Computing and Informatics, Vol. 30, 2011, 1225–1246

A SECURE GROUP-ORIENTED FRAMEWORK FOR INTELLIGENT VIRTUAL ENVIRONMENTS

Jose M. Such, Juan M. Alberola
Antonio Barella, Ana Garcia-Fornes

Departament de Sistemes Informàtics i Computació
Universitat Politècnica de València
Camí de Vera s/n. 46022, València
e-mail: {jsuch, jalberola, tbarella, agarcia}@dsic.upv.es

Communicated by Jacek Kitowski

Abstract. In this paper a multiagent system for intelligent virtual environments using the Magentix Multiagent Platform is presented. It is based on a previous framework which has been improved with agent groups, security and efficiency concerns. Therefore, the framework presented can be used in common Intelligent Virtual Environment domains such as education, commercial games and simulation.

Keywords: Intelligent virtual environments, multiagent systems

1 INTRODUCTION

As stated in [10], Multiagent Systems (MAS) provide strong models for representing complex and dynamic real-world environments. Typical application areas include the simulation of economies, societies, biological environments, medical diagnosis systems and city traffic management, etc. (refer for example to [13] and [6]). On the MAS research scene there are methodologies that aim to structure agent development but it may be difficult to implement theoretical designs. Hence, it is important to stress the significance of applying theoretical development improvements in real scenarios (applications).

Lastly, a new field has emerged for simulating a physical (or real) world inhabited by autonomous intelligent entities, called Intelligent Virtual Environments (IVEs) [9]. These entities have to interact in/with the virtual environment as if they

were real entities in the real world. In addition, entities and the virtual environment have to be displayed to users in an appropriate way.

JGOMAS (Game Oriented Multiagent System based on Jade [5]) is a framework for IVEs which provides an environment for developing and running intelligent agents over simulated 3D worlds [3]. JGOMAS allows agents to be run by different teams that compete with each other to achieve their own objectives as well as team objectives. One of its applications, such as the one currently being used, is the Capture the Flag (CTF) game, where agents are grouped into two teams (allies and axis). The allies must go to the axis base, capture the flag, and take it to their base, in order to win the game. The axis agents must defend their flag against the allies and, if the flag is captured, they must return it to their base.

JGOMAS has been successfully proven in an educational environment to teach Artificial Intelligence (AI) [4]. The main reason for its success is not only due to it being a suitable framework for teaching AI techniques, but also because it is an appealing 3D simulated world for the students where they are able to watch the behavior of the agents based on the AI techniques they are using.

Although JGOMAS makes intensive use of agent groups (teams), there is no explicit definition of agent groups in the JGOMAS design. Moreover, the JGOMAS architecture does not provide any support for agent groups and so JGOMAS programmers have to implement ad-hoc mechanisms in order to achieve this functionality. Also, JGOMAS is supposed to be a large system composed of thousands of agents, so a way to structure and coordinate the agents in the system is needed.

What is more, IVEs are commonly used in other domains beside education, such as commercial games or simulation. In this sense, JGOMAS currently lacks some concerns which arise in such domains, i.e., security (in the case of commercial games) and efficiency (in both commercial games and simulation domains). On the one hand, security is a key issue since an agent's incorrect or inappropriate behavior may cause non-desired effects such as money and data loss. On the other hand, efficiency is needed when building complex, open and large-scale applications with a huge number of agents with a lot of interactions with each other.

In this paper, we propose a framework for IVEs. The framework is called MGOMAS (Game Oriented Multiagent System based on Magentix) and it extends JGOMAS taking into account agent groups, security and efficiency.

The rest of the article is organized as follows. Section 2 shows an overview of JGOMAS and points out its limitations and main drawbacks. Section 3 introduces MGOMAS. Section 4 describes an example of the Capture the Flag game with MGOMAS. Section 5 presents a performance evaluation of JGOMAS versus MGOMAS. Finally, in Section 6 we present some concluding remarks and some improvements we could make as future work.

2 JGOMAS: DESIGN AND LIMITATIONS

This section presents JGOMAS, a framework that integrates a MAS and a virtual reality system as a scalable solution for IVEs. After presenting JGOMAS, we also analyze the limitations of the JGOMAS design and its main drawbacks.

2.1 JGOMAS Framework

The architecture of JGOMAS, as shown in Figure 1, is composed of three subsystems:

A Multiagent System (MAS). There are two kinds of agents: *simulation controller* and *inhabitant agents*. Simulation controller is in charge of keeping the virtual environment's data, maintaining the consistency at any given time, while the inhabitant agents simulate beings (humans, animals, etc.) situated in the virtual world. These agents move, look, listen, etc. in the virtual scenario. Furthermore, they can communicate with each other to achieve their goals. Thus, an inhabitant agent interacts with other inhabitant agents and with the scenario. As a result, the virtual world can be changed. The simulation controller generates events for inhabitant agents involved in world's changes.

A Visualization Module (VM). One of the JGOMAS' main goals is for artificial intelligence and virtual reality systems to work independently. In fact, it is possible to run the JGOMAS MAS even if there are no graphic viewers connected. In order to make things easier for IVE designers, a basic graphic viewer displaying the 3D agents, objects, and the scenario in JGOMAS has been implemented. This viewer is composed of renders which are in charge of displaying the information of the JGOMAS world.

A Multiagent Platform (MAP). This simplifies the implementation of MASs through a middleware, so that an IVE designer only has to implement the intelligence of his/her agents, thus not wasting time on low-level technical issues, such as inter-agent communication. Jade [5] is the MAP of JGOMAS.

2.2 Limitations of JGOMAS

The JGOMAS MAS subsystem only takes into account individual agents (simulation controller and inhabitant agents). There is no means to somehow organize the agents. JGOMAS is supposed to be a large system composed of thousands of agents, so a way to structure and coordinate the agents in the system is needed. Indeed, JGOMAS applications, like CTF, implicitly use the concept of agent groups to organize the inhabitant agents into two troops (allies and axis) but there is no definition of any kind of agent organizations in the JGOMAS MAS subsystem design.

