Meta-modelling in Agent Oriented Software Engineering

Jorge J. Gómez-Sanz, Juan Pavón

[1]Dep. Sistemas Informáticos y Programación,
Univ. Complutense, 28040 Madrid, Spain
`{jjgomez,jpavon}@sip.ucm.es`
`http://grasia.fdi.ucm.es`

Abstract. The MESSAGE methodology has shown that the application of meta-models in the development of Multi-Agent Systems (MAS) facilitates the integration of current software engineering processes and the use of agent technology concepts in analysis and design activities. This paper presents an extension to the MESSAGE meta-models and how to conduct the modelling (i.e., instantiation of the meta-models into a specific MAS) following the Unified Software Development Process, especially considering the importance of an incremental development and the role of the different meta-model entities in this process. This process is illustrated by an example.

Keywords: Multi-agent system, Analysis and Design Methodology, Meta-modelling
Topics: Multi-Agent Systems

PAPER TRACK

Meta-modelling in Agent Oriented Software Engineering

1. Introduction

Most of the existing agent development methodologies consider a development process of a short number of steps for analysis and design of the MAS, which may seem rather simplistic, especially when compared with standard development processes, such as the Unified Software Development Process [9]. Developing a Multi-Agent System (MAS) is more complex than a conventional object oriented application, as it has to cope with distributed intelligent entities. It is true that in some cases, like MaSE [4], the presence of a tool, such as an agentTool, simplifies the development process. However, this means risking a loss of flexibility because of the constraints of the underlying agent model. For instance, the idea of an agent in agentTool is that of a conventional process, whose behavior is specified as state machines (which is not convenient for deliberative agents). ZEUS [13] facilitates the development process by providing implemented solutions for planning, ontologies management, and communication between agents. When the MAS is a small one and when the problem is restricted to the academic domain, and for rapid prototyping, such tools are useful and a methodology of just a short number of steps sounds reasonable. However, an industrial development, involving a team of several engineers requires management of activities, a more detailed software development process, and the control of the cost involved in making a MAS work under real world workload.

A trend in current software methodologies is the provision of methods for incremental development. Though the idea is present in the development processes of existing agent methodologies, it is not clear how to achieve an incremental development using existing formalisms. For instance, GAIA [16] propose iterations of some development steps in order to comply with the specification of *roles*; however, the formalism that gathers the information from iteration to iteration is not well suited for that task. GAIA uses card-like specifications that are textual descriptions of what is being modelled. Changes to such specifications are not trivial and may involve high costs. This lesson is well known in software engineering and that is why today developers use other formalisms, such as UML.

From this perspective of industrial software engineering, and trying to avoid the constraints of a specific agent model, the MESSAGE project [2] addresses the definition of an agent development methodology by identifying the generic elements required to build a MAS, organizing them in five views, and expressing them using a meta-modelling language. The resulting meta-models (one for each view) can be applied by integration into a well-proven software development process model, which in the case of MESSAGE is the Unified Software Development Process [9], although others could be considered.

The work presented in this paper is an extension to MESSAGE in several aspects. First, by providing a more detailed definition of the agent, organization, interaction, tasks and goal views of a MAS, and by adding the *Environment* view, which substitutes the *Domain* view from MESSAGE. Meta-models have been described using a meta-tool (METAEDIT+ [10]) to create specific editors for building MAS models (illustrations in this paper are taken from the application of such a tool), which allows the developer to work directly with agent concepts instead of specific implementation entities such as classes or rules. Another improvement to MESSAGE methodology is a more detailed definition of the activities required in the MAS analysis and design phases in the context of the Unified Software Development Process, and the provision of a path and supporting tools to obtain an implementation.

The next section discusses how meta-modeling supports the definition of a language for the development of a MAS using agent technology concepts, and which elements have to be considered in each view of the MAS. The third Section explains how meta-models are integrated into the development process, and which kind of activities are required for building a MAS. This is illustrated with a working example, in order to provide a practical insight into the proposed methodology. Finally, we present some conclusions. For further information and a detailed description of all the meta-models and the activities of the methodology visit our web site at http://grasia.fdi.ucm.es.

