A STUDENT MODEL IN A MULTI-AGENT INTELLIGENT SYSTEM

Cornelia Novac – Ududec, Rodica Birla
cornelia.novac@ugal.ro, arondica@yahoo.com

University "Dunarea de Jos" of Galati, Department of Computer Science,
111 Domneasca Street, Galati, Romania

Abstract: The paper presents an intelligent tutoring system which provides a Student Model using the "overlay method". The system's tasks are distributed among the intelligent agents (software agents), each having clearly specified individual roles as following: administrator (Cerberus Agent); interface (InterfaceAgent); tutor (TutorAgent); student (StudentAgent). Moreover, the system is a "three-tier" architecture application and was implemented in Java language.

Keywords: multi-agent system, educational system, student model, intelligent system

1. INTRODUCTION

The current thought patterns related to intelligent tutoring systems refer to creation of distributed applications, which can be accessed by a great number of users and do not necessarily require the physical presence of the tutor.

Moreover, the responsibilities of the tutor (as real, physically present person) are proposed to be distributed to the intelligent system's modules, with a view to automating the tutoring process (coe.sdsu.edu/..) The computer network provides the option of identifying another system or human partner able to assist the student solve its problem. This tendency can be found in the collaborative tutoring system approaches, capable of finding suitable partners for either assistance or collaboration, for setting up teams or initiate group activities (Weiss et al, 1999), (www.aaai.edu/..).

Therefore, it becomes necessary that tutoring systems (or their modules providing adaptive assistance) were capable of communicating information about their users, their available resources and their objectives in order to be able to identify a suitable partner (Martin et al., 1999).

We may well imagine software agents attached to every application or tutoring environment, each having an explicit representation of the users' objectives, of their plans and resources. These agents communicate and negotiate with each other in order to fulfill their individual or group objectives (Kelly et al, 2006).

The present paper aims to describe a multi-agent tutoring system (MTS) designed and implemented within the Computer Science Department, in which research focused upon "modeling" the student with a view to identifying the most suitable methods and tutoring strategies by taking into account individual knowledge and abilities.

The "modeling student process" used the *overlay* architecture (common for many tutoring systems) combined with the Bayesian Theory of Probability (www.cs.umbc.edu/..)

2. THE PROTOTYPE MTS (MULTI-AGENT TUTORING SYSTEM)

The tutoring system is designed as a distributed application, providing the option of distance

This paper was recommended for publication by Severin Bumbaru

learning/tutoring by means of a computer network or the Internet. The application can be classified as having a "three - tier" architecture, the Client part being represented by the configuration module and the tutoring material presentation module. The user connects to the system by means of a Login module, monitored by an Administrator agent (CerberusAgent). After authentication the user may have access to a configuring instrument (if the user connected to the system is the tutor) or to the user interface by means of which the learning material will be presented (if the user is a student). This part of the application will be monitored by an interface agent (InterfaceAgent). The application server contains the agent platform that monitors the student activity, and a web server responsible for delivering the learning material to the user interface (Apache Tomcat). The third level consists of a xml database (Apache Xindice), which stores information about the structure of the course, student models, etc. (Fig.1)

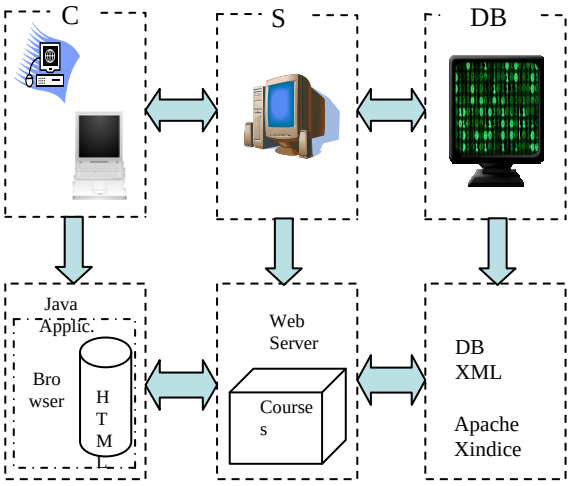

Fig. 1. System's Architecture

The server component of the application has a double functionality: agent platform and web server. The user goes over the content of the course, as the user interface reacts to his demands and requires the web server to communicate the html page that corresponds to the client demand.