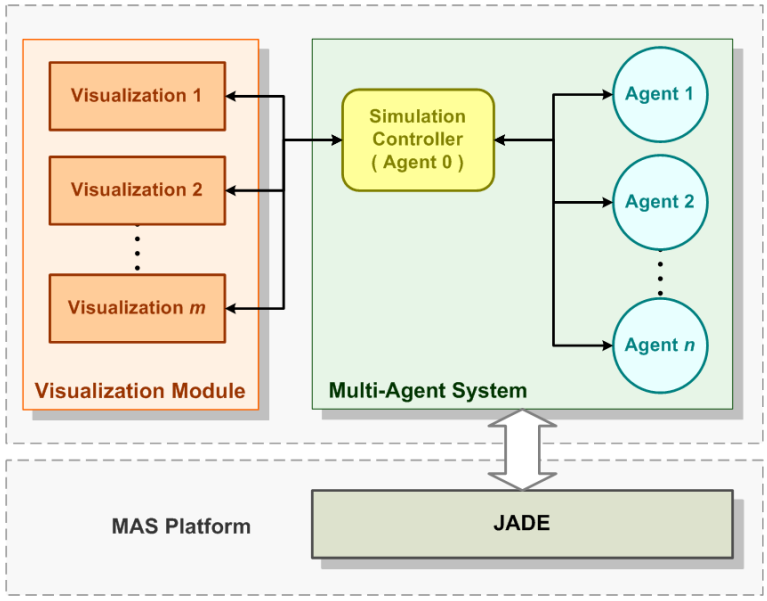

Fig. 1. Architecture of the JGOMAS framework

Apart from including the agent group concept in the MAS subsystem, some support for managing agent groups at runtime is required. This support should be provided by the MAP subsystem, which is the infrastructure that allows the execution of the MAS subsystem.

Moreover, there are other concerns regarding the MAP subsystem so that the framework can be used in other domains apart from education. In such domains, like commercial games or simulation there is a need to achieve some levels of security and efficiency.

Security is a key issue since an agent's incorrect or inappropriate behavior may cause non-desired effects such as money and data loss. Assuring the identity of the agents (authentication), preserving the integrity and confidentiality of the messages they exchange, as well as controlling access control to resources are requirements needed to prevent such non-desired effects [8]. In JGOMAS, a cheating agent can impersonate the simulation controller thereby sending messages to the rest of the agents containing incorrect or biased information about the 3D virtual world. This is actually a major drawback in online commercial games, where a cheating agent would exploit this to cheat other agents to win the game. Therefore, the new framework must include agent authentication and authorization as well as secure message exchange.

As stated in [2], agent technology demands efficiency when building complex, open and large-scale applications (such as commercial games, simulation, etc.). In

these environments the scalability, efficiency and robustness of the multiagent platform (MAP) which executes these applications become essential features.

Finally, the simulation controller agent is in charge of the 3D virtual world while the inhabitant agents simulate beings (humans, animals, etc.) situated in the virtual 3D world. All of the inhabitant agents ask the simulation controller about the world or ask it to modify their position in the world. Moreover, all of the visualization modules attach themselves to the unique simulation controller. This makes the simulation controller a single point of failure (SPOF) regarding reliability as well as a clear bottleneck regarding efficiency.

In the following section, we present a new framework, called MGOMAS, which is an evolution of JGOMAS that incorporates new functionalities in oder to address the limitations stated above. Thus, it provides the agent group abstraction, security and efficiency.

3 MGOMAS

This section presents MGOMAS that is a framework for IVEs. Figure 2 shows the architecture of the MGOMAS framework. MGOMAS is composed of two subsystems which are explained in the following sections: the Multiagent System (MAS) subsystem and the Multiagent Platform (MAP) subsystem.

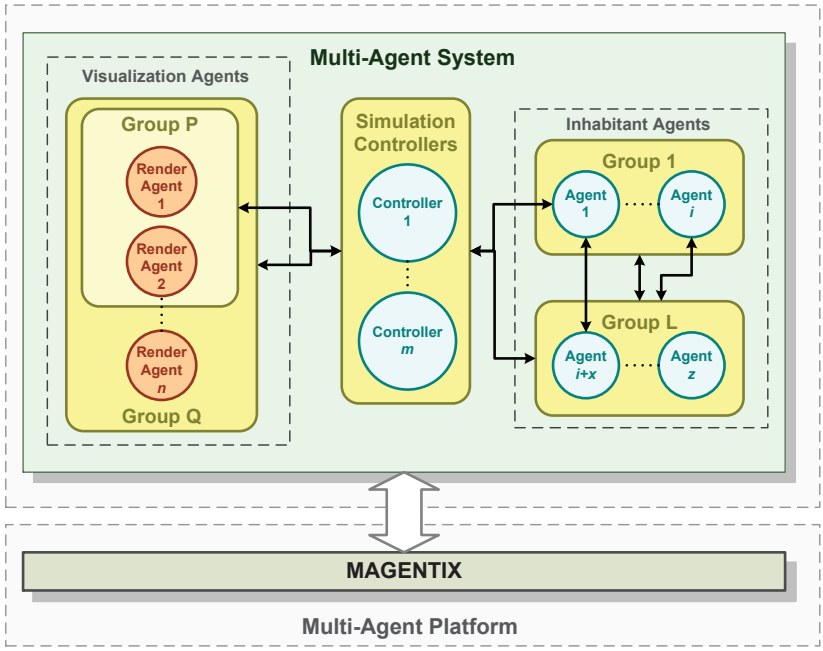

Fig. 2. Architecture of the MGOMAS Framework

3.1 MGOMAS MAS subsystem

The MGOMAS MAS subsystem extends the JGOMAS MAS subsystem by including the concept of agent group (from this moment on, group). In this sense, the MGOMAS MAS subsystem is made of two concepts, i.e., agents and groups. As stated in Section 2, there are two kinds of agents: simulation controller and inhabitant agents. With MGOMAS we add the possibility of organizing these agents into groups. Moreover, visualization agents are added to the MGOMAS MAS to display the information of the 3D world.

3.1.1 Inhabitant Agents

Inhabitant agents have a set of basic predefined behaviors regarding their physical model and intelligence. The developer of applications using MGOMAS is in charge of modifying these behaviors or adding new ones in order to implement her application. Therefore, in MGOMAS there is always a set of inhabitant agents IA that is defined as follows:

$$IA = \{a_1, \ldots, a_n\}.$$

The number N of inhabitant agents is defined by the developer who also programs the inhabitant agent's behavior to fit the final application.

The MGOMAS MAS subsystem allows agents to organize themselves into agent groups. These groups can be both static and dynamic, i.e., they can be specified before running the application (by the developer) and they can also emerge at runtime when an agent (or a group of agents) decides to create it. Agent groups can also be composed not only of agents but also of nested groups. Therefore, a group G is defined as:

$$G \subseteq IA \cup \{G_1, G_2, \ldots\}$$

where G_i are groups as well, and IA is the set of inhabitant agents.

