Multi-agent Systems as Intelligent Virtual Environments

George Anastassakis[2], Tim Ritchings[1], and Themis Panayiotopoulos[2]

[1] School of Sciences, University of Salford,
Newton Building, University of Salford, Salford, M5 4WT
T.Ritchings@salford.ac.uk
[2] Knowledge Engineering Lab, Department of Informatics,
University of Piraeus, Piraeus, 185 34, Greece
{anastas, themisp}@unipi.gr

Abstract. Intelligent agent systems have been the subject of intensive research over the past few years; they comprise one of the most promising computing approaches ever, able to address issues that require abstract modelling and higher level reasoning. Virtual environments, on the other hand, offer the ideal means to produce simulations of the real world for purposes of entertainment, education, and others. The merging of these two fields seems to have a lot to offer to both research and applications, if progress is made on a co-ordinated manner and towards standardization. This paper is a presentation of VITAL, an intelligent multi-agent system able to support general-purpose intelligent virtual environment applications.

1 Introduction

Probably one of the most exciting and promising scientific fields ever, the field of *intelligent agents* is still the subject of major controversy over its origin, formal background, definitions, methods, applications and future directions. The notion of an intelligent agent, indisputably challenging to define precisely, has been used to characterize a vast number of approaches and applications, ranging from simple softbots to complex, large-scale industrial control systems.

Recent attempts to merge intelligent agent approaches with virtual reality and artificial life have given birth to the field of *intelligent virtual environments (IVEs)*. An IVE is a virtual environment resembling the real world (or similar), inhabited by autonomous intelligent entities exhibiting a variety of behaviours. These entities may be simple static or dynamic objects (a revolving sun, traffic lights, etc.), virtual representations of life forms (virtual animals and humans), avatars of real-world users entering the system, and others. In fact, the structure and contents of a virtual environment are only restricted by the nature of the target application and the designer's imagination - and, of course, the amount of computing power available.

Today, IVEs are employed in a variety of areas, mainly relating to simulation, entertainment, and education. Sophisticated simulated environments of different types (open urban spaces, building interiors, streets, etc) can significantly aid in architectural design, civil engineering, traffic and crowd control, and others. In

F. Baader, G. Brewka, and T. Eiter (Eds.): KI 2001, LNAI 2174, pp. 381–395, 2001.

addition, precisely modelled simulations of real-world equipment (vehicles, aircrafts, etc) not only can be tested at reduced cost and risk, but also more accurate results can be obtained thanks to the additional element of control by and interaction with intelligent, thus closer to real life, entities. Moreover, IVEs have set new standards in computer-aided entertainment, through outstanding examples of computer games involving large, life-like virtual worlds (where imaginative scenarios are to be challenged), interactive drama (where the user is an active participant in the plot) virtual story-telling, and many other areas where immersion and believability are key factors. Concluding, IVE-based educational systems incorporate believable tutoring characters and sophisticated data representation techniques, resulting in the stimulation of user interest and perceptual ability, thus providing a novel, effective and enjoyable learning experience.

Despite the fact that an intelligent agent is the ideal metaphor for representing intelligent inhabitants inside an IVE, surprisingly little effort has been directed towards a formal and co-ordinated merging of intelligent agent systems and virtual reality techniques to produce IVEs fully exploiting the advantages of both fields. This paper is a presentation of our attempt to contribute to such an initiative: a fully functional intelligent agent system with the ability to support virtual intelligent agents embodied inside simulated worlds represented using VR techniques. Along with a number of other features of both practical and scientific value, the system can be used for a variety of purposes, acting as either an IVE for one of the application areas mentioned above, or a typical multi-agent system employed in classical application domains, such as control systems, distributed problem solving, resource allocation and many others. In the rest of this paper, a discussion of relevant research work is given in section two. Section three is a thorough presentation of the proposed system, while section four is a layout of additional work carried out towards multi-agent support. An example of the system's operation is given in section five.

2 Related Work

The Beliefs-Desires-Intentions (BDI) model [4] is probably the most popular approach towards the design of intelligent agents, mainly due to its ability to trigger behaviours driven by conceptually modelled intentions and goals rather than explicit procedural information. In addition, it seems to be a functional abstraction for the higher-level reasoning processes of the human mind, those that are related to action selection and the focusing of intelligent reasoning processes on specific desired states. The BDI model has been adopted in a significant number of implementations:

In [3], Bratman et al. present the Intelligent Resource-bounded Machine Architecture (IRMA), an architecture for resource-bounded (mainly in terms of computational power) deliberative agents, based on the BDI model. IRMA agents consist of four main modules: a means-end planner, an opportunity analyser, a filtering process and a deliberation procedure. In addition, they contain a plan library, and data structures to store beliefs, desires and intentions.

