How Do Agents Affect Modifiability? A Comparison between Two Architectures for Intelligent Virtual Environments for Training

Gonzalo Méndez and Angélica de Antonio

Computer Science School
Technical University of Madrid
gonzalo@gordini.ls.fi.upm.es, angelica@fi.upm.es

Abstract. The use of agents is spreading as a means to develop different kinds of software systems, among which we can find Intelligent Virtual Environments for Training. The agent community has already started to pay attention to software engineering issues to develop agent-oriented systems, but they are mainly focused on methodologies and, to some extent, design patterns. However, not much attention has been paid to software architecture for the moment. We compare two agent-based software architectures for Intelligent Virtual Environments for Training that are intended to be easily extended and modified. The first one was designed using an organizational approach recommended by some agent oriented methodologies. The second one is a redesign of the first architecture using more formal principles and methods of software architecture design. A comparison between both architectures highlights the need to pay more attention to software architecture design in this field.

1 Introduction

An Intelligent Tutoring System (ITS) is an application of computer science to education that has a particular structure shown in Fig 1 [1,2]. They are different from other educational software in that they are *intelligent*, since their purpose is to adapt teaching to the abilities and characteristics of every student. In addition, their structure was thought to make it possible to easily change the teaching domain (expert module), the tutoring strategy (tutoring module) or the way students are modeled in the system (student module). However, the biggest success of ITSs is the fact that a great deal of researchers in educational software use the structure shown in Fig. 1, with research groups specializing in the development of each of the different modules.

Educational Virtual Environments are software systems that make use of three dimensional Virtual Environments (VEs) for education and training. Their development has a quite short history, dating from the mid-nineties, and in some cases they have evolved from the necessity to integrate a Virtual Environment with an ITS. The use of Virtual Environments in fields such as military or industrial training has proven to be a very promising application area, so part of

R. Morrison, D. Balasubramaniam, and K. Falkner (Eds.): ECSA 2008, LNCS 5292, pp. 82–97, 2008.

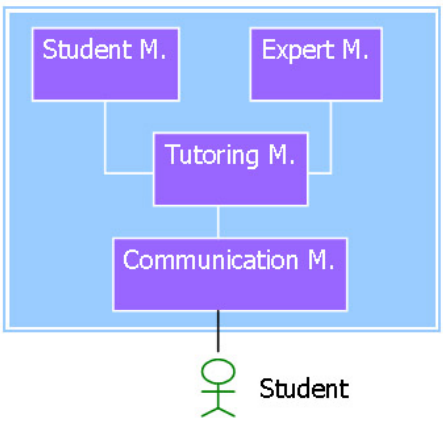

Fig. 1. Architecture of an ITS

the education community are making use of them to develop Intelligent Virtual Environments for Training (IVET), which are the conjunction of an ITS and a Virtual Environment for training.

The youth of the field, together with the complexity and variety of the technologies involved, have led to a situation in which neither the 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 specific solutions and monolithic architectures (even if they allegedly make use of the ITS structure), and in many cases forgetting the principles and techniques of the Software Engineering discipline [3].

The MAEVIF project (*Model for the Application of Intelligent Virtual Environments to Education*) was the result of several experiences integrating VEs and intelligent tutors [4,5] that served to point out the problems that commonly arise in such integrations. The objective of the MAEVIF project was to define a model for the application of intelligent virtual environments to education and training, which involved: the definition of a generic model for intelligent learning environments based on the use of virtual worlds; the definition of an open and flexible agent-based software architecture to support the generic model of an Intelligent Virtual Environment for Training; the design and implementation of a prototype authoring tool that simplifies the development of IVETs, based on the defined architecture; and the definition of a set of methodological recommendations for the development of IVETs.

In this paper we present two different approaches to the design of a software architecture for IVETs. The first one is the result of applying a specific Agent Oriented Software Engineering methodology, while the second is the result of applying more specific, architecture centric techniques. Our aim is to use our system as a case study for the application of general software engineering techniques to the development of agent-oriented software, since we believe the agent

community is not making enough use of the knowledge produced by the software engineering community in general, and the software architecture community in particular.

In the remainder of the paper we briefly describe the first version of the architecture and the results of evaluating it (section 2). Then, we describe how the agent-based software architecture has been designed using software architecture principles (section 3) and how it is being evaluated (section 4). After that, we present some related work where agents have been used to develop IVETs (section 5). Finally, we present some conclusions and ongoing work (section 6).

2 An Agent-Based Architecture for IVETs

There are two main reasons why we have chosen agents to develop this system instead of a more traditional approach, either object or component oriented. The first reason, as described in [6] is the fact that, given the increasing complexity that the development of IVETs involves, agents represent a powerful tool to use abstraction as a way to face complexity. In addition, the fact that many agent platforms are developed on top of object oriented languages makes it possible to take advantage of all the possibilities provided by these languages (i.e. JADE and Java).