Apart from structuring purposes, both agent to group and group to group interactions can be defined. In this sense, an agent group can be seen as a blackbox from the point of view of agents outside the group, i.e., an agent interacts in a transparent way with both agents and groups. To achieve this functionality, an agent group G has a set of *representative* agents denoted as R_G that is defined as:

$$\forall x \in R_G, x \in G \wedge x \in IA.$$

Therefore, agents defined as *representative* (members of R_G) must be members of the group (G) which they are representative agents of and must be inhabitant agents (members of IA). A group cannot be defined as *representative* of other groups.

Representative agents can receive the messages addressed to the group and can send messages on behalf of the group. R_G can also be empty if the agent group does not need to interact with agents (or groups) from outside G.

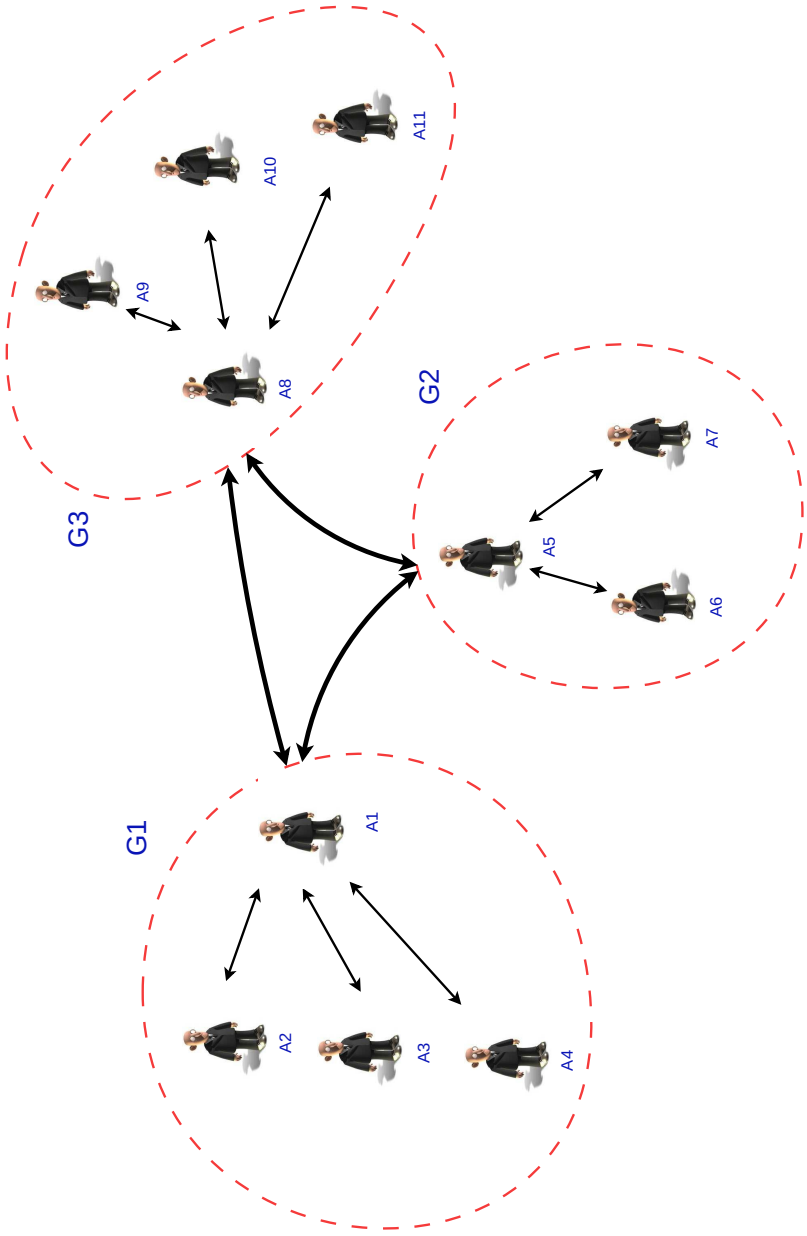

Fig. 3. Federation in the MGOMAS MAS

By means of defining the structure (the members of the group) and the interactions among agents and groups (the representative agents of the group), agent groups are shaped as some well-known agent organizations [7], such as *teams*, *coalitions*, *hierarchies*, *federations*, etc.

For example, Figure 3 shows a federation in the MGOMAS MAS subsystem. A federation is composed of groups of agents which have ceded a certain amount of autonomy to a single delegate which represents the group. The set of inhabitant agents in our example is:

$$IA = \{a_1, a_2, \ldots, a_{11}\}.$$

There are three groups (G_1, G_2, G_3) which compose the federation G_{fed} so that each group is a federate:

$$\begin{aligned} G_{fed} &= \{G_1, G_2, G_3\} \\ G_1 &= \{a_1, a_2, a_3, a_4\} \\ G_2 &= \{a_5, a_6, a_7\} \\ G_3 &= \{a_8, a_9, a_{10}, a_{11}\}. \end{aligned}$$

In each group there is a *representative* agent which represents the group. The other members of each group only interact with the *representative* agent of the group which acts as an intermediary between the group and the outside world. In this example, the *representative* agents are:

$$\begin{aligned} R_{G_{fed}} &= \emptyset \\ R_{G_1} &= \{a_1\} \\ R_{G_2} &= \{a_5\} \\ R_{G_3} &= \{a_8\}. \end{aligned}$$

$R_{G_{fed}}$ is empty because this group has no interaction with the outside world, so a *representative* agent for this group is not needed.

3.1.2 Simulation Controllers

In MGOMAS, we take advantage of the agent group concept presented above. In this sense, in MGOMAS there are more than one simulation controller agents that are grouped in a special group: the simulation controller group (SCG). We define SCG as:

$$SCG = \{c_1, \ldots, c_m\}$$

where c_i are simulation controller agents and M is the number of simulation controller agents. Moreover, we define the set of *representative* agents of SCG, denoted as R_{SCG}, as $R_{SCG} = SCG$. Therefore, all of the simulation controller agents c_i that make up the SCG are also *representative* agents of SCG. As a result, SCG is shaped as a team of agents where all of the simulation controller agents are equally important and can act on behalf of SCG.