Jennings in [10] proposes GRATE, an architecture clearly focused on co-operative problem solving through agent collaboration. Central to the entire architecture is the notion of joint-intentions. In fact, even though GRATE is a deliberative architecture based on the BDI model, it is specifically referred to as a belief-desire-joint-intention architecture.

The BDI model has provided valuable theoretical grounds upon which the development of several other architectures and approaches, such as hybrid and layered agents, was based [9]:

The Procedural Reasoning System (PRS) [7] is a hybrid system, where beliefs are expressed in first-order predicate logic and desires represent system behaviours instead of fixed goals. PRS includes a plan library containing a set of partial plans, called knowledge areas, each associated with an invocation condition. Knowledge areas might be executed due to goal-driven reasoning or as a response to sensory data; this way, the agent is capable for both deliberative and reactive behaviours.

Muller in [12] proposes INTERRAP, a layered agent architecture focusing on the requirements of situated and goal-directed behaviour, efficiency and co-ordination. INTERRAP agents consist of a world interface, a behaviour-based, a plan-based and a co-operation component, each affecting agent behaviours at a different level of social and functional abstraction..

In [16], Sycara et al. present the Reusable Task Structure-based Intelligent Network Agents (RETSINA) architecture. The architecture consists of three types of agents: interface, task agents and, information agents.

Due to its apparent focus on high-level reasoning and generation of elaborate behavioural patterns, the BDI model seems to be inadequate to efficiently and effectively model all aspects of intelligent reasoning. However, any system that needs to exhibit goal-driven behaviour should incorporate, among others, a BDI-based or equivalent component.

The merging of intelligent agent systems, artificial life and classical VR techniques has given birth to the field of *Intelligent Virtual Environments (IVEs)*. Typical examples involving IVEs and general virtual agents include Humanoid [2], Creatures [8], Artificial Fishes [17], and others.

The CoMMA-COGs [5] project (Cooperative Man Machine Architectures - Cognitive Architecture for Social Agents) is an architecture for Multi-Agent systems and animated virtual environments, developed by the German Research Center for Artificial Intelligence. The system employs traditional multi-agent research approaches. Furthermore, it supports self-organization of agent societies, so that external users perceive them as units, and, thus, being unaware of the underlying organization processes. In addition, resource-awareness allows agents to perform in unpredictable environments while flexibly managing their resources. In general, IVEs tend to focus on either the virtual representation and embodiment side, or the intelligence side. Full benefit has not yet been taken of the combined advantages of intelligent multi-agent systems and virtual environments. A complex, accurately modelled and general-purpose IVE, inhabited by numerous believable entities driven by strong and effective AI reasoning processes, is yet to be presented.

A predecessor to the VITAL system and a first effort towards an intelligent agent system architecture with the ability to support IVE applications, the DIVA architecture, developed by the Knowledge Engineering Lab of the University of Piraeus, was presented in [18].

3 Overview of the VITAL System

The system presented in this paper is called *VITAL*, an acronym for *Virtual InTelligent Agents with Logic*. It represents an attempt to explore the world of

intelligent agent systems and intelligent virtual environments, initiated in UMIST, UK [1]. VITAL is a simulation of a maze world. Agents are required to explore a given maze, locate specific items inside it and process them in response to user instructions, e.g. move them to given locations. Agents have no initial knowledge of either the structure of the maze or of the items' locations. The system is monitored using a 3D viewer supporting free world navigation and several configuration options.

The system was designed and implemented with the following goals and requirements in mind:

1. *Wide range of applications:* the system should support a wide range of areas.
2. *Agent effectiveness, independence and agility:* for a given area, agents should be effective, that is, achieving what they have been assigned; furthermore, they should require no user intervention while doing so; finally, they should be agile, maintaining their effectiveness even when the environment changes unpredictably within the limits of a defined domain.
3. *Distributed and modular structure:* the system should be distributed, consisting of discrete co-operating components executing on possibly different machines across a network; in addition, modularity should allow the insertion and removal of components at runtime without interrupting the system's operation.
4. *Significant research potential:* the system should be designed so that scientific experimentation is inherently.
5. *Intuitive observation and monitoring capabilities.*
6. *Sophisticated implementation:* the system should be implemented using modern, specialized software tools and latest technologies, essentially comprising a high-quality software product.
7. *Extendibility and reusability:* the system should be highly extendible so that it can be advanced along with ongoing research, by allowing the introduction of new concepts and approaches, such as multi-agent characteristics.

