AN AGENT-BASED ARCHITECTURE FOR VIRTUAL ENVIRONMENTS FOR TRAINING

ANGELICA DE ANTONIO

Computer Science School, Universidad Politécnica de Madrid

Campus Montegancedo, 28660 Boadilla del Monte (Madrid), SPAIN

(+34) 913366925 (Phone), (+34) 913366917 (Fax)

angelica@fi.upm.es

JAIME RAMIREZ

Computer Science School, Universidad Politécnica de Madrid

Campus Montegancedo, 28660 Boadilla del Monte (Madrid), SPAIN

(+34) 913367449 (Phone), (+34) 913366917 (Fax)

jramirez@fi.upm.es

GONZALO MENDEZ

Computer Science School, Universidad Politécnica de Madrid

Campus Montegancedo, 28660 Boadilla del Monte (Madrid), SPAIN

(+34) 913366941 (Phone), (+34) 913366917 (Fax)

gonzalo@gordini.ls.fi.upm.es

AN AGENT-BASED ARCHITECTURE FOR VIRTUAL ENVIRONMENTS FOR TRAINING

ABSTRACT

This chapter proposes an architecture for the development of Intelligent Virtual Environments for Training (IVETs), which is based on a collection of co-operative software agents. The first level of the architecture is defined as an extension of the classical Intelligent Tutoring System architecture that adds a new World Module. Several software agents are then identified within each module. They communicate among them directly via messages and indirectly via a common data structure which is used for the collaborative development of plans. Some details are provided about the most remarkable interactions that will be established among agents during the system's execution. The proposed architecture, and its realization in a platform of generic and configurable agents, will facilitate the design and implementation of new IVETs, maximizing the reuse of existing components and the extensibility of the system to add new functionalities.

KEYWORDS

Software Architecture, Computer-Based Training, Agent Technology, 3-D Graphics, Virtual Reality

INTRODUCTION

Computer-Based Training is a promising application area of three dimensional virtual environments(VEs). These environments allow the students to navigate through and interact with a virtual representation of a real environment in which they have to learn to carry out a certain task. They are especially useful in situations where the real environment is not available for training, or it is very costly or risky. A good example is training of nuclear power plant operators. A multi-user virtual environment also allows for the training of teams. An Intelligent Virtual Environment for Training (IVET) results from the combination of a Virtual Environment and an Intelligent Tutoring System (ITS). IVETs are able to supervise the actions of the students and provide tutoring feedback. The Intelligent Tutoring component of an IVET usually adopts a virtual representation (a Pedagogical Virtual Agent) that inhabits the environment together with the virtual representations of the students (avatars).

The development of three dimensional Virtual Environments has a quite short history, dating from the beginning of the 90s. The youth of the field, together with the complexity and variety of the technologies involved, have led to a situation in which neither the software architectures nor the development processes have been standardized yet. Therefore, almost every new system is developed from scratch, in an ad-hoc way, with very particular solutions and monolithic architectures, and in many cases forgetting the principles and techniques of the Software Engineering discipline (Munro et ál, 1999). Some of the proposed architectures deal only partially with the problem, since they are centered on a specific aspect like the visualization of the VE (Alpdemir and Zobel, 1998) (Demyunck et ál., 1999) or the interaction devices and hardware (Darken et ál, 1995).

As a result, current VEs lack many of the desirable quality attributes of any software system, such as flexibility, reusability, maintainability or interoperability.

The size and complexity of VEs will continue to increase in the future, making this situation even worse. Many researchers and developers of VEs are starting to recognize the need of a Software Engineering approach to the development of VEs (Brown et ál., 1999) (Fencott, 1999) (Sánchez, 2001).

In particular, there is a need to define “standard” architectures in order to facilitate the development of individual components by different teams or organizations which may have different skills and knowledge. The development of a new VE will then consist of the selection and adaptation of existing components, and their assembly and integration. In this way, components will be reusable, the system will be flexible to be extended with new components, and the interfaces among components will be clearly defined to facilitate interoperability of possible very heterogeneous components.

Unfortunately, we are still very far from this ideal state. Given the broad variety and diversity of VEs and their applications, and taking into account that they may require different architectures, we have decided to restrict the scope of our research to a certain type of VEs, namely Virtual Environments for Training (VETs).

The goal of this kind of VEs is to train one or more students in the execution of a certain task. They are especially useful in those situations in which training in the real environment is either impossible or undesirable because it is costly or dangerous. Lets consider as an example training the operators of a nuclear power plant in the execution of maintenance interventions. In the real environment the trainees would be subject to radiation, which is of course unacceptable for their health, and additionally it would be impossible to reproduce some maintenance interventions without interfering with the normal operation of the plant.

In VETs, the supervision of the learning process can be performed by human tutors or it can be performed by intelligent software tutors, also known as pedagogical agents (in this case we will call it an IVET). Those pedagogical agents, in turn, can be embodied and inhabit the virtual environment together with the students or they can be just a piece of software that interacts with the student via voice, text or a graphical user interface. Some pedagogical agents have been developed to date, in some cases with quite advanced tutoring capabilities. One of the best known is STEVE, developed in the Center for Advanced Research in Technology for Education (CARTE) of the Intelligent Systems Division of the University of Southern California (USC) (Rickel and Johnson, 1999) (Rickel and Johnson, 2000).

Recently, we conducted an experiment (Méndez et ál., 2003) in reusing an intelligent animated agent, namely Steve, in a new virtual world, HeSPI, which was developed independently. Steve was carefully designed to be easy to be applied to new domains and virtual worlds. It was originally applied to equipment operation and maintenance training on board a virtual ship. Subsequently, it was significantly extended and applied to leadership training in virtual Bosnia. However, the leadership training application was designed with Steve in mind. In contrast, HeSPI (de Antonio et ál., 2003) was developed independently as a tool for planning maintenance interventions in nuclear power plants. HeSPI's Virtual avatars were developed using Jack, a human simulation tool distributed by EDS. HeSPI's design was also carefully thought so that different user interfaces could be easily connected to the system. Two user interfaces were developed, one of them using voice recognition and conventional mouse, keyboard and monitor, and another one using a 3D mouse and shutter glasses. Thus, HeSPI+Steve looked like a good test bed for evaluating the degree in which nowadays VEs and pedagogical agents can be easily ported and made them interoperate.