Simultaneously, the agent monitoring the student interface communicates messages to the other agents involved in the tutoring process, as they react accordingly. For example, InterfaceAgent may communicate to TutorAgent a message that requests initialization of the navigation assistance board. The tutor agent generates a list of recommended subjects and communicates it to the interface agent, who will update the content of the respective board accordingly. Plus, the student model will be changed in order to include this information.
When the student proceeds to studying a new subject, the student interface communicated a request to the web server that will send the correspondent html page. A message will be simultaneously sent to TutorAgent with information related to the subject previously studied. TutorAgent will update the status information referring to the learning level the student reached and will carry out the necessary steps to update the student model.
The Client consists of two modules:

A *maintenance module* that allows adding of new users (be they tutors or students), setting the level to be reached by a certain student and deleting the model of a specific student form the database;

An *interface module* that will interact with the student and will provide the educational content (plus a testing sub-module).

The maintenance module is accessible only to the "tutor"- type users, providing them with a series of configuration instruments. After performing all operations requested by the tutor, the changes he/she proposed will be stored in the xml Apache Xindice database.

The student interface module is monitored by the interface agent, and communication between the Java application and agents will be carried out using *sockets.*

When designing this module one used an instrument called JACK (Java Agent Compiler and Kernel), which is a multi-agent application development system, based on the Java language (actually carried out as an extension of that), originating in the BDI Model (Belief-Desire-Intention).

3. THE AGENT STRUCTURE FOR THE SYSTEM

The agent platform that monitors this application consists of four agents:

CerberusAgent – has an administrator role (verifies the validity of the user name and password). After successful authentication it will send a message to the interface agent who – depending on the type of user connected to the system - will initiate the configuration module or the student interface module;

InterfaceAgent – initializes the application modules, the student and tutor agents, monitors the student interface, requests the Tutor Agent initiation of the navigation assistance board, etc;

StudentAgent – monitors the student's actions and sends to the other agents messages that contain information used for updating the student model;

TutorAgent – sends messages for updating the navigation assistance board, initiation of the testing phase, etc.

4. THE STUDENT MODEL

A *student model* is defined as a "*representation of the tutoring system's beliefs about the student"*.
The Student Models can be classified taking into account a number of factors such as: the manner they are generated, their content or their application. Within this section the classification is based on two factors:
1. Persistence of representation. The beliefs about the student either cover a short period of time (as long as the answer to a question, for example), either they are retained in order to help build a long-term Student Model. Most intelligent tutoring systems implement both representations, using the short–term representation to update the long-term model.
2. Content. Which are those beliefs about the student that actually model the intelligent tutoring system? The answer to this question generally depends upon persistence. Short-term beliefs, by nature, must be very specific (for example „Student X broke rule Y of the issue Z"), as they can be observed rather than observed. Long-term models on the other hand consist of a much larger proportion of inferred beliefs. These beliefs may be more abstract also, as they relate to the knowledge level of the student, professional misunderstandings or the student conduct.
The Overlay Models (Fig.2) are long-term models and they show the expert's knowledge level related to a specific domain. The student's knowledge is therefore represented as a subset of the expert knowledge. Moreover, the overlay model can only be constructed as long as the expert knowledge can be decomposed in generic units (i.e. rules, facts, concepts).
The knowledge level of every unit varies from 0 (beginner) to 1 (expert). The expert is represented by a knowledge level equal to 1 attached to every unit, while the student attaches to every unit a knowledge level smaller than 1.

Interestingly, there are two very different approaches of the *knowledge level* within specialized literature. Some systems consider it a binary variable that can only get the „learned, not learned" values and that represents the belief that the unit is known. This could be qualified as a probabilistic approach and is representative for the intelligent tutoring systems using Bayesian probabilities for student modeling.

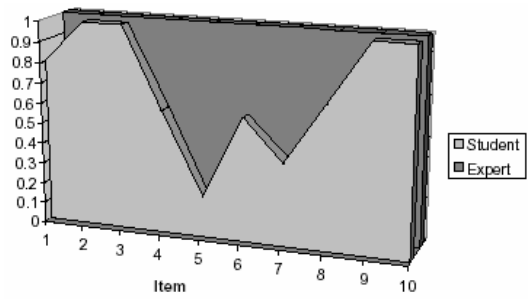

Fig. 2. Overlay Model

On the other hand, the knowledge level may be interpreted as the current intellectual status of the student. For example, a 0.5 knowledge level for a unit means that the system believes that the student only learned half of it and needs more practice, but is no longer a beginner; the other approach would have presumed that the probability for the student to have learned the x unit were 50%, therefore the initial status.

It is interesting to also note that a value of 0.5 for a unit represents *the maximum uncertainty state* if probabilistic approaches are used, therefore diagnose strategies might produce different results, depending upon the qualification of the measure: probabilistic or absolute.

An overlay model with a more varied structure that was used is *the differential one*. It compares the student's level of learning to a value that indicates the desired knowledge level that the student should have reached at one time.

The Differential Model is much less restrictive than the Overlay one, since it considers as significant just the difference until the desired knowledge level, not inferring upon the student's knowledge beyond that limit.

