.: Service-oriented computing: key concepts and principles. IEEE Internet Computing **9**(1), 75–81 (2005).
37. Huhns, M., Singh, M., Burstein, M., Decker, K., Durfee, K., Finin, T., Gasser, T., Goradia, H., Jennings, P., Lakkaraju, K., Nakashima, H., Van Dyke Parunak, H., Rosenschein, J., Ruvinsky, A., Sukthankar, G., Swarup, S., Sycara, K., Tambe, M., Wagner, T., Zavafa, L.: Research directions for service-oriented multiagent systems. IEEE Internet Computing **9**(6), 65–70 (2005).
38. Iglesias, C., Garijo, M., Gonzalez, J.: A survey of agent-oriented methodologies. Lecture notes in computer science **1555**, 317–330 (2000)
39. Jarraya, T., Guessoum, Z.: Towards a model driven process for multi-agent system. In: CEEMAS (ed.) Multi-Agent Systems and Applications V, vol. 4696, pp. 256–265. Springer (2007)
40. Jayatilleke, G., Padgham, L., Winikoff, M.: A model driven component-based development framework for agents. International Journal of Computer Systems Science & Engineering **20**(4), 273–282 (2005)
41. Jennings, N., Wooldridge, M.: Applications of intelligent agents. In: Agent Technology: Foundations, Applications, and Markets, pp. 3–28 (1998)
42. Jennings, N., Wooldridge, M.: Agent-oriented software engineering. Lecture notes in computer science **1647**, 4–10 (1999)
43. Julian, V., Rebollo, M., Argente, E., Botti, V., Carrascosa, C., Giret, A.: Using THOMAS for Service Oriented Open MAS, pp. 56–70. Springer (2009)
44. Kirby, J.: Model-Driven Agile Development of Reactive Multi-Agent Systems. In: Computer Software and Applications Conference, 2006. COMPSAC'06. 30th Annual International, vol. 2 (2006)
45. Kleppe, A., Warmer, J.B., Bast, W.: MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley Professional (2003)
46. Kolp, M., Giorgini, P., Mylopoulos, J.: Multi-agent architectures as organizational structures. Autonomous Agents and Multi-Agent Systems **13**(1), 3–25 (2006)
47. Longbing, C., Zhang, C., Ruwei, D.: Organization-oriented analysis of open complex agent systems. International Journal of Intelligent Control and Systems **10**(2), 114–122 (2005)
48. Luck, M., McBurney, P.: Computing as interaction: agent and agreement technologies. In: Proc. of the 2008 IEEE International Conference on Distributed Human-Machine Systems, pp. 1–6 (2008)
49. Luck, M., McBurney, P., Shehory, O., Willmott, S.: Agent Technology: Computing as Interaction. A Roadmap for Agent Based Computing (2005)
50. Maamar, Z., Mostefaoui, S., Yahyaoui, H.: Toward an agent-based and context-

oriented approach for Web services composition. IEEE Transactions on Knowledge and Data Engineering pp. 686–697 (2005)
51. Minsky, N., Ungureanu, V.: Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. ACM Transactions on Software Engineering and Methodology (TOSEM) **9**(3), 273–305 (2000)
52. Object management group (OMG). meta object facility (MOF) 2.0 core specification. http://www.omg.org/docs/ptc/04-10-15.pdf (October 2004)
53. Odell, J., Nodine, M., Levy, R.: A metamodel for agents, roles, and groups. Agent-Oriented Software Engineering V pp. 78–92 (2005)
54. Ohtani, T., THINT, M.: An Intelligent System for Managing and Utilizing Information Resources Over the Internet. International Journal on Artificial Intelligence Tools **11**(1), 117–138 (2002)
55. Omicini, A.: SODA: Societies and infrastructures in the analysis and design of agent-based systems. In: Agent-Oriented Software Engineering 1957, pp. 185–193. Springer (2001)
56. Omicini, A., Ricci, A., Viroli, M.: RBAC for organisation and security in an agent coordination infrastructure. Electronic Notes in Theoretical Computer Science **128**(5), 65–85 (2005)
57. Papazoglou, M., Georgakopoulos, D.: Service-oriented computing. Communications of the ACM **46**(10), 25–28 (2003)
58. Papazoglou, M., Traverso, P., Dustdar, S., Leymann, F.: Service-oriented computing: State of the art and research challenges. IEEE Computer Society **40**(11), 38–45 (2007)
59. Pavón, J., Gómez-Sanz, J.: Agent oriented software engineering with INGENIAS. In: Proceedings of the 3rd Central and Eastern European conference on Multi-agent systems, pp. 394–403. Springer-Verlag (2003)
60. Selic, B.: The pragmatics of model-driven development. IEEE software **20**(5), 19–25 (2003)
61. Silva, V., Garcia, A., Brandão, A., Chavez, C., Lucena, C., Alencar, P.: Taming agents and objects in software engineering. Software engineering for large-scale multi-agent systems. LNCS **2603**, 103–136 (2003)
62. Singh, M., Huhns, M.: Service-oriented computing: semantics, processes, agents. John Wiley & Sons Inc (2005)
63. Skarmeas, N., Clark, K.: Component based agent construction. International Journal on Artificial Intelligence Tools **11**(1), 139–164 (2002)
64. Sturm, A., Dori, D., Shehory, O.: Single-model method for specifying multi-agent systems. In: Proceedings of the second international joint conference on Autonomous agents and multiagent systems, pp. 121–128. ACM (2003)
65. Sudeikat, J., Braubach, L., Pokahr, A., Lamersdorf, W.: Evaluation of Agent–Oriented Software Methodologies–Examination of the Gap Between Modeling and Platform. Agent-Oriented Software Engineering V pp. 126–141 (2005)
66. Wooldridgey, M., Ciancarini, P.: Agent-oriented software engineering: The state of the art. In: Agent-Oriented Software Engineering, LNAI 1957, pp. 55–82. Springer (2001)
67. Zambonelli, F., Jennings, N.R., Wooldridge, M.: Developing multiagent systems: The GAIA methodology. ACM Trans. Softw. Eng. Methodol. **12**(3), 317–370 (2003)