All through the integration, we encountered several problems. For example, there were undesired behaviors due to the fact that both Steve and HeSPI performed redundant actions. There were control and synchronization problems. Steve required some information that HeSPI could not provide and vice versa, etc. Many of these difficulties stemmed from the fact that their underlying architectures and their external interfaces were not totally compatible. Our conclusion from this experiment was that there is effectively a need for standard architectures designed to facilitate reusability, extensibility and interoperability among components.

Component-based standard architectures in the field of educational software are not new. In the last years we witnessed the activity of several standardization organizations all around the world, like the IEEE Learning Technology Standards Committee (LTSC) (http://www.ltsc.ieee.org), which proposed the Learning Objects Meta-data Specification (LOM); the IMS Global Learning Consortium Inc. (http://www.imsglobal.org), creators of the Learning Resource Meta-data Information Model; ARIADNE (Alliance of Remote Instructional Authoring and Distribution Networks for Europe (http://www. ariadne-eu.org); or the AICC (Aviation Industry Computer Based Training Committee, http://www.aicc.org), authors of the Computer Managed Instruction (CMI) Specifications, Course Structure Format, and CMI Data Model. These standards and specifications were further integrated by the ADL (Advance Distributed Learning) co-Laboratory, created by an US initiative, to give birth to SCORM (Sharable Content Object Reference Model, http://www.adlnet.org), which is increasingly being recognized as an international standard for e-learning applications. However, these standards have never taken into account the special characteristics and needs of educational software based on VEs. They are oriented towards web-based e-learning courses in which the interactivity with the student is restricted to navigating through the materials (web pages) and answering tests.

The work that we present in this paper is a first step towards the goal of defining standard architectures for IVETs.

AN ARCHITECTURE FOR VETS BASED ON THE ARCHITECTURE OF INTELLIGENT TUTORING SYSTEMS

Our approach to the definition of an architecture for VETs is based on the agent paradigm, as opposed to the SCORM and other standards approach which is based on the object oriented paradigm. The rationale behind this choice is our belief that the design of highly interactive VETs populated by intelligent and autonomous or semi-autonomous entities, in addition to one or more avatars controlled by users, requires higher level software abstractions. Objects and components (CORBA or COM-like components) are passive software entities which are not able to exhibit the kind of pro-activity and reactivity that is required in highly interactive environments. Agents, moreover, are less dependent on other components than objects. An agent that provides a given service can be replaced by any other agent providing the same service, or they can even co-exist, without having to recompile or even to reinitiate the system. New agents can be added dynamically providing new functionalities. Extensibility is one of the most powerful features of agent technology. The way in which agents are designed make them also easier to be reused than objects.

Our work draws from the results obtained in the MAPI project ("Modelo Basado en Agentes Cooperativos para Sistemas Inteligentes de Tutoría con Planificación Instructiva", funded by CICYT from 1996 to 1999) and is currently being funded by the Spanish Ministry of Science and Technology through project MAEVIF (TIC00-1346). In MAPI we designed an architecture based on co-operative agents for the tutoring component of Intelligent Tutoring Systems (ITSs) in which communication among agents took place through a shared blackboard structure. Starting from the idea that a VET can be seen as an ITS (an IVET, Intelligent Virtual Environment for Training), and the pedagogical agent in an IVET can be seen as an embodiment of the tutoring module of an ITS, our first approach towards extending the MAPI architecture for IVETs was to define an agent for each of the four modules of the generic architecture of an ITS (see figure 1).

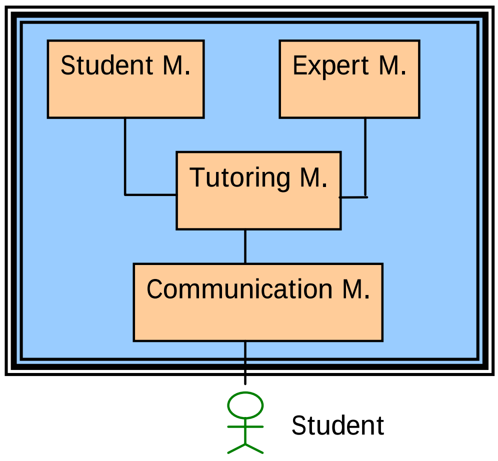

Figure 1. Architecture of an ITS

The ITS architecture, however, does not fit well with the requirements of IVETs in several aspects:

- IVETs are usually populated by more than one student, and they are frequently used for the training of teams. An ITS is intended to adapt the teaching and learning process to the needs of every individual student, but they interact with the system one at a time. However, in a multi-student IVET the systems would have to adapt both to the characteristics of each individual student and to the characteristics of the team.
- The student module should model the knowledge of each individual student but also the collective knowledge of the team.
- The student is not really out of the limits of the ITS, but immersed in it. The student interacts with the IVET by manipulating an avatar within the IVET, possibly using very complex virtual reality devices such as HMD (head mounted displays), data gloves or motion tracking systems. Furthermore, each student has a different view of the VE depending on their location within it.

- The communication module in an ITS is usually realized by means of a GUI or a natural language interface that allows the student to communicate with the system. It would be quite intuitive to consider that the 3-D graphics model is the communication module of an IVET. However there is a fundamental difference among them. In an IVET some of the learning goals may be directly related to the manipulation and interaction with the 3D environment, while the communication module of a classical ITS is just a means, not an end. For instance, a nuclear power plant operator in an IVET may have to learn that in order to open a valve he has to walk to the control panel, which is located on the control room, and press a certain button. Therefore, the ITS needs to have explicit knowledge about the 3D VE, its state, and the possibilities of interaction with it.

As a first step we decided to modify and extend the ITS architecture by considering some additional modules (see figure 2).

First of all, we split the communication module into a set of different views for all the students, plus a particular communication thread for each student and a centralized communication module to integrate the different communication threads. Then we added a World Module, which contains geometrical and semantical information about the 3D graphical representation of the VE and its inhabitants, as well as information about the interaction possibilities. The tutoring module is unique to be able to make decisions that affect all the students as well as tutoring decisions specific for a certain student. The expert module will contain all the necessary data and inference rules to maintain a simulation of the behavior of the system that is represented through the VE (for example the behavior of a nuclear power plant). The student module, finally, will contain an individual model for each student as well as a model of the team.

