A Multi-Agent Architecture for Intelligent Tutoring

Nicola Capuano[1] Massimo De Santo[2] Marco Marsella[1] Mario Molinara[2] Saverio Salerno[1]

[1] CRMPA, Salerno, Italy

[2] DIIIE - Università di Salerno, Italy

Abstract

One of the most interesting realm among those ones brought up to success by the development of the Internet is Distance Learning. A key issue in such a field is the development of systems for supporting Tutoring activities. This paper is concerned with the presentation of an innovative architecture for Intelligent Tutoring which make use of Software Agents. The way in which the knowledge is represented and stored is discussed together with the ability of our system to manage individual learning paths for different users. The rationale for using Agents is presented and the implementation of the system is discussed.

1. Introduction

The great amount of information available across the Internet brought to the development of new sophisticated information-based technologies; interests in knowledge management, in information retrieval and information filtering are becoming *hot topics* in several areas for different applications across Internet.

Among the enormous number of such applications, one of the most interesting is the *Distance Learning*. The potential of the Web for providing rich materials and experiences, the possibility and capability to learn more knowledge implied by digital technologies are factors of increasing importance in a world where the amount of information that needs to be learned grows very rapidly and becomes obsolete very quickly.

As a matter of fact, the proliferation of Local Area Networks (LANs), and Wide Area Networks (WANs) for telecommunications, information and data applications has brought the enabling technological framework needed to bring network-based multimedia training to full availability of millions of people world-wide. Electronic mail and electronic bulletin boards are familiar tools of today's business and educational environment. When coupled with the rapid improvements in graphic user interfaces now available on the Internet via the World Wide Web, new applications in training and education are emerging daily.

Although the use of internet/intranet delivery systems for training and education is already underway and the potential for expansion is enormous, there are a number of challenges to overcome.

Recent research has been geared towards the development of *Intelligent Tutoring Systems* that respond directly to the student and assume the active role of instructor/tutor. Intelligent Tutoring Systems (ITS) [1] allow the emulation of a human tutor in the sense that an ITS can know what to teach (domain content), how to teach it (instructional strategies), and learn certain teaching relevant information about the student being taught. This requires the representation of a domain expert's knowledge (called the *Expert Model*), an instructor's knowledge (called the *Instructional Model*), and the particular student that is being taught (called the *Student Model*).

Through the interaction of these models, Intelligent Tutoring Systems are able to make judgements about what the student knows and how well the student is progressing. The Instructional Model to the student's needs can then tailor instruction, automatically, without the intervention of a human instructor. The ITS acts as the student's private tutor, while the human trainer or tutor is then free to focus on more complex and individualized student needs. [2]

With the use of *Artificial Intelligence* techniques, ITS can identify a student's strengths, weaknesses and preferred style of learning. These systems are able to qualitatively process information, recognize patterns of behavior, identify misconceptions or *bugs* in performance, and establish a plan of instruction. Instruction can be tailored to the student's learning style and remedy is based on student errors and the "computer perceived" misconceptions. [3]

ITS have the capability of reasoning, justifying, interpreting, predicting, diagnosing, monitoring, planning, and controlling student behavior. As we seen, the three major components of an Intelligent Tutoring System are the Expert Model, the Student Model and the Instructional Model. [4] The three following paragraphs are dealt with an introduction of such models.

1.1. The Expert Model

There are a variety of knowledge architectures. The ***Black Box*** model establishes a criterion referenced knowledge base. The content domain is naturally organized in an existing simbology, which the computer understands. The computer assess student performance without the need of "human intelligence". The criterion for acceptable performance is clearly identified. If the input behavior does not meet the criterion, the computer will inform the student of the performance error and recommend possible solutions. The dialogue between the student and ITS is very simplistic. The computer does not provide detailed explanation about its reasoning. Mathematical equations lend themselves to this type of architecture.

Issue-based architectures compare the student input to the "expert" model and the "student" model. Instructions are programmed to specific issues, which are observable in the behavior of the "expert" model and the "student" model. If the student's performance fails to meet the prescribed behavior criterion, the student receives immediate feedback. Feedback includes an explanation of the rule. The dialogue can be very superficial (i.e. simple feedback on the correct action) or very complex by providing detailed reasoning behind the behavioral rule.

Cognitive Models realistically simulate human problem solving. The three levels of knowledge are procedural, declarative, and qualitative. Procedural Knowledge relates to how a task is performed. Declarative Knowledge is a set of facts organized to permit reasoning. The underlying premise of using this type of knowledge architecture is the assumption that the student has the procedural knowledge base to make inferences from the content domain. Qualitative knowledge involves the causal understanding, which allows humans to reason about behavior using the models of the system. Expert models using qualitative knowledge are in the developmental stage.

Knowledge is structured in a Production System (known as a rule-based system). A production system analyses and synthesizes large quantities of knowledge to solve problems. Declarative and procedural knowledge are integrated in a production system. The declarative knowledge formulates a database of facts and the procedural knowledge provides the rule base in an IF-THEN or WHEN-THEN relationships.

Intelligent Tutoring Systems have expanded the application of the production system by recognizing that humans have the capacity to solve problems using various reasoning methods. The IT systems assesses student performance to determine if the behavior conforms to the prescribed "expert" rule(s). If the performance does not match the rule, the ITS assumes the role of instructor, diagnoses the student's weaknesses and prescribe appropriate remedy. Intelligent Tutoring Systems also use meta-rules to make inferences concerning what to teach independent of the content domain.

1.2. The Student Model

The Student Model establishes the framework for identifying the student's misconceptions and sub-optimal performance. The structure of the student model can be derived from (1) the problem-solving behavior of the student, (2) direct questions asked from the student, (3) historical data (based on assumptions of the student's assessment of his skill level, novice to expert), and (4) the difficulty level of the content domain.

ITS compares the student's actual performance to the Student Model to determine if the student has mastered the content domain. Advancement through the curriculum is dependent upon the IT system's assessment of the proficiency level of the student. The Student Model contains a database of student misconceptions and missing conceptions. This database is known as the *bug library*.