3.1 Architecture

The VITAL system consists of three types of conceptually discrete components: *worlds*, *agents* and *viewers*. These are implemented as separate software applications according to the client-server approach. A world component represents a virtual environment inside which the entire agent system's activity takes place. Agent components represent actors inside an environment. Agents perceive the environment and act upon it according to goal-driven behaviours. Viewers offer the means to human supervisors to observe the environment and all activity inside it in a domain-specific manner. The system's component-based nature is outlined in Figure 1 below:

During system operation, a number of interactions take place between components. In particular, when an agent wants to sense its environment, it requests sensory information from the world component – a *sense request*. The world component then replies, providing the requested information. Similarly, when an agent wants to perform an action, the corresponding agent component provides all necessary action information to the world component – an *action request* – and then the world component responds regarding whether the requested action was successfully applied. In the case of successful action application, the world component sends world change data to all viewer components so that actions are correctly visualised. In addition,

when a viewer needs to build an entirely new visualisation, it requests a full description of the world model – a *world description* request.

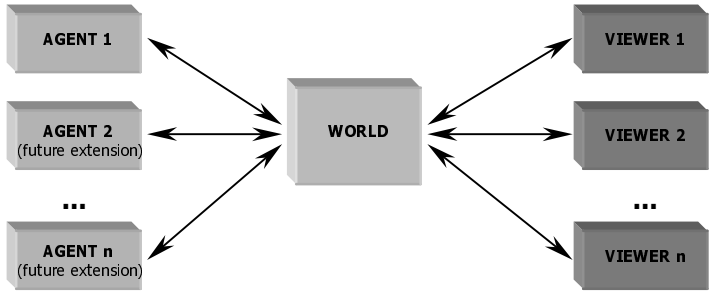

Fig. 1. VITAL system architecture outline

The separation of the world and the agent into two discrete components essentially enables the separation of the logical layer from the physical, real-world layer. This way, agent design can remain focused on the abstract properties and characteristics of the domain addressed and the conceptual specifics of the problem, hence, significantly increasing the system's modelling capacity. On the other hand, the world component can deal with all the realization details, for example, the control of equipment or hardware, giving physical substance to agents' virtual actions.

Adopting the client-server approach directly serves the goal of extendibility, since there is no limitation as to how many agent clients can be connected at any time to a world server. In addition, multiple simultaneous viewer client connections are supported, allowing the system to be observed and monitored from different views, and possibly in different ways, depending on the visualisation capabilities of each viewer client.

3.2 World Modelling

Each component maintains a complete or partial internal representation of the world and, in the case of an agent component, additional properties about itself. The representation used can be either symbolic or object-oriented, according to the component's nature. An additional type of representation, called *pseudo-symbolic*, is used to transfer symbolic facts between applications as well as to translate between symbolic and object-oriented representations.

In the VITAL system, worlds are modelled from the object-oriented point of view as sets of interconnected *locations*. Each location contains one or more *items*. Each item has a *name*, belongs to an item *class* and has *properties*. Agents are also represented as world items. To enrich the modelling scheme with spatial features, each location includes a two- or three-dimensional co-ordinate pair, according to the applications' needs.

This modelling abstraction is adequate to describe a substantial number of different environments: simple mazes, real-world buildings, streets, networks, etc. A world modelled from the object-oriented approach is essentially represented as a partially connected non-directed graph.

To implement this representation as well as to equip it with the necessary management functionality, a specialised class-based hierarchy was defined. According to it, each item has a property of name 'class', which denotes its *item class*, that is, the conceptually wider class of entities the item belongs to. Item classes can be defined according to the application's modelling requirements to any level of abstraction desired. Their selection depends on the properties selected to adequately discriminate types of items in a world. For instance, an agent could be denoted by an item property of name 'class' and value 'agent', whereas a non-agent item could have a 'class' property of value 'object'.

The symbolic modelling methodology defined by the architecture borrows syntactical and semantics elements from predicate logic and logic programming. The VITAL system uses the symbols shown in Table 1 below, with their respective interpretations:

Table 1. World modelling symbols used in the VITAL system

Symbol	**Interpretation**
connects(X1, Y1, X2, Y2)	'(X1, Y1)' and '(X2, Y2)' are connected
at(Item, X, Y)	The location of 'Item' is '(X, Y)'
location(X, Y)	'(X, Y)' is a valid maze location
item(Item)	'Item' is a valid maze item
class(Item, Class)	The class of 'Item' is 'Class'
<property_name>(Item, Value)	'Item' has a property with a name of '<property_name>' and a value of 'Value'

To handle asynchronous network transmission of symbolic information, the architecture introduces a *pseudo-symbolic representation*, according to which, facts and functors are broken down to a series of strings, the first of which being the fact or functor name and the rest arguments. Non-atomic terms, i.e. variables, are represented by an appropriate keyword selected by the agent system designer, for instance, '#VAR'. A *terminating dot* is appended after the last argument to denote that no more arguments should be expected. If a series of facts or functors is to be transmitted, an additional terminating dot is appended after the last fact or functor; thus, two terminating dots should be expected at the end of a series of facts or functors in pseudo-symbolic representation.

