A Modifiable Agent-Based Software Architecture for Intelligent Virtual Environments for Training

Gonzalo Méndez
Computer Science School
Technical University of Madrid
gonzalo@gordini.ls.fi.upm.es

Angélica de Antonio
Computer Science School
Technical University of Madrid
angelica@fi.upm.es

Abstract

During the last years, Intelligent Virtual Environments for Training have become a quite popular application of computer science to education. However, little attention is being paid to software engineering issues, and most systems are developed in an ad-hoc way that does not allow the reuse of their components or an easy modification of the application, even though some authors claim that the use of agents makes systems be more modifiable. We describe an agent-based software architecture that is intended to be easily extended and modified. This architecture is a redesign of a previous one using more formal principles and methods of software architecture design.

1. Introduction

Intelligent Virtual Environments for Training (IVET) are a combination of a 3D Virtual Environment (VE) and an intelligent tutor, be it an Intelligent Tutoring System (ITS) [11] or not. They are mainly used to train students in situations where training in the real environment may be dangerous, very expensive or difficult. Although still not widely used, they are experiencing an increase of popularity for military, industrial and medical training.

The development of IVETs has a quite short history, dating from the mid-nineties. 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, and in many cases forgetting software engineering principles and techniques.

The MAEVIF project (*Model for the Application of Intelligent Virtual Environments to Education*) was the result of several experiences integrating VEs and intelligent tutors [8, 9] 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 an open and flexible agent-based software architecture to support IVETs, the design and implementation of a prototype authoring tool, based on the defined architecture, that simplifies the development of IVETs and the definition of a set of methodological recommendations for the development of IVETs.

In the remainder of this paper we describe the architectural design (sections 2, 3 and 4) and the resulting architecture (section 5). Then, we present some evaluation results (section 6) and the current and future work lines (section 7).

2. Overview of the Architectural Design

The architecture described in this paper is the evolution of a former one [4], based on an organizational approach, that did not offer the expected modifiability. After its implementation and subsequent analysis of the resulting system, it was decided to redesign it making use of not specifically agent oriented techniques. The redesign aimed at obtaining an easily modifiable architecture, that allowed us to substitute virtually any agent by a different one with a low impact on the other agents. An additional objective was for developers to be able to extend the system with as few modifications as possible.

The main theoretical support have been the methods developed at the Software Engineering Institute (SEI) [1, 2, 3], such as Attribute Driven Design (ADD), Architecture Tradeoff Analysis Method (ATAM) and Views and Beyond (V&B). The purpose of the first two methods is to design and evaluate a software architecture driven by quality attributes instead of only functionality, while the third one helps to organize the documentation. The other important support has been provided by the use of *information hiding* [10], which establishes that a division in submodules must be such that each submodule encapsulates a design decision that must remain hidden from the rest, and communication among submodules is carried out using an interface as abstract as possible. The design decisions that are encapsulated in each module are related to the changes that are likely to happen over the system's life.

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. One problem we had with the first architecture was the fact that some agents were not clearly under the supervision of another agent, so a hierarchical structure was not what we wanted to obtain. 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 with the ITS.

Agent systems are intrinsically peer-to-peer, where each agent is a peer that makes use of services offered by other agents. 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. However, instead of following a decomposition approach, we have worked using an iterative an incremental approach. A sketch of the architecture was always present in a whiteboard in the architectural design sessions. 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 architectural design.

3. Quality Attributes

The design process started with the definition of a set of quality scenarios to establish the changes to be considered by the design. These 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.

The system is also required to be easily extended, so that new agents that provide new functionality can be added without having to make big changes in the existing ones.

Taking into account that training is carried out in a VE, 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. Therefore, it is important to keep performance close to real time. If not, the training experience may be frustrating for the student, which may cause the training to be less effective than 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 objectives of an ITS. The student modelling is used to customize the training process to the student's abilities and needs, so it has not been necessary to consider it as an additional quality attribute.

4. 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 providing the services they need. To achieve this, during system startup, the agents announce in the system's yellow pages the services they are capable to provide. 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 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. This requirement is similar to Liskov's substitution principle in object oriented design [7].

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.

Agents communicate with each other exchanging FIPA ACL messages. We have designed a fairly simple communication protocol for an 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 communication centre has been designed to communicate the agent platform and the VE. Each application subscribes to the messages it is interested in receiving, so that they only receive the messages they know how to handle, no matter who sends them.

