MATHTUTOR: A MULTI-AGENT INTELLIGENT TUTORING SYSTEM

Janétte Cardoso
Institut de Recherche Informatique de Toulouse (IRIT) - UT1
jcardoso@univ-tlse1.fr

Guilherme Bittencourt, Luciana B. Frigo, Eliane Pozzebon and Adriana Postal
Departamento de Automação e Sistemas (DAS) - UFSC
{ gb | lu eliane | apostal }@das.ufsc.br

Abstract

In this paper we propose a multi-agent intelligent tutoring system building tool that integrates different formalisms in order to facilitate the teacher task of developing the contents of a tutorial system and at the same time to provide adaptiveness and flexibility in the presentation. The adopted formalisms are ground logic terms for the student model, data-bases for the domain model and object Petri nets for the pedagogical model. The interaction between the student and each agent of the system is controlled by an object Petri net, automatically translated into a rule-based expert system. The object Petri net tokens are composed by data objects that contain pointers to the student model and to the domain knowledge, stored into a data-base of texts, examples and exercises. The object Petri net transitions are controlled by logical conditions that refer to the student model and the firing of these transitions produce actions that update this student model.

Keywords: Intelligent Tutoring Systems, Intelligent Agents, E-Learning, Expert Knowledge-based Systems, Design Methodologies

Introduction

The field of Artificial Intelligent in Education (AI-ED) includes several paradigms, such as Intelligent Computer Aided Instruction (ICAI), Micro-world, Intelligent Tutoring Systems (ITS), Intelligent Learning Environment (ILE) and Computer Supported Collaborative Learning

conference (CSCL), that originated many systems [12],[15],[14],[4]. Moreover, innovative computer technology, such as hyper-media, Internet and virtual reality, had an important impact on AI-ED [3],[2].

Nevertheless, the task of building ITSs that cover a rich domain and at the same time are adaptive to user level and interests continue to be a very complex one. On the one hand, the domain knowledge must be structured and its presentation planned by a human teacher in an attractive and interesting way. On the other hand, differences among the students, such as background knowledge, personal preferences and previous interactions with the system, must be taken into account.

In [11], Mizoguchi and Bourdeau identifies several drawbacks of current AI-ED tutoring systems, e.g. the conceptual gap between authoring systems and authors, the lack of intelligent authoring methodologies, the difficulty of sharing and reusing components of ITS, the gap between instructional planning and tutoring strategy for dynamic adaptation of the ITS behavior. Furthermore, they note that all these drawbacks are content-related and do not depend directly on issues such as representation and inference formalisms.

In this paper we propose a multi-agent ITS building tool, called MathTutor, that integrates different formalisms in order to facilitate the teacher task of developing the contents of a tutorial system and at the same time to provide adaptiveness and flexibility in the presentation. Multi-Agent Systems (MAS) technology have been of great help in reducing the distance between ideal systems and what can really be implemented, because it allows to simplify the modeling and structuring tasks through the distribution, among different agents, of the domain and student models. The proposed tool is based on a conceptual model, called MATHEMA [6], that provides a content-directed methodology for planning the domain exposition and teaching strategies. The main contribution of the paper, besides instantiating the MATHEMA model into a working tool, is to propose an original way of integrating the student and pedagogical models using *Object Petri Net (OPN)* [16].

The MathTutor tool has been implemented using the Java programming language environment. The expert systems included in the system were implemented using Jess [9], a Java Expert System Shell based on the widely used Clips system. The multi-agent society is supported by Jatlite [10], a Java platform that implements the KQML agent communication language [8]. The interface is based on Servlets and uses the HTTP protocol and can be accessed through any browser, what allows the use of the system as a distance learning tool. Using this tool, a prototype of an Intelligence Tutor System, aimed to teach *Information Structure* was implemented. The course is based on the programming

language Scheme [1, 7] and covers the program of an undergraduate discipline of the Control and Automation Engineering course at the University of Santa Catarina, Brazil.

The paper is organized as follows. In Section 1, the ITS building tool architecture is described. In Section 2, the architecture of the agents that compose the multi-agent society and the tutor systems that they contain are presented. In Section 3, the authoring mechanism that allow the design of new courses to be included into the system is described. In Section 4, some related work is presented. Finally, in Section 5, we present some conclusions and future work.