3.3 World Server

The world server can be viewed as the central component of the architecture; it provides the grounds on which all action takes place. In addition, it ensures that this action follows specific rules, maintaining consistency throughout the system at all times. Furthermore, from a functional point of view, the world server co-ordinates data interchange between applications ensuring that the agent's model for the world and the viewer's visual representation are valid. The world server is divided into three layers, as shown in Figure 2 below:

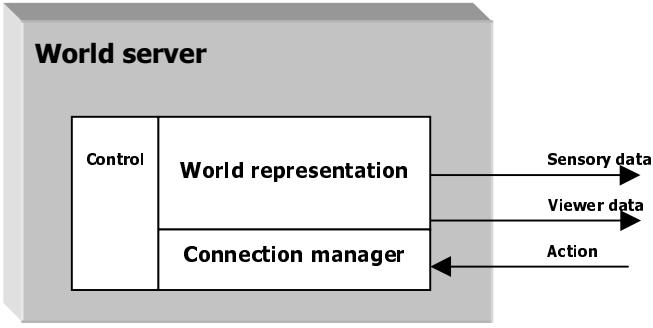

Fig. 2. World server structure

The *world representation* layer contains all data and functionality required to represent and manage the world. The *connection manager* layer is an encapsulation of the server's communications layer, so that the connection management programming interface specifically manages agent and viewer client connections as well as sensory and action requests – a useful programming abstraction offered to agent system developers. Finally, the *control layer* uses the functionality of the world representation layer to receive data from clients, carry out requests and transmit results. The world representation approach used in the world server is object-oriented.

The control layer is responsible for coordinating multiple connections (an agent and one or more viewers). In addition, all data receipt and transmission must be appropriately co-ordinated so that visualisations by viewer clients are consistent with the world model at all times. To achieve that, the control layer operates on a state-dependent basis.

3.4 Agent Client

The agent client is the component with the most vital contribution to the architecture. It stands as an implementation of an actor inside an environment simulated by a world server; it is the ingredient that brings the system to life. It essentially introduces the element of intelligence thanks to innate support for intelligent problem solving. As shown in Figure 3, an agent client consists of an *intelligence layer* and an *interface layer*. The intelligence layer is further divided into the *knowledge base*, the *decision engine* and the set of *sensors* and *effectors*.

Agent clients operate on a *sense-decide-act* cycle. During the *sense stage*, a 'SENSE' keyword is sent to the world server; results received in pseudo-symbolic representation are then processed by sensors and appended to the knowledge base. During the *decision-making stage*, the decision engine reasons upon the contents of the knowledge base and creates a plan for one of the agent's goals. Finally, during the *action* stage, the plan's next action is sent to the world server in pseudo-symbolic representation and effectors apply effects concerning the agent's own internal state.

The knowledge base is the agent's memory. It contains all data that has been perceived by the agent or produced as a result of its reasoning processes. Data in the knowledge base are formulated in a symbolic fashion, thus following the symbolic

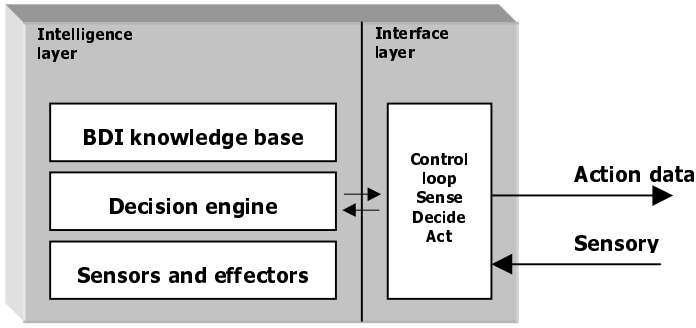

Fig. 3. Agent client structure

modelling approach. Consequently, the knowledge base readily supports the agent'sintelligent reasoning processes. Furthermore, to support intentional reasoning, the knowledge base is structured according to the *BDI (Beliefs-Desires-Intentions)* architecture [4]. An example knowledge base is shown below:

```
[
            beliefs([
              holding([])
            ]),

            desires([
              explore(X, Y),
              pickup(theRedBall),
              drop(theRedBall, 1, 2),
              pickup(theBlueBall),
              drop(theBlueBall, 3, 2),
              move(2, 2)
            ]),

            intentions([

            ])
]
```