The second reason is that, although a widely accepted definition of agent does not currently exist, many authors agree on a set of features that agents must have, among which we can find both proactivity and situatedness. Given the fact that IVETs are a highly interactive kind of application, proactivity is a feature that is very well suited for their development, since it facilitates the design of tutors that interact with the students in a human-like way. In addition, training inside an IVET makes it necessary for the tutor to be aware of the structure and state of the environment where that training is taking place. Therefore, situatedness is a feature that makes it possible to manage that information in a more natural way.

This does not mean that the mere use of agents is the solution for all problems, but properly used they are likely to ease the design and implementation of a suitable solution.

2.1 A Hierarchical Approach

Taking the structure described in the previous section as a starting point, the next step was to decide which software agents were necessary to transform it into an agent-oriented architecture, which has been designed using the GAIA methodology [6]. In this methodology, the authors suggest the use of the *organizational metaphor* to design the software architecture, which requires the analysis of the real world organization in order to emulate its structure. This approach does not always work (depending on particular organization conditions), but in this case, considering the architecture of an ITS as the organization to reproduce, it is possible to imitate its structure to develop the architecture.

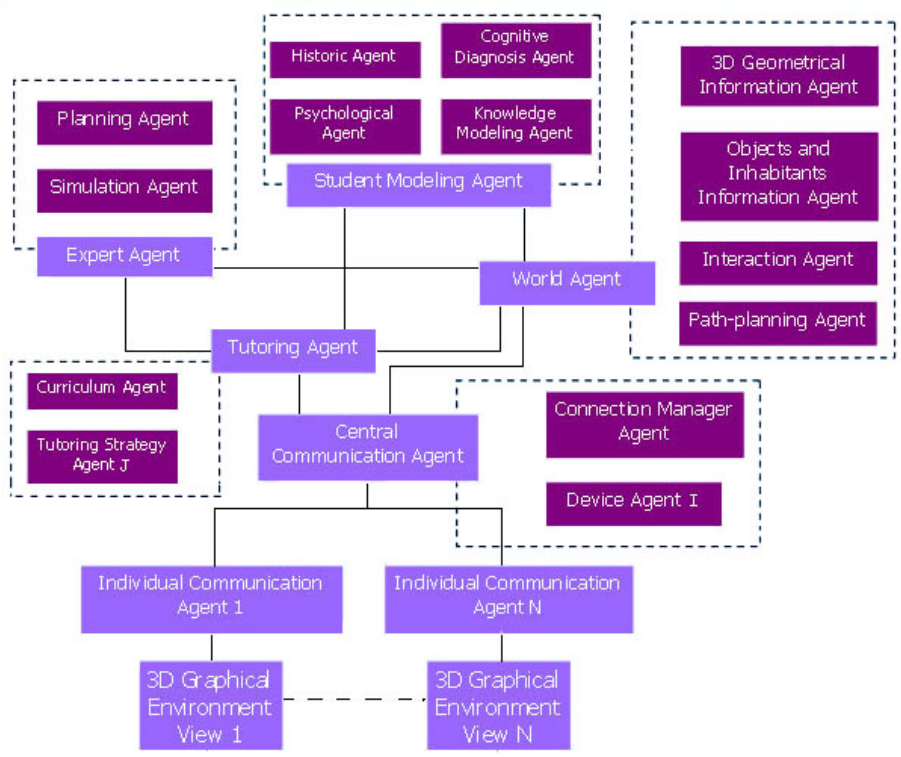

Fig. 2. Decomposition view of the agent-based architecture

There is an additional reason to use an ITS structure as a starting point: the ITS architecture shown in Fig. 1 is widely used by the educational software community. Therefore, if we aim at being able to exchange elements between different applications, making use of a widely used, well known structure is likely to facilitate this task.

The ITS architecture was transformed, from a modular point of view, into an agent-based architecture. It has five agents corresponding to the four modules of the ITS architecture plus an agent that represents the Virtual World: Communication Agent, Student Modeling Agent, Expert Agent, Tutoring Agent and World Agent.

Analyzing the responsibilities of these agents, some additional roles can be identified that point to the creation of new, subordinate agents that can carry them out, subsequently giving rise to a hierarchical multi-agent architecture. Each subordinate agent is in charge of managing some process and the information related to it, while each supervisor agent is in charge of coordinating its subordinate agents and communicating them with other subordinate agents through their respective supervisor. A decomposition view of this architecture can be seen in Fig. 2.

For more details about this architecture, we suggest reading [7] or a more detailed description in [8].

2.2 Discussion

All along the design and development of the architecture, one of the aspects that has had a bigger impact on it has been the planning process, since, due to the fact that it is a collaborative task, a change in the planning method or in the way that knowledge is represented may imply changes in all the agents that take part in it. At the beginning, a simple STRIPS planner [9] was implemented. However, trying to substitute it with one based on SHOP2 (Simple Hierarchical Ordered Planner 2) [10] showed that it was far more complicated and required more changes than expected.