1. Tool Architecture

The tool architecture is based on the conceptual model MATHEMA [6] and consists basically of three modules: the society of tutorial agents (TAS), the student interface and the authoring interface. The interface provides access to the system through any Internet browser. The authoring interface module allows the definition of the course structure and contents and is discussed in more detail in Section 3. Finally, the society of tutorial agents consists of a multi-agent system where each agent, besides communication and cooperation capabilities, contains a complete intelligent tutorial system focused on a sub-domain of the target domain. The fact that the system consists of a multi-agent society allows the distribution of domain contents and student modeling data among the several agents that cooperate in the tutoring task.

The MATHEMA conceptual model [6] provides a partitioning scheme, called the *external view,* leading to sub-domains. This partitioning scheme is based on epistemological assumptions about the domain knowledge and consists of a three dimensional perspective. The proposed dimensions are: *context:* along the context dimension the domain knowledge is partitioned according to a set of different points of views about its contents; *depth:* given one particular context, the associated knowledge can be partitioned according to the methodologies used to deal with its contents; *laterality:* given one particular context and one particular depth, complementary knowledge can be pointed to in order to allow the student to acquire background knowledge not in the scope of the course or to reach related additional contents. Each sub-domain defined according to this scheme is under the responsibility of a different agent, that contains a complete ITS specialized in the sub-domain. This fact facilitates the specification of the course contents, because the teacher can concentrate in each sub-domain. During the execution of the system, if

one agent concludes that the next tutoring task is out of its capabilities, it asks the other agents of the society for cooperation.

In the case of the implemented ITS prototype, the domain knowledge – Information Structure – is divided into two contexts – theoretical and practical –, and each of these contexts is worked out in two depths – procedural abstraction and data abstraction. Therefore, the tutorial agent society consists of four agents, each one responsible for one of the following sub-domains: *TP* - theoretical procedural abstraction; *PP* - practical procedural abstraction; *TD*- theoretical data abstraction; *PD*- practical data abstraction. Lateral knowledge includes computer architectures, programming languages courses, in particular about the Scheme language [7], complexity analysis, software engineering techniques, among other.

According to the *internal view* of the MATHEMA conceptual model, the knowledge associated with each sub-domain (TP, PP, TD and PD in the case of the ITS prototype) is organized into one or more curricula. Each *curriculum* consists of a set of pedagogical units and each *pedagogical unit* is associated to a set of *problems.* In the first interaction of a new student with the system, the interface module asks for identification data, basic preferences and background knowledge, and builds the initial student model. The control is then passed to one of the TAS agents. In the first interaction, typically this agent would be the one that is responsible for the initial pedagogical unit of the course, the *theoretical procedural abstraction (TP),* in the case of the implemented ITS prototype.

2. Agent Architecture

Each agent in the TAS has the architecture shown in fig. 1. The behavior of the agent is controlled by the *Coordinator* module and consists of the following activities: (i) According to the information in the student model, one predefined curriculum is chosen, (ii) The rules that implement the pedagogical model of the chosen curriculum are loaded and the expert system shell inference engine is started. (iii) The inference engine, based on the rules and on the information in the student model, infers which pedagogical unity should be used next. (iv) The coordinator extracts the appropriate interaction data associated with the inferred pedagogical unity, typically a HTML page, from the domain knowledge data base and sends it to the interface. (v) If the pedagogical unity does not need any interaction with the Scheme program, the result of the interaction with the student is used to directly update the student model. (vi) If some interaction with the Scheme program is

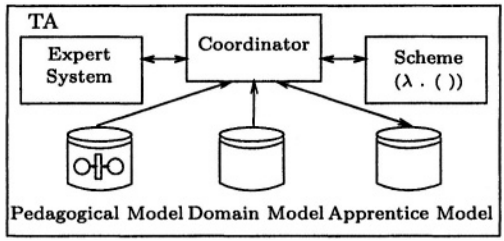

Figure 1. Tutorial Agent

needed, the Coordinator performs it and returns the results to the interface, that would typically show to the student these results in the same HTML page. (vii) When the interface finishes the interaction with the student, the coordinator runs again the inference engine and the process is repeated.

