Using Natural Interfaces for Human-Agent Immersion

Angel Sanchis[1], Vicente Julián[1], Juan M. Corchado[2], Holger Billhardt[3], and Carlos Carrascosa[1]

[1] Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València, Spain
[2] Department of Computer Science, University of Salamanca, Spain
[3] CETINIA, Universidad Rey Juan Carlos, Spain
angel2esoc@hotmail.com, {vinglada,carrasco}@dsic.upv.es,
corchado@usal.es, holger.billhardt@urjc.es

Abstract. Multi-agent technology allows the development of current AmI applications. Specifically, a multi-agent system allows the formation and management of applications where the main components can be humans and software agents interact and communicate with humans in order to help them in their daily activities. This kind of applications are what we call a *Human-Agent Society*, where agents provide services to humans or to other agents in an environment of whole integration. This paper presents a solution for the problem of human immersion presented in this kind of systems, providing the use of natural interfaces for the interaction among humans and software agents.

Keywords: Virtual Agents, Human-Agent Societies.

1 Introduction

Ambient Intelligence (AmI) imagines a future where technology surrounds the users [1], and helps them in their daily lives. The AmI scenarios described by the Information Society Technologies Advisory Group (ISTAG) have intelligent environments capable of recognizing and responding to the presence of different individuals in a simple, unobtrusive and often invisible way [5]. AmI is heavily based on the concept of Ubiquitous Computing (UC), introduced by Weiss in the 90s, which describes a world where a multitude of computational objects interact and communicate in order to help humans in daily activities [10]. The main goal of AmI systems is to be invisible, but very useful. This raises some requirements for AmI -based [9] systems. The technology must be transparent to users, services must be adapted to the context and user preferences and the applications must provide intuitive interfaces and must be friendly to users. This situation attributes to Intelligent Systems (IS) a key role in achieving the AmI goals [8]. AmI provides distributed complex problems by applying methodologies inspired by human techniques for solving problems. That is, to provide machine learning procedures, interaction protocols, distributed communication, coordination

J.M. Corchado et al. (Eds.): PAAMS 2014 Workshops, CCIS 430, pp. 358–367, 2014.

and cooperation and adaptive behavior models to the knowledge representation formalisms of AmI.

Recent trends in AmI, based on intelligent systems, have not had much success. There are two main reasons as the cause of this failure. For one side, intelligent systems have not reached the maturity level of other information technologies, and for a long time, they have forgotten traditional industry [6]. On the other hand, it is required an interdisciplinary perspective, which is difficult, since a considerable amount of resources (scientific, economic and human) would be required.

Agent technology, although still immature in some ways, allows the development of systems that support the requirements of AmI applications. Specifically it allows the formation and management of systems where the main components can be humans and software agents providing services to humans or other agents in an environment of whole integration. This kind of applications are what we call a *Human-Agent Society*, which can be defined as a computing paradigm in which the traditional notion of application disappears. Rather than developing software applications that accomplish computational tasks for specific purposes, this paradigm is based on an immersion of the users in a complex environment that enables computation.

This paper deeps in the immersion problem of this kind of systems providing the use of natural interfaces for the interaction among humans and software agents. When a human is completely immersed into a system of this kind, the human can interact with the system using natural gestures. Moreover, agents inserted in the system can learn about human actions adapting its behaviors and taking decisions about future situations. Examples of these systems can be domotic scenarios, production lines in an industry, entertainment industry, ... where humans interact with the rest of components only moving, for example, their arms or hands. In order to show this, the paper presents a proof of concept of this kind of immersion.

The rest of the document is structured as follows. Section 2 describes what we have called *Human-Agent Societies*. Next, in Section 3, we describe JGOMAS, a framework for 3D simulated worlds where we situate our proposal. After that, Section 4 presents the proposal we have made for a *Human-Agent Society* proof of concept. Finally, the conclusions of this paper are commented in Section 5.

2 Human-Agent Societies

With the term Human-Agent Society (HAS) we refer to a computing paradigm in which the traditional notion of application disappears. Rather than developing software applications that accomplish computational tasks for specific purposes, this paradigm is based on an immersion of the users in a complex environment that enables computation. The environment itself is populated by computational entities with different capacities and intellectual properties, ranging from simple devices that offer specific capacities or rudimentary information, like screens or sensors, to autonomous artificial agents that provide high level services and are

able to engage in complex interaction protocols. This view of a system is closely related with the development of AmI applications.