The *SCG* group can be viewed as a wrapper of the virtual environment, because it translates all of the inhabitant agents' actions to the virtual world. It ensures that these actions follow specific rules, maintaining consistency of the virtual world. It also provides all of the necessary information for the graphic viewers. Both inhabitant agents and visualization agents interact in a transparent way with this group when asking/modifying the 3D virtual world. In the end, these requests are dealt with by the one of the simulation controller agents that make up the simulation controller group. Therefore, we are distributing the load among the simulation controller agents, so that efficiency is improved. Moreover, if one of the simulation controller agents fails, inhabitant agents and visualization agents will not be aware of that and the application will keep on running seamlessly, so that reliability is also improved.

The number of simulation controllers M is expected to be chosen by the framework user and is a parameter that must be defined at framework start-up. Depending on the domain of the final application, M can vary a lot. For instance, M will be small if we are using MGOMAS to implement an example in a class of AI, but it will be huge if we are implementing an online commercial game with a huge number of users playing it.

3.1.3 Visualization Agents

In order to make things easier for IVE designers, a basic graphic viewer displaying the 3D agents, objects, and the scenario in MGOMAS has been implemented. This viewer is composed of visualization agents which are in charge of displaying the information of the MGOMAS world. Visualization agents are render applications "agentified", i.e., they are able to communicate with other agents of the MGOMAS MAS subsystem. Using these communicative skills they can ask the information about the 3D virtual world to the SCG group. Visualization agents do not have intelligent capabilities, they only display the 3D virtual world so that the artifical intelligence and virtual reality systems work independently.

Visualization agents can also group themselves into groups. In this sense, there is ongoing work to take advantage of this feature. The main idea is, in a nutshell, to divide the 3D virtual world into sections, each one managed by a simulation controller agent. Then, several visualization agents (each one acting as a camera), grouped depending on the section they are displaying, interact with the simulation controller agent in charge of that section to obtain the information needed. Moreover, other cameras (visualization agents) interact with more than one visualization controller agents to act as a bird's-eye view camera.

3.2 MGOMAS MAP subsystem

In some previous studies, we identified a lot of MAPs that have been developed over the last few years in the agent technology research field. However, each MAP offers its own set of features for developing and executing MAS. In this way, it is

necessary to choose from among them a suitable MAP that fulfills the MGOMAS requirements for this subsystem. We have previous MAP evaluation experience, in [11] for example, and this has helped us in this study.

All of the requirements for the MGOMAS MAP subsystem are summarized below:

Communication. The agents must be able to interact with each other by sending and receiving messages.

Agent Groups. In the design of MGOMAS there are agent groups. In order for the programmers to use this concept, the MAP subsystem would provide some support so that the programmers do not have to implement ad-hoc mechanisms for grouping agents.

Security. In a real scenario, in which multiple players may take part in the game, a method for avoiding security flaws and cheating agents is required.

Efficiency. In domains such as commercial games or simulation the efficiency of the resulting application is crucial.

Open Source. MGOMAS is a framework intended to be open source (as JGOMAS is), so it is important that all of the subsystems are open source as well.

All of the MAPs studied obviously provide some kind of mechanism to allow agents to *communicate* with each other by sending and receiving messages.

Regarding *agent groups*, *security* and the *code openness*, Table 1 sums up all of the MAPs studied and whether they provide these three features (MAPs offering a feature are marked with $\surd$ while MAPs not offering it are marked with –).

Although there are many MAPs available, only a few of them provide agent groups, security, and are open source. In this sense, only Magentix [2] and Zeus [12] are suitable to be the MAP subsystem for MGOMAS.

As MGOMAS is aimed at being used in real scenarios, *efficiency* and scalability also become major concerns. As stated in [14], Zeus performs worse when exchanging messages compared with other MAPs while Magentix is a MAP aimed at being efficient and scalable, as shown in [2]. Therefore, Magentix MAP is chosen to be the MAP subsystem for MGOMAS, replacing the Jade MAP used in JGOMAS.

3.2.1 Magentix as the MAP Subsystem

Magentix[1] architecture focuses on offering good performance and scalability especially when running large systems. Apart from being a high-efficient and scalable MAP, Magentix provides a secure group-oriented model (explained in-depth in [15]). This model is based on the concept of organizational units and giving identities to all of the entities in the MAP.

[1] Magentix can be downloaded from `http://www.dsic.upv.es/users/ia/sma/tools/Magentix/index.html`.

Platform	**Agent Groups**	**Security**	**Open Source**
3APL	–	–	√
AAP	–	–	√
ABLE	–		
ADK	–	√	–
AgentBuilder	–		–
AgentScape	–	√	√
AgentOS	–	√	–
Aglets	–	√	√
AGlobe	–	–	√
Ajanta	–	√	√
Ara	–	√	√
Bond	–	–	√
CAPA	–	–	–
CapNet	–	√	–
Concordia	–	√	–
Cougaar	–	√	√
CrossBow	–	√	
Cybele	–		√
Decaf	–		√
Genie	–	–	√
Grasshopper	–	√	–
Gypsy	–	√	√
Hive	–	√	√
Jade	–	√	√
Jack	√	√	–
Jackal	√	–	√
Jacomma	–		√
Jat	–		–
Mage	–		√
MadKit	√	–	√
Magentix	√	√	√
Sage	–	√	√
Semoa	–	√	√
Spyse	–		√
Voyager	–	√	–
Zeus	√	√	√

Table 1. Platform functional analysis

Organizational units are an advanced message routing mechanism. In each unit there is a *manager* that creates the unit and decides which agents can join as *members* of that unit. Then, the *manager* of the unit also decides which of the *members* can send and receive messages on behalf of the unit. The agents allowed to communicate on behalf of the unit are called *contact* agents. The *manager* also defines the routing type of the unit, i.e., the way that *contact* agents will receive messages sent to the unit. There are five routing types:

Unicast: The messages addressed to the unit are delivered to a single *contact* agent.

Multicast: Several *contact* agents can be appointed to receive messages. When a message is addressed to the unit, this message is delivered to any contact agent in the unit.

Round Robin: There can be several *contact* agents appointed to receive messages. Each message addressed to the unit is delivered to a different *contact* agent, defined according to a circular policy.

Random: Several agents can be defined as *contact* agents. Each message addressed to the unit is delivered to a randomly selected *contact* agent.

Sourcehash: Several agents can be the *contact* agents. Each message addressed to the unit is delivered to one of the agents responsible for receiving messages according to a hash function about a property of the message sender (e.g. the host where the message sender is situated).

A direct map can be done from agent groups in the MAS subsystem to the Magentix organizational units in the MAP subsystem. In this sense, the agent groups can be implemented as organizational units where the set of *representative* agents of the MGOMAS MAS can be mapped to *contact* agents of the organizational unit. Moreover, the routing type of the organizational unit can be deduced from the agent organization that the agent group is shaping.