Two special events may stop the above behavior: (i) the present curriculum ends, or is interrupted by the student, and in this case either a new curriculum is chosen or the session is terminated; (ii) the present curriculum rules infer that the next interaction should be controlled by another agent, e.g., to switch between theoretical and practical contexts, or to change the depth of the present context, and in this case, the coordinator uses its communication and cooperation capabilities to inform the other agent that it should take the control. In the following subsections the three models (see fig. 1) are presented in detail.

2.1 Domain Model

The adopted formalism for the domain model is a data base of Servlets definitions, i.e. interactive HTML pages, structured according to the pedagogical approach proposed in the MATHEMA conceptual model [6]. This approach is inspired both by the constructivism [13] and by the Vygotsky's theory of social knowledge [17]. The idea is to allow the student to acquire and construct knowledge through the interaction with the tutor system, that is designed with the aim of reinforcing the active participation of the student in the learning process. To achieve this goal, the interaction is based on cooperative problem solving activities, combining *learning by doing* and *learning by being told* [5].

According to the internal view of the MATHEMA conceptual model, the knowledge associated with each sub-domain (TP, PP, TD and PD in the case of the ITS prototype) is organized into one or more curricula. Each curriculum consists of a set of pedagogical units ordered according to prerequisites. For instance, in the Information Structure ITS prototype, the pedagogical units associated with the sub-domains

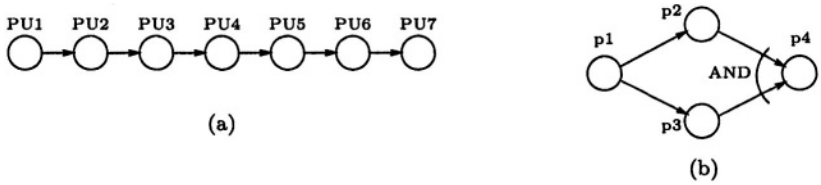

Figure 2. Prerequisite graphs: (a) Curriculum CV1, (b) Pedagogical Unit PU1

TP and PP (Procedural abstraction) are: PU1 - Primitive procedures; PU2 - Compound Procedures; PU3 - Interaction and recursion; PU4 - Higher-Order Procedures; PU5 - Procedures as Arguments; PU6 - Procedures as General Methods; PU7 - Procedures as Returned Values. In this example, both contexts – theoretical and practical – have the same pedagogical units, what changes is the point of view about the subject.

A possible curriculum prerequisite graph for the "theoretical procedural abstraction" domain is represented in fig. 2(a). Each pedagogical unit is associated with a set of problems, in the sense that, if the student is able to solve the problems associated with a given pedagogical unit, the system considers that the contents of the unit have been learned. The problems are also ordered according to prerequisite. The prerequisite order graph is defined by the teacher using the authoring interface (see Section 3). A possible prerequisite graph for the problems in pedagogical unit "primitive procedures" (UP1) is represented in fig. 2(b). Note that p_4 has two prerequisites – p_2 and p_3 – and they can be done in any order. Each problem is associated with a set of interactive HTML pages that support the problem solving activities. These pages are of three types – explanations, examples and exercises – and the pages of each type are ordered by difficulty.

The interactive pages that implement a problem have a standard format with some navigation controls, such as *exit, proceed, repeat,* etc. Explanations and examples are just HTML text pages to be read by the student and typically would be available in different levels of complexity for each content. Exercises can be (multiple) choice questions, questions that ask for some symbolic or numerical answers or questions asking for some Scheme implementation, that need the interaction with the Scheme program to be tested. Below, we present a fragment of the domain model structure for the pedagogical unit "primitive procedures" of the TP and PP sub-domains, in the Jess knowledge base syntax:

```
(deffacts domain
(ped-unit (name ped-unit1)
          (title "Primitive procedures")
          (problems (p1 p2 p3 p4))) ...
(problem (name p1)
         (ped-unit ped-unit1)
         (question "What are computers and programming languages?")
```

```
        (explanations (p1-exp1 p1-exp2 p1-exp3))
        (examples (p1-exa1 p1-exa2 p1-exa3))
        (exercices(p1-exe1 p1-exe2 p1-exe3))) ...
(page (name p1-exe1)
      (type exercice)
      (title "Fibonacci number")
      (location "/var/www/mathtutor/exercices/p2/e1.html")
      (return number)) ...)
```