2. MAS meta-models

The meta-models describe the entities that should be part of a MAS and their relationships. As such, the task of the MAS developer is to define models with specific instances of the entities from the meta-models. In the case of an object-oriented application a model is defined by a set of classes (instances of meta-classes) and their relationships. In the case of a MAS we are interested in the identification of types of organization, groups, workflows, agents, perceptions, interactions, mental state, goals, tasks, resources, etc., which are instances of the entities in the MAS meta-models. In this sense, the MAS meta-models provide a high level language for development MAS in terms of agent technology concepts (although in the end they have to be translated into computational terms such as classes, rules, neural networks, depending on the implementation technology).

In order to structure the specification of the MAS, we can consider several views (this separation of concerns is also applied in most of the existing MAS methodologies, for instance, Vowel engineering [14] and MASCommonKADS [7]), and therefore one meta-model for each one:

- **Agent meta-model.** Describes agent's responsibilities with tasks and roles. It also takes into account the control of the agent defining its goals and mental states required during execution. With instances of this model, we can define constraints in the freedom of action of the agent without it being restricted to a specific control paradigm.
- **Organization meta-model.** Organization is the equivalent of a system architecture. Following Ferber [5] and MESSAGE [6], there are structural relationships that are not restricted to hierarchies between roles. These structures are delegated to specialized entities, *groups*. In the organization model there are also power relationships among *groups*, *organizations*, and *agents*. Functionality of the organization is expressed using workflows which show consumer/producer

associations between tasks as well as assignment of responsibilities for their execution, and resources associated to each.

- **Environment meta-model.** The environment defines the sensors and effectors of the agents. It also identifies available resources as well as already existing agents and applications.
- **Tasks and Goals meta-model.** As we base this on a BDI model and Newell's *principle of rationality* [12], it is important to justify the execution of tasks in terms of the goals that are achieved. We provide decomposition of tasks and goals. To relate both, there are specialised relationships that detail which information is needed to consider a goal to be solved or failed. Finally, this meta-model also provides low level detail of tasks in the system, describing which resources are needed in the execution, which software modules are used throughout the process, and which are the inputs and outputs in terms of entities of these meta-models.
- **Interaction meta-model.** Describes how coordination among agents takes place. It goes a step further than sequence diagrams (UML) in the sense that it reflects the motivation of the interaction and its participants. It also includes information about the mental state required in each agent throughout the interaction as well as tasks executed in the process. In this way, we can justify at design level why an agent engages in a interaction and why it should continue.

The generation of models from these meta-models is not trivial, since there are dependencies between different views. For instance, tasks appearing in workflows in an organization model should also appear in a Tasks and Goals model. That is why meta-models need to be integrated into a consistent software development process, as described in the next section.

The example in section 4 shows some of the concepts defined in the above meta-models in some detail. Note that for each entity in the meta-model there is a graphical representation that allows the developer to work at a higher level, i.e., with agent concepts instead of implementing specific artefacts. This is supported by a visual tool generated by a meta-tool from the meta-models specifications.

3. Integration in the Unified Software Development Process

For each meta-model, we have defined a set of activities (around seventy) in the software development process that lead to the final MAS specification. Initially, activities are organised in UML activity diagrams showing dependencies between them. Instead of showing these activities here, **Fig. 1** summarises the results required in each phase of the Unified Software Development Process. Meta-models are used as specification language of the MAS the same way as UML does for object oriented applications. We have used a meta-tool, METAEDIT+, that takes as input the meta-models specifications and generates graphical tools that are used by the developer to produce models using the concepts described in the meta-models (for which we have also associated a graphical representation). Models developed using these graphical tools consist of the agent concepts described in the meta-models. The example in the next section shows some diagrams for a particular case study, and the iterative and incremental nature of the development process.

		PHASES		
		Inception	**Elaboration**	**Construction**
WORKFLOWS	**Analysis**	o Generate use cases and identify actions of these use cases with interaction models. o Sketch a system architecture with an organization model. o Generate enviroment models to represent results from requirement gathering stage	o Refined use cases o Agent models that detail elements of the system architecture. o Workflows and tasks in organization models o Models of tasks and goals to highlight control constraints (main goals, goal decomposition) o Refinements of environment model to include new environment elements	o Refinements on existing models to cover use cases
	Design	o Generate prototypes perhaps with rapid application development tool such as ZEUS o Agent Tool.	o Refinements in workflows o Interaction models that show how tasks are executed. o Models of tasks and goals that reflect dependencies and needs identified in workflows and how system goals are achieved o Agent models to show required mental state patterns	o Generate new models o Social relationships that perfect organization behaviour.