Another aspect we tested was how easy it was to add new functionality to the IVET. To do this, we added an embodied tutor whose goal was to observe what happened in the VE and follow the student to supervise him. It was necessary to add two new agents and, although it was quite easy to make these changes, it soon became clear that any non-trivial change was likely to affect at least one of the supervisor agents (the one that supervises the modified agent) if not more. In addition, all the agents knew of the existence and identity of the agents they had to communicate with, so they were easily affected by changes.

There were some other factors that made us think that a redesign of the architecture was necessary, both at design and implementation levels. One of them was the poor performance the system offered when several students were taking part in a training session. Several tests pointed out that the agent platform presented a fairly good performance, and so did the VE. The problem arose when running both of them at the same time in different machines, which made us think of poor communication performance.

In addition, we were having problems when trying to add new functionality, since it was not clear whether some responsibilities were to be assigned to the expert agent or to the world agent, both of which started to be too coupled for the system to be modifiable. This is a problem that usually arises when establishing classifications and hierarchies: if the criteria used for classification changes or some elements dont fully fall under one of the categories, then the decomposition degrades quite quickly. This was the case with our architecture.

Finally, there were other facts that pointed out the unsuitability of the architecture. Among them, some are described in [11] as an indication of a poor design, such as the proliferation of agents to carry out small tasks, the difficulty to assign responsibilities to an agent or the existence of agents that carry out actions for which an agent is not needed. These problems were caused, at least partially, by the lack of an architectural design method in a not very mature field like IVETs.

3 Architectural Redesign

The main theoretical support to redesign the architecture has been the body of work on software architecture developed at the Software Engineering Institute (SEI) [12,13], such as Attribute Driven Design (ADD) and Architecture Tradeoff Analysis Method (ATAM). Their purpose is to design and evaluate a software architecture driven by the quality attributes desired for it, instead of only the functionality. A set of scenarios is used to help identify the quality attributes that are relevant for the architecture, based on the stakeholders interests, and to evaluate the architecture in order to identify potential risks.

The other important support has been provided by the use of *information hiding* [14]. The design decisions that are encapsulated in each module are related to the changes that are perceived to be likely over the system's life. The way to design is to use abstraction as a means to face complexity and facilitate changes.

Although we planned to use ADD as the architectural design method, we discarded it after a few design sessions because of two reasons. The first reason is the fact that ADD is based on a hierarchical system decomposition, and one that does not allow elements to have more than one father. After the experience gained with the previous architecture, we did not think a hierarchical structure was what we wanted to obtain. In addition, one of the problems we had with the first architecture was the fact that some agents were not clearly under the supervision of one of the five supervisor agents. The second reason has to do with the complexity of decomposing a module in more than four or five elements, which was likely to be the case (in the current design, the ITS consists of nine different kinds of agents).

We soon found two more grounded reasons to discard ADD as a design method. The first one is Parnas argument expressed in [15], where he clearly states that, although maybe desirable, information hiding and hierarchical structure do not always go together. On the contrary, we consider information hiding to be a design criterion, and not just a decomposition one. The other reason can be found in [16], where the author analyzes Simon's "Architecture of Complexity" and identifies the historical reasons that made hierarchical structure a predominant design mechanism.

Agent systems are intrinsically peer-to-peer (after all, they were born in the distributed artificial intelligence field), where each agent is a peer that makes use of services offered by other agents to carry out the responsibilities assigned to it. Therefore, this is the approach we have followed to design the new architecture.

Like ADD suggests, we have started by selecting the architectural drivers for our application, some of which were very well identified thanks to the analysis of the previous system. However, instead of following a decomposition approach, we have worked using an iterative an incremental approach.

If we think of a system we need to extend, we don't face this task by decomposing some part of the system. Instead, we usually try to make the new parts fit using the architectural mechanisms we used to design the existing architecture. Thus, this is the approach we have followed. We have started by designing a core

architecture with a very small functionality, and we have proceeded by adding new functionality in each iteration.

As Haythorne states in [17], a system is only modifiable in the points that are designed to accept modifications, and we can have such a design only if we know the modifications we expect to happen. For the initial architectural design, we had a fairly clear idea of the modifications we were going to add (those corresponding to the functionality we wanted the system to have), so in each iteration we proceeded by adding one of the modules we had identified as likely to be substituted, along with what the existing modules would need to use from the new ones. Every time a feature had to be added, or a change had to be made, it was tested against the architectural drivers until a way was found to satisfy them. At that moment, the change was added to the architecture.

This approach to the architectural design has been useful in two ways. First, it has allowed us to make sure the architectural drivers have been taken into account or where and why they have not. Second, it has also made it possible to identify existing dependencies between agents, so we have obtained a list of possible changes we may have to make when substituting a specific agent.

In addition, this design method has allowed us to test the service oriented approach we have used to design the system (which we describe below) and we have been able to make sure that, with the information we have about the system and possible changes, the architecture may be easily modified to include them.

This is probably one of the issues ADD still has to address, since it is not always possible to face a new design or modification as a (hierarchical or not) decomposition.