When defining a HAS, it can be seen as the next evolution of Multi-agent Systems, where there is an immersion at two levels of agents and humans respectively. Thus, in the MAS level, humans are situated and integrated into the system in such a way that they appear as agents for the other agents. Whereas in the human level (from the perspective of humans) interaction is performed with objects and actions a person is accustomed. This is because the MAS is modeled as an ubiquitous system. Thus, integration of both types of features is achieved allowing a maximum level of immersion of users in the MAS, minimizing the level of difficulty of the interaction. Moreover, considering all this, the satisfaction level of the user is maximized.

The overall objective of this work is to deep in the development of Human-Agent Societies through the use of natural interfaces that allow immersion in the two previously mentioned levels.

The ability of a *Human-Agent Society* to establish connecting links and achieve goals together in a dynamic environment, allows to develop flexible and dynamic systems where individuals, by themselves, are unable to achieve the goals that emerge in a society. It is necessary to provide mechanisms and interfaces for such double immersion at MAS. To achieve this extension is necessary to have information based on the context that allows to perform a more realistic integration of the human situation, including possible actions and behaviors that are part of a human society. Furthermore, at human level, extension mechanisms should be raised allowing access to agent-based systems in an ubiquitous way.

3 JGOMAS

JGOMAS[1] [2] [4] (acronym for *Game Oriented Multi-Agent System* based on *JADE*) is a framework that develops and executes agents in 3D simulated worlds. In this framework, the social interaction to simulate is based on the rules of a "Capture the Flag" type of game: agents belong to one of two teams that compete to capture the opponent's flag. This game modality has become a standard that is included in almost all multiplayer games that have appeared since Quake.

It is very easy and intuitive to apply MAS to games of this type because each soldier can be viewed as an agent. Moreover, agents on a team must cooperate with each other to achieve the team's objective, thus competing with the other team. In fact, it is not uncommon to find applications of agent technology to the game field in general (e.g., the board game developed by Offerman et al. [7]) and to Capture the Flag in particular.

The framework allows designers to incorporate intelligence in agents that interact in a VE and to follow the evolution of these agents through an unlimited number of visualization modules in a distributed fashion.

In summary, JGOMAS is composed of two subsystems (see Figure 1):

[1] `http://jgomas.gti-ia.dsic.upv.es`

– JGOMAS MAS: This subsystem works on a JADE[2] platform. There are two different kinds of agents: Manager Agents (that control the game's logic, and interface with visualizers) and Player Agents (one for each member of each team). Player Agents can play one of three roles: soldier, medic or field operations, each of which is provided with a set of basic behaviors. JGOMAS supports interactions between Manager Agents (virtual world) and Player Agents, so that it abstracts these peculiarities to the users.
– Render Engine (RE): The RE is a 3D multi-platform graphic engine that allows users to view the evolution of JGOMAS agents in the VE, to observe how the components of each team behave and the outcome of the game. More specifically, each RE provides a single window to observe the VE containing the agents. It is also possible to have different RE configurations to satisfy different visualization needs, as explained in [3].

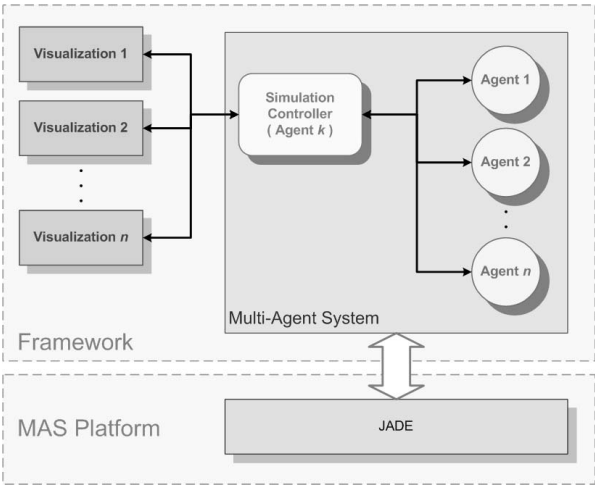

Fig. 1. JGOMAS architecture overview

3.1 Multi-agent System

There exists two main classes of agents defined in JGOMAS: the *simulation controller* and *inhabitant agents* (as *player* agents).

Simulation controller, that is also called *Manager Agent*, is in charge of keeping the virtual environment's data, maintaining the consistency at any time, along with controlling the rules of the game. On the other hand, the *inhabitant agents* simulate players (humans, animals, etc.) situated in the virtual world. These agents are moving, looking, hearing, etc. . . in the virtual scenario. Furthermore, they can comunicate each other in order to achieve their goals. Thus, an *inhabitant agent* interacts with other *inhabitant agents* and with the scenario. As a

[2] `http://jade.tilab.com/`

result, the virtual world can be changed. And the *simulation controller* generate events for those *inhabitant agents* involved in the changes.