"A missing conception is an item of knowledge that the expert has but the student lacks. A misconception is an item of knowledge that the student has but the expert does not." Bugs are identified from the literature, observation of student behavior and learning theory of the content domain.

1.2. The Instructional Model

ITS actively interacts to student inputs and diagnoses the student's level of understanding or misunderstanding of the knowledge domain. The tutorial gives some control over the selection and sequencing of information by responding to student questions concerning the subject domain and in determining when the student needs help and what kind of help is needed.

An effective ITS will meet the ever changing needs of the student. ITS diagnoses the student's characteristic weaknesses and adapts the instruction accordingly. As the student's level of proficiency increases, ITS will ideally conform to the evolving skill level of the student. ITS adapts as the novice evolves into a subject matter expert.

2. What is ABITS

ABITS stands for "Agent Based Intelligent Tutoring System". It is a Multi-Agent System (MAS) able to extend

a traditional Course Management System (CMS) with a set of "intelligent" functions allowing student modeling and automatic curriculum generation.

The purpose of such functions is the improvement of the learning effectiveness based upon the adaptation of the didactic material to student skills and preferences.

This chapter is thought as an introduction to these functions. In particular, paragraph 2.2 is dealt with student modeling while paragraph 2.3 describes the ABITS implemented algorithm for curriculum generation. Such functions are depicted in the UML Use Case Diagram of figure 1 where the *Evaluate Curriculum* case is dealt with curriculum generation while the *Evaluate Preferences* and the *Evaluate Cognitive State* cases are related to user modeling.

ABITS functions found their effectiveness on a set of rules for knowledge indexing based on Metadata and Conceptual Graphs. This point is treated in paragraph 2.1.

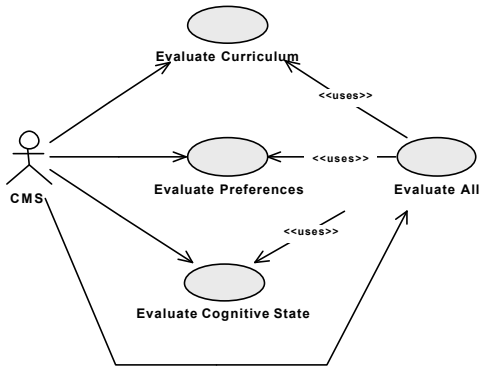

Figure 1. ABITS Use Case Diagram

2.1. Knowledge indexing

ABITS didactic material is organized in *Learning Objects* and is stored in a Course Material File System. A Learning Object is any entity which can be used, re-used or referenced during technology-supported learning [5].

Learning Objects must be indexed in order to let the system know what each one of them is about and how it can be used during the learning process. Some kind of information about Learning Objects is so required. This is *Metadata*.

"Metadata is information about an object, be it physical or digital and its main goal is to locate in efficient and effective way resources over a system or a computer network" [7].

In the field of learning materials, several organizations such as IEEE, EDUCOM etc focused their attention on the creation of Metadata standards specifying the syntax and the semantics of the so-called *Learning Object Metadata* (LOM).

A LOM standard defines the minimal set of properties needed to allow Learning Objects to be managed, located, and evaluated. They accommodate, moreover, the ability for locally extending the basic properties.

ABITS adopts the IEEE LTSC LOM standard [7] to index learning material. Many advantages come in fact from referring to a Learning Object Metadata standard:

- to take advantage of a complete syntax and semantic created by experts of the Learning Technology;
- to enable the automatic importation of extern learning objects that adopt the same Metadata standard;
- to enable the exportation/sale of learning objects to extern systems/clients that adopt the same Metadata standard;

Metadata not only have to provide information about a single Learning Object. They have to provide information about object relations and interdependency too. For this purpose the IEEE LOM standard has a Metadata element called *Idea* that supports *Domain Conceptualizations*.

A Conceptualization is an abstract, simplified view of the world that we wish to represent. A *Conceptual Graph* is an explicit specification of a Conceptualization [8].

Conceptual Graphs are graph-like structures composed by *Concepts* and *Conceptual Relations* where every arc links some Conceptual Relation *r* to some concept *c* [5].

With Concept we intend an abstract notion that refers to a particular Conceptual Graph. Conceptual Graphs are used to link Concepts underlying the knowledge domain with several kinds of relations: (prerequisite, sub-concept, general relation, etc).

As we will see, Conceptual Graphs are massively used by ABITS functions in conjunction with Metadata fields for Cognitive State modeling and automatic Curriculum Generation.

2.2. Student Modeling

ABITS student models are composed by a *Cognitive State* and a set of *Learning Preferences*.

The ***Cognitive State*** contains the knowledge degree, reached by a particular student, of every ABITS tested domain Concept [6]. We represent this information by using an array of *Fuzzy Numbers* (one for each concept).

The decision to use Fuzzy Numbers [9] in ABITS Cognitive States arises from the necessity to manage uncertainty in the student evaluation process. In this way, in fact, we can admit different kind of evaluations with different degree of reliability.

As an example, when a student reads an ABITS delivered expositive Learning Object (i.e. a lesson) with a particular set of Concepts involved, ABITS can infer that there will be a little increase in the student Cognitive

State relating to these concepts but with a very large degree of uncertainty.

Conversely, when the same student answers correctly to a test relating to the same Concepts, his Cognitive State will be increased too in relation to involved concepts but with a lower and lower degree of uncertainty. To represent this kind of added information we use more and more narrow fuzzy numbers.

Moreover, in order to model the attitude that have humans to forget what they learn, ABITS applies a *Forgetting Function* to Cognitive States. Cause that not all humans forget in the same manner, this algorithm don't decrease concept knowledge degrees but only wides the amplitude of representing fuzzy number signifying that evaluations are more and more unreliable.

Within ***Learning Preferences*** we enclose information about the student perceptive capabilities i.e. to which kind of resources a specified student is shown to be more receptive [6]. To evaluate student preferences ABITS exploits Metadata elements contained in the *Educational* IEEE Metadata Category such as: *Format* (kind of media), *Difficulty*, *Pedagogical Approach*, *Interactivity Level* and *Semantic Density*.