3.1 Quality Attributes

The design started with the definition of a set of quality scenarios to establish what kind of changes were to be considered by the design. In general, this changes have to do with the ability to substitute an agent with a different one that provides a similar functionality, or to move some responsibility from one agent to another. This is required because one of the objectives of the system is to be used as a test-bed for teams developing just some of the elements of the ITS (e.g. the student modelling or the tutoring strategy).

Another kind of change is the possibility to turn off some functionality, such as supervision, so that the student can use the system in an exploratory way without the tutor interrupting him (although the system would continue registering his actions), or even disabling tutoring completely.

The system is also required to be easily extended, so that new agents that provide new functionality can be added without having to make changes in the existing ones (at least, in the ones that don't make direct use of the new functionality).

Taking into account that training is carried out in a VE, all these modifiability requirements cannot be an obstacle for the main objective of the system, which is to provide students with a training environment as similar as possible to the real one. For that, it is of utmost importance to keep performance close to real

time. If not, training in the IVET may be somehow frustrating for the student, which may cause the training experience to be less efficient than other, more traditional, methods.

There is a usability attribute, adaptation to the user, that has not been considered explicitly because it is already included in the features of an ITS. The student modelling is used to personalize the training process to the student's abilities and needs, so it has not been necessary to consider it as an additional quality attribute to take into account.

3.2 Design Decisions

With the described modifiability objectives in mind, the approach we followed was to keep the agents as anonymous as possible, so that no agent directly knows which agents are carrying out the actions they need to successfully complete their responsibilities. To achieve this, during system startup, the agents announce in the system's yellow pages the services they are capable to provide to other agents. Thus, an agent does not know how many or what kind of agents there are in the system; they just know that there is an agent that can provide a certain service they need. This way, it is easier to change the agent that provides a service, as long as the service is provided in the same terms the original one was.

Once the agent finds the service it is looking for, it can act in two different ways. If the service involves frequent updates, the agent subscribes to an update list, so that every time an update arrives, it is immediately informed about it. If, on the contrary, the agent only needs to request the service at specific times, it annotates which agent it has to request the service to. In both cases, the decision is made at runtime, so changes in the design are easier to carry out. We considered the possibility of giving the chance to change the service provider any time during runtime, but we discarded it because it is not likely to happen in a system of this kind.

The agents communicate with each other exchanging FIPA ACL messages. Since it is a quite extended formalism, the difficulties may come from the communication protocol. We have designed a fairly simple communication protocol for a given agent to request a service from another agent. Agent A sends a request to agent B, who acknowledges the reception of the request. Then agent B carries out the required actions and sends agent A the result of the execution of the service (or a message with the reasons why it could not be carried out). Agent A acknowledges the reception of the result and the communication stops until another service request is required.

A similar mechanism has been used to communicate the agent platform with the VE, but with an even simpler communication mechanism. A communication centre has been designed where both the agent platform and the VE send their messages for other applications to receive them. Each application subscribes to the messages it is interested in receiving, so that, for example, different VEs or different versions of a VE only receive the messages they know how to handle.

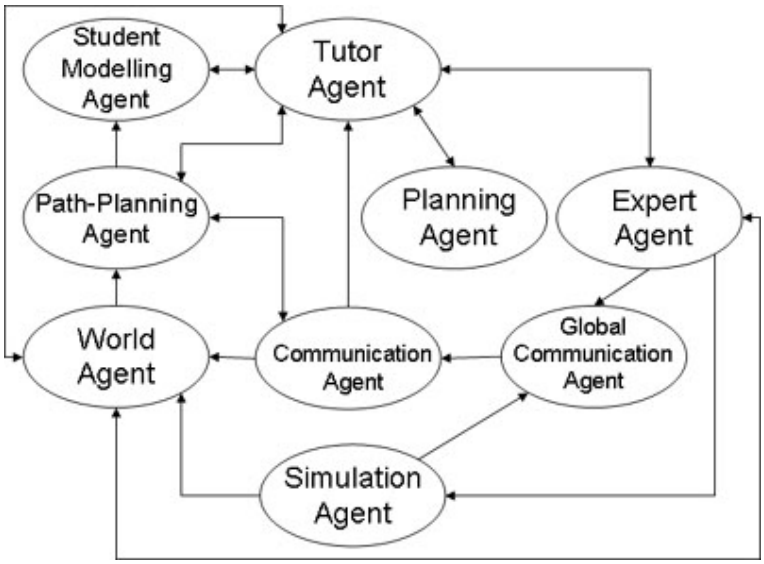

Fig. 3. Peer-to-peer view of the new architecture. Ellipses represent agents and arrows show communication channels.

We have used this possibility all along the development of the agent platform, so we could send the messages we wanted to test from a console instead of having to run the VE and carry out a specific procedure before the message we were interested in could be sent.