Fig. 1. Results to be obtained in each phase of the development process

In the analysis-inception phase, organization models are produced to sketch how the MAS looks like. This result, equivalent to a MAS architecture, is refined late in the analysis-elaboration phase to identify common goals of the agents and relevant tasks to be performed by each agent. Task execution has to be justified in terms of organizational or agent's goals (with task-goal models). This leads to identify the results that are needed to consider a goal as satisfied (or failed). In the design-elaboration phase, more detail is added, by defining workflows among the different agents (with organization models), completing workflow definition with agent interactions (with interaction models), and refining agent's mental state as a consequence (with agent models). According to the Unified Software Development Process, the goal of elaboration phase is to generate a stable architecture, so only of the most significant use cases should be considered (the key functionality of the system). Remaining use cases, which are supposed to deal with special situations but that do not provide changes in the system architecture, are left to the construction phase.

According to Unified Software Development Process, the different iterations would point to a certain level of detail. We perform activities with an increasing level of detail in the products obtained so far. Testing and implementation phases have not been included in this paper. Testing should not be different from conventional software testing. We assume that use cases determine core functionality to be developed. From these use cases, test suites can be generated. Regarding implementation, we envision two choices: (1) consider generated models as a specification of the system like those generated by UML and perform the implementation manually; and (2) try to generate the code automatically from the specification. The first approach was carried out in the MESSAGE project [3]. The second option is oriented towards the encapsulation of the implementation so designers of MAS are not that interested in the code. Work in this line has already been done by ZEUS [13], AgentBuilder [8] and, recently, agentTool [15]. However, in most cases the target platform cannot be changed. The only one that supports this feature is agentTool.

To facilitate automated code generation, we use a complete MAS architecture made of components implemented in different languages. Current target languages include JAVA, Java Expert Ssystem Shell (JESS), April or PROLOG. We have also tried to generate a code for agent platforms, specifically JADE [1]. Our approach does not embed the final code into the application. Instead, it assumes that the source code of the architecture is marked up with tags. These tags are later substituted by data from the models according to a process defined by the developer. With meta-models and the supporting tool (METAEDIT+), it is easy to generate a representation of the models in other languages. In our case, we use PROLOG as intermediate representation of the meta-models. Later, and with another PROLOG program, we run the parameterisation procedure to generate the final system.

4. A structured development example

The example shows a model of an organisation of personal agents to hep the user in managing the personal computer. The diagrams shown in this section have been generated directly from meta-models. The tool that supports meta-modeling allows work to be carried out in the same way as in a conventional software engineering tools. Also, the tool checks that during the development, models are defined exactly as conceived at the meta-model level.

Problem statement

Managing a personal computer is a tiring task, as we know. Though operative systems facilitate the management of the computer, there is not too much management support for the tons of programs that today can be installed in a PC. Conventional support only includes tools to install/uninstall the program and detect collisions in the access to system resources (e.g. printers, files).

However, it is well known that there is also information overload in the PC. There are many programs producing information (e-mail, chat-programs, internet monitors, bookmarks, word-processors) without any more control than the user's orders. This leads to an information overload which causes user's desidia. For instance, users tend to spread their documents throughout the folders in one or many virtual (e.g. NFS) or physical (e.g. current hard disk) storage media. E-mail messages are stored forever in a forever growing in-box. Notifications of changes in some monitored URL are ignored again and again. Agents have been applied to solve some of these problems, especially those concerning email (Maxims [11]). However, what should be designed is one or several organizations of agents able to collaborate among themselves to deal with this information overload in the PC.

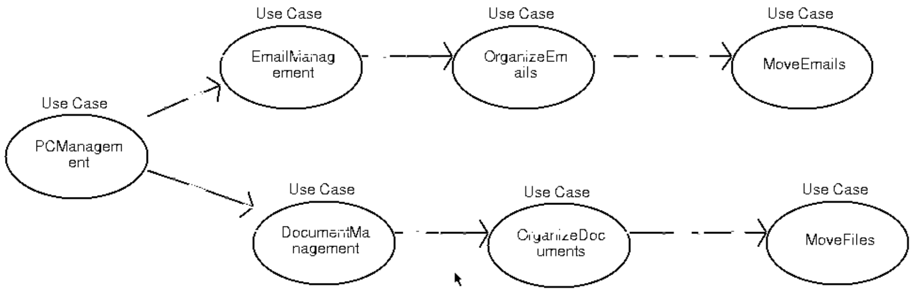

Fig. 2. Use case diagram that identifies relevant use cases in this case study

Analysis-inception

We start identifying initial use cases oriented towards PC management (see **Fig. 2**). With these use cases we want to cover email and document management, and more specifically, organize information in the PC when the information is emails and files on the hard disk.