Abilities represent the ways an agent can act upon a world. In a VITAL agent, abilities are also defined within the agent's beliefs. In the presented architecture, abilities are defined as (*N, P, E*), where *N* stands for the ability *name*, a functor that identifies it and provides access to all necessary arguments, *P* for the list of *preconditions* and *E* for the list of *effects*. Preconditions are functors that must be included in an agent's knowledge base for an ability to be usable. Effects are functors processed by an agent's effectors to update its beliefs. An example of an ability defined according to the scheme described above follows:

```
ability:       move(X, Y)
preconditions: [at(CX, CY), connects(CX, CY, X, Y)]
```

```
delete:        [at(CX, CY)]
add:           [at(X, Y)]
```

The above example shows the definition for the 'move' ability, that is, the agent's ability to move from one location to another. The preconditions list denotes that the ability is usable only when there is a connection between the agent's current location with co-ordinates (CX, CY) and the new location with coordinates (X, Y). Effects are divided into the *delete* and *add* lists; these denote the facts that must respectively be removed from and added to the agent's beliefs to reflect the new state produced.

The decision engine is responsible for controlling the agent's behaviour; it is an encapsulation of its very intelligence. On every decision-making stage, the engine validates the current plan, and if this fails, or if there is no current plan, the engine attempts to generate a new plan for the goal of top priority. If no plan can be generated for the top goal, the engine attempts to generate a plan for the next one, and keeps doing so until a plan is generated or until no more goals are available.

In VITAL agents, the basic component of the decision engine is the plan generator, or *planner*. The planner consists of two layers: a) a general-purpose means-end planner, and b) a problem-specific second level, which employs heuristics to perform action selection and hence take full advantage of available knowledge into a specific application area, something that significantly reduces computational load and agent response times.

Sensors and effectors encapsulate the agent's sensing and acting functionality; essentially, they are the only access points to its knowledge base. At the end of every sense stage sensors process sensory data received by the world and finally append them to the agent's beliefs. Moreover, after transmission of an action request, effectors update its knowledge base as to reflect its new internal state.

It is important to note that the agent does not have unlimited sensing abilities. What is available to its sensors on any given location is decided upon by the world, following certain rules for vicinity, obstruction, etc. This allows agents to be employed in situations where environment knowledge is obtained gradually. It also gives a certain amount of realism to experimental simulations. The rules are appropriately designed to suit the needs of each application.

Furthermore, an agent's model of the world might not only be a subset of, but inconsistent with the 'real' one, that is, the one maintained by the world server. This may be a desired effect, or the result of erratic operation, most probably due to limited or faulty sensing and reasoning.

3.5 Viewer Client

The viewer client is the encapsulation of the system's visualisation functionality. Similarly to the world server, it maintains an object-oriented world model, and contains a communications layer with sufficient translation functionality. The viewer client's structure is shown in Figure 4 below:

The viewer displays the maze and its contents as a 3D scene, through which a user can navigate freely to obtain a suitable observation angle. Agents are displayed using avatars loaded from VRML files. A number of additional features are also supported,

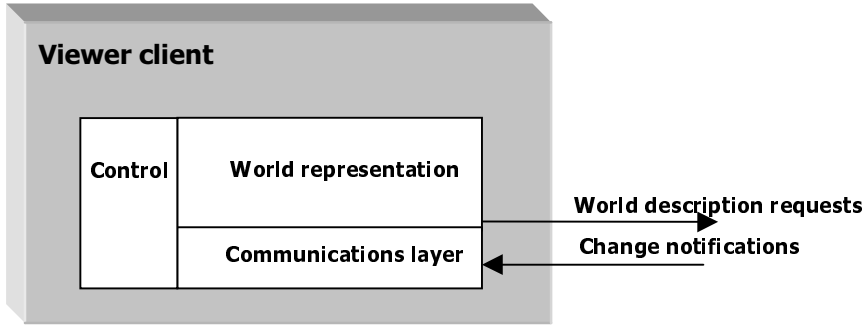

Fig. 4. Viewer client structure

such as lighting, avatar file configuration, as well as locking of the viewpoint on an avatar from various viewing angles, so that agent movement and actions can be easily tracked. The viewer client's user interface as well as the VITAL system in full operation is shown in Figure 5 below.

3.6 Implementation

Reasoning processes within the VITAL agent client have been implemented using SICStus Prolog [15], an implementation of the Prolog language developed at the Swedish Institute of Computer Science (SICS). Apart from being a fully functional Prolog system, offering multiple useful features such as constrained solving, access to operating system resources, parallel solving and many others, SICStus Prolog supports compilation of the Prolog code. Compiled predicates will run faster, using memory more economically. Compiled predicates can be called from within source code in another programming language, such as C++, thanks to an interface provided by the SICStus system. This is an essential feature if reasoning mechanisms built in Prolog are to serve as parts of another Win32 application.

The planner developed for the purposes of the VITAL system is a breadth-first, state-reducing, means-end planner, called *StateExplorer*. Due to its breadth-first nature, it systematically explores a given state space by applying all *available actions* to produce *next* states for a list of *current* ones, hence, the StateExplorer name. The planner was written entirely in the Prolog language; it is defined as a *plan/4* predicate.