The way an *inhabitant agent* achieves a goal is carrying out tasks. A *Task* is defined as something to carry out in an specific position in the virtual environment. So, the three main states of an *inhabitant agent* are:

- Standing: the agent has no current task to do, so he is waiting to do something.
- Go To Target: the agent has one current task that has been selected from its task list (according to tasks priority). This task has to be carried out in a different position to the one the agent is in, so it is moving to this place. While the agent is moving, it may encounter enemy agents, and react to them (aiming and shooting them).
- Perform Target Reached: the agent has reached the position where he has to carry out his current task, and so it has to make the specific actions related to the task.

There is a set of predefined tasks though the designer may change this set and / or change the actions to carry out when the target position is reached (tasks priority can also be dynamically set). This set includes, but is not limited to:

- GO TO TARGET: the agent moves to the specified position.
- GET OBJECTIVE: all the agents in the attacking team have this task since game beginning. This task makes the agents to go to the initial position of the flag in the defending base. If the agent arrives, and the flag is still there, the agent gets it and the task changes to return to base.
- PATROLLING: all the agents in the defending team have this task since game beginning. This task is not related to a position but to a set of random positions generated around the defending base. The task makes the agent to go visiting them one after the other. When the agent arrives to the last random position, it begins anew with the first one. So this task never ends, and the agent is patrolling defending its base and the flag.

There is also some tasks related to the special abilities of some of the roles agents may play in the game.

There are three different roles for *inhabitant agents* (though the user may add more if he wants it): medic, fieldops and soldier. The medic can create *medic packs* that allow to recuperate health to any *inhabitant agent* by consuming them. In the same way, the fieldops can create *ammo packs* that allow to recuperate ammo to any *inhabitant agent* by consuming them. The special ability of the soldier is that he is very skilled at shooting and he makes more damage than the rest. Any agent registers a service to offer to the other agents in his team his abilities. And each one of these abilities have a task related to carry them out. So, there exist a *GIVE MEDIC PACKS* task, a *GIVE AMMO PACKS* task and a *GIVE BACKUP* task (this last one for the soldier, to go to help their teammates).

The JGOMAS package includes the multi-agent platform, the RE, maps, documentation and a sample that is ready to use. Fig. 2 shows an execution example of this package, with JADE GUI, text console, and some instances of the RE.

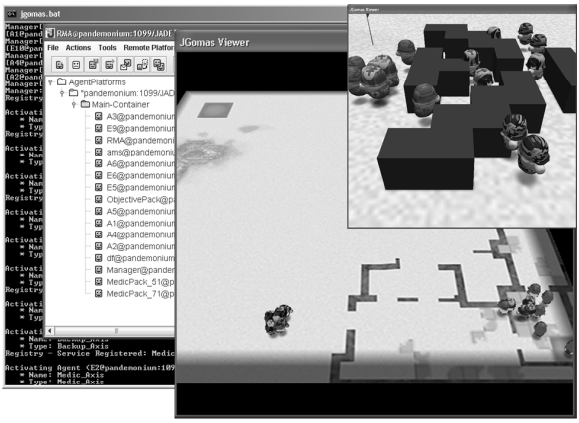

Fig. 2. JGOMAS execution example

4 HAS Case Study

4.1 Design

The present proposal supposes a proof of concept about a *Human-Agent Society*, that is, a system with a double immersion in it. The user is immersed in the system interacting with it by means of a natural interface, and the agents in the system see the user as one of them, interacting with him in the same way.

So, we have added a user to the JGOMAS system as one more of the agents that interacts with the rest of the system (environment –Manager Agent–, and the rest of agents –of their same team and of the other team–).

To have the human user immersed in the multi-agent system we have used a natural interface that allows the user to easily interact with the system. This natural interface is provided through the *Kinect*[3] sensor device.

The Kinect sensor is composed of the following elements:

- an IR emitter and an IR depth sensor: The first one emits a pattern of points over the elements in front of the sensor through a laser diode. The reflex pattern is captured by the IR depth sensor and it is sent to an inner chip to be compared with an original pattern. The result of this comparison is the depth information.
- A color sensor: It is a RGB camera with a resolution of 640x480 px at 30fps.