For example, let $H = \{a_1, a_2, a_3\}$ be an agent group of the MAS subsystem that is shaping a hierarchy and a_1 be the *representative* agent of the group ($R_H = \{a_1\}$). Therefore, a_1 is the supervisor of the hierarchy (the only one that can act on behalf of H) while a_2 and a_3 are subordinates.

We implement H using a Magentix organizational unit as follows. a_1 is both the *manager* and the unique *contact* agent of the unit. Moreover, the routing type of the unit must be unicast. Therefore, a_1 controls what agents can join the unit (and throw them out if they do not act as expected). It is also the unique entity allowed to act on behalf of the unit when interacting with other agents or units.

Regarding security concerns, with the use of Magentix the fact that any agent in the system can act on behalf of another is avoided. Magentix provides a security model that is based on identities that are assigned to all of the different entities that can be found in a Magentix MAP, i.e., users, agents, services and organizational units. The Kerberos[2] authentication protocol is used to authenticate the identities.

[2] `http://web.mit.edu/Kerberos/`

Using this model, when an agent receives a message from a unit, this agent does not have to check which agents are able to send messages using the unit identifier, because this identity is sure to be used by an authorized agent. For instance, in the previous example, what prevents a_2 from acting on behalf of the a_1? It could impersonate a_1 and order a_3 to do an action. What is more, what prevents a_2 from acting on behalf of the H group? It could interact with other agents or groups in the system on behalf of H. All of these actions are avoided by the security model of Magentix.

The authentication of these identities by any entity of the MAP is the basis for achieving the other desired features (integrity, confidentiality and authorization). In this sense, all of the information that the agents exchange by means of messages is assured by signing (integrity) and ciphering (confidentiality) all of the messages exchanged. Thus, this information cannot be tampered with or disclosed by unauthorized agents. Moreover, the security model allows access to resources (such as files, databases, and the like) to be granted or denied, depending on the identity of the entity trying to access the resource by means of access control lists (ACLs).

4 CAPTURE THE FLAG

A Capture the Flag (CTF) game involves agents that are grouped into two teams (allies and axis). The allies must go to the axis base, capture the flag, and take it to their base, in order to win the game. The axis agents must defend their flag against the allies and, if the flag is captured, they must return it to their base. There is a time limit for the allies to bring the flag to their base. If the time limit expires, the axis team wins the game.

There are three kinds of agents which provide specific services: Soldiers are the main agents in the game, Medics heal Soldiers and FieldOps provide Soldiers with munitions.

Figure 4 shows a screen capture of the MGOMAS framework executing a CTF game. There are three visualization agents displaying the 3D world. There is also the GUI of the Magentix MAP which shows the agents and the groups running in the framework.

In the remainder of this section, we focus on detailing how inhabitant agents in the CTF game (Soldiers and the like) are grouped using the MGOMAS framework. To this aim, we detail an example of the CTF game in which the Allies and the Axis groups are composed of five Soldiers, one Medic and one FieldOps each one.

The Allies group is defined as a two-level hierarchy shown in Figure 5. It is composed of the agent sold1 and the simple hierarchies Soldiers and Support. The supervisor (the unique *representative* agent) of Allies is the agent sold1. Using the MGOMAS MAS notation for inhabitant agents, we specify the Allies, Soldiers and Support groups as follows:

$$\begin{aligned} \text{Allies} &= \{\text{sold1}, \text{Soldiers}, \text{Support}\} \\ R_{\text{Allies}} &= \{\text{sold1}\} \end{aligned}$$

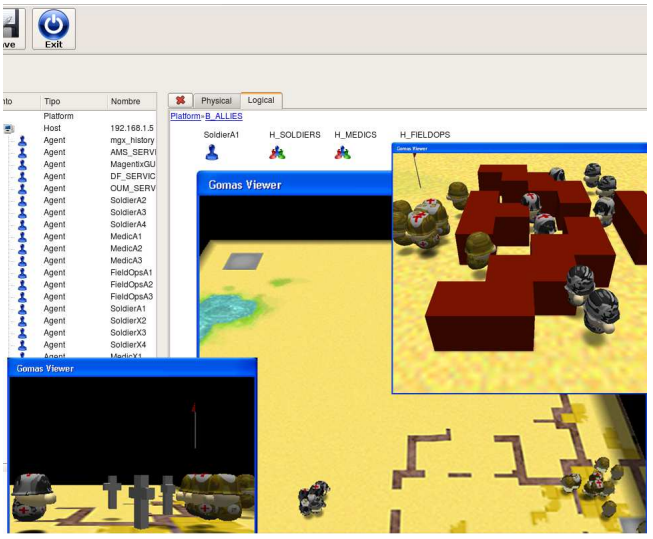

Fig. 4. CTF in the MGOMAS framework

$$\begin{aligned} \text{Soldiers} &= \{\text{sold2}, \text{sold3}, \text{sold4}\} \\ R_{\text{Soldiers}} &= \{\text{sold2}\} \\ \text{Support} &= \{\text{sold5}, \text{med1}, \text{field1}\} \\ R_{\text{Medics}} &= \{\text{sold5}\} \end{aligned}$$

As explained in the previous sections, agent groups are directly mapped to Magentix units in order to carry out their implementation. Therefore, in some point of the code of agent sold2, it has to create the corresponding Magentix Soldiers group using the Magentix API for units as follows:

```
mgx_new_unit("Soldiers",UNICAST);
mgx_new_member("Soldiers","sold2");
mgx_new_member("Soldiers","sold3");
mgx_new_member("Soldiers","sold4");
mgx_new_contact_agent("Soldiers","sold2");
```

Note that the Soldiers unit is created with the routing type `UNICAST` and sold2 is defined as the unique `contact_agent` so that sold2 is the unique agent allowed to interact on behalf of Soldiers. Similar code is needed in agent sold5 in order to create the Support group. Finally, the code for creating the Allies group is in the sold1 agent and is as follows:

```
mgx_new_unit("Allies",UNICAST);
mgx_new_member ("Allies","sold1");
mgx_new_member ("Allies","Soldiers");
```