We have also made use of configuration files to set up the training session. Thus, the description of the procedures to be trained, the composition of the scenarios, the objectives of the activity, its participants or the parametrization of the tutoring strategy are all read from several configuration files, which allows changes in the way the system behaves without further changes in the design.

3.3 Resulting Architecture

The resulting architecture is the one shown in Fig. 3. The picture shows the structure of the architecture as it is currently designed, where all the agents are represented along with the communication channels (the yellow pages are not represented, since all the agents communicate with them).

During runtime, there is only one agent of each kind, except for the agents that are directly related with students: the Student Modelling Agent and the Communication Agent. This is so because, having several students, the system can handle the communication with them in parallel. In addition, if the agent platform needs to be distributed in different machines, the distribution can be made in terms of the number of students.

The main differences with the former architecture are:

- There is no hierarchical structure. Since the existence of the supervisor agents was due to modifiability reasons, and it was only achieved in a low degree, it has been considered preferable to use a peer-to-peer style.
- It uses a publish-subscribe style to offer a service oriented behaviour. Agents advertise their services in the yellow pages and other agents can subscribe to the services they are interested in. This is one of the mechanisms that introduces a higher degree of modifiability in the system.
- Task planning is not a collaborative task any more. The planning agent acts as a wrapper [18] for the planner, hiding details of its functioning to the rest of the system and enabling the change of the planing algorithm with a lower impact than collaborative planning had.
- Extended support for a simulator. Some systems such as the one described in [19] simulate environmental events that may be caused by external factors, such as changes in the state of a patient. In the cited system, events are directly simulated in the VE, but it may also be desirable to use an external simulator in cases where it has already been implemented or when it has a complex behaviour. The simulation agent can now simulate simple systems, but it can also receive information from a simulation running together with the VE or act as a wrapper of an external simulation.
- The tutoring strategy can be adjusted by changing some parameters that are read from a configuration file during the initialization of the system. These parameters are expected to change dynamically with the new design of the student modelling agent.
- The world agent is responsible for maintaining an ontology that stores the state of the VE. A simple reasoning engine has been added so that the world agent is able to provide richer answers to the student.
- A message centre is now used to communicate the different subsystems that form the training system. Each subsystem registers in the message centre and requests the kind of information it is interested in. Currently, in addition to the VEs and the ITS, a command line console can also connect to the message centre with debug purposes.

After the redesign, the implemented system has not shown any of the undesired properties the former one had. Performance is quite good, with no apparent latencies, two new agents and some new responsibilities have been quite easy to add and all agents seem to have well defined roles in the architecture. We have not been able to test the effects of changing the planner, since that change has not been required for the moment, although the evaluation we have run suggests it should not have a big impact on the architecture, given that it is now under the responsibility of a single agent.

There is one sensitivity point we have not been able to avoid, which has to do with the dependency there is between the tutoring agent and the student modelling agent. However, this dependency has to do with the original ITS architecture, and trying to solve it would require a completely different design approach that would probably make the substitution of these two agents a very problematic task.

4 Evaluation

In addition to the more formal design and documentation of the software architecture, we are currently evaluating the suitability of the architecture to our needs both at architectural and runtime levels.

At the architectural level, the evaluation requires the use of quality scenarios provided by the stakeholders to identify relevant quality attributes. We are trying to gather a thorough collection of scenarios that gives us a better understanding of the implications of the design decisions we have made. In order to do it, we have run an ATAM session and we are planing to run another one in the context of a 3-year research project, ENVIRA, that has already started in conjunction with two other research groups that will be using the multiagent system to develop their own training systems.

In the first session, the participants were the members of the development team, and we only made use of some steps of Phase 0 and of Phase 2 of ATAM, as described in [13], since all of them were familiar with the architecture. The main objectives of this session were to identify and evaluate *use case scenarios* and *growth scenarios*.

Given the composition of the group that took part in the evaluation, no significant results were obtained in terms of use case scenarios, although we have been able to check that the design decisions we made while designing the architecture were still valid. As for the growth scenarios, we were able to identify sensitivity points that we will have to cope with in the ENVIRA project. These sensitivity points have to do with the addition of a new student modelling scheme and a cognitive architecture for virtual characters managed by agents.

We have already scheduled a second ATAM session where the members of the other two research teams will also take part. In this session, we expect to get more results about growth scenarios related to their assignments in the project and a few exploratory scenarios provided by the members of the three research groups that are taking part in the ENVIRA project.

To test the system at runtime level, we are developing it in an iterative way. Each agent is being developed apart from the rest of the system, and the agents they need to communicate with have been substituted by 'dummy' agents. At the end of each iteration, the dummy agents are removed and substituted by the agents that are under development. This way, the development keeps focused on three aspects: adherence to the designed communication protocols; change of one agent by a different one, even if it is as simple as the dummy agents are; turning some functionalities on and off, with the aid of the dummy agents (although another mechanism is to be designed so that the dummy agents are not necessary to turn off functionalities).