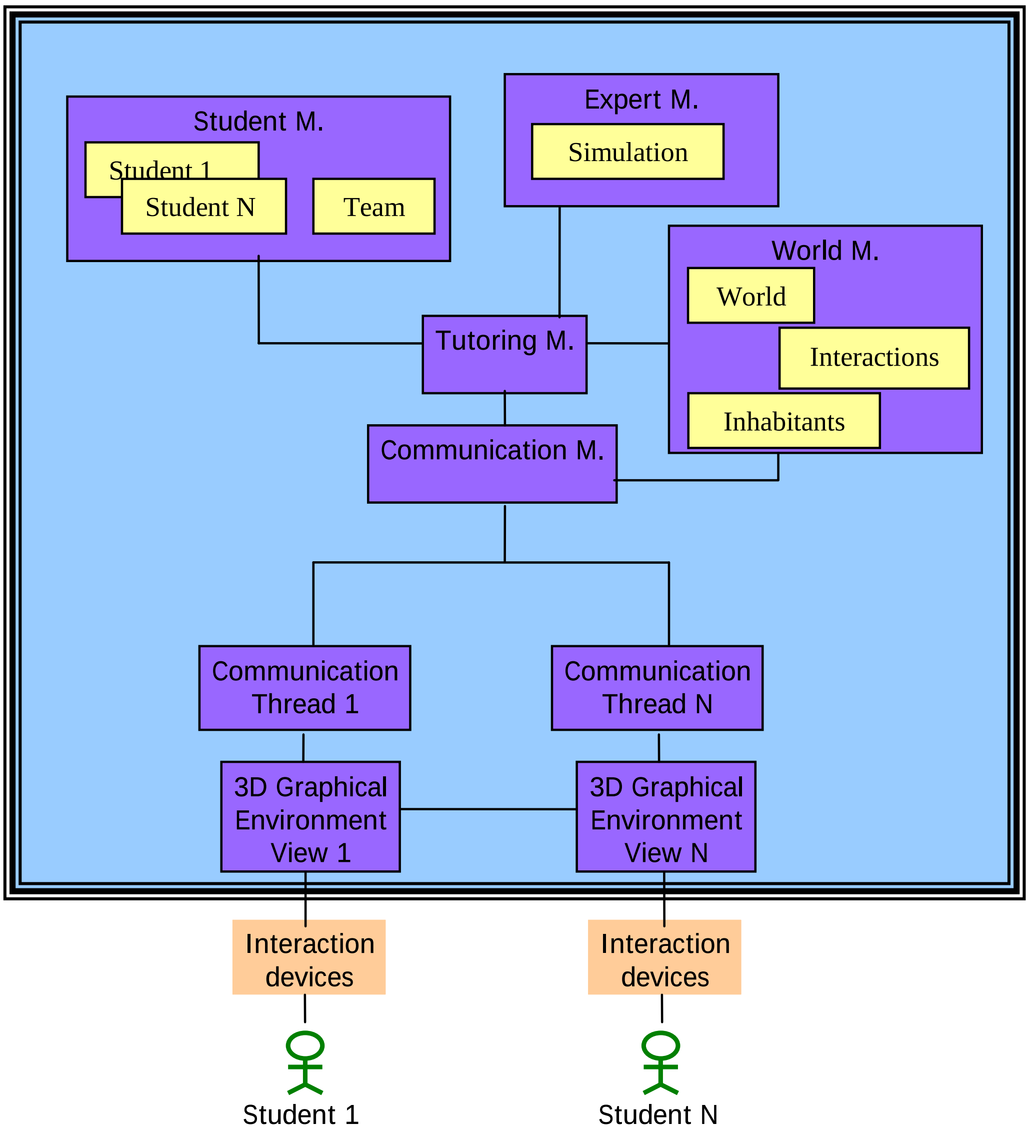

Figure 2. Extended ITS architecture for IVETs

AN AGENT-BASED ARCHITECTURE FOR IVETS

Taking the extended architecture of the previous section as a reference, the next step was to decide which software agents would be necessary to transform this component-oriented architecture into an agent-oriented architecture. In an agent-oriented architecture, each agent is capable of performing a certain set of tasks, and is capable of communicating with other agents to co-operate with them in the execution of those tasks.

Our agent-based architecture has five principal agents corresponding to the five key modules of the extended ITS architecture:

- A Communication Agent
- A Student Modeling Agent
- A World Agent
- An Expert Agent
- A Tutoring Agent

Each of these principal agents may relate to, communicate with and delegate some tasks to other subordinate agents, giving rise to multi-level agent architecture.

In this way, the Communication Agent will delegate on a set of Individual Communication Agents dedicated to each student. The students can choose among several interface devices for the interaction with the environment, ranging from the simple monitor+mouse+keyboard combination to the most complex and immersive virtual reality combination: head mounted display + motion tracking + data glove + voice recognition. There is a set of Device Agents to manage the different devices that can be used to interact with the environment and make the system independent of any specific combination of interaction devices. There is also a Connection Manager Agent which is responsible of coordinating the connections of the students.

The Student Modeling Agent is assisted by:

- A Historic Agent, which is responsible of registering the history of interactions among the students and the system.
- A Psychological Agent, which is responsible of building a psychological profile of each student including their learning style, attentiveness, and other personality traits, moods and emotions that may be interesting for adapting the teaching process.

- A Knowledge Modeling Agent, which is responsible of building a model of the student’s current knowledge and its evolution.
- A Cognitive Diagnostic Agent, which is responsible of trying to determine the causes of the student’s mistakes.

The World Agent is related to:

- The Objects and Inhabitants Information Agent, which has geometrical and semantical knowledge about the objects and the inhabitants of the world. This agent, for instance, will be able to answer questions about the location of the objects or their utility.
- The Interaction Agent, which has knowledge about the possible interactions with the environment and the effects of those interactions. For instance, it will be able to answer questions like “What will it happen if I push this button?”
- The Path Planning Agent, which is capable of finding paths to move along the environment without colliding with objects or walls.