To evaluate student Preferences ABITS exploits this idea: during the learning process there are *Milestones* (points in the student Curriculum) chosen by tutors where the Cognitive State is updated with respect to activities performed by students. After this point, a new evaluation is given for each Concept involved in student performed activities.

ABITS can evaluate the pedagogical effectiveness of Learning Object typologies by exploiting the *Conceptual Variation* and the Educational Metadata information about visited Learning Objects between couples of subsequent Milestones.

ABITS calculated information about Student Models can be exploited directly by tutors or re-used by ABITS in the Automatic Curriculum Generation procedure.

2.3. Automatic Curriculum Generation

Each student can be assigned to one or more different *Courses*. An ABITS Course is composed by a set of *Learning Goals* and by a *Curriculum*.

With ***Learning Goals*** (that are strongly different from Learning Objects) we intend a set of key Concepts necessary to be learnt to successful complete a specific Course. Such Concepts (as all other Concepts) are part of a Domain and are represented inside the Conceptual Graph of such Domain.

With ***Curriculum*** we intend, instead, an ordered list of Learning Objects that can be used to provide to a specific student all necessary knowledge to complete a specific Course. While Learning Goals indicate what (which Concepts) a student has to learn, Curriculum specify how these Concepts has to be learnt.

Different students can require different Curriculum to learn about same Learning Goals depending on their Cognitive States and Learning Preferences. For this reason a Curriculum Generation procedure is also provided by ABITS.

Curriculum Generation is done by constructing the best Concept sequence from the Conceptual Graph of the domain in relation to target Learning Goals and to student Cognitive State (each learning goal and each prerequisite not already known by student is inserted in this sequence) and by transforming this concept sequence into a Learning Object sequence looking at student preferences and inserting testing activities. A full description of such algorithm can be found in [6]; an example of Curriculum Generation is presented in chapter 5.

The Curriculum Generation facility can be used simply to help tutors during the course management phase or directly to change in run-time student Curriculum basing on their performed activities.

3. The Multi-Agent paradigm

As we already seen, ABITS is conceived as a Multi-Agent System. The purpose of this chapter is to provide fundamentals about Agents, Multi-Agent Systems (MAS) and Agent Oriented Software Engineering (AOSE).

In particular, our main goal is to justify our choice to implement ABITS using a Multi-Agent paradigm. As we will see in 3.2, this choice is based mainly upon Agent and MAS characteristics. This involve that an introduction to the concept of Agent and MAS is needed (paragraph 3.1).

Finally (paragraph 3.3) we will discuss about which tools can help in developing MAS application and why we adopted JAFMAS [11].

3.1. The concepts of Agent and MAS

Arguably, the most significant improvements in the field of software engineering have resulted from the introduction of powerful abstractions for managing the software inherent complexity. The key advances in program design and development over the past three decades – procedural abstraction, abstract data types, and most recently, object oriented programming (OOP) – all represent increasingly powerful examples of such abstraction. Probably the single most compelling argument in favor of agents for software engineering is that they represent yet another such abstraction.

No consensus on what is an agent, but several key concepts are important to this emerging paradigm. A software agent:

- is an autonomous, goal-directed process;
- is situated in, is aware of, and reacts to its environment;
- co-operates with other agents (software or human) to accomplish tasks.

Software agents offer a new paradigm for large scale distributed heterogeneous applications. The paradigm focuses on the interactions of autonomous, cooperating processes which can adapt to human & other agents [12].

Mobility is an orthogonal characteristic which many, but not all, consider important [13]. Intelligence is always a desirable characteristic but is not strictly required by the paradigm. The paradigm is still forming.

The roots of agent oriented (AO) methodology attain from OOP methodology [14], [15], [16] and AI studies [17]; hence, an intelligent agent may be defined as *"a decision making system that acts on and reacts to the environment"*.

Agent-to-agent communication is the key to realize the potential of the agent paradigm, just as the development of human language was the key to the development of human intelligence and societies. Agents use an *Agent Communication Language* (ACL) to share information and knowledge. This lead to the concept of MAS.

MAS can be defined as loosely-coupled networks of communicating and cooperating agents working together to solve problems that are beyond their individual capabilities.

In order to obtain coherent system behavior, individual agents in a MAS are not only able to share knowledge about problems and solutions, but also to reason about the processes of coordination among other agents.

3.2. Why we adopted an AOSE approach

As we seen in chapter 2, ABITS is a MAS able to extend a traditional Course Management System with a set of "intelligent" functions. It is a software module able to acts transparently as an intelligent engine with respect to functionality normally exhibited by a CMS.

This means that, when an "intelligent" function about Student Modeling or Curriculum Generation is needed, then the CMS requests a service to ABITS. Such behavior is depicted in figure 1.

In order to fulfil its tasks, there are two abilities that ABITS must have:

- an intelligent behavior;
- the ability to react to concurrent service requests.

Given that, it is easy to describe the reasons motivating our choice in favor of a MAS technology to provide ABITS services.

- The agent paradigm is inherently distributed in nature, the intrinsic concurrency of our task (many students can require concurrent evaluations) can take advantage from the presence of a computer network (agents can be placed on different machines).
- The MAS approach ensures an high level of workload scalability. This means that a Multi Agent System is intrinsically able to share the workload ("intelligent" services requested by the CMS) through the pool of agents (variable in number) distributed on different machines in order to optimize the reply time and the workload of all servers.
- The MAS approach, moreover, ensures an high level of functional scalability. New "intelligent" functions can in fact be added to the system simply by adding new agents providing such functions. No modification must be made to the system behavior. Simply a new correspondence event – agent service must be added.
- The MAS architecture allows to obtain optimum solutions through cooperation between various kind of agents. As an example, to update the whole student model, four ABITS agents must interact. Each one of them can in fact obtain only partial solutions. If such agents are placed on different machines, the final solution can be reached in ¼ of the total needed time.
- The MAS system, thanks to the high redundancy that is able to allow (many agent can be placed on many machines), ensures an high level of robustness too. Exploiting such feature, ABITS is able to manage many error situation that can be verified during its activity (a server goes down, a sub-net is unreliable, an agent expires, etc).

