Atomic and Molecular Intelligent Tutoring Systems

A new architecture for intelligent and interoperable open educational resources

Gustavo Santos, Joaquim Jorge
INESC-ID / Department of Computer Science and Engineering
IST, Technical University of Lisbon
Lisbon, Portugal
{gustavossantos, jorgej}@ist.utl.pt

Abstract—Due to interoperability issues, Intelligent Tutoring Systems are hard to deploy in the different educational platforms available nowadays. This represents a considerable limitation, since tutoring systems require substantial time and a lot of resources to be implemented. This paper describes a new architecture for developing open source and interoperable intelligent tutors. In our approach, “atomic” tutoring systems are grouped to create “molecular” tree structures covering the curriculum of courses. Based on standards, our approach provides interoperable and open source ITSs that can be retrieved from repositories of educational content, grouped when necessary, loaded into different educational platforms and have their code reused for incremental development.

Keywords- Computers and Education; Distance learning; E-Learning Standards; Intelligent Tutoring Systems

I. INTRODUCTION

ITSs are interactive educational software developed with artificial intelligence techniques, and concepts from the learning sciences. Most tutoring systems employ theories of learning-by-doing [1]. Accordingly, they can engage students into reasoning interactive activities, by having a considerable understanding of the domain being taught. The traditional architecture of a tutoring system comprises four elements [2-4], as we can see in Fig. 1.

The conventional instructional model of an ITS relies on engaging students in problem solving activities, via the user interface. The expert module evaluates the activities performed by the students. The pedagogical module, determines the instructional interventions and the feedback that should be given to the apprentices. Finally, the user model records what the ITS knows about the students.

When it comes to functionalities, ITSs are described as having two main loops [5]: (1) the Outer Loop, (2) the Inner Loop. Pseudo Code 1 exemplifies the basic algorithm for each loop. First, the Inner Loop provides hints, personalized feedback, direct problem solving assistance and also evaluates students’ competencies recording it on the student model. Second, the Outer Loop uses the information about the student to perform task selection.

until tutoring is complete, repeat { // Outer Loop
  tutor selects a task;
  until task is complete, repeat { // Inner Loop
    tutor may present a hint;
    student executes a step;
    tutor presents feedback on the step;
    tutor updates the student model;
  }
  student submits the solution to the task;
}

Pseudo Code 1. ITS Loops

This functional approach to ITSs described above characterizes the key functionalities that should be implemented by the ITS loops, and also explains what services are required to assist students [5]. For instance, Assessment of Knowledge, Next Step Hints, Error-Specific Feedback, Minimal Feedback and Review of Complete Solutions [5]. This paradigm describing ITSs in terms of functionalities has become very well known. It has served as inspiration for our approach to implement interoperable tutors, and also to implement tutors by Programming with Demonstration [6,7].

Using the set of E-Learning standards defined on the Sharable Content Object Reference Model (SCORM), we have developed a novel approach that can assure interoperable and open source ITSs based on securing their functionalities as described above [8]. However, since the traditional architecture of ITSs does not properly fit the paradigm of E-Learning standards, we had to develop our own special architecture that is described on this paper.
II. ARCHITECTURE DESCRIPTION

The key to our architecture is the focus on what defines a tutor in terms of its behavior and functionalities: Inner Loops and Outer Loops. Therefore, our method organizes tutors in tree structures that collect two types of constructs, as shown in Fig. 2: (1) Molecular Tutors (MTs) and (2) Atomic Tutors (ATs).

It is very important to be clear why tutors need to be structured this way. First of all, ATs are responsible for Inner Loops. Consequently, they represent small assignments, tasks and exercises. This is fundamental because ITSs are associated to step-based problem solving and ATs implement this functionality. Second, MTs are intended to implement task selection. Accordingly, they perform Outer Loops and select the appropriate AT that should be delivered to a student at a specific moment. Whereas ATs provide problem-solving support and also update the Student Model, MTs aggregate ATs and consequently use the Student Model to select tasks. Overall, Atomic Tutors have to be grouped to create Molecular Tutors.