The Expert Agent, in turn, is related to other agents that are specialists in solving problems related to the subject matter that is being taught to the students. This is one of most variable components in an IVET. Underlying the virtual environment, one or more Simulation Agents are in charge of simulating the behaviour of the system that is represented through the Virtual Environment (VE) (for example the behaviour of the nuclear power plant). In many IVETs the goal of the system is to train students in the execution of procedures. In our prototype for nuclear power plants, for instance, the goal is to teach a team of operators to execute some maintenance procedures. In this case, the Expert Agent should be able to find the best procedure to solve a given malfunctioning situation, and this is achieved by a Planning Agent that is able to apply intelligent planning techniques like STRIPS. If the IVET was to be used for teaching

Chemistry, for instance, the Expert Agent should have knowledge about the chemical elements and should be able to plan and simulate reactions.

The Tutoring Agent, finally, will be assisted by:

- A Curriculum Agent, which has knowledge of the curricular structure of the subject matter.
- Several Tutoring Strategy Agents, which implement different tutoring strategies.

Figure 3 shows how the extended ITS architecture is transformed from a modular point of view to an agent-based architecture.

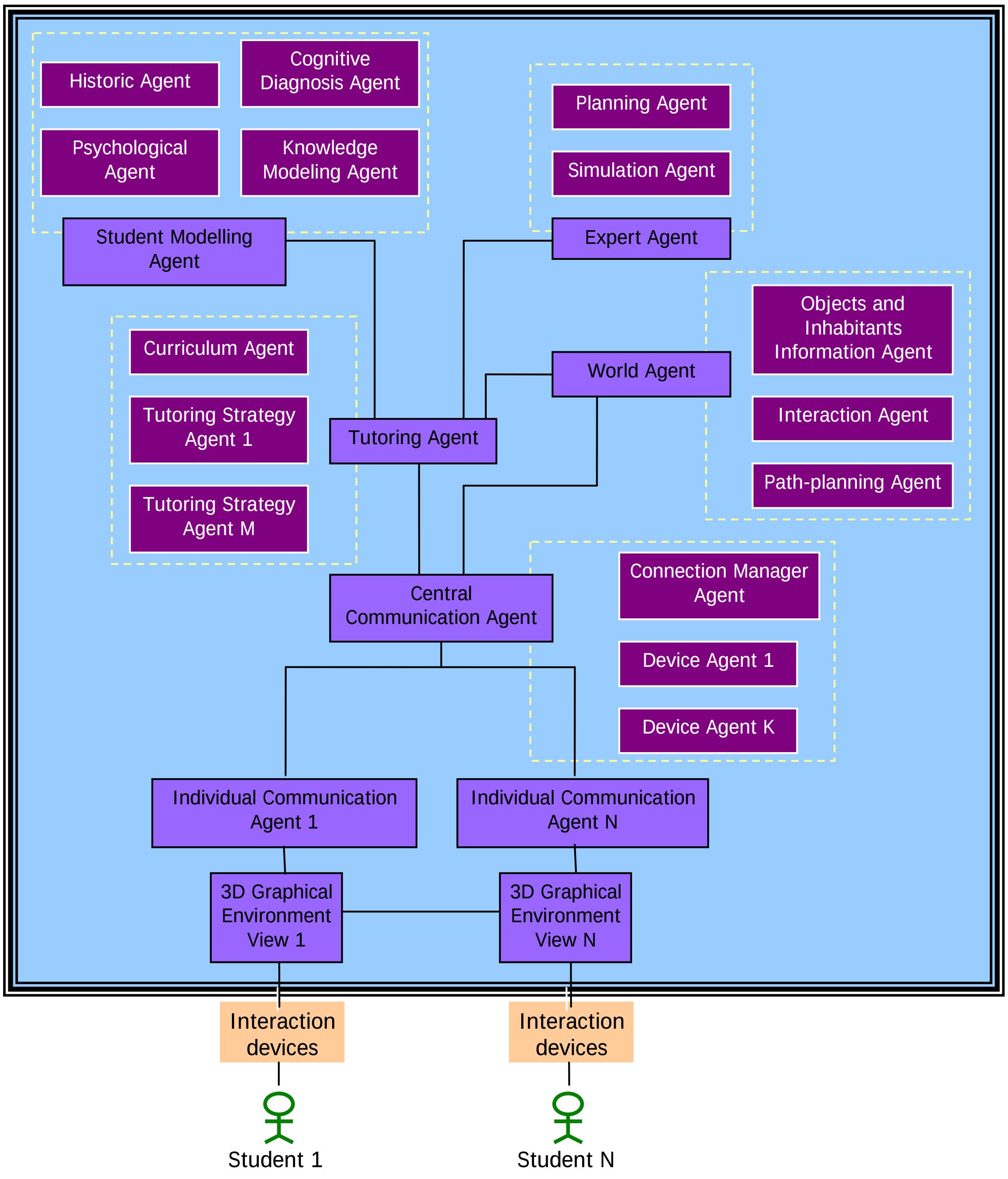

Figure 3. Agent-based architecture for IVETs

In the next sections we are going to discuss in more detail some of the more important aspects and functionalities that have to be considered in the design of an IVET, and how we have dealt with them in our agent-based architecture. Throughout these sections more details will be provided about the role played by some of the agents and the interactions that will be established among different agents.

MANAGEMENT OF MULTIPLE VIEWS

The fact that multiple users will be simultaneously connected to the system poses interesting challenges to the system architecture. As we mentioned before, each student will be provided with a particular view of the VE. We assume that each student will be represented in the environment by a graphical avatar, and his/her point of view will be located on the avatar's eyes. The interaction of the student with the environment will be performed mainly with his/her hands (pushing buttons, picking up objects, etc.). In our prototypes we have decided that one student will only see the hand/s of his/her avatar, while he/she will see the full body of the other students' avatars.