The VITAL agent client's second-level planner is implemented as a *plan2/3* Prolog predicate. It is responsible for exploiting domain-dependent knowledge into the nature of the agent's abilities, to minimize computational effort.

The visualisation engine of the VITAL viewer has been implemented using OpenGL. Avatars are loaded from VRML [19] files thanks to a parser implemented using Lex and YACC tools for the Win32 environment, namely the Parser Generator package by Bumble-Bee Software. The parser is capable of processing VRML files containing a subset of the language sufficient to represent scenes and objects exported from major 3D design packages, such as Kinetix 3D Studio and Macromedia Poser.

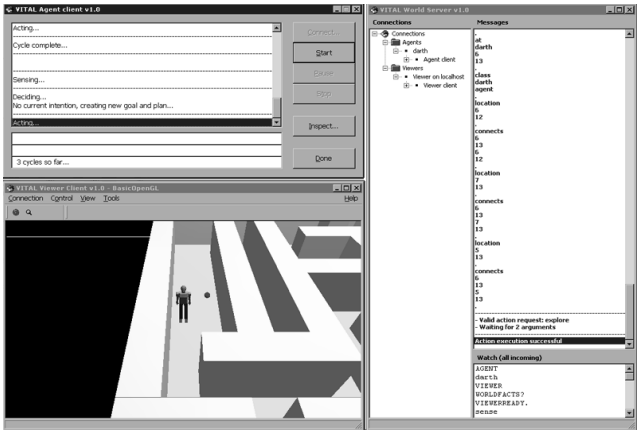

Fig. 5. The VITAL system in full operation

4 Multiple Agent Support and Inter-agent Communication

The VITAL system has recently been extended to support co-existence and simultaneous operation of multiple agents. The new system is named 'mVITAL' (multi-agent VITAL). It enables the development of simulations where intelligent agents communicate through simple speech acts, co-operate and help each other to achieve goals, reason on other agents, exchange beliefs, etc.

As discussed in [1], the agent client is probably the most significant component of the VITAL system, since it introduces the feature of intelligence, thus allowing the properties of agility and dynamic behaviour to emerge. Agent clients in the extended mVITAL system are of even more crucial importance, since they enable the definition of agent societies, introducing the elements of inter-agent interaction, inter-agent communication, co-ordination, and distributed problem solving.

In mVITAL, an agent can identify other agents by sensing their 'class' property and checking if it has a value of 'agent'. In addition, they are able to refer to each other using the value of their 'name' property, just as they refer to themselves using the 'me' keyword. For instance, an agent would believe that 'at(smith, 1, 1)' about agent Smith's position, and 'at(me, 10, 10)' about its own. Social reasoning is from that point on a matter of modelling, and agent interactions emerge as a result of properly defined abilities.

Agents are able to communicate using virtual, non-visual *speech items*. Speech items belong to a special class denoted by a 'speech' value to the item's 'class' property. Speech items have a 'text' property, whose value can be anything an agent wishes to communicate to another, usually text in some natural language. A speech act between two agents can then be modelled as the consecutive application of two actions, one adding a speech item to the world through the effects part, and the other enabling the perception of the item through the preconditions part, thus allowing the receiving agent to respond. Communication can be made even more complex, with

additional speech item properties denoting implied or 'hidden' information, as well as information related to mood, tone, and other expression-related parameters. KQML [6] performatives can be modelled using a suitable set of properties; such a set could include properties such as 'MSG-TYPE', 'VERB', etc, to denote intent, tone, and other information exchanged according to KQML format. To allow more intuitive observation, an extended viewer client can appropriately handle speech-objects and use property values to display communicated text on-screen, reproduce speech using a voice generation engine, etc.

A crucial issue with speech items is that they need to be automatically removed from the world after a certain period of time, to simulate the real world analogy, where a person's speech is heard only while it is spoken. In mVITAL, speech items last a little longer (five seconds) so that all agents in a given range can sense them during a following sense-decide-act cycle. Eventually, five seconds after insertion to the world, speech items are automatically removed by the world server.