![Figure 2. Tree Structure of a Sample ITS](image)

This reference to chemistry concepts is appropriate because ATs cannot be decomposed without loss of functionality, since they are the smallest building blocks of a complete ITS and they must be grouped to create MTs. Conceptually, MTs correspond to educational units. Thus, each one of the ATs aggregated in a MT corresponds to an individual task in its educational unit.

A particular MT is a complete ITS for a specific topic. However, we can build larger ITSs composed by more than one MT e.g. to provide training on more than one subject. As we can see in Fig. 2, the sample ITS is composed by two MTs that approach different topics and also many ATs that are specific to each of these topics.

To give a concrete example, a MT could address an educational unit such as “Calculating the Area of Triangles”. Consequently, the ATs that compose this unit must include activities and exercises for calculating the area of triangles. In addition, a MT for the Area of Triangles could be merged with other MTs, such as “Calculating the Area of Parallelograms”, “Calculating the Area of Squares”, “Calculating the Area of Ellipses”, “Calculating the Area of Circles” and so on, to create an ITS composed by many MTs, for a larger educational unit such as “Calculating the Area of Plane Shapes”.

To give another example of how to group ATs and MTs, let’s consider an ITS for the domain of “Sorting Algorithms”. This ITS could be composed by MTs for the “Bubble Sort Algorithm”, “Insertion Sort”, “Selection Sort” and “Quick Sort”. Each MT mentioned above would have their own set of ATs to exercise their specific topics (see Fig. 3). Additionally, this large ITS for “Sorting Algorithms”, could also be used as a component within an even larger ITS to teach “Algorithms and Data Structures”. Unsurprisingly, larger educational units that cover many topics correspond to more complex ITSs.

![Figure 3. Sample Tree Structure of an ITS for Sorting Algorithms](image)

Our approach applies therefore a “divide and conquer” strategy, to decompose large tutors into smaller tutoring artifacts, that can grouped to create a complete ITS. This is similar to organizing an educational book in chapters and sections within chapters. The decomposition strategy fits perfectly to E-Learning standards, since these standards focus on the aggregation LOs, which are "a collection of content items, practice items and assessment items that are combined based on learning objectives" [8]. Decomposing domains is a method that has been used before to implement ITSs, for instance, to implement tutors with Programming by Demonstration [6,7].

A. Architecture of Atomic Tutors

Since ATs resemble miniature-tutoring systems for specific problems, each AT should include their own expert and pedagogical models (see Fig. 4). In addition, they must also include every component required to run: graphical widgets, hint sequences, error-specific feedback, both incorrect and correct step representations, etc., which must be coded in the same LO, in a similar way as Example-tracing Tutors [6,7].
Designing the AT models appropriately is a key factor to the pedagogical success of the ITS. Consequently, it is essential to follow proper educational guidelines. Cognitive Task Analysis (CTA) has been effectively used to design ITSs for many years, since it provides many important guidelines to the pedagogical soundness of an ITS [9,10]. Accordingly, performing a CTA before implementing any ITS is strongly recommended.

B. Architecture of Molecular Tutors

Given that MTs are in charge of aggregating ATs and performing task selection, their architecture is significantly different from the architecture of ATs (see Fig. 5). First, MTs do not necessarily need to interact with the students (they can, if desired by the instructional designers, for example, by using a navigable menu of contents). Accordingly, having a User Interface is not a structural requirement for MTs. However, they must have a rich set of task selection rules, to query the Student Model and determine the most suitable activity.

Every time a student starts a course, or finishes working on an exercise, MTs consult the task selection rules to choose the next AT to be delivered. As we can see in Fig. 5, to make decisions, MTs have their own expert and pedagogical models, coded in the task selection rules. Furthermore, using the same principle used for ATs, the models should be designed based on a CTA.

III. Conclusions

This paper presents a new architecture to implement interoperable ITSs, based on what defines a tutor in terms of behavior and functionalities (Outer Loops and Inner Loops). Our approach focuses on developing Atomic Tutoring Systems that must be grouped to create Molecular Tutors, using a divide-and-conquer strategy to organize the syllabus of an ITS. As a result, due to the usage