For example, if the student is expected to know the A fact, but not the B fact, then – if the student makes mistakes when proving both A and B, then the differential model may only infer that the student doesn't know A, not being able to infer anything about B.

In order to store the information about the student (the student model) and some information about the educational material (structure, questions, concept list), one used a xml database. Namely, the 1.0 implementation of the Apache Xindice database was chosen, as a native xml database designed to store data in the xml format. The benefits of this solution relate to the fact that the programmer avoids the "underground" implication of data storage, as both

the information inside the database and the data to be extracted from it will be in xml format.

Among the multiple facilities of this database there are: the possibility of interrogating the content of the database using the Xpath language, the possibility of updating the information in the database by XMI:DB Update, the possibility of implementing XML:DB API for developing Java applications.

The memorized data are grouped in collections of documents that can be indexed in order to increase the speed of XPath interrogations' execution.

The facilities provided by this xml database recommends it for this type of application, since the reduced response time and the capacity of storing data in the xml format are of peculiar importance.

The information necessary for the system's functioning are grouped in three collections within the database: users, model and course.

- The *users collection* consists of a xml document that summarizes data about the users: name, password, type;
- The *model collection* contains xml documents that correspond to the model of each student who uses the application;
- The *course collection* consists of three documents: *syllabus* – the complete structure of the course, *questions* – the collection of questions that will make up the tests and *concepts* – a concept list.

The student model is that part of the tutoring system that contains information about the student such as: the knowledge level mandatory for completing the course, the position within the course material, etc.

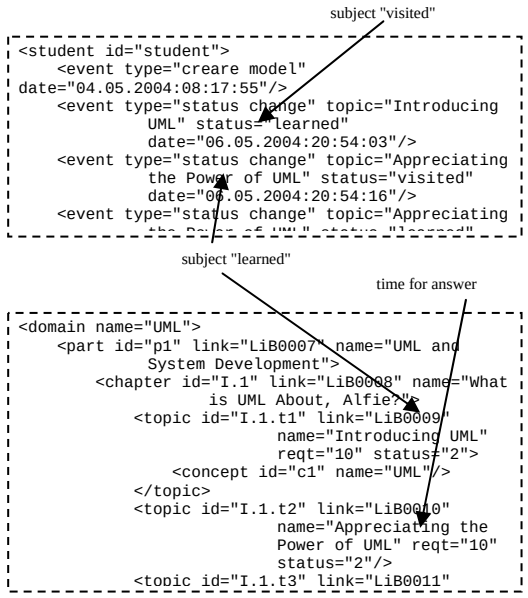

Fig. 3. The Student Model

The first of them comprises of the structure of the sub-domain that has the knowledge that the student must acquire for completion of the course, and the second one consists of a history of the student actions, used by the tutor in order to set up a strategy for navigation assistance.

The Student Model within the MTS is an overlay model and is made up of two documents which are to be found in the database from the *model* collection. The documents are to be found under the label of: <name_student>_model and <name_student>_log.

5. REPRESENTATION OF THE KNOWLEDGE

The knowledge in the field is represented within the xml database by a collection named "course" that consists of three documents:

- *syllabus* - the complete structure of the course plus control information;
- *concepts* – the complete list of questions together with the necessary explanations;
- *questions* – the question base for test generation (with questions, answers, clues to be used for student assistance).

All of them combined represent the expert knowledge in the field, which are to be used for student model generation, his/her knowledge being therefore considered a sub-set of the expert knowledge (the *overlay* type model) (coe.sdsu.edu/..).

An example from Syllabus:

```
<domain name =”UML”>
 <part id =”p1” name=”UML and System
Development” link=”Lib0007”>
    <chapter id=”I.1” name=”What is UML
About, Alfie?” link=”Lib008”>
         <topic       id=”I.1.t1”       name
“Introducting UML” link=”Lib0009”>
          <concept name=”UML” id=”c1”/>
               </topic>
                  …
                </chapter>
                  …
                </part>
                  …
                </domain>
```

An example from Concepts:

```
<concepts>
<concept name=”UML” id=”c1”> UML stands for
the Unified Modeling Language.
   </concept>
      <concept   name=”Structural   Diagrams”
id=”c2”> Structural diagrams are diagrams
that show the building block of the system
features that don’t change with time.
         </concept> …
       </concept>
```

An exampple from Questions:

```
<question_base>
```

```
    <question text="What does UML stand
for?" cor_ans="1" part="1">
      <answer text= "UML stands for the
Unified Modeling Language"/>
       <answer text="UML stands for
Unilateral Marketing Language"/>
       <hint text="It involves
modeling..."/>
      </question>
        ...
</question_base>
```

6. UTILIZATION SCENARIOS FOR THE MTS

In order to be able to utilize the system, every user must have a user-name and a password, both of them valid. After authentication, depending on the type of user connected to the system, one of the following modules will be accessed: the *maintenance module* (if the user is a tutor) or the *student interface module* (if the user is a student).