In order to build and update a given 3D Graphical Environment View, we need several essential pieces of information:

- The position and orientation of the student within the VE
- The direction of the student's gaze
- The position and gesture of the student's hand/s
- The position of other students
- The actions performed by other students

One possibility for dealing with these information requirements in the system architecture is to have a centralized component that collects this information from every student, builds a common representation of the environment and then sends to each student's view the updates. This task could be performed by the Central Communication Agent in the architecture of figure 3. The problem is that the centralized component becomes a bottleneck and has to deal with synchronization and consistency problems. Other possibility is to have all the student's views communicate directly among them. Whenever an important event occurs in one of the views, the important information is broadcasted to the other views. This option is illustrated by the link among the 3D

Graphical Environment Views in figure 2 and 3, and it is the one that we have chosen to optimize the performance of the system.

INDIVIDUALITY VERSUS COLLECTIVENESS

Having multiple simultaneous users raises a new question that needs to be addressed in the design of an IVET: the degree of individuality versus collectiveness that will be allowed in the tutoring component. In the most general case we can find a system that can be used both and simultaneously by individual students and by teams. In our prototype for nuclear power plant operators, for instance, there must be a first phase in the training in which each individual student uses the system to acquire or confirm general knowledge about radiation and radioprotection. Then, several activities can be posed that require the intervention of a team of operators. Two issues have to be taken into account in this kind of situation:

- First, how to coordinate the different students that conform a team, so that one team activity can be initiated. Different students may be located on different places, they may connect to the IVET at different times, and their learning process may be on different stages. On the other hand, different students may have different profiles and learning goals. For instance, one operator may be learning to operate a crane while other operator may be learning to measure radiation.
- Second, how the tutoring component is going to supervise the activity of individual students and teams. Are all of them going to share a tutor or is it going to be only one tutor for all the students? And what is the effect of this decision on the architecture of the system?

In order to solve the first question, the Curriculum Agent must know, for each collective activity, the number of students involved and the role to be performed by each of them. In turn, the Knowledge Modeling Agent must have knowledge about the learning profiles or roles of each student. Then, when a new student tries to connect to the IVET, the Connection Manager Agent asks for the name or identification of the student and informs the Tutoring Agent. If the Tutoring Agent decides that some students are ready to learn a certain activity, it asks the Curriculum Agent for the number and roles of the participants involved in the activity. Each student must choose one participant with one of his/her roles. . Then we can have two possibilities:

- to wait for the required number of students with the proper roles to be connected to the system,
- to substitute any missing student with an Student Role Agent that has been programmed to be able to play that role.

In the first case, the Connection Manager Agent is endowed with the goal of registering newly connected students for the pending activity, after checking with the Tutoring Agent if they are ready to learn that activity, and informing the Tutoring Agent as soon as the required number of students with the proper roles is connected.

This solution has the disadvantage of having to adapt the learning speed of one student to the others', but the advantage of promoting real team learning and social exchange among the students.

The second option may be advisable if the learning speed of the students is very different or if there is a role in one activity for which there is not any student enrolled in the course.

Regarding the issue of having only one or many tutors, we believe that having many embodied tutors moving around the environment, one for each student, may be very disturbing. On the other hand, each student should receive individualized advice. Having only one embodied tutor that has to supervise and talk to many students may imply that the tutor is all the time running from one student to another. We have chosen an intermediate solution. Since each student has a particularized view of the system, it is possible to show something only in one student's view but not in the others'. Then, each student will only see one tutor that is dedicated to him/her. The tutor will follow the student along the VE, it will observe the student's actions and it will talk to him. For one student it will look as if the other students were not being supervised.

PROBLEM SOLVING WITHIN THE ENVIRONMENT

The main kind of learning activity in an IVET consists of the tutoring agent describing an initial state of the system to the team of learners and asking them to find a way to reach a desired goal. For instance, the tutoring agent may ask the team to stop the reactor, or to change a contaminated filter. Solving the problem will require each of the team members to execute a certain set of actions in an appropriate order. For instance, changing the filter requires a cleaning operator to enter the controlled area and clean the surroundings, a radioprotection operator to measure the radiation level close to the filter at certain points during the change procedure, a couple of mechanical operators to disassemble the filter cartridge and extract the filter, and so on.

A straightforward solution for the Expert Agent is to have a predefined plan or sequence of actions for each possible problem. The Tutoring Agent will then have to check whether the students' actions adjust to the plan or not. However, this solution restricts the number of possible problems that the tutoring agent can pose to the student to the ones that have predefined solution plans. A more critical drawback of this solution is that many times different plans may be valid to reach the desired goal (even if they are not equally optimum in terms of time spent or radiation exposure, for instance). Whenever a student executes an action that was not in the predefined solution plan, the system should be able to determine whether it is possible to reach the desired goal from the resulting state or not. The kind of tutoring action to be taken greatly depends on this.

The only way to make the system flexible enough to deal with a possibly unlimited number of problems and with unpredicted student actions is to provide the Expert Agent with the capability of finding the solution for any problem in real time. The expert in solving planning problems will be the Planning Agent (a planning problem can be defined as finding an optimal sequence of actions to reach a desired goal state from a given initial state).

Initially, the Tutoring Agent is interested in finding out a plan to reach a certain final state in the VE from the current state. The plan consists of a sequence of actions that the student can perform in the VE. Three kinds of agents will be involved in the planning process: the Tutoring Agent, the Planning Agent and what we call Action Agents. Each Action Agent is specialized in a certain set of goals, so that it knows one or more actions that can satisfy each one of goals belonging to this set. As a working hypothesis, we assume there is not more than one Action Agent that can satisfy a goal (hypothesis 1). In our system, we have three Action Agents: the Simulation Agent, the Path-Planning Agent and the Interaction Agent. The interaction among these agents will be carried out by means of a shared blackboard and asynchronous message passing. During the planning process, the Planning Agent will coordinate Action Agents.

The Path-Planning Agent can determine whether the avatar that models the student in the VE can walk from a position of the VE to another position of the VE. Hence, this agent will be in charge of satisfying goals of the type *Is_In_Position(X, Y)*, and for that it will use the action *Move_To*$((X_i, Y_i), (X_f, Y_f))$. Although the VE is a 3D virtual world, the displacements of the avatar will always be done over a floor or plane. For that reason, we will only use two coordinates to specify the position of the avatar. Besides, as we assume it is always possible to move the mannequin from a position to any one in the VE, the preconditions of the operator *Move_To*$((X_i, Y_i), (X_f, Y_f))$ is true.