There is already a functional application that offers much of the functionality that the previous version provided. For the moment, the student modelling is quite simple, as well as the simulation agent. In contrast, the tutoring agent is capable of supervising the student, providing different levels of hints and answers to the student's questions. The planning agent is already capable of planning a procedure and replanning alternatives in response to the student's actions, and

the world an expert agents provide support to the tutoring agent, so that it can provide better assistance to the student according to the state of the environment and the characteristics of the procedure the student is training. Both the agent platform and the VE show an adequate performance when running at the same time in the same or different machines, either with one or two students. Further testing is needed to add more students, but with the current results we expect the system to behave better than the previous version.

5 Related Work

There are several projects aiming at the use of VR for education and training supported by intelligent agents. The first ones were developed over a decade ago, and the most representative among them are Steve [20], Adele [21], Cosmo [22], Herman the Bug [23] and Vincent [24]. What all of them have in common is the fact that the primary objective in all of them was to develop an embodied pedagogical agent to support education and training. Each of them tried to solve some of the problems that this emerging discipline posed.

None of these systems are structured as multiagent systems, but as a single agent that inhabits a particular virtual world, and each of them exhibits its own internal architecture. Even so, they have been the key to identify some of the issues that researches are still trying to solve in a satisfactory way.

There are some examples of multiagent systems that support education and training without using VEs. That is the case of FILIP, a multiagent system for training based on simulations [25] to provide training for air controllers. The system is composed by seven agents that cover the modules of an ITS: one for the student, one for the expert, three for the tutor (skill development, curriculum and instructor agents) and two other agents related to the communication with the learning environment and the user.

Baghera is another example of multiagent system used to teach geometry [26]. The aim of this system is to study emergent behaviours in multiagent systems. What makes this system more interesting is the fact that agents are organized in two levels, and the number of agents is not fix, but varies according to the number of students connected to the system. Each student is assisted by three agents: the personal interface agent, which monitors the student's actions, the tutor agent and the mediator agent. In addition, the tutor is assisted by two agents: the personal interface agent and the assistant agent. All these agents are supported by second level agents of four different kinds, which are in charge of evaluating the student's actions. This is made through a voting mechanism that causes the emerging behaviours that are the subject of study.

The systems that are closer to the one described in this paper are those that are based on multi-agent systems and make use of VEs to support training. A good example is MASCARET (Multi-Agent Systems to simulate Collaborative, Adaptive and Realistic Environments for Training) [27], an agent-based IVET that has been used to train firemen in operation management. In this system, agents are divided in organizations, each of which controls different aspects of the organization: physical, social, pedagogical, mediation, and human interaction.

The agents that integrate the pedagogical organization cover the four modules of an ITS, plus a fifth module that is in charge of controlling the mistakes an student may make. The expert agent communicates with the social and physical organizations to be able to know what to to and what objects and agents are involved in an action.

Lahystotrain is an IVET developed to train surgeons in laparoscopy and hysteroscopy interventions [19]. This systems contains five agents which help the student in the training process. One of them is the tutor, which supervises the student and registers his actions. An assistant agent provides explanations and interrupts him when he makes a mistake. These agents have an ad-hoc architecture tailored to suit their responsibilities. The other three agents take the role of an auxiliary surgeon, a nurse and an anaesthetist that play their role within the team. Their architecture is the same for all three, and it is a kind of BDI architecture with a perception module, a reasoning engine and an action control module. The student must learn what the role of these three agents is and how to coordinate them.

What most of these systems have in common is the fact that they have been developed to solve a specific problem, but only a few of them have been designed to be reusable, at least to some extent [20,28], and apparently none of them have taken advantage of the existing knowledge on software architecture. This is an aspect that was already put forward in [29], where three main problems are mentioned: the fact that most research groups develop only part of the systems, which does not give them a view about the whole design; systems are tailored to solve specific problems; and designs are not evolutive. They claim that an ITS can be developed as a set of independent agents that exchange messages in a predefined language, and they use the concept of federated architecture [30] to articulate their system GIA. To some extent, the architecture described in this paper follows those guidelines, although the federated architecture has been substituted by a service oriented approach and a more formal software engineering support has been used.

There are few examples of multiagent systems that have been evaluated using ATAM. Among these systems, we can highlight the work presented in [31], where the authors report a successful utilization of an ATAM workshop to evaluate an agent-based software architecture for an industrial transportation system. Although they do not explain the method that was used to design the architecture, they suggest they took quality attributes into account when designing it.

Another work reporting the use of ATAM is the one described in [32], which is the first reference we have found about the use of ATAM to evaluate a multiagent system. In this work, the authors put forward what they think to be relevant attributes in agent-based systems: performance predictability, security against data corruption and spoofing, resilience to modifiability of the environment and availability and fault tolerance. Although not all of them are applicable to the system presented in this paper, this work constitutes an important approach from agent-oriented software development to more traditional software construction.

6 Conclusions and Ongoing Work

It is getting common for Virtual Environments for Training to be designed as Multi-Agent Systems, since agents provide a higher level of abstraction than objects and this helps to face the increasing complexity that involves the development of these systems.