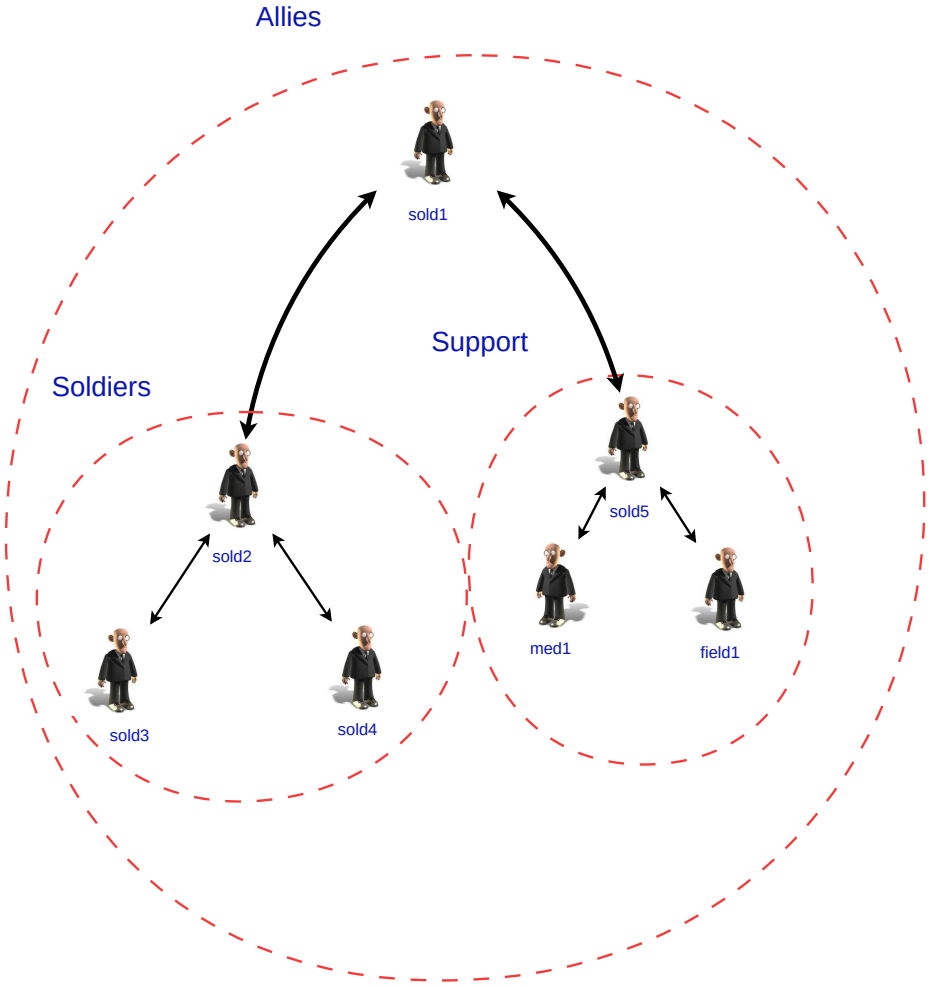

Fig. 5. Allies team

```
mgx_new_member ("Allies","Support");
mgx_new_contact_agent ("Allies", "sold1");
```

Axis group is defined as a team (Figure 6) so that all of its members are equally important and can interact with each other. The MGOMAS MAS notation for the Axis group is:

$$\begin{aligned} \text{Axis} &= \{\text{sold6}, \text{sold7}, \text{sold8}, \text{sold9}, \text{sold10}, \text{med2}, \text{field2}\} \\ R_{\text{Axis}} &= \{\text{sold6}, \text{sold7}, \text{sold8}, \text{sold9}, \text{sold10}, \text{med2}, \text{field2}\} \end{aligned}$$

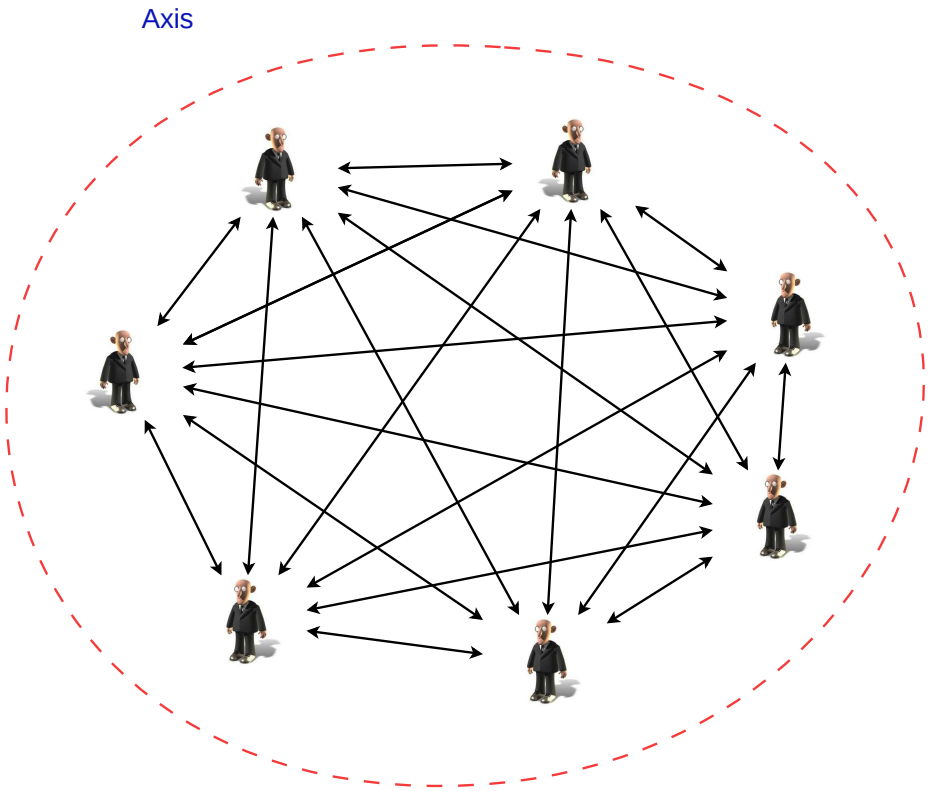

Fig. 6. Axis team

In order to carry out the implementation of the Axis group, one of the agents that make up the group must create the corresponding Magentix unit. In our example, sold6 is in charge to create the Axis group as follows:

```
mgx_new_unit("Axis",MULTICAST);
mgx_new_member ("Axis","sold6");
mgx_new_contact_agent ("Axis", "sold6");
                    .
                    .
                    .
mgx_new_member ("Axis","field2");
mgx_new_contact_agent ("Axis", "field2");
```

Note that Axis is defined with the routing type `MULTICAST`, and all of the agents of Axis are also defined as `contact_agents`.