In addition, the actions of the Interaction Agent are the basic actions that the avatar can do in the VE except for moving from a position to another one, for instance, press a button, pick up an object, insert a card in a card reader, etc. In order to make hierarchical planning possible, some basic actions can be grouped into tasks or higher-level actions. The Simulation Agent will use the latter type of actions. Lets see an example to clarify the difference between basic actions and tasks. We suppose it is necessary to raise the temperature of a reactor in a nuclear power plant, and to carry out that, an operator must go to the control room and press a certain button. We consider *raise the temperature of the reactor* to be a task, and *go to the control room* and *press button X* to be basic actions.

The planning process is inspired in the STRIPS algorithm (Fikes and Nilsson, 1971). However, our planner accomplishes a breadth search in the state space instead of a depth search; the reason of this is that we are not interested in obtaining any plan, but the best plan. To implement the breadth search, the planning process will maintain a search tree in which each node will include a stack of goals and actions in STRIPS style, a state, and the plan to reach this state (see figure 4). In some domains, the computational cost of building such a search tree may be too high; therefore we will work with domains in which the size of the state space is manageable. In order to allow the agents to do concurrent operations over the search tree, the tree is encapsulated in a blackboard.

The planning process will begin when the Tutoring Agent introduces in the blackboard an empty parent node, and a child node for each different order of the goals that describe the desired final state and are not in the initial state. In addition, each child node will contain the description of the initial state and an empty plan. Then, the Tutoring Agent asks the Planning Agent to begin the planning process. Next, the

Planning Agent notifies the Action Agents that there are new goals in the blackboard (the tops of the stacks in the leaf nodes of the tree). We call these goals *active goals*. Now, the Action Agents read all the active goals, and check whether they can satisfy any of them by using one of the actions that they know. It is noteworthy to mention that these read operations can be executed concurrently. If an Action Agent can satisfy an active goal by means of an action, this agent will carry out the following steps:

1. Add child nodes to the node that comprises the active goal; a different child node will be created for each possible different order of the preconditions of the action. If any child node is already present in the search tree, it must not be added to the tree.
2. For each child node:
    - 2.1. The ground action, the ground conjunction of the action preconditions and a different order of the ground preconditions are pushed in its stack.
    - 2.2. Check whether the goal in the top of the stack holds in the state included in the node. If it does, the goal is deleted, and the step 2 is repeated if the top of the stack is a goal. Otherwise, if the goal does not hold, the step 3 is executed.
    - 2.3. If the top of the stack is a conjunction, check whether the conjunction holds in the state included in the node. If it does, the conjunction is deleted. Otherwise, a *fail node notification* is sent to the Planning Agent, and this operation ends.
    - 2.4. If the top of the stack is a ground action, the action is removed from the stack, it is introduced in the plan of the node, and it is applied to the state of the node; next, go to step 2.2.
3. Notify this operation to the Planning Agent (*satisfied goal notification*).

In the step 2.4 we must distinguish a particular case related to the action *Move_To*$((X_i, Y_i), (X_f, Y_f))$. This action will be not completely ground in the stack, but only the latter two arguments will be constants. This is due to the fact that this action is used to satisfy a goal with the scheme *Is_In_Position(A, B),* so X_f/A and Y_f/B. In order to bind the free variables of the action before introducing it in the plan, the plan will be examined to find the last bound action *Move_To* included in the plan. From this bound action, the latter two arguments (constants) will be extracted, and they will be used to bind the free variables. Otherwise, that is, if there is not a bound action *Move_To* in the plan, the initial position of the avatar will be used to bind the free variables.

It is remarkable, thanks to the hypothesis 1, several agents may execute these steps over the blackboard concurrently because different leaf nodes are involved. On the other hand, if a Action Agent cannot satisfy a certain active goal, the agent will report it to the Planning Agent. In this way, the Planning Agent will be able to find out about whether there is a node in the tree whose active goal cannot be satisfied by any agent (*fail node*). In this case, the Planning Agent will delete the fail node, performing the backtracking; if the deleted node is the last not-empty node, the Planning Agent will detect the initial problem has no solution. Upon the Planning Agent receives a satisfied goal notification, it will tell Action Agents that the set of active goals has been modified. Then, as soon as possible, the Action Agents must read the new set of active goals. It could happen that a Action Agent is using an obsolete set of active goals. However, it is not difficult to see this situation will not be problematic due to the hypothesis 1. Moreover, the Planning Agent must be able to notice the search tree has been completed. In that situation, the set of active goals is empty; hence, when the Action Agents try to obtain this set, after receiving a satisfied goal notification, they will obtain an empty set. Then, they have to notify this event to the Planning Agent, so that this agent will be able to determine the best plan for the initial problem. Finally, the Planning Agent will report the best plan to the Tutoring Agent.