3.3. MAS implementation issues

The choice of a proper tool to implement a Multi-Agent System can arm the developer with many advantages while, being careless about it, can prove to be constricting in the long run. The inherent difficulties encountered in implementing coordinated behavior in any MAS are essentially the following:

- *Communication*: how to enable agent communication, what communication protocols to use;
- *Interaction*: what language the agents should use to interact with each other and combine their efforts;
- *Coherence and Coordination*: how to ensure that the

agents coordinate with each other to bring about a coherent solution to the problem they are trying to solve.

Another important issue is the programming language. We chose Java because of the following advantages: architecture neutral and portable, multithreading, network savvy, security, object orientation, database connectivity, native methods.

Concerning the MAS developing tool, after a careful examination of the state of the art [18], we selected JAFMAS (Java-based Agent Framework for MAS) [11], because it pays special attention in satisfying all the discussed points. Furthermore, JAFMAS provides a well defined methodology to support all phases of MAS software development (from requirement specification to coding), according to most commonly accepted AOSE approaches [12], [18].

4. How ABITS is implemented

ABITS is a module of a greater system for Computer Based Training. Its task is limited to add "intelligence" to a classical Course Management System. This means that some kind of communication must be allowed between ABITS, CMS and external shared data sources. This led to a set of relations and interdependencies as shown in figure 2.

As you can see, ABITS and CMS modules are strongly separated, communication happens only in one way (from CMS to ABITS) using directed RMI invocations. Both modules, moreover, must have the ability to access to a shared Courses Database. CMS-ABITS interaction is matter for paragraph 4.2. Now we will describe the meaning of data sources shown in figure 2.

- *Course Material Database* contains Learning Object in the form of Web-deliverable files (we suppose that the course delivery happens via Web);
- *Metadata Base* contains all Metadata indexing Course Material in XML/RDF format [10] (such database can be modified using a *Metadata Authoring Tool*);
- *Log Database* contains all student activities performed during the learning experience (visited pages, test results, permanency times, etc);
- *Learner Models Database* contains ABITS-calculated Cognitive States and Learning Preferences for each student;
- *Users Database* contains all information about system users (log-in names, roles, assigned Courses, etc);
- *Courses Database* contains all information about Courses Learning Goals and Curriculum.

ABITS is conceived to be a MAS. The following paragraphs will describe the whole process of ABITS production. In particular, according to the most common AOSE approach ([12], [18]), the production of Agent software requires the following three steps:

1. Identification of the Agents;
2. Identification of the Logic Interaction among Agents;
3. MAS Implementation.

The following three paragraphs describe each one of such steps.

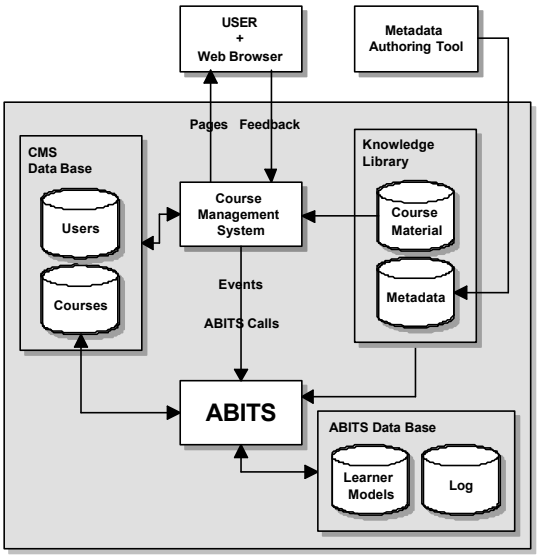

Figure 2. ABITS / CMS Interrelations

4.1. Identifying the Agents

The starting point to identify which kind of agents have to be implemented inside ABITS is the UML Use Case Diagram shown in figure 1. From this diagram all services that ABITS has to provide arise. There are:

- *Cognitive State Evaluation* (see 2.2);
- *Preferences Evaluation* (see 2.2 too);
- *Curriculum Evaluation* (see 2.3);
- *Complete Student Evaluation* (the *Evaluate All* case that is a macro-function calling the preceding ones).

Our system have exactly three kind of agents, one for each base function typology. In particular ABITS is composed by:

- ***Evaluation Agents*** that are interested of evaluating and updating Cognitive States (to do this, they interact with the *Metadata Base* and the *ABITS Database*);
- ***Affective Agents*** that are interested of evaluating and updating Learning Preferences (to do this, they interact with the *Metadata Base* and the *ABITS Database*);
- ***Pedagogical Agents*** that are interested of evaluating and updating Curriculums (to do this, they interact with the *Metadata Base*, the *ABITS Database* and the *Courses Database*).

Agent to agent communication is ensured by JAFMAS, the agent developing tool we adopt. Despite that, the CMS isn't an Agent: it is a stand-alone software application. In order to implement CMS-Agent communication a fourth kind of Agent is so necessary.

This Agent must have the capability to be invoked directly by the CMS and to act as an interface through CMS and ABITS. We name this fourth kind of Agent *Spooler Agent*.

Spooler Agent is embedded in the CMS module and can be instanced as any other Java class. CMS to ABITS communication happens simply by locally calling Spooler Agent methods. In other words, when CMS requires an ABITS service, it requests this service to the Spooler Agent. Spooler Agent then finds all available *Slave* Agents on the net (Evaluation, Affective or Pedagogical as needed) and requests the service to one of them (the choice is made to minimize Agent workload). Figure 3 depicts such kind of agent interaction.