It is worth mentioning that, at runtime, the MGOMAS framework assures the message exchanges and the identities of both agents and groups. As the agent and

group identities are assured, an agent can avoid cheating agents that try to act as members of the allies team but are really members of the axis team. The message exchanges are also secured, assuring their confidentiality and integrity. In this sense, MGOMAS avoids that agents from a team (allies/axis) can overhear conversations of agents from the other team (axis/allies).

By assuring identities and message exchanges, the CTF game implemented on top of MGOMAS can be used as an online game with multiple players each one acting as an inhabitant agent (Soldier, Medic or FieldOps) avoiding cheating agents. This is a key feature for online games, because cheating agents could discredit the game, discouraging users from playing it.

5 PERFORMANCE EVALUATION

In the previous sections, we present how the agent group concept is incorporated into the framework and also how security concerns are taken into account. In this section, we present a performance evaluation in order to assess the fulfillment of the efficiency requirement. We present a set of tests in order to measure the efficiency and scalability of both MGOMAS and JGOMAS frameworks. We have previous experiences in evaluating the efficiency and scalability of MASs (refer to [11] and [1]).

We first check how the frameworks perform when the number of agents in the system is increased (simulating an on-line game with a big amount of players). In this sense, we launch the allies and the axis teams with the goal of exchanging a big amount of messages (a lot of interactions among agents). We increase the number of agents of each team so that both the number of agents and the messages exchanged increase. Each agent (being Medic, Soldier or FieldOp) communicates with the rest of the agents of the group in a sequential way. Each agent sends and receives m messages. Moreover, if an agent receives a message from other agent, it answers the message. We measure the time elapsed between the first message is sent by the first agent and the last message is received by the last agent. We start the experiment with 100 agents in the system (50 agents in each group) and we increase this number up to 1 000. The number of messages sent by each agent is specified to $m = 1\,000$.

In Figure 7 we can see the time of both frameworks. Actually, there is a performance degradation as the number of agents (and the message traffic) are increased. However, MGOMAS performance degrades less than JGOMAS performance. As an example, we can see that the elapsed time in MGOMAS when the system is composed of 1000 agents is less than the elapsed time in JGOMAS when the system is composed of 200 agents. Furthermore, as we increase the number of agents, we can see that the time difference between both frameworks increases. Therefore, we can conclude that MGOMAS framework is more scalable than JGOMAS.

Another typical scenario is the massive message sending to a specific agent. In this second test, we measure the ability of the frameworks when a lot of agents send messages to a single one. This specific agent could become a bottleneck in

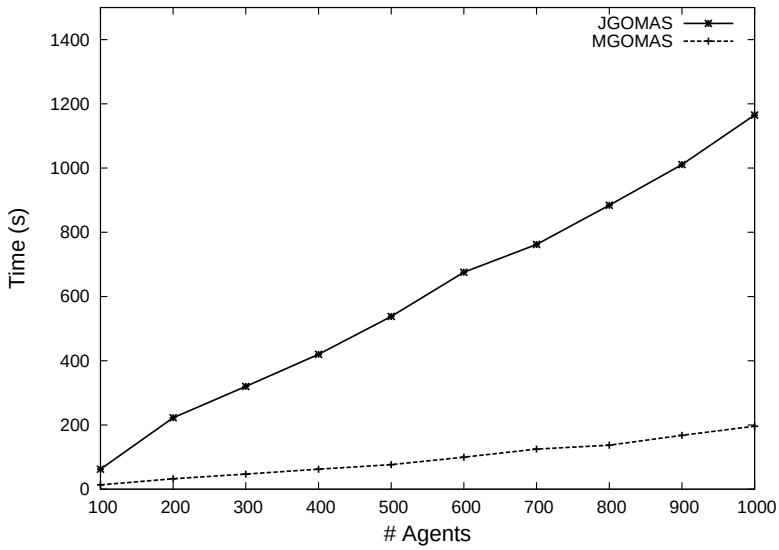

Fig. 7. Test 1

the system when multiple messages are addressed to it. This scenario appears, for example, when soldiers are requesting the same medic agent to heal them. This agent has to serve every received request. As the number of incoming requests is increased, the time for processing these requests may also increase.

In order to simulate this situation, we present a test in which a medic agent is requested by a lot of agents of the same team. In this test, we launch a single medic agent and n soldier agents whose goal is to send m messages to the medic agent. The elapsed time between when the Medic agent receives the first message and when she answer the $n \times m$ messages can be seen in Figure 8. In this experiment we launch an allies team composed by the Medic and one Soldier agent ($n = 1$) and we increase the number of Soldier agents up to 6. Each soldier agent sends an amount of 10 000 messages ($m = 10\,000$) to the Medic agent.

We can see that the elapsed time increases in both frameworks as the number of requests increases. However, as in the previous test, we can see that the performance degradation is less in the MGOMAS framework. The time difference between both frameworks gradually increases as we increase the number of agents. Therefore, MGOMAS framework is also more scalable and efficient than JGOMAS in this scenario.

With the results provided in these tests, we can conclude MGOMAS improves efficiency and scalability provided by JGOMAS. In these tests, we have simulated two typical scenarios in order to check both features in the MGOMAS and the JGOMAS frameworks. These tests represent critical situations so that we can see more clearly the degree of performance improvement achieved.

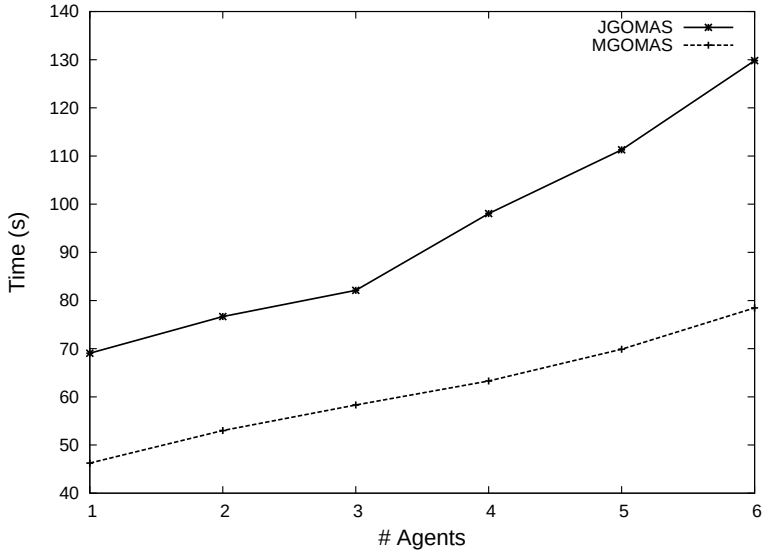

Fig. 8. Test 2

6 CONCLUSIONS

In this paper we present a framework called MGOMAS which is a framework for IVEs. MGOMAS extends JGOMAS taking into account agent groups, security and efficiency. Design of MGOMAS explicitly defines agent groups which can be used both at the application and programmer levels. We have shown the benefits of adding the group perspective into the framework: to simplify the simulation controller functionality, to help programmers structure their battalions, and of course, for managing the different teams in the game. Authentication, secure message passing and efficiency are also important features for widening the framework application. That is, to prevent inappropriate agent behaviors or cheating agents and to build large applications.