Organization is seen, in this case, as the allocation of files in different folders (email folders or hard disk folders). As readers may imagine, there is a strong chance that the system will grow, by adding new agents to deal with other PC management tasks or improving agent functionality by adding some collaboration between agents allocated in different PCs.

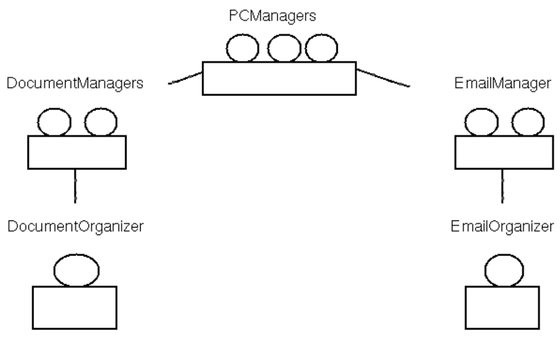

Fig. 3. Organization model of agents in the PC. Rectangles with one circle denote agents, rectangles with two circles, groups, and rectangles with three circles, organizations.

According to the initial specification, initially, there should be two kinds of agents: agents that organize user's email (*EmailOrganizer)* and agents that organize user's documents on the hard disk (*DocumentOrganizer*). As the system is expected to grow with different assistants for other PC management issues, it seems reasonable to start grouping these agents into two organizational structures: *Document Manager* and *Email Manager*. This tentative system architecture is sketched in **Fig. 3**.

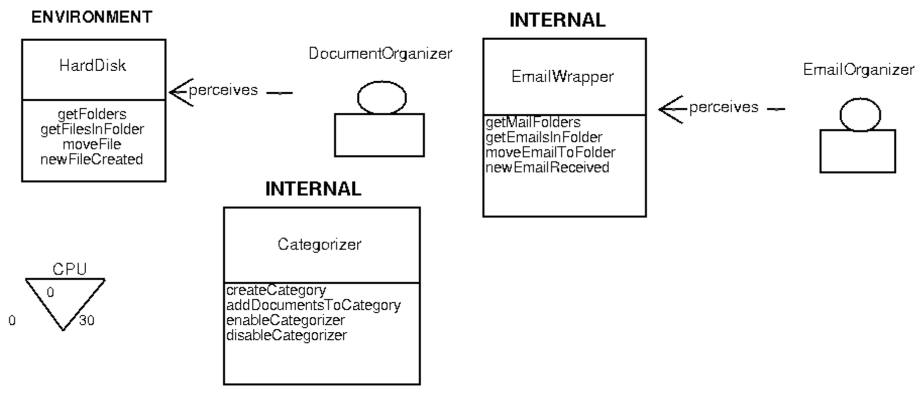

Fig. 4. Tentative environment model. Inverted triangles denote resource. Left number is the lower usage threshold, center number denotes expected initial value, right number is higher threshold. System applications are represented with the same symbol as UML objects.

The environment, the user's PC in this case, is modelled focussing on the aspects that are going to be modelled. These are files on the hard disk and an email application. To handle these aspects (see **Fig. 4**), we define application entities (*HardDisk* and *EmailWrapper*) and some expected methods. Among these, we highlight *newFileCreated* and *newEmailReceived* operations since with them we are able to define agent's perception (*perceives* associations). To be able to organize information, emails and files, we decided to apply a text mining tool that provides clustering and classification capabilities. As a requisite, we established that the use of CPU should not exceed 30 %, so that user's work is not disturbed.

After this study, we should return to the organization model and include this new information. Though it will not be shown here, changes include integrating *EmailWrapper* in the *EmailManager* group, *HardDisk* in *DocumentsManager*, and the *Categorizer* to both groups.

Design and Implementation–inception

In this stage we make some prototypes to test the viability of the proposal using text-mining tools to classify emails and other text documents. To facilitate experiments, we assume that the user uses the Eudora email client, which stores emails and attachments separately (this facilitates the work of the text mining tool since binary files distort clustering results). To test user interaction, we develop an HTML prototype. Since agent interaction is not considered at this point in the development, we do not see any need to use any rapid application development agent tool (like ZEUS [13] or agentTool [15]).