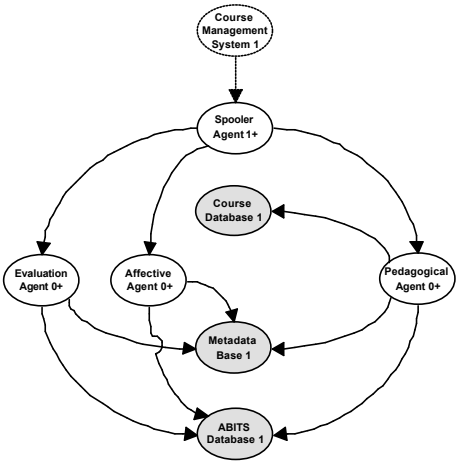

Figure 3. Agents-CMS-Data Sources Interaction

4.2. Identifying the logic interaction

ABITS Agents are divided in several multicast groups where each Agent kind corresponds to a specific group. Scalability in terms of ability to reply to concurrent requests is obtained simply by putting in each group an higher number of Agents (eventually on different hosts) as higher is the forecasted workload.

Spooler agents have to communicate with every agent typology so they are part of their group and of all other groups. Figure 4 depict the group subdivision of ABITS Agents.

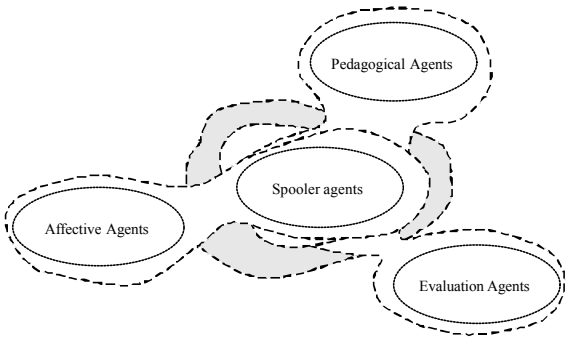

Figure 4. ABITS Agent Groups

In practice, the Spooler Agent has the role of *Agent Directory*: it identifies other agents on the net by sending multicast messages in an automatic fashion. When a *Slave* Agent is launched, it become visible to the Spooler Agent that adds the found *Slave* address in a list maintained inside its workspace. Conversely, when a *Slave* Agent is closed (by the user or because some kind of error occurs) it is cancelled from this list.

It is important to note that all agents (including the Spooler Agent) can be duplicated in order to increase the workload that ABITS can handle.

We described how ABITS Agent interaction happens; let's now formalize what we said. The interaction between Agents in a MAS are modeled in form of *Conversations*.

A ***Conversation*** is an Agent plan to achieve some goal, based on interactions with other agents. We identify every possible conversation that each Agent in our system can engage in, and we represent such Conversations by developing a Finite State Machine (FSM) model for each one of them.

In our MAS we identified only one Conversation type among a Spooler Agent and a *Slave* Agent (Pedagogical, Evaluation or Affective) in order to request and provide services. Such conversation can be seen in figure 5 form the Spooler Agent point of view (5a) and from the *Slave* Agent point of view (5b).

The Conversation is very simple but it allows to realize a scalable, robust and efficient system: if an error occurs, it is signaled to the Spooler Agent. When an error signal arrives through a Conversation, the Spooler Agent is able to perform a limited number of new attempt to the same or to a different Agent. If the error condition persists then

the exception is passed to the CMS for handling.

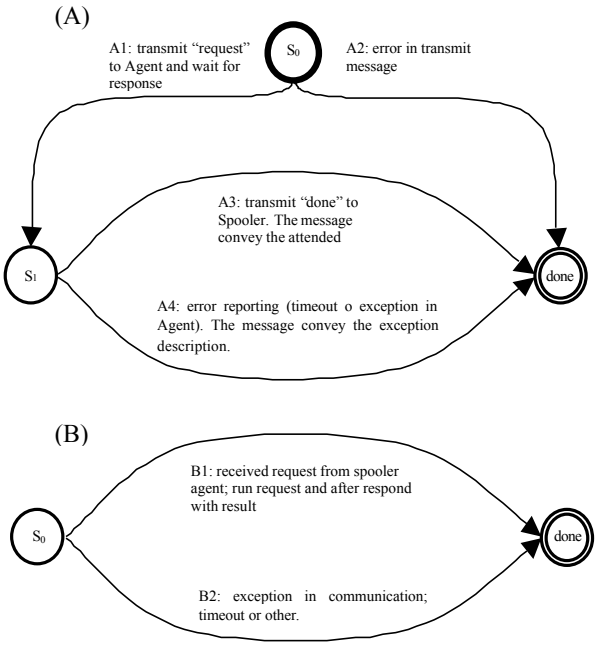

Figure 5. Spooler-Slave Agents Conversation : the Spooler (A) and the Slave (B) points ov view

4.3. MAS implementation

The implementation phase is beyond the purposes of this paper so we will introduce only some basic concept about this step. First of all it is important to remember that the purpose of this phase is the generation of the Java infrastructure for ABITS, starting from extension points introduced by the JAFMAS framework.

In particular, we were asked to extend some already defined JAFMAS class (e.g. `Agent` or `Conversation`) and implement some other new Java class to provide the system with the data structures needed to work properly.

In order to design a MAS able to find a coherent solution to the entire system problem, it is important to check the logical consistency of all Agent Conversations. Automata models like concurrent finite state machines and Petri Nets can provide useful tools for checking system coherency by analyzing the conversation models.

5. ABITS in action

In this section we will present some examples of how ABITS software works. Figure 6 shows how an ABITS Agent looks like. It is a Java SWING application (so able to be loaded in many Operating Systems) with the main window composed by two principal elements: on the left side there is the Agent Conversation tree while on the right side there is a big text box displaying user selected items.

The conversation tree is composed by two folders: *Multicast Group* and *Conversations*. The *Multicast Group* folder contains an item for each group the Agent is registered (the Agent in figure 6 is a Pedagogical Agent so it is registered only to the Pedagogical Group). Selecting such item, all messages related to the Multicast Group will appear in the right box.

The *Conversations* item, instead, contains three sub-folders:

- *Running* containing all agent-to-agent conversations still in action;
- *Ended With Error* containing all agent-to-agent conversations already terminated but in an error state;
- *Ended Without Error* containing all agent-to-agent conversations terminated in a correct ending state,

Selecting a conversation from the preceding folders, all messages relating to the agent-to-agent conversation will appear in the right box.