[3] Kinect For Windows: `http://www.microsoft.com/en-us/kinectforwindows/`

- An inclination motor: It allows an inclination of 27 degrees (positive or negative) with respect to the horizontal.
- A multi-array microphone: It is a set of four microphones allowing the sensor to capture audio and to localize the sound sources. It generates 16 bit audio to 16 khz.

The user is seen in the MAS as any other agent, belonging to one of the competing teams. The user may change the predefined behaviour of his avatar in the virtual world by means of a pre-defined set of gestures that are captured by the Kinect sensor. Figure 3 details the different gestures used in the implementation of the Kinect interface. These gestures are related to the following behaviours:

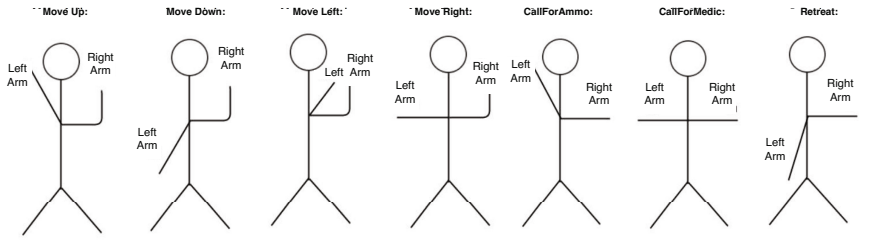

Fig. 3. Kindle gestures used

- Move Up: the user agent is moved up in the virtual environment map.
- Move Down: the user agent is moved down in the virtual environment map.
- Move Left: the user agent is moved left in the virtual environment map.
- Move Right: the user agent is moved right in the virtual environment map.
- CallForAmmo: the user agent sends a message to all living agents in his team playing the field operations rol asking for ammo packs.
- CallForMedic: the user agent sends a message to all living agents in his team playing the medic rol asking for medic packs.
- Retreat: the user agent sends a message to all living agents in his team ordering them to retreat to their base.

Figure 4 shows how the JGOMAS architecture is left now, where the *User Agent* represents the agent that is able to receive orders from the user, having as an input mechanism the *Kinect*.

User gestures are translated into messages sent through a TCP-IP connection to the *User Agent*. These messages are the most prioritary messages that this agent may receive. *User Agent* translates them to tasks and messages to other agents. Move Up, Move Down, Move Left and Move Right orders provoke the user agent to add a priority *Go To Target* task with the corresponding position. The rest of the orders, as have been commented above, provoke the agent to send different messages to their teammates.

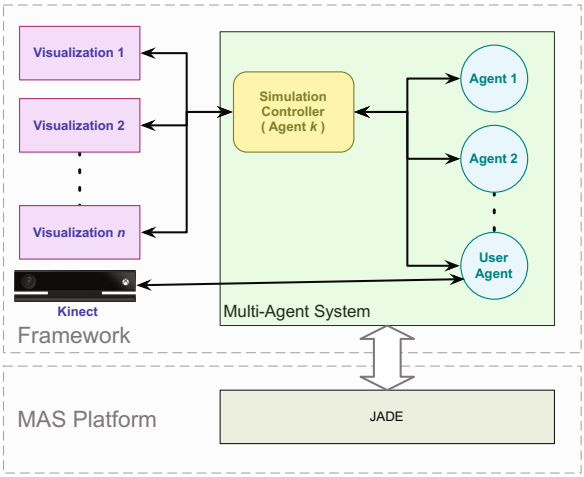

Fig. 4. JGOMAS extended architecture

4.2 Evaluation

The proposal has been implemented over an scenario example in order to validate the whole immersion of the users and software agents. To do this, the scenario is formed by two human players which are located in different rooms playing the same JGOMAS game each one in a different team. Moreover, the game includes nine more software agents playing in each team. These agents have been implemented in JADE (see figure 5).

As before commented, each human player interacts with the system through a Kinect sensor and a JADE agent (User Agent) which represents the human in the virtual scenario. The Kinect sensor is controlled through a driver implemented in C# that must take into account the main technical limitations of the sensor, which are:

- The Horizontal field of view is only a maximum of 57 degrees.
- The Vertical field of view is a maximum of 43 degrees.
- There is a low resolution in the image depth. The detection of small elements is worse when increasing distance.
- There is a range of + - 27 degrees of vertical tilt .
- Regarding depth sensitivity there are two operating modes:
  - In normal mode, the range of distances for the optimal operation of the sensor varies from 0.8m to 4m , and in the range of 4 -8m its performance degrades. Beyond 8m and closer than 0.4m the data has no validity.
  - In the operating mode called *near*, the distance for an optimum performance is reduced to 0.4 m, having an optimal range of 0.4- 3m, being unreliable data for 3 -8m and without any validity in 0-0.4m and beyond 8m.