As far as we are concerned, MGOMAS design and architecture can be used in a broad range of IVE applications, from educational purposes to simulation and real scenarios such as commercial games. It can also be suitable to any other domain in which there is a large and complex system that requires some organization structures (that simplify the problem) and where security and efficiency also matter.

We also present a performance evaluation comparing MGOMAS and JGOMAS frameworks. By means of two tests we check the efficiency and scalability of both frameworks. We conclude that MGOMAS outperforms JGOMAS.

There is ongoing effort in order to also include norms in the MAS subsystem, and its corresponding support mechanisms in the MAP subsystem, so that the behavior of the members of a group can be restricted.

There is also ongoing work in order visualization agents to take advantage of agents groups when displaying the 3D virtual world.

Acknowledgement

This work has been partially supported by Consolider-Ingenio 2010 under grant CSD2007-00022, and projects TIN2008-04446 and PROMETEO/2008/051. Jose M. Such has received a grant from Conselleria d'Empresa, Universitat i Ciència de la Generalitat Valenciana (BFPI06/096). Juan M. Alberola has received a grant from Ministerio de Ciencia e Innovación de España (AP2007-00289).

REFERENCES

[1] Alberola, J. M.—Such, J. M.—Espinosa, A.—Garcia-Fornes, A.—Botti, V.: A Performance Evaluation of Three Multi-Agent Platforms. Artificial Intelligence Review, Vol. 34, 2010, No. 2, pp. 145–176.

[2] Alberola, J. M.—Such, J. M.—Espinosa, A.—Botti, V.—Garcia-Fornes, A.: Scalable and Efficient Multi-Agent Platform Closer to the Operating System. Artificial Intelligence Research and Development, Vol. 184, 2008, pp. 7–15.

[3] Barella, A.—Carrascosa, C.—Botti, V.: Agent Architectures for Intelligent Virtual Environments. In 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, pp. 532–535, IEEE 2007.

[4] Barella, A.—Valero, S.—Carrascosa, C.: JGOMAS: New Approach to AI Teaching. IEEE Transactions on Education, 2008.

[5] Bellifemine, F.—Poggi, A.—Rimassa, G.: JADE: A FIPA2000 Compliant Agent Development Environment. In AGENTS '01: Proceedings of the fifth international conference on autonomous agents, ACM Press 2001, New York, pp. 216–217.

[6] Burguillo, J.-C.—Rodriguez, P. S.—Costa, E.—Gil, F: History-Based Self-Organizing Traffic Lights. Computing and Informatics, Vol. 28, 2009, No. 2, pp. 157–168.

[7] Horling, B.—Lesser, V.: A Survey of Multi-Agent Organizational Paradigms. The Knowledge Engineering Review, Vol. 19, 2004, pp. 281–316.

[8] Longstaff, T.—Ellis, J.—Shawn, H.—Lipson, H.—Mcmillan, R.—Pesante, H.—Simmel, D.: Security of the Internet. The Froehlich/Kent Encyclopedia of Telecommunications, Vol. 15, 1997, pp. 231–255.

[9] Luck, M.—Aylett, R.: Applying Artificial Intelligence to Virtual Reality: Intelligent Virtual Environments. Applied Artificial Intelligence, Vol. 14, 2000, No. 1, pp. 3–32.

[10] Luck, M.—Mcburney, P.—Shehory, O.—Willmott, S.: Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing). AgentLink, 2005.

[11] Mulet, L.—Such, J. M.—Alberola, J. M.: Performance Evaluation of Open Source Multiagent Platforms. In Proceedings of the Fifth International Joint

Conference on Autonomous Agents and Multi-agent Systems (AAMAS 06), 2006, pp. 1107–1109.

[12] Nwana, H. S.—Ndumu, D. T.—Lee, L. C.—Collis, J. C.—Re, I. I.: Zeus: A Tool-Kit for Building Distributed Multi-Agent Systems. Applied Artificial Intelligence Journal, Vol. 13, 1999, pp. 129–186.

[13] Paunovski, O.—Eleftherakis, G.—Cowling, A.: Disciplined Exploration of Emergence Using Multi-Agent Simulation Framework. Computing and Informatics, Vol. 28, 2009, No. 3, pp. 369–391.

[14] Rollo, M.—Pechoucek, M.: A-Globe: Agent Platform with Inaccessibility and Mobility Support. In Cooperative Information Agents VIII, No. 3191, LNAI, Springer-Verlag 2004.

[15] Such, J. M.—Alberola, J. M.—Espinosa, A.—Garcia-Fornes, A.: A Group-Oriented Secure Multiagent Platform. Software: Practice and Experience (to appear).

Jose M. Such is a Ph. D. candidate at the Departament de Sistemes Informàtics i Computació of the Universitat Politècnica de València. His interest areas include privacy, trust, reputation, and security in multiagent systems as well as multiagent platform design, benchmarking, and performance evaluation.

Juan M. Alberola is a Ph. D. student at the Departament de Sistemes Informàtics i Computació of the Universitat Politècnica de València. His interest areas include organizations and reorganization in multiagent systems, prediction markets, case-based reasoning and multiagent platforms design.

Antonio Barela is a Ph. D. student at the Departament de Sistemes Informàtics i Computació of the Universitat Politècnica de València. His interest areas include multiagent systems, intelligent virtual environments and 3D simulation. He has collaborated in teaching practical work in various advanced AI subjects at the UPV.

Ana Garcia-Fornes is a Professor at the Departament de Sistemes Informàtics i Computació of the Universitat Politècnica de València. Her interest areas include: real-time artificial intelligence, real-time systems, development of multiagent infrastructures, tracing systems, operating systems based on agents and negotiation strategies.