The system knowledge base is represented by Jess *unordered facts* with the following structure: $(\langle object \rangle \ (\langle slot \rangle \ \langle value \rangle) \cdots (\langle slot \rangle \ \langle value \rangle))$, where $\langle object \rangle$ is the type of the fact, $\langle slot \rangle$ the name of an attribute and $\langle value \rangle$ its associated value. The three levels of the domain model structure can be clearly seen in the representation: a pedagogical unit contains a set of problems and each problem is associated with sets of explanations, examples and exercises objects, represented by `page` objects, where a pointer to the associated interactive HTML page is stored.

2.2 Student Model

The adopted formalism for the student model is the subset of first-order logic supported by the Jess knowledge base mechanism. This mechanism consists of a base of facts containing ground logic terms and a query language that may contain terms with variables. The model contains static and dynamic information. The static information consists of identification data (name, origin, background knowledge, etc.) and preferences (theoretical oriented, practical oriented, first overview than detail, step by step, etc.). The dynamic information consists of descriptions of the student activities during all the interaction sessions of the student with the system. The dynamic information is stored into a distributed knowledge base, where each one of the TAS agents stores the details of its own interactions with the student and just summaries of the interactions of the other agents. These summaries contains basically which pedagogical units have been visited and the degree of advance in each of them.

Below, we present a fragment of the student model in the Jess knowledge base syntax:

```
(defglobal ?*app* =
 (assert (student (name "Maria")...(doing pu1-p1)(what examples))))
(deffacts student
(problem (student ?*app*)
         (ped-unit ped-unit1)
         (name p1)
         (to-be-done-explanations (p1-exp2 p1-exp3))
         (to-be-done-examples (p1-exa3))
         (to-be-done-exercices (p1-exe2 p1-exe3))) ...
(explanations (student ?*app*)
              (name p1-exp1)
```

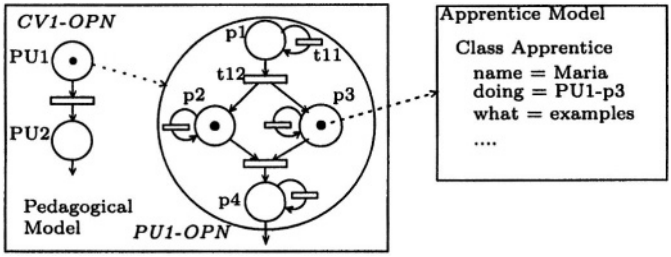

Figure 3. Object Petri Net

```
              (session 1)) ...
(example (student ?*app*)
         (name p1-exa1)
         (session 1)) ...
(exercice (student ?*app*)
          (name p1-exe1)
          (session 2)
          (result ok)) ...)
```

The student identification data is stored in a fact assigned to a global variable (`?*app*`). In particular, this fact contains the present interaction situation of the student: in the slot `doing`, the current pedagogical unit/problem and, in the slot `what`, the current activity. The student model consists of objects of type `problem`, `explanation`, `example` and `exercise`, linked to the student identification data by the `student` slot, where the details of the interaction, such as the session number in which the object was accessed, the result obtained, etc, are stored.

2.3 Pedagogical Model

The pedagogical model controls the interaction between the student and each agent of the system and is implemented by an object Petri Net (OPN) [16], automatically translated into a rule-based expert system. The OPN and its associated expert system are generated by the Authoring Interface (see Section 3) based on the description, provided by the teacher, of courses sequences (curricula and prerequisite order of pedagogical units and problems) and contents (explanations, examples and exercises, ordered by difficulty). An important point in the proposed approach is that the teacher does not need to specify the interaction with the student model, this interaction is automatically included in the OPN using the prerequisite and difficulty orders defined by the teacher.