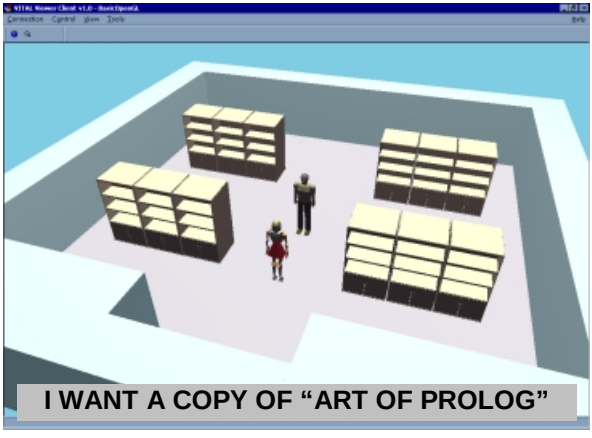

Fig. 6. The client agent requests a copy of "Art of Prolog" from the librarian agent

5 A Working Example

A simple simulation has been modelled in the mVITAL system to demonstrate agent communication and information exchange. The example could originate from the field of computer games or simulation.

The presented scenario involves two agents, a librarian and a library client. The client is looking for a book. In order to get it, it approaches the librarian and asks for it. Then, the librarian sets off and starts looking for the requested book among the library's shelves. If the book is available, the librarian brings it to the client.

According to the VITAL architecture, the above sequence can be modelled as follows: Initially, the client-type agent is committed to requesting a certain book from the librarian. This is denoted by a specific goal in its knowledge base, e.g. 'request_book("some_title")'. Among the effects of this goal, there is the creation of a speech object with a text value of 'I want a copy of "some_title"', or something

similar. After the speech object's creation, the librarian-type agent is able to sense it, and extract book title information from the sensed speech object's text value. Then, a goal is adopted, forcing the librarian to locate the requested book, that is, move close to it. The scenario then follows similarly with the adoption and execution of appropriate goals, forcing the librarian to return with the requested book and pass it to the client, or inform the client that the book is unavailable by creating an appropriate speech object.

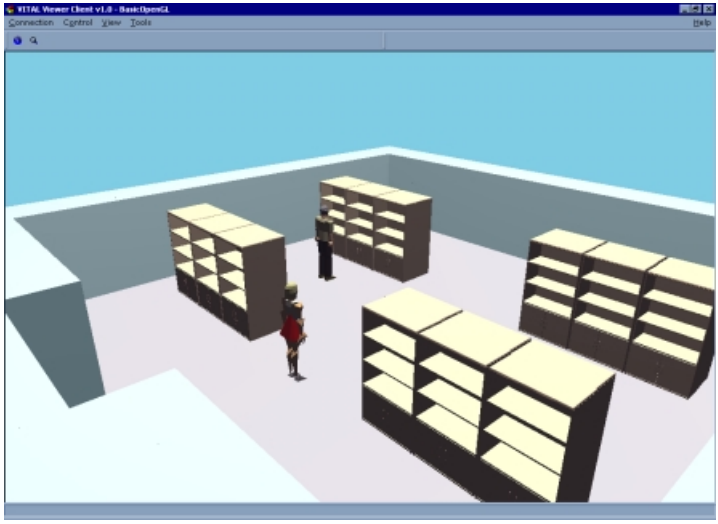

Fig. 7. The librarian is searching for the requested book while the client is waiting

6 Conclusions and Future Work

This paper has been a presentation of the VITAL intelligent agent system, a framework for developing a variety of applications ranging from typical agent-based control systems to VR-based simulations, to IVEs. Despite the fact that numerous issues still need to be addressed for the system to be used in complex evaluation environments such as the RoboCup-Rescue simulator [11], the system is fully functional, and all design requirements have already been fulfilled. In particular, the system is distributed, thus able to exploit the benefits of today's sophisticated networking technologies and the Internet; it employs formal AI techniques - logic programming, planning, intentional reasoning - to support intelligent agent behaviours; it is modular and component-based, enabling the deployment of persistent applications; different types of agents – not necessarily built according to the structure proposed by the architecture, but using the same communication scheme – can be connected to a world server, providing openness and extendibility, as well as enabling dynamic alteration of a simulation's structure and experimentation with other reasoning approaches; the system incorporates sophisticated VR techniques to produce intuitive and believable visualisations; finally, the system comprises a set of

state-of-the-art software applications, built using latest and specialized software engineering technologies.

The VITAL system contributes to original AI and IVE research not by actually extending current methods and approaches, but by providing a robust and well-designed architecture serving as the grounds on which research work and extension can take place. This way, even though VITAL is still at an experimental stage, it eases work towards improvement of the underlying formal approaches, standing as a research testbed where changes are applied in a straightforward manner and results are instantly and intuitively observed; this is the system's most important and original contribution to today's research.

As mentioned in 4, the mVITAL system is still under construction and experimentation. However, evaluation of the VITAL system's performance as a single agent starting point has justified the continuation of the effort towards a multi-agent level, showing that there is strong potential in using MAS technology in the field of IVEs.