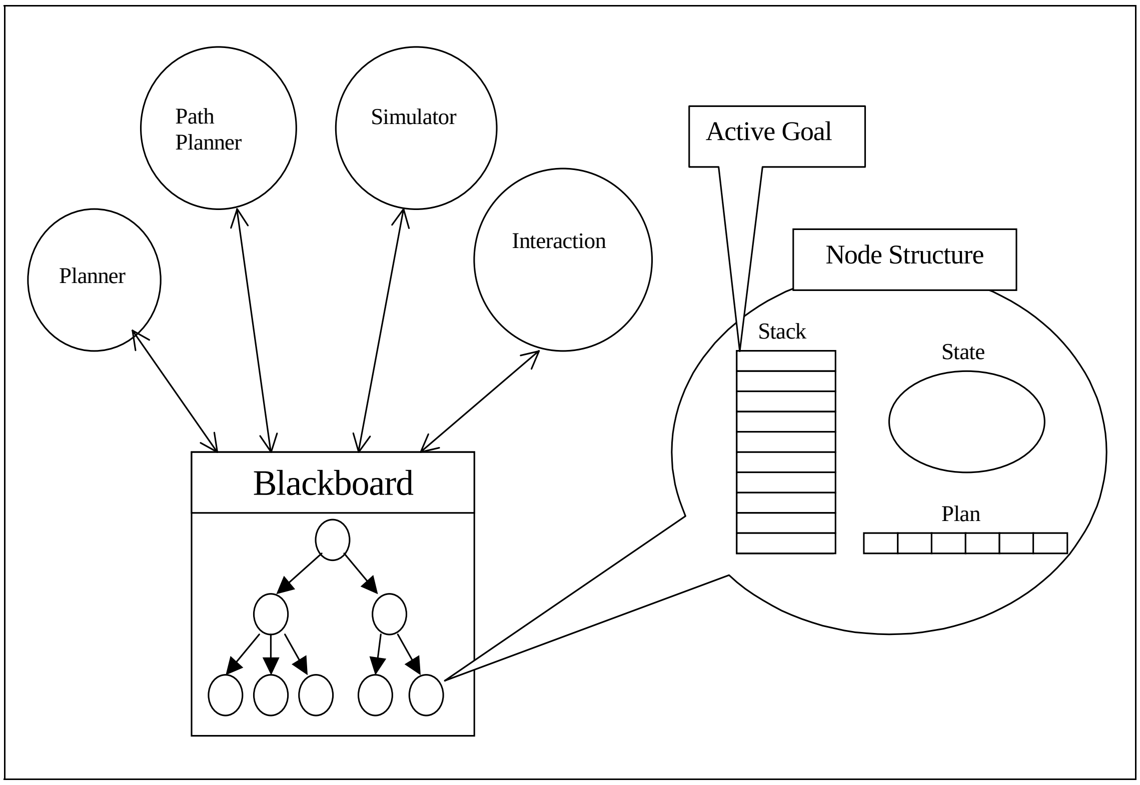

Figure 4. Planning Agents and Blackboard

According to the explanation presented above, the blackboard must support four operations:

- Initialize the blackboard.
- Obtain the active goals that were not examined by a certain agent yet.
- Satisfy some active goals: for each goal, a ground action must be provided.
- Obtain the best plan.

PATH PLANNING

After obtaining the plan, the tutoring agent will have the sequence of actions that each student has to carry out in the VE to solve a problem. However, the tutoring agent will also need to know the best trajectory for each displacement that the avatar must do in the VE. In this way, the tutoring agent will be able to check whether the trajectory

followed by the student in the VE is acceptable, that is, whether it is close enough to the best trajectory for each movement. The agents in charge of generating the trajectories will be the Path-planning Agent and the Objects and Inhabitants Information Agent.
The Objects and Inhabitants Information Agent contains a geometrical model of the VE expressed by means of several graphs. These graphs will have been obtained prior to the planning process from geometrical information related to the VE.
We assume the VE to be divided into several rooms (sub-environments) joined by doors. Then, we use a graph to model the accessibility among the rooms (*Environment Graph*). Furthermore, we use a different graph for each room to model the accessibility of the different positions (*Room Graph*). In order to obtain the graph of each room, we model each room as a 2D-grid in which each free square corresponds to a node of the graph. We decide whether a square is free by using the projection over the floor of all the objects existing in the room. In this way, if a part of the projection of an object is inside an square of the grid, this square will not be free. Besides, an edge between two nodes A and B is added to the graph if the two squares associated to the nodes share a side in the grid. The size of the squares must be adjusted correctly. Otherwise, if the squares are too big, some feasible trajectories may be considered wrong; or if the squares are too small, the computational cost of the search process that will be explained later will increase unnecessarily. In addition, the process to obtain the graphs from the 3D information of the VE can be done automatically.
The trajectory for each 2D-movement included in the plan will be determined after working out the whole plan. Before starting to generate the trajectory for a certain 2D-movement, it will be necessary to update the graphs according to the actions to be executed prior to the 2D-movement in the plan. For example, if the position of a table is changed, the previous position of the table may be traversed by a trajectory afterwards.

As the Planning Agent does not know the semantics of the actions, it will tell the Interaction agent to decide if it is necessary to update the graphs according to each bound action. In turn, the Interaction Agent will notify the Objects and Inhabitants Information Agent the changes in the position of the objects in the 3D-world, so that the Objects and Inhabitants Information Agent can modify the graphs.

After the update of the graphs, the Path-planning Agent must find out whether the movement traverses more than one room. If it does, the Path-planning Agent will use the Environment Graph to obtain, if it exists, the sequence of rooms that the avatar must traverse from his initial position to his final position. Next, the trajectory across each room must be obtained. For this, the Room Graphs will be used. If the movement traverses just one room, the unique graph to use will be the graph associated to this room. In order to work out the path between two nodes in the graphs, the A* algorithm (Hart et ál., 1968) is applied under the assumption that all the edges' weights are one, and using as heuristic function the euclidean distance. Each time the Path-planning Agent needs to know the nodes directly connected to a node in a graph, it will ask it to the Objects and Inhabitants Information Agent.

The A* algorithm outputs a sequence of nodes that the Path-planning Agent will translate into a sequence of points or trajectory in the VE with the help of the Objects and Inhabitants Information Agent. This trajectory will be saved into a XML file, so that the Tutoring Agent can employ this information to supervise the movements of the students later on.

COORDINATION AMONG AGENTS DURING THE SUPERVISION STAGE

Once the expert solution for a given activity has been calculated, the IVET enters the Supervision Stage, in which:

- the Individual Communication Agents will observe the behaviour of the different team members and will inform about it to the Tutoring Agent through the Central Communication Agent,
- the Tutoring Agent will compare the real behaviour to the expected behaviour provided by the Expert Agent and it will evaluate the adequacy of each student's behaviour,
- the Historic Agent will register the actions performed by each student,
- the Knowledge Modeling Agent will infer and model the state of knowledge of each student, with the help of the Cognitive Diagnostic Agent in case of errors.
- the Psychological Agent will use the behaviour of the student to infer psychological chararacteristics,
- the Tutoring Strategy Agent will decide on the next step to be taken, considering the last actions of the student (provided by the Historic Agent), his/her state of knowledge (provided by the Knowledge Modeling Agent), his/her psychological state (provided by the Psychological Agent), and the learning objectives and structure of the subject matter (provided by the Curriculum Agent). The decision might be to wait for new actions of the student, to use the embodied tutor to give a hint, to congratulate the student, to explain why something was wrong, etc.