In an OPN, the tokens are object instances. A OPN is defined by a control structure (places, transitions and arcs connecting places to/from transitions) and by the data structure of its tokens. In our case, all places and tokens have the same class (type): student. The generated OPN is hierarchical, because nodes of the curriculum graph are peda-

gogical units, and each pedagogical unit is itself a graph whose nodes are problems (see fig. 3). The tokens are instances of the student class whose associated data structures are defined in Section 2.2. A token in a place PU_i of the curriculum OPN represents that the student is doing pedagogical unity PU_i. Each place PU_i is *exploded* in a PU_i-OPN whose places are problems p_i. A student token in a place p_i means that the student *can* do it. If the token attribute `doing` is set to PU_i-p_j, the student is actually *doing* problem p_j of pedagogical unit PU_i. Considerer fig. 3, the student is doing PU_1 (m(PU_1)=**Maria**), can do PU_1-p_2 and PU_1-p_3 (m(p_2)=m(p_3)=**Maria**) and is actually doing p_3 (**Maria.doing**=PU_1-p_3), in particular doing an example associated with problem p_3, because `Maria.what` = examples.

The arcs represent the prerequisite order between pedagogical units, in curriculum OPN, and between problems, in the problem OPN. Besides these prerequisite conditions, transitions have also extra firing conditions controlled by the student model. The firing of transitions produce actions that update the student model. These extra conditions, predefined into the OPNs, are not specified by the teacher and allow the system to be adaptive with respect to the different students.

Below, we present a fragment of the pedagogical model in the Jess knowledge base syntax, in the situation shown in fig. 3:

```
(deffacts pedagogical
(place (name pu1-p1)(ped ped-unit1)(probl p1))
(place (name pu1-p2)(ped ped-unit1)(probl p2)(token ?*app*))
(place (name pu1-p3)(ped ped-unit1)(probl p3)(token ?*app*))
(place (name pu1-p4)(ped ped-unit1)(probl p4))
(place (name pu2-p1)(ped ped-unit2)(probl p1))
...
(trans (name t11)(place-in pu1-p1)
                 (place-out pu1-p1)
                 (condition repeat))
(trans (name t12)(place-in pu1-p1)
                 (place-out pu1-p2) (place-out pu1-p3)
                 (condition continue)) ...)
```

Places and transitions are represented by Jess objects (classes `place` and `trans`). The tokens are the global variables associated with students and are stored in the slot `token` of the class `place`. Conditions are functions that access the student model to determine whether the transition should fire or not and are stored in the slot `condition` of class `trans`.

3. Authoring Interface

The complexity of the MathTutor architecture is a consequence of the intended goal of presenting the domain knowledge in an attractive and interesting way and, at the same time, to provide adaptiveness to

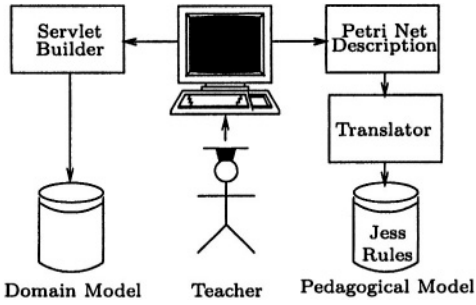

Figure 4. Authoring Mechanism

user level and interests. Nevertheless, this complexity makes the task of designing a course for the system equally complex.

To facilitate this task, an authoring mechanism is proposed (see figure 4). This mechanism, following the internal view of the MATHEMA conceptual model, has three levels. At the first level, the teacher specifies the curricula of the course. Each curriculum is composed by a set of pedagogical units and their possible sequences of execution. To specify each curriculum the teacher disposes of a graphical interface that allows the construction of graphs (see fig. 2(a)). Each graph is associated with a curriculum and each node of a graph is associated with a pedagogical unit. An edge from node pu_1 to node pu_2 means that pedagogical unit pu_2 has pedagogical unit pu_1 as prerequisite. Each node may have the following input edges: (i) none: the pedagogical unit has no prerequisite and can be executed anytime; (ii) one: the pedagogical unit has only one pedagogical unit as prerequisite and this one must be executed before it is available for execution; (iii) two or more *necessary* edges: the pedagogical unit has several prerequisite pedagogical units and all must be executed, in any order, before it is available for execution; (iv) two or more *alternative edges:* the pedagogical unit has several prerequisite pedagogical units but only one of them must be executed before it is available for execution.

Nodes and the different types of edges may be combined in a complex graph, according to the intended course sequences. The output of the interface first level consists of a graph represented as an expression in a pre-defined formal language.