When the user is a tutor, activating the maintenance module will provide him/her with the possibility of adding new users, initiating or deleting new student models.

When the user is a student, the course material presentation window will be displayed, and the agent platform for monitoring the student activity will be initiated. The student will be able then to read the course material, request information about a certain concept, etc.

After going over the recommended material, the navigation assistance board will be updated by the interface agent, by request of the tutor agent.

Depending upon the student knowledge level, the tutor agent may request the interface agent to initialize the testing sub-module. This may generate a new test based on the questions within the question-database. Taking into account the time spent by the student to answer a question, supplemental clues will be provided and the final grade will be diminished accordingly.

After having taken the test, if the results are satisfactory the student will be allowed to reach the next level of the course. If the results are negative (below a certain established limit), the display of the navigation assistance board will read "Testing", the student being unable to reach the next level until the test gets a positive grade. After finalizing going over the course material, the student will be congratulated upon completion of the course.

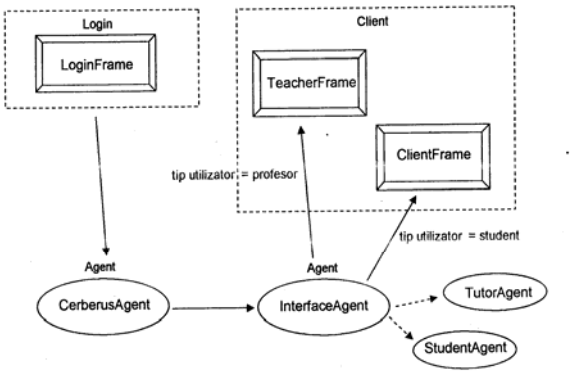

Figure 4. Utilization Scenarios

Exemple: The schedule used by InterfaceAgent for the initialisation of the TutorAgent.

```
import java.util.Vector;
import util.GestionarModel;
import util.XMLHelper;
plan InitTutorAgentPlan extends Plan{
       #handles event InitTutorAgentEvent
initEvent;
       #uses interface TutorAgent self;

       body(){
       System.out.println("TutorAgent init:
"+initEvent.username);
               self.tutorData =
                       XMLHelper.getDOMAsVec
                       tor(initEvent.usernam
                       e.trim());
               self.topics =
XMLHelper.getTopics(initEvent.username.trim(
));
               self.partLimits =
                       XMLHelper.getPartLimi
                       ts(initEvent.username
                       .trim());
               Vector links =
(Vector)self.topics.get(1);
               self.lastLink =
(String)links.get(links.size()-1);
               self.currentPart =
XMLHelper.getCurrentPart(self.username);

       self.lastPart=((Vector)self.partLimit
s.get(0)).size();
               self.checkSum=0;
               self.test1=false;
               self.test2=false;
       }
}
```

7. CONCLUSIONS

The presented system is a prototype of an intelligent tutoring system using software agents. Its purpose is to "model the student", by student being understood any individual undertaking a learning process. This approach over the design of a multi-agent system

originated from the separation of roles in a process of assisted tutoring, which led to defining the systems' agents together with their responsibilities. These roles are static all along the system's execution.

The MTS was tested on disciplines and students in Computer Science, presuming XML knowledge for the introduction of new disciplines and tests.

Consequently, the next objective aiming to ease the utilization of the system will consist of designing an agent that will provide assistance for the tutor during uploading the information (the course material) and elaborating the tests for specific domains.

REFERENCES

Intelligent Tutoring Systems- http://www.aaai.org/AITopics/html/agents.html

Intelligent Tutoring Systems – http://coe.sdsu.edu/eet/Articles/tutoringsystem/start.html

Kelly, D., Tangney, B.- Using Multiple Intelligence Informed Resources in a Adaptive Systems, Intelligent Tutoring Systems: 8th International Conference, pp. 412-421, Jhongli,Taiwan, 2006

Martin, D, et al. (1999), The Open Agent Architecture, Artificial Intelligence Journal, vol.13

Software Agents – An Overview- www.davidreilly.com/topics/software_agents

UMBC Agent Web: http://www.cs.umbc.edu/agents http://www.contrib.andrew.cmu.edu/~plb/AIED97_workshop/Brusilovsky/Brusilovsky.html

Weiss, G., (1999), Multi-agent Systems, A Modern Approach to Distributed Artificial Intelligence