Many authors claim, without further proof, that their systems are flexible because they are using agents to build them, and to some extent we may have made the same mistake in our first version of the architecture. Although it had been designed with modifiability in mind, it soon became clear that successive modifications were making the architecture degrade quite quickly. That experience is one more example to show that the mere use of agents (or any other technology) does not guarantee that the application developed using them will have certain properties. On the contrary, the result may be even worse if the design decisions have not been made with care.

In the second version of the architecture, we have tried to take advantage of the growing experience in the field of software architecture, even if it is not specifically agent oriented – something that is not considered to be necessary at the architectural level [12] –. Even so, we have used agents in the architectural design because they involve the use of certain artifacts, such as yellow pages or asynchronous message passing, that are relevant in the architectural design.

However, we have not been able to use ADD for the architectural design, given the fact that a hierarchical decomposition does not seem to suit our needs. A review of the new version of ADD [33] shows it is based on the same design strategy, so we still need a different design approach that is not based on hierarchical structure and decomposition. Even so, designing with quality attributes as architectural drivers as ADD promotes has resulted in a design that, up to now, has proven to be more modifiable than the previous design was.

As for the use of ATAM, it is a valuable tool from which we still expect to obtain useful results as soon as the second workshop is carried out

We are already making changes to the system to test to what extent it can be modified, and they are being evaluated both on the architectural design and on the implemented system. In addition to the modification of the student modelling agent, there are two main changes that will prove the suitability of the architecture. The first one is the inclusion of a model of human-like perception [34] to use the student's attention as part of the student's model and as an additional source of information for tutoring decisions. The second one is the inclusion of a cognitive architecture that allows us to make use of virtual tutors and teammates with complex, emotional behaviours [35].

Acknowledgements. The research presented in this paper has been funded by the Spanish Ministry of Science through projects MAEVIF (TIC2000-1346), ICEVAPI (TIN2004-07946) and ENVIRA (TIN2006-15202-C03-01) and has been supported by the INTUITION NoE.

References

1. Sleeman, D., Brown, J. (eds.): Intelligent Tutoring Systems. Academic Press, London (1982)
2. Wenger, E.: Artificial Intelligence and Tutoring Systems. Computational and Cognitive Approaches to the Communication of Knowledge. Morgan Kaufmann Publishers, Los Altos (1987)
3. Munro, A., Surmon, D., Johnson, M., Pizzini, Q., Walker, J.: An open architecture for simulation-centered tutors. In: Proc. of AIED 1999: 9th Conference on Artificial Intelligence in Education, Le Mans, France, pp. 360–367 (1999)
4. Mendez, G., Rickel, J., de Antonio, A.: Steve meets jack: the integration of an intelligent tutor and a virtual environment with planning capabilities. In: Rist, T., Aylett, R.S., Ballin, D., Rickel, J. (eds.) IVA 2003. LNCS (LNAI), vol. 2792, pp. 325–332. Springer, Heidelberg (2003)
5. Mendez, G., Herrero, P., de Antonio, A.: Intelligent virtual environments for training in nuclear power plants. In: Proc. of the 6th Intl. Conf. on Enterprise Information Systems (ICEIS 2004), Porto, Portugal (2004)
6. Zambonelli, F., Jennings, N.R., Wooldridge, M.: Developing multiagent systems: The gaia methodology. ACM Transactions on Software Engineering and Methodology (TOSEM) 12(3), 317–370 (2003)
7. Mendez, G., de Antonio, A.: Training agents: an architecture for reusability. In: Panayiotopoulos, T., Gratch, J., Aylett, R.S., Ballin, D., Olivier, P., Rist, T. (eds.) IVA 2005. LNCS (LNAI), vol. 3661, pp. 1–14. Springer, Heidelberg (2005)
8. de Antonio, A., Ramirez, J., Mendez, G.: An Agent-Based Architecture for Virtual Environments for Training. In: Developing Future Interactive Systems, pp. 212–233. Idea Group (2005)
9. Fikes, R.E., Nilsson, N.J.: Strips: a new approach to the application of theorem proving to problem solving. Artificial Intelligence 2(3-4), 189–208 (1971)
10. Nau, D., Au, T., Ilghami, O., Kuter, U., Murdock, W., Wu, D., Yaman, F.: Shop2: An htn planning system. Journal of Artificial Intelligence Research (JAIR) 20, 379–404 (2003)
11. Wooldridge, M., Jennings, N.R.: Software engineering with agents: Pitfalls and pratfalls. IEEE Internet Computing 3(3), 20–27 (1999)
12. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice, 2nd edn. SEI Series in Software Engineering. Addison Wesley Professional, Reading (2003)
13. Clements, P., Kazman, R., Klein, M.: Evaluating Software Architectures. The SEI Series in Software Engineering. Addison-Wesley, Reading (2002)
14. Parnas, D.L.: On the criteria to be used in decomposing systems into modules. Communications of the ACM 15(12), 1053–1058 (1972)
15. Parnas, D.L.: On a 'buzzword': Hierarchical structure. In: Information Processing 1974, Proceedings of IFIP Congress 1974, pp. 336–339 (1974)
16. Agre, P.E.: Hierarchy and history in simon's "architecture of complexity". Journal of the Learning Sciences 12(3), 413–426 (2003)
17. Haythorn, W.: What is object-oriented design? Journal of Object Oriented Programming 7(1), 67–78 (1994)
18. Hayden, S., Carrick, C., Yang, Q.: Architectural design patterns for multi-agent coordination. In: Proc. of the 3rd Intl. Conf. on Agent Systems (Agents 1999) (1999)
19. los Arcos, J.L., Muller, W., Fuente, O., Orúe, L., Arroyo, E., Leaznibarrutia, I., Santander, J.: Lahystotrain: Integration of virtual environments and its for surgery training. In: Gauthier, G., VanLehn, K., Frasson, C. (eds.) ITS 2000. LNCS, vol. 1839, pp. 43–52. Springer, Heidelberg (2000)