DEVELOPMENT OF A NEW IVET

One of the advantages of the proposed architecture is that it has allowed us to build a basic infrastructure of agents that work as a runtime engine. In order to develop a new IVET, it will be the author's responsibility: to select the desired agents among the available ones (for instance selecting the Tutoring Strategy Agent that implements the desired tutoring strategy); to configure the parameters that govern the behaviour of those agents (for instance the duration of the session, the number of mistakes that will be allowed before the Tutoring Agent tells the student the correct answer, etc.); to provide the data specific to the new IVET and subject matter (the geometrical model of the VE, the curriculum, the actions that are possible in the new VE and their effects on the simulation, etc.); and in the worst case to create new agents and register them in the platform.

The requirements for the new IVET under development should be driven by real world studies (Economou et ál., 2001), and these requirements will drive, in turn, the design decisions regarding the configuration and adaptation of agents.

As a prototype application of our tool we have developed a training system for nuclear power plants operators. We had previously developed this system from scratch in 1999, during a one year period. The re-development using our infrastructure has just taken a few weeks, and the achieved functionality is superior. For instance, the previous implementation was for only one user, the tutor was not embodied, and the communication tutor-student was restricted to correction feedback. The decrease in development time and the increase in functionality suggest that we have successfully achieved our aim. A thorough experimental evaluation of the effectiveness of the solution is out of the scope of this paper.

PRACTICAL REALIZATION

The agent-based architecture for VETs that was described in the previous sections has been implemented with a combination of quite heterogeneous technologies.

The agents have been implemented in Java, and the direct communication among them has been realized using the Jade platform with FIPA ACL messages. The 3D VE and avatars have been modeled with 3D Studio Max and imported into OpenGL format with a specific tool developed for that purpose. The visualization of the 3D models, animations and interactions are managed by a program in C++, making use of the

OpenGL graphical library. Microsoft’s DirectPlay library has been used for direct communication among the different 3D graphical environment views in order to take into account the movements and actions of the other students for real-time update of each view. Microsoft’s DirectInput has been used to manage some interaction devices, namely the mouse, keyboard and joystick. The head mounted display and data glove inputs and outputs are managed by specific libraries. Communication among the C++ VE and the Java agents is performed by using a middleware of CORBA objects.

CONCLUSIONS

An agent-based architecture is proposed in this paper for the design of Intelligent Virtual Environments for Training. The roots of this architecture are in the generic architecture of an Intelligent Tutoring System, that has been firstly extended to be applicable to IVETs, and has been then transformed into an agent-based architecture by the identification of the set of generic agents that would be necessary to accomplish the tasks of each module. Some details are provided about the most remarkable interactions that will be established among agents during the system’s execution, namely the collaborative planning to find expert solutions to the situations and goals posed to the students, the determination of trajectories for the movements across the VE, and the supervision of the student’s behaviour. Some peculiar aspects are also discussed, such as the way to manage multiple views in a multi-user environment or the way to balance the individual versus the collective aspects in the system’s functioning.

The proposed architecture, and its realization in a platform of generic and configurable agents, will facilitate the design and implementation of new IVETs, maximizing the reuse of existing components and the extensibility of the system to add new functionalities.

REFERENCES

Antonio, A.D., Ferre, X., & Ramirez, J. (2003). Combining virtual reality with an easy to use and learn interface in a tool for planning and simulating interventions in radiologically controlled areas. 10th International Conference on Human - Computer Interaction (HCI 2003), Creta, Greece.

Alpdemir, M.N., & Zobel, R.N. (1998). A component-based animation framework for DEVS-based simulation environments. Simulation: Past, Present and Future. 12th European Simulation Multiconference (ESM'98), 79-83. Manchester, UK.

Brown, J., Encarnaçao, J., & Shniderman, B. (1999). Human-Centered Computing, Online communities, and Virtual Environments. IEEE Computer Graphics and Applications, 19(6), 70-74.

Darken, R., Tonessen, C., Passarella, & Jones K. (1995). The bridge between developers and virtual environments: a robust virtual environment system architecture. Proceedings of the SPIE – The International Society for Optical Engineering, vol.2409, 234-240.

Demyunck, K., Broeckhove, J., & Arickx, F. (1999). Real-time visualization of complex simulations using VEplatform software. Simulation in Industry'99. *11th European Simulation Symposium (ESS'99)*, 329-333.

Economou, D., Mitchell, W.L., Pettifer, S.R., Cook, J. & Marsh, J. (2001) User Centred Virtual Actor Technology. Proceedings of Virtual Reality, Archaeology and Cultural Heritage (VAST'2001), Athens, Greece, European Association for Computer Graphics.

Fencott, C. (1999). Towards a Design Methodology for Virtual Environments. Workshop on User Centered Design and Implementation of Virtual Environments. University of York.

Fikes, R., & Nilsson, N. (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. Artificial Intelligence 2, 1971.

Hart P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Miniman Costs Paths. IEEE Transactions Systems Science and Cybernetics 4(2).

Méndez, G., Rickel, J., & de Antonio, A. (2003). Steve Meets Jack: the Integration of an Intelligent Tutor and a Virtual Environment with Planning Capabilities. 4th Internacional Workshop on Intelligent Virtual Agents (IVA'03).

Munro, A., Surmon, D.S., Johnson, M.C., Pizzini, Q.A., & Walker, J.P. (1999). An open architecture for simulation-centered tutors. Open Learning Environments: New Computational Technologies to Support Learning, Exploration and Collaboration. (Proceedings of AIED99: 9th Conference on Artificial Intelligence in Education), 360-367. Le Mans, France.

Rickel, J., Johnson, W.L. (1999). Animated agents for procedural training in virtual reality:Perception, cognition and motor control. Applied Artificial Intelligence 13, 343–382.

Rickel, J., Johnson, W.L. (2000). Task Oriented Collaboration with Embodied Agents inVirtual Worlds. MIT Press.

Sánchez, M.I. (2001). Una Aproximación Metodológica al Desarrollo de Entornos Virtuales, Ph.D. dissertation, Universidad Politécnica de Madrid, Spain.