The following two paragraphs will present an example of CMS-ABITS interaction in two levels of details. Main data flows between user browser, CMS and ABITS agents will be described in paragraph 5.1.

Moreover, paragraph 5.2 is dealt with the analysis of a single Conversation between the Spooler Agent and a Pedagogical Agent during a request for the Curriculum Evaluation service.

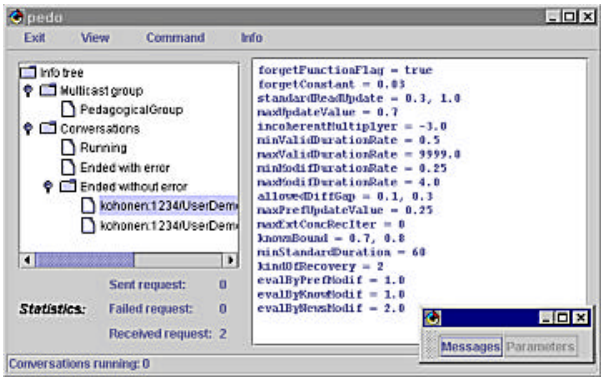

Figure 6. An ABITS Pedagogical Agent

5.1. CMS-ABITS Interaction Example

ABITS Agents can be launched on several servers and can be accessed from the CMS through its built-in Spooler Agent. As an example see figure 7.

A student learns from the system via Web using his own browser. When the student reaches a point on his curriculum (7a) where an ABITS function must be called

(this point is named Milestone) then the CMS is noticed about that by the browser and, on the server side, a Spooler Agent is invoked to provide such service (7b).

The service is requested by the Spooler Agent to an available *Slave* Agent (7c). The *Slave* Agent performs all calculations involved in the service, modifies something in some database (Curriculum sequence or Student Model) and then replies to the Spooler Agent with the *Done* performative (7d).

The control returns then to the CMS (7e) that sends to the browser the current page for the student (7f). It is important to note that, after the ABITS activity such page can be changed as the remaining part of the Curriculum.

5.2. An Example of Conversation

In this paragraph we will analyze in details, as an example of how ABITS works, a Spooler-Pedagogical Agent Conversation. Spooler Agent is requesting the *Update Curriculum* service and the conversation is shown from the Pedagogical Agent point of view.

Courier text inside boxes is text extrapolated directly from the Pedagogical Agent message box and represents Agent messages. Text in Times represents comments to such messages.

```
Conversation
kohonen:1234/UserDemoSpoolerAgentTwo-961688896904

Trying to execute rule r1

r1 looking for a recvd msg of type EvaluateCurr in msgQueue
r1 could not find expected recvd msg in the queue
Input condition not satisfied for rule r1
r1 not executed

Trying to execute rule r2

r2 looking for a recvd msg of type UpdateCurr in msgQueue
r2 found expected recvd msg in the queue
Input condition satisfied for rule r2
Executing rule r2
UpdateCurr service requested
```

The conversation starts. The string `kohonen...` in the second line represents the name of the conversation created in real time by the Spooler Agent.

After that the conversation starts, the Pedagogical Agent tries to apply rule ***r1*** but the service requested doesn't match (rule ***r1*** serves the `EvaluateCurr` request while this is an `UpdateCurr` request).