20. Rickel, J., Johnson, W.L.: Animated agents for procedural training in virtual reality: Perception, cognition, and motor control. Applied Artificial Intelligence 13, 343–382 (1999)
21. Shaw, E., Johnson, W., Ganeshan, R.: Pedagogical agents on the web. In: Proceedings of the Third Annual Conference on Autonomous Agents, Seattle, WA, USA, pp. 283–290. ACM Press, New York (May 1999)
22. Lester, J., Voerman, J., Towns, S., Callaway, C.: Cosmo: A life-like animated pedagogical agent with deictic believability. In: IJCAI 1997 Workshop on Animated Interface Agents: Making them Intelligent, Nagoya, Japan (August 1997)
23. Lester, J., Stone, B., Stelling, G.: Lifelike pedagogical agents for mixed-initiative problem solving in constructivist learning environments. User Modeling and User-Adapted Interaction 9(1–2), 1–44 (1999)
24. Paiva, A., Machado, I.: Life-long training with vincent, a web-based pedagogical agent. International Journal of Continuing Engineering Education and Life-Long Learning 12(1) (2002)
25. Zhang, D., Alem, L., Yacef, K.: Using multi-agent approach for the design of an intelligent learning environment. In: Wobcke, W., Pagnucco, M., Zhang, C. (eds.) Agents and Multi-Agent Systems Formalisms, Methodologies, and Applications. LNCS (LNAI), vol. 1441, pp. 221–230. Springer, Heidelberg (1998)
26. Webber, C., Pesty, S.: A two-level multi-agent architecture for a distance learning environment. In: de Barros Costa, E. (ed.) Workshop on Architectures and Methodologies for Building Agent-based Learning Environments (ITS 2002), pp. 26–38 (2002)
27. Buche, C., Querrec, R., Loor, P.D., Chevaillier, P.: Mascaret: A pedagogical multi-agent system for virtual environments for training. International Journal of Distance Education Technologies 2(4), 41–61 (2004)
28. Evers, M., Nijholt, A.: Jacob - an animated instruction agent in virtual reality. In: Tan, T., Shi, Y., Gao, W. (eds.) ICMI 2000. LNCS, vol. 1948, pp. 526–533. Springer, Heidelberg (2000)
29. Cheikes, B.: Gia: An agent-based architecture for intelligent tutoring systems. In: Finin, T., Mayfield, J. (eds.) Proceedings of the CIKM 1995 Workshop on Intelligent Information Agents, Baltimore, Maryland (1995)
30. Genesereth, M.: An agent-based approach to software interoperability. Technical Report Logic–91–6, Logic Group Computer Science Department, Stanford University (1993)
31. Boucke, N., Weyns, D., Schelfthout, K., Holvoet, T.: Applying the atam to an architecture for decentralized control of a transportation system. In: Hofmeister, C., Crnković, I., Reussner, R. (eds.) QoSA 2006. LNCS, vol. 4214, pp. 181–199. Springer, Heidelberg (2006)
32. Woods, S.G., Barbacci, M.: Architectural evaluation of collaborative agent-based systems. Technical Report CMU/SEI-99-TR-025, CMU/SEI (1999)
33. Wojcik, R., Bachmann, F., Bass, L., Clements, P., Merson, P., Nord, R., Wood, B.: Attribute-driven design (add), version 2.0. Technical Report CMU/SEI-2006-TR-023, CMU/SEI (2006)
34. Herrero, P., de Antonio, A.: Keeping watch: Intelligent virtual agents reflecting human-like perception in cooperative information systems. In: Proc. of the 11th Intl. Conf. on Cooperative Information Systems. Springer, Heidelberg (2003)
35. Imbert, R., de Antonio, A.: Using progressive adaptability against the complexity of modeling emotionally influenced virtual agents. In: Proc. of the 18th Intl. Conf. on Computer Animation and Social Agents (CASA 2005) (2005)