Analysis-elaboration

In this stage we continue adding details to each agent's specification. As a refinement of the *email organization* use case, we add a new scenario that considers relationships between emails and other user's documents. These related documents may improve email organization by providing more documents whose allocation can serve as a quality test.

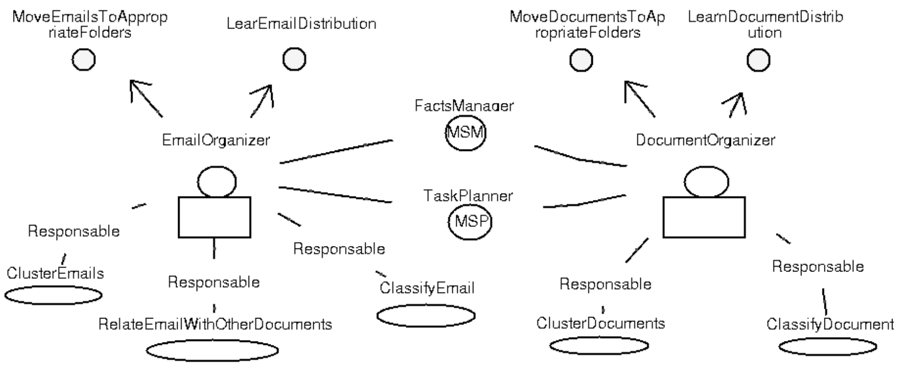

Fig. 5. *EmailOrganizer* and *DocumentOrganizer* description. MSM means *Mental State Manager* and MSP *Mental State Processor*. Ovals denote tasks. Circles denote goals.

Each agent is represented in **Fig. 5**. As one can seen, the functionality of both agents is quite similar. However, the domain of application of each task is completely different. Email structure includes mail headers and MIME types. Hard disk files, however, can be word documents, HTML pages, or GIF files, among others. As the categorizer tool performs final classification on ASCII documents, we need to process the different sources of information accordingly, and this is the purpose of these tasks.

To begin considering agent control, we assume that it will be composed of a mental state manager (to manage the knowledge of the agents) and a mental state processor (in charge of taking decisions upon current knowledge). In this case, we established that we would use facts as a unit of knowledge and that a task planner would decide what to do next. In order to justify this planner, a proper definition of tasks should be provided.

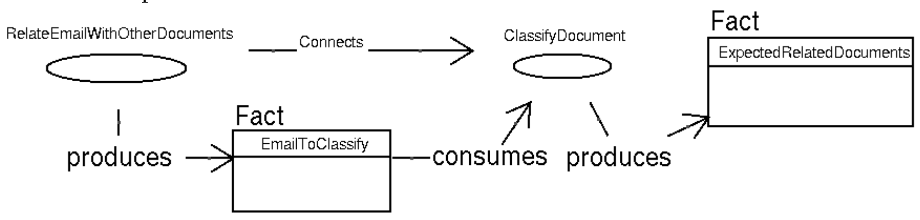

Fig. 6. Description of consumer-producer relationships between two tasks. Facts produced and consumed belong to the mental state of the agent.

What **Fig. 6** shows is that there is a workflow that relates *EmailOrganizer* tasks with *DocumentOrganizer* tasks. It also defines each task by the required inputs and the outputs that are produced. Information is lacking here about how new facts contribute to goal satisfaction, but this is part of the incremental specification. For the moment, it may be enough to know that these two tasks are related and what the nature of the relationship is. Note that during the specification of these tasks we discover mental entities that should exist in the mental state of the agent, so in fact we are defining control restrictions (*ClassifyDocument* cannot work until an *EmailToClassify* fact is present in the mental state of its responsible agent).

The process would continue identifying new goals and tasks, and modifying organization accordingly (indicating the existence of workflows and new resources, for instance). During the process, key interactions, like the one added between the two agents are associated with use cases and existing goals. These interactions are expected to be fully detailed during their design. Of course, the environment model may also change, since new applications may be required, for instance, to perform the transformation from word processing documents to ASCII documents.

Other stages

In the design stage, the detail of the interactions increases, specifying which tasks are executed throughout the interaction and which mental states are required from each agent in each stage. Again new tasks or mental entities can be discovered during the design and so existing models should be modified to maintain consistence.