– It is able to monitor at most two active persons simultaneously, being able to follow up to 20 joints for each active person.

Humans can control his agent through the use of the before commented gestures. In absence of them, the User Agent can take the initiative and interact with the rest of the system without intervention of the human. After several executions the system worked in a correct way, but some problems were detected. Regarding the recording position process, the recognition system which has been used requires that the positions were stored in a file that is uploaded to a serializable classes in a certain structure. In the absence of any tool to record gestures it has been implemented a block that allows to create a file with the gestures to be recognized for each one of the possible actions to be performed. Occasionally, the system suffered short disconnections forcing to restart the Kinect driver. A reconnection module has been added in order to automatically manage these short disconnections avoiding this situation. Moreover, during the development was observed that the number of events launched by the Kinect sensor was too low to properly recognize gestures. To solve this problem different versions of the Kinect for Windows controller were tested over different platforms without getting improvements. Finally, it was decided to limit the number of actions taken by the Kinect taking into account at any time only one active sensor (voice, rgb, or depth).

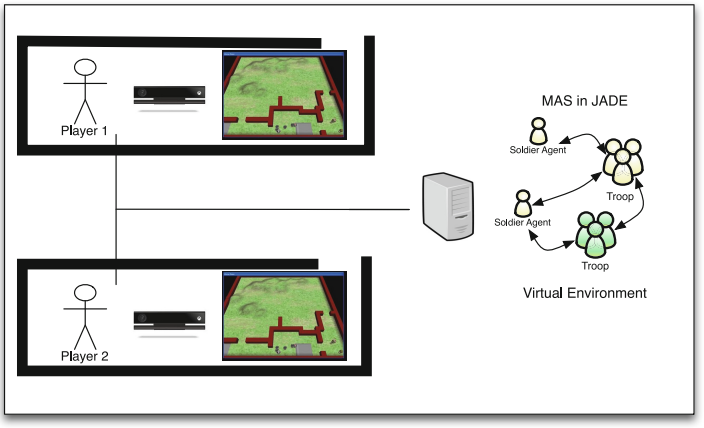

Fig. 5. Real implementation test

5 Conclusions

This work presents the problem of the double immersion in Human-Agent Societies, where human and agents must coexist in a framework of maximum integration. In order to achieve this kind of immersion, a case study has been developed over the JGOMAS framework using natural interfaces allowing an easy integration of the human in the MAS. Future work in this research area will focus on

developing a learning module which will allow to modelize the human behavior. This learning mechanism can be seen as a training process allowing software agents to anticipate to human actions. This will allow humans to minimize the number of gestures needed to react in front of typical situations.

Acknowledgments. This work was supported by the Spanish government grant MINECO/FEDER TIN2012-36586-C03-01,

References

1. Augusto, J.: Ambient Intelligence: the Confluence of Ubiquitous/Pervasive Computing and Artificial Intelligence. Intelligent Computing Everywhere (2007)
2. 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)
3. Barella, A., Mart, M., Carrascosa, C., Botti, V.: Multi-Agent Systems applied to Virtual Environments: a Case Study. In: ACM Symposium on Virtual Reality Software and Technology, pp. 237–238. ACM SIGGRAPH (2007)
4. Barella, A., Valero, S., Carrascosa, C.: JGOMAS: New Approach to AI Teaching. IEEE Transactions on Education 52(2), 228–235 (2009)
5. Group, I.A.: The European Union report: Scenarios for Ambient Intelligence in 2010 (2001), `ftp://ftp.cordis.europa.eu/pub/ist/docs/istagscenarios2010.pdf`
6. Hendler, J.: Where Are All the Intelligent Agents? IEEE Intelligent Systems 22, 2–3 (2007)
7. Offermann, S., Ortmann, J., Reese, C.: Agent based settler game (2005), `http://x-opennet.org/netdemo/Demos2005/aamas2005/_netdemo/_settler.pdf`, part of NETDEMO, demonstration at the international conference on Autonomous Agents and Multi Agent Systems, AAMAS 2005
8. Ramos, C., Augusto, J., Shapiro, D.: Ambient Intelligence?the Next Step for Artificial Intelligence. IEEE Intelligent Systems 23, 15–18 (2008)
9. Satyanarayanan, M.: Pervasive Computing: Vision and Challenges. IEEE Personal Communications 8(4), 10–17 (2001)
10. Weiser, M.: The Computer for the 21st Century. Scientific American 265, 94–104 (1991)