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

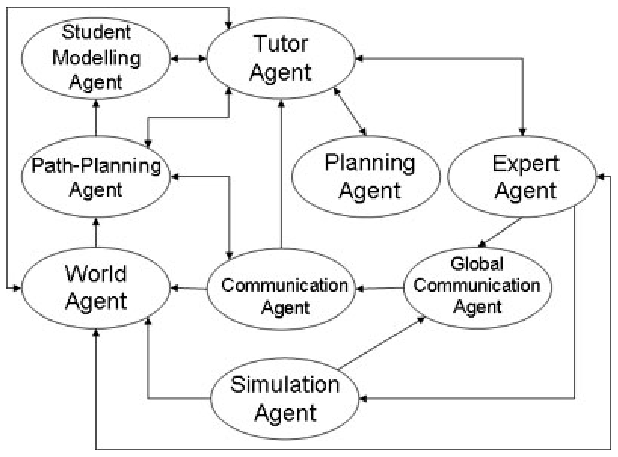

Figure 1. Peer-to-peer view.

files, which allows changes in the way the system behaves without further changes in the design.

5. Agent-Based Architecture

The resulting architecture is the one shown in Fig. 1. The picture shows the structure of the architecture as it is currently designed, where all the agents are represented along with the communication channels.

During runtime, there is only one agent of each kind, except for the agents that are related to students: the Student Modelling Agent and the Communication Agent. This way 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 features of the architecture are:

- There is no hierarchical structure; 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. Although we were not aiming at obtaining a Service Oriented Architecture, this is one of the mechanisms that introduces a higher degree of modifiability in the system.

- Extended support for a simulator. Some systems simulate environmental events that may be caused by external factors, such as changes in the state of a patient. 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 simulate simple systems, 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 the 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.

6. Evaluation

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 to identify relevant quality attributes. We are gathering 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 multi-agent 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, 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. As for the growth scenarios, we have identified sensitivity points that we will have to cope with in the ENVIRA project. They have to do with the addition of a new student modelling scheme and a cognitive architecture for virtual characters controlled by agents.

A second ATAM session has been scheduled where 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.

To test the system at runtime level, we are developing it in an iterative way. Each agent is being developed apart from the rest, 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.

There is already a pilot 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 able to plan a procedure and replan alternatives, and the world an expert agents provide support to the tutoring agent. Both the agent platform and the VE show a good performance when running at the same time in the same or different machines, either with one or two students. Further testing is still needed to add more students.

7. 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 IVETs.

Many authors claim that their systems are flexible because they are using agents to build them. On the contrary, the result may be worse than without agents if the design decisions have not been made with care. We have tried to take advantage of the growing experience in the field of software architecture both to design and evaluate our architecture, although 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 [12] 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 has resulted in a design that has proven to be more modifiable than the previous one. As for the use of ATAM, it is a valuable tool from which we still expect to obtain useful results.

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 [5] to use the student's attention as part of the student's model. 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 [6].

Acknowledgements

This research has been funded by the Spanish Ministry of Science through projects MAEVIF (TIC2000-1346), ICE-VAPI (TIN2004-07946) and ENVIRA (TIN2006-15202-C03-01) and has been supported by the INTUITION NoE.

References

[1] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. SEI Series in Software Engineering. Addison Wesley Professional, 2nd edition, 2003.

[2] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford. *Documenting Software Architectures: Views and Beyond*. SEI Series in Software Engineering. Addison Wesley Professional, 2002.

[3] P. Clements, R. Kazman, and M. Klein. *Evaluating Software Architectures*. SEI Series in Software Engineering. Addison Wesley Professional, 2002.

[4] A. de Antonio, J. Ramirez, and G. Mendez. *An Agent-Based Architecture for Virtual Environments for Training*, chapter VIII, pages 212–233. Idea Group, 2005.

[5] P. Herrero and A. de Antonio. 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-Verlag, 2003.

[6] R. Imbert and A. de Antonio. 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.

[7] B. Liskov and J. Wing. Family values: A behavioral notion of subtyping. Technical Report MIT/LCS/TR-562b, MIT, Cambridge, MA, USA, 1993.

[8] G. Mendez, P. Herrero, and A. de Antonio. Intelligent virtual environments for training in nuclear power plants. In *Proc. of the 6th Intl. Conf. on Enterprise Information Systems (ICEIS 2004)*, Porto, Portugal, April 2004.

[9] G. Mendez, J. Rickel, and A. de Antonio. Steve meets jack: the integration of an intelligent tutor and a virtual environment with planning capabilities. In *4th Intl. Working Conf. on Intelligent Virtual Agents (IVA03)*, volume 2792 of *LNAI*, pages 325–332. Springer-Verlag, September 2003.

[10] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053 – 1058, December 1972.

[11] E. Wenger. *Artificial Intelligence and Tutoring Systems. Computational and Cognitive Approaches to the Communication of Knowledge*. Morgan Kaufmann Publishers, Los Altos, California, 1987.

[12] R. Wojcik, F. Bachmann, L. Bass, P. Clements, P. Merson, R. Nord, and B. Wood. Attribute-driven design (add), version 2.0. Technical Report CMU/SEI-2006-TR-023, CMU/SEI, 2006.