After that, the Pedagogical Agent tries to apply rule ***r2***

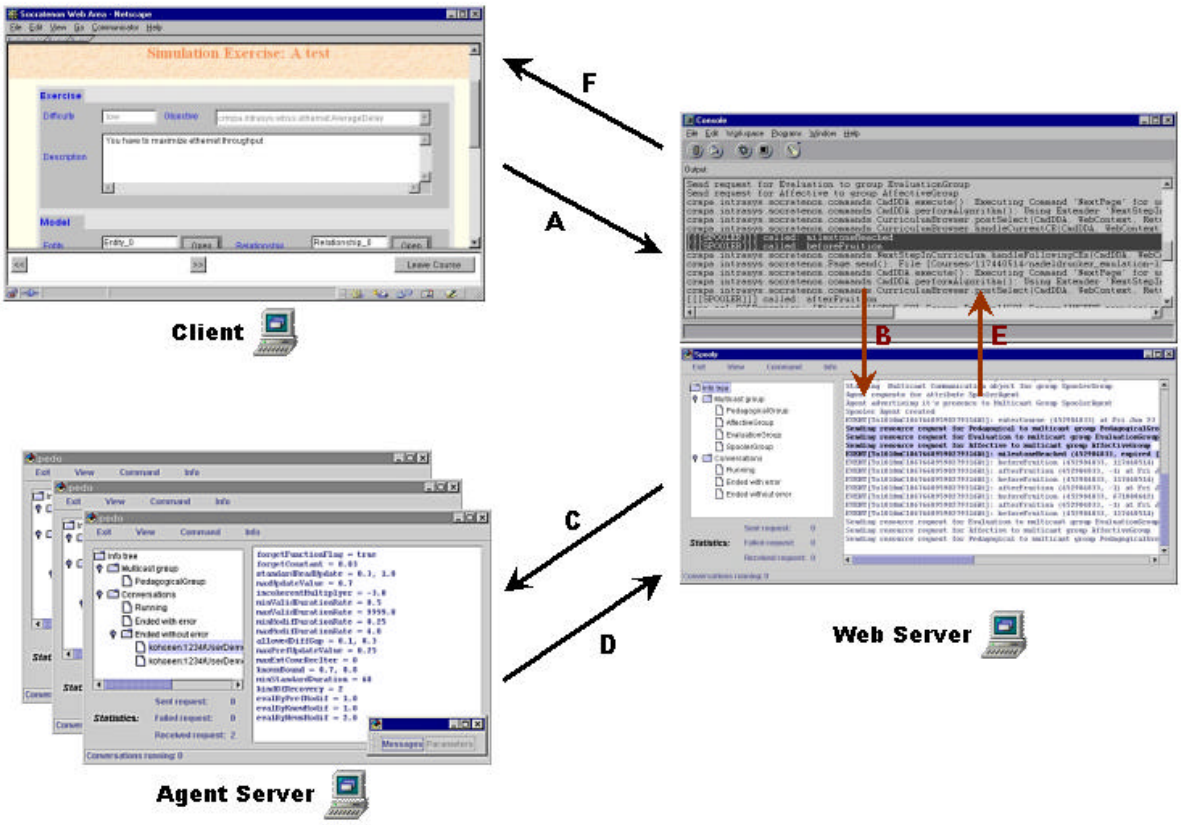

Figure 7. An example of Student-CMS-ABITS Interaction

that, serving the `UpdateCurr` request, can be applied. The curriculum evaluation starts.

```
Course Information
ID         = 452984833
Student ID = 50331653
Date       = 22/06/2000 06:06:09 PM
Position   = 21

Curriculum Sequence:
00 Expired [CM]
01 Marketing Definition (385875969) [PP]
02 Test 01 (385876012) [MCT]
03 Expired [CM]
04 Strategic Marketing Definition (385875970) [PP]
05 Test 02 (385876013) [MCT]
06 Expired [CM]
07 Sector (385875971) [PP]
08 Test 03 (385876014) [MCT]
09 Expired [CM]
10 Sector Structure (385875972) [PP]
11 Corporate Planning (385875973) [PP]
12 Test 05 (385876016) [MCT]
13 Test 04 (385876015) [MCT]
14 Expired [CM]
15 Macro-Segmentation (385875975) [PP]
16 Test 08 (385876019) [MCT]
17 Test 10 (385876021) [MCT]
18 Test 07 (385876018) [MCT]
19 Test 11 (385876022) [MCT]
20 Test 09 (385876020) [MCT]
21 Update All [CM] ←

Learning Goals:
Macro_Segmentation from Marketing
```

The Pedagogical Agent loads all information about the actual Course. First of all there are general information (course identifier, current position, etc) followed by the old Curriculum (viewed as a list of Learning Objects) where [PP] means Passive Presentation, [MCT] means Multiple Choice Test and [CM] means Curriculum Milestone.

Actual position is on the last Curriculum Milestone. For this reason the control passes to the Pedagogical Agent. The last line represents the Learning Goal associated with this Course. It is a Concept identifier.

```
Cognitive State Information
ID             = 335544331
Student ID     = 50331653
Date           = 22/06/2000 06:32:09 PM
Visited Frames = 6
Duration Rate  = 0,921

Evaluated Concepts:
[1,000; 0,251] Marketing_Definition from Marketing
[1,000; 0,251] Strategic_Marketing_Definition from Marketing
[1,000; 0,250] Sector from Marketing
[1,000; 0,250] Sector_Structure from Marketing
[0,000; 0,650] Corporate_Planning_Definition from Marketing
[0,000; 0,212] Macro_Segmentation_Definition from Marketing
[0,700; 0,500] ASA from Marketing
[0,930; 0,500] SBU from Marketing
[0,090; 1,000] BCG from Marketing
[0,000; 0,400] GE from Marketing
```

The Second thing to do is to load the actual Cognitive State for the student under examination (i.e. the list of Concepts already evaluated for this student). In this list the first numeric couple of each row is a triangular fuzzy number representing the knowledge degree reached by the student for the concept in that row.

The first number of the couple is strictly the knowledge degree (from 0 = worst to 1 = best) while the second one is the reliability of such evaluation (from 0 = certain to 1 = uncertain).

In particular, first four concepts are fully known by the student as the 7^{th} and 8^{th}. The student shows some difficult to understand the other concepts.

```
Preferences Information
ID             = 352321543
Student ID     = 50331653
Date           = 22/06/2000 06:32:15 PM
Max Frame Size = 1000 Kb
Languages      = en it

Courseware Genre Preferences:
[-0,014; 0,761] hypertext
[-0,014; 0,580] image
[ 0,750; 0,474] text

Pedagogical Approach Preferences:
nothing

Interactivity Level Preferences:
[-0,028; 0,761] medium
[ 0,750; 0,474] very low

Semantic Density Preferences:
[ 0,722; 0,433] low
[ 0,000; 0,638] very low

Difficulty Preferences:
[ 0,222; 0,574] low
[ 0,500; 0,497] very low
```

Thirdly, information about user preferences on learning material is required. In the list, an element with a positive associated fuzzy number represents a preferred genre while an element with a negative associated fuzzy number represents a disagreed one.

```
Required Concepts
ASA from Marketing
BCG from Marketing
Macro_Segmentation_Definition from Marketing
GE from Marketing
Corporate_Planning_Definition from Marketing
```

The Pedagogical Agent starts his curriculum evaluation. All concepts required to reach the learning goals minus those already known are calculated.

```
Found material about Goals
Macro-Segmentation (385875975) [PP]
Corporate Planning (385875973) [PP]
```

The Pedagogical Agent finds in the Metadata base all material explaining required Concepts (some Learning Object explains more that one Concept). Found Learning Objects has to match student Preferences too.

```
First Curriculum

Curriculum Sequence:
00 Corporate Planning (385875973) [PP]
01 Macro-Segmentation (385875975) [PP]

Learning Goals:
Macro_Segmentation from Marketing
```

The first Curriculum approximation is generated by the Agent simply ordering found material.

```
Second Curriculum

Curriculum Sequence:
00 Corporate Planning (385875973) [PP]
01 Test 05 (385876016) [MCT]
02 Test 08 (385876019) [MCT]
03 Update Cognitive State [CM]
04 Macro-Segmentation (385875975) [PP]
05 Test 10 (385876021) [PP]
06 Test 07 (385876018) [PP]
07 Test 11 (385876022) [PP]
08 Update All [CM]

Learning Goals:
Macro_Segmentation from Marketing
```