The interface second level allows the definition of the pedagogical units. Each pedagogical unit consists of a set of problems whose definitions are specified through a specific interface. The problem definition includes a question that the student should be able to answer, after the interaction with the problem, and the specification of the number of

explanations, examples and exercises that will be associated with the problem. The prerequisite ordering among the problems of the same pedagogical unit is defined through the same graphical interface used to define the ordering of pedagogical units (see fig. 2(b)) and is represented by an expression in the same formal language.

Based on the information obtained in these two first levels, the interface generates an object Petri net (OPN) description of the course sequences, taking into account the defined prerequisites. In this OPN, each problem is associated with a place and each pedagogical unit with a sub-net. The use and update mechanisms of the apprentice model are automatically integrated into the OPN, leaving to the teacher only the task of providing the associated explanations, examples and exercises. The OPN is automatically translated into Jess expert system rules that implement it.

The interface third level is where these explanations, examples and exercises are specified by the teacher. They can be directly typed into the interface or copied from previously prepared files. The texts are incorporated into Servlets HTML pages in a standard format that already include the navigation controls. Presently, the exercise pages that include interaction with the Scheme system must be defined manually.

4. Conclusion

We presented MathTutor, a multi-agent tool for building intelligent tutoring systems based on a principled model for content exposition and learning strategy planning. The tool also includes a three level authoring interface, through which the structure of on-line courses can be defined. It was used to implement a prototype ITS to teach Information Structure as an undergraduate course. The implemented ITS prototype consists of a multi-agent society composed by four ITSs, each one specialized on a sub-domain.

Future work includes the refinement of the student model use and update mechanisms embedded into the object Petri net control. Future work also includes the development of other courses based on the same architecture and the evaluation of the implemented ITS prototype by the students of the discipline of Information Structure of the Control and Automation Engineering course at the University of Santa Catarina.

References

[1] H. Abelson and G. J. Sussman. *Structure and Interpretation of Computer Program.* The Mit Press, 1996.

[2] S. R. Alpert, M. K. Singley, and P. G. Fairweather. Deploying intelligent tutors on the web: An architecture and an example. *J. of AI in Education,* 10:183–197, 1999.

[3] P. Brusilovsky. Adaptative hypermedia: From intelligent tutoring systems to web-based education. *LNCS 1839,* June 2000. ITS 2000, Montreal, Canada.

[4] W.J. Clansey. *Knowledge-Based Tutoring: The GUIDON Program.* The MIT Press, 1987.

[5] E. Costa. Artificial intelligence and education: the role of knowledge in teaching. In *Machine and Human Learning,* pages 249–258. 1991.

[6] E. de B. Costa, M.A. Lopes, and E. Ferneda. MATHEMA: A learning environment based on a multi-agent architecture. In *LNAI,* volume 991, pages 141–150, October 1995.

[7] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. *How to Design Programs An Introduction to Computing and Programming.* The MIT Press, 2001.

[8] T. Finin, Y. Labrou, and J. Mayfield. *KQML as an Agent Communication Language.* MIT Press, 1995.

[9] Ernest J. Friedman-Hill. *Jess, The Rule Engine for the Java Platform.* Sandia National Laboratories, Livermore, CA, distributed computing systems edition, September 2003. http://herzberg.ca.sandia.gov/jess/.

[10] JATLite. Java agent template lite. Technical report, Stanford University, 1997.

[11] R. Mizoguchi and J. Bourdeau. Using ontological engineering to overcome common AI-ED problems. *J. of AI in Education,* 11(2):107–121, 2000.

[12] T. Murray. Authoring intelligent tutoring systems: An analysis of the state of the art. In *J. of AI in Education,* volume 10. 1999.

[13] Jean Piaget. *The Psychology of Intelligence.* Routledge Classics, Sept. 2001.

[14] John Self. *Artificial intelligence and human learning : intelligent computer-aided instruction.* Chapman and Hall, 1988.

[15] John Self. Theoretical foundations for intelligent tutoring systems. *J. of AI in Education,* 1990.

[16] C. Sibertin-Blanc. High-level Petri nets with data structures. In *European Workshop on Application and Theory of Petri nets,* pages 141–170, Helsinki, Finland, June 1985.

[17] Lev Semyonovich Vygotsky. *Mind in Society: The Development of Higher Psychological Processes.* Harvard University Press, 1978.