Future work directions include, but are not restricted to, making the system more open and configurable, with the ultimate goal being a powerful IVE-authoring tool that will effectively contribute to formalization and standardization in the field. To achieve that, we are already working on the merging of the system with VAL (Virtual Agent Language) [13], to enable abstract and dynamic definitions of agent personalities. VAL is a C/C++-like agent-oriented programming language based on logic programming that was initially developed for the system presented in [14]. Moreover, we are addressing the issue of a world-modelling scheme general enough to enable precise definitions of a variety of scenarios and simulations. Furthermore, we are enriching the agents' planning abilities by introducing explicit temporal and spatial references. In addition, we are investigating more sophisticated visualization and embodiment techniques, as well as the issue of user intervention to simulations through embodiment and presence in the system. Finally, from an implementation point of view, various networking technologies are evaluated in an effort to optimise the system's performance so that real-time execution is guaranteed.

Acknowledgments. This work has been partially supported by the Greek Secretariat of Research and Technology under the PENED 99 project entitled "Executable intensional languages and intelligent multimedia, hypermedia and virtual reality applications", contract no. 99ED265.

References

1. Anastassakis, G.: Intelligent Agents in Vitual Worlds. MSc thesis, UMIST UK (2000)
2. Boulic, R., Capin, T., Huang, Z., Moccozet, L., Molet, T., Kalra, P., Lintermann, B., Magnenat-Thalmann, N., Pandzic, I., Saar, K.,Schmitt, A., Shen, J., Thalmann, D.: The HUMANOID Environment for Interactive Animation of Multiple Deformable Human Characters. Proc. Eurographics '95, Maastricht (1995) 337-348
3. Bratman, M. E., Israel, D. J., Pollack, M. E.: Plans and resource-bounded practical reasoning. Computational Intelligence, 4 (1988) 349-355

4. Bratman, M. E.: Intentions, Plans and Practical Reason. Harvard University Press (1987)
5. Burt, A. D., Fischer, K., Siekmann, J. H.: COGS : Cognitive Architecture for Social Agents. Computational Logic Magazine – The NewsLetter of the European Network in Computational Logic, http://www.cs.ucy.ac.cy/compulog/issue7/areakrr/agents.htm
6. Finin, T., Labrou, Y.: KQML as an Agent Communication Language. Software Agents, Bradshaw, J. M. (ed), MIT Press, Cambridge (1997) 291-316
7. Georgeff, M., Lansky, A.: Reactive reasoning and planning. Proceedings of the Sixth National Conference on Artificial Intelligence (1987) 677-682
8. Grand, S., Cliff, D., Malhotra, A.: Creatures: Artificial Life Autonomous Software Agents for Home Entertainment. Autonomous Agents 97, California USA (1997) 22-29
9. Haddadi, A., Sundermeyer, K.: Belief-Desire-Intention Agent Architectures. Foundations of Distributed Artificial Intelligence, O' Hare, G. M. P., Jennings, N. R., eds. Wiley & Sons, Inc (1996) 169-185
10. Jennings, N. R.: Specification and implementation of a belief desire joint-intention architecture for collaborative problem solving. Journal of Intelligent and Cooperative Information Systems (1993) 289-318
11. Kitano, H.: RoboCup-Rescue: A grand challenge for multiagent systems. Proc. 4th International Conference on MultiAgent Systems (2000)
12. Muller, J. P.: The Design of Autonomous Agents - A Layered Approach. Lecture Notes in Artificial Intelligence, vol. 1177, Springer-Verlag (1996)
13. Panayiotopoulos, T., Anastassakis, G.: Towards a Virtual Reality Intelligent Agent Language. Advances in Informatics (D. Fotiadis, S. Nikolopoulos, eds.), World Scientific (2000) 249-259
14. Panayiotopoulos, T., Katsirelos, G., Vosinakis, S., Kousidou, S.: An Intelligent Agent framework in VRML Worlds. Advances in Intelligent Systems : Concepts, Tools and Applications (S. Tzafestas, ed.), Kluwer Academic Publishers (1999) 33-43
15. SICS Institute: SICStus Prolog User's Manual. SICStus Prolog v3.8 (1999)
16. Sycara, K., Decker, K., Pannu, A., Williamson, M., Zeng, D.: Distributed Intelligent Agents. IEEE Expert, December (1996)
17. Terzopoulos, D., Tu, X., Grzeszczuk, R.: Artificial fishes: Autonomous locomotion, perception, behavior, and learning in a simulated physical world. Artificial Life 1, 4 (1994) 327-351
18. Vosinakis, S., Anastassakis G., Panayiotopoulos, T.: DIVA: Distributed Intelligent Virtual Agents. Extended abstract, presented at the Virtual Agents 99 workshop on Intelligent Virtual Agents, University of Salford UK (1999)
19. VRML Consortium: VRML97 International Standard (ISO/IEC 14772-1:1997). http://www.vrml.org/Specifications/VRML97 (1997)