The Agent generates the second approximation of the best Curriculum by adding testing material and evaluation Milestones about explained Concepts to the first one.

```
Last Curriculum
ID            = 452984833
Student ID    = 50331653
Date          = 22/06/2000 06:06:09 PM
Position      = 22

Curriculum Sequence:
00 Expired [CM]
01 Marketing Definition (385875969) [PP]
02 Test 01 (385876012) [MCT]
03 Expired [CM]
04 Strategic Marketing Definition (385875970) [PP]
05 Test 02 (385876013) [MCT]
06 Expired [CM]
07 Sector (385875971) [PP]
08 Test 03 (385876014) [MCT]
09 Expired [CM]
10 Sector Structure (385875972) [PP]
11 Corporate Planning (385875973) [PP]
12 Test 05 (385876016) [MCT]
13 Test 04 (385876015) [MCT]
14 Expired [CM]
15 Macro-Segmentation (385875975) [PP]
16 Test 08 (385876019) [MCT]
17 Test 10 (385876021) [MCT]
18 Test 07 (385876018) [MCT]
19 Test 11 (385876022) [MCT]
20 Test 09 (385876020) [MCT]
21 Expired [CM]
22 Corporate Planning (385875973) [PP] ←
23 Test 05 (385876016) [MCT]
24 Test 08 (385876019) [MCT]
25 Update Cognitive State [CM]
26 Macro-Segmentation (385875975) [PP]
27 Test 10 (385876021) [PP]
28 Test 07 (385876018) [PP]
29 Test 11 (385876022) [PP]
30 Update All [CM]

Learning Goals:
Macro_Segmentation from Marketing
```

The last Curriculum is then generated by merging old and new Curriculums (in order to ensure visibility on all previous seen material) an by positioning the student on the next page.

It is important to note that the Milestone generating the Agent call is transformed in an *Expired* Milestone.

```
r2 transmitting msg type Done
UpdateCurr service provided
r2 executed
Final State reached
```

Finally the new Curriculum is stored in the Courses database, the *Done* performative is sent back to the Spooler Agent and the Conversation is closed.

6. Conclusions

We introduced in this paper an Intelligent Tutoring Systems based on the adoption of Software Agents. The main innovative issues can be found in the distributed nature of the implementation and in the flexibility of the Agent approach (different Agents have been defined and cooperate to solve tutoring tasks).

A prototype version of ABITS has been implemented and is currently undergoing to significant experimental campaigns in the framework of a European Esprit Project called InTraSys [19].

ABITS was integrated with a core engine for Web Courses Delivery and with some simulation tools. The implementation choices reveled to be suitable for integrating heterogeneous Distance Learning modules based on the use of the Java language.

During the experimental campaigns, ABITS facilities will be tested in detail so to determine the real effectiveness of our system in supporting Tutors and improving the quality of the learning process.

Further study will concern both the architecture and the model of the Intelligent Tutoring Systems and they will be defined in details only after the end of the experimentation and of a careful analysis of the obtained results.

References

[1] Winkels R., "Explorations in Intelligent Tutoring and Help", *IOS*, Holland, 1992

[2] Mark M.A. and Greer J.E., "Evaluation Methodologies for Intelligent Tutoring Systems", *Artificial Intelligence and Education 4*, 1993

[3] Alem L., "Intelligent tutoring system: a knowledge based approach", *Proc. 8th Int. Conf. Industrial and engineering applications of artificial intelligence and expert systems*, 1995

[4] Soloway E., "Interactive learning environments: where they've come from where they're going", *Proc of CHI '96*, 1996

[5] N. Capuano, F. Cirillo, S. Salerno, A. Cozzolino, M. Marsella, "Educational Knowledge Manipulation Through Metadata: a Real Case"

[6] N. Capuano, M. Marsella, S. Salerno, "ABITS: An Agent Based Intelligent Tutoring System for Distance Learning", *Proceedings of ITS 2000, Montreal,Canada, June 19-23 2000*, Springer-Verlag.

[7] W. Hodgins et al., "Learning Object Metadata, Working Draft Document 2.5", *IEEE Learning Technology Standards Committee (LTSC)*, http://ltsc.ieee.org, 1999.

[8] Conceptual Graphs "Conceptual Graph Standard", *draft proposed dpANS*, http://www.bestweb.net/%7Esowa/cg, 1999.

[9] D. Dubois, H, Prade "Fuzzy Sets and Systems – Theory and Applications" *Academic Press*, 1980.

[10] W3C, "Resource Description Framework (RDF) Model and Syntax Specification", *World Wide Web Consortium Proposed Recommendation*, http://www.w3.org/RDF, 1999.

[11] A. Galan, "JAFMAS, a Java-based Agent Framework for Multi-Agent Systems", *Department of Electrical and Computer Engineering and Computer Science University of Cincinnati* (PhD thesis) http://www.ececs.uc.edu/~abaker/JAFMAS/

[12] T. Finin, Y. Labrou, *"Tutorial on Agent Communication Languages*", University of Maryland Baltimore County, First International Symposium on Agent Systems and Applications and the Third International Symposium on Mobile Agents, ASA/MA'99, (featuring the Third Dartmouth Workshop on Transportable Agents), October 3 - 6, 1999, Rancho Las Palmas Marriott Resort and Spa, Palm Springs, California, U.S.A.

[13] C.G.Harrison, D.M. Chess, A.Kershenbaum *"Mobile agents: are they a good idea?",* IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY 10598

[14] M.J. Pont, E. Moreale, "Towards a practical Methodology for Agent Oriented Software Engineering with C++ and Java", *Technical Report 96-33,* Department of Engineering, Leicester University Dec. 1996.

[15] Z. Guessoum, J..P. Briot, "From Active Objects to Autonomous Agents", *IEEE Concurrency*, July-September, 1999, pp.68-76

[16] C. Hayes, "Agents in a Nutshell – A Very Brief Introduction", *IEEE Transactions on Knowledge and Data Engineering*, Vol.11, No.1, January-February 1999.

[17] M. Wooldridge, "Agent-based software engineering", *IEEE Proc.-Softw. Eng., Vol.144, No.1*, Frebruary 1997

[18] http://www.AgentLink.org/activities/sigs/sig2.html

[19] http://dbs.cordis.lu/cordis-cgi/srchidadb?ACTION=D&SESSION=227102000-6-26&DOC=2&TBL=EN_PROJ&RCN=EP_RCN:48519&CALLER=EN_UNIFIEDSRCH