For reasons of brevity, other stages have been omitted. However, we would like to point out that the specification continues incrementally and that, in parallel, implementation starts from the construction stage.

5. Conclusions

The paper shows the generation of the specification of a MAS using meta-models. Though the development of the example specification has been summarised, it shows that it is possible to build MAS following an incremental and iterative software development process. Though the example cannot be compared with more serious problems, like coordination of robots in a factory, there has been experiments with problems of a similar degree of complexity, such as the coordination of hundreds of agents distributed throughout several computers in order to perform information personalization. Current application domains range from interface agents and planning agents to collaborative filtering systems.

Our experiments show that there are key differences with existing approaches. The most important is how the analysis and design are carried out in a similar way to conventional software engineering without simply falling into an object oriented trend.

From current specifications we have performed automated code generation of specific parts, like the interaction model. In future work we intend to complete code generation with different target MAS architectures, like ZEUS or JADE. Another goal is to include knowledge from experts in different agent fields in the methodology, such as real time or mobility.

6. References

[1] Bellifemine, F., Poggi, A., and Rimassa, G. *JADE - A FIPA-compliant Agent Framework*. The Practical Application of Intelligent Agents and Multi-Agents. 1999.

[2] Caire, G., Leal, F., Chainho, P., Evans, R., Garijo, F., Gomez-Sanz, J. J., Pavon, J., Kerney, P., Stark, J., and Massonet, P., *Agent Oriented Analysis using MESSAGE/UML*, in Wooldridge, M., Weiss, G., and Cianciarini, P. (eds.) *Agent-Oriented Software Engineering II* Springer Verlag, 2001.

[3] Caire, G., Leal, F., Chainho, P., Evans, R., Garijo, F., Gomez-Sanz, J. J., Pavon, J., Kerney, P., Stark, J., and Massonet, P. *Eurescom P907: MESSAGE - Methodology for Engineering Systems of Software Agents*. http://www.eurescom.de/public/projects/P900-series/p907/default.asp . 2002.

[4] DeLoach, S. *Analysis and Design using MaSE and agentTool.*. Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conferece (MAICS). 2001.

[5] Ferber, J. and Gutknecht, O. *A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems*. Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS98), IEEE CS Press. 1998.

[6] Garijo, F., Gomez-Sanz, J. J., and Massonet, P. *Multi-Agent System Organization. An Engineering Perspective.*. MAAMAW 2001, Springer Verlag. 2001

[7] Iglesias, C., Mercedes Garijo, M., Gonzalez, J. C., and Velasco, J. R., *Analysis and design of multiagent systems using MAS-CommonKADS*, in Singh, M. P., Rao, A., and Wooldridge, M. J. (eds.) *Intelligent Agents IV* LNAI Volume 1365 ed. SpringerVerlag: Berlin, 1998.

[8] IntelliOne Technologies. *AgentBuilder*. http://www.agentbuilder.com . 2002.

[9] Jacobson, I., Rumbaugh, J., and Booch, G., *The Unified Software Development Process* Addison-Wesley, 1999.

[10] Lyytinen, K. S. and Rossi, M. *METAEDIT+ --- A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment*. LGNS#1080.. Springer-Verlag. 1999

[11] Maes, P., *Agents that Reduces Work and Information Overload.*, *Readings in Intelligent User Interfaces* Morgan Kauffman Publishers, 1998.

[12] Newell, A., *The knowledge level*, *Artificial Intelligence*, vol. 18 pp. 87-127, 1982.

[13] Nwana, H. S., Ndumu, D. T., Lee, L. C., and Collis, J. C., *ZEUS: A Toolkit for Building Distributed Multi-Agent Systems*, *Applied Artificial Intelligence Journal*, vol. 1, no. 13, pp. 129-185, 1999.

[14] Ricordel, P. M., *Programmation Orientée Multi-Agents , Développement et Déploiement de Systèmes Multi-Agents Voyelles*. Institut National Polytechnique de Grenoble, 2001.

[15] Wood, M. and DeLoach, S. *Developing Multiagent Systems with agentTool*. 2000. ATAL 2000. LNAI 1986. Castelfranchi, C. and Lespérance, Y.

[16] Wooldridge, M., Jennings, N. R., and Kinny, D., *The Gaia Methodology for Agent-Oriented Analysis and Design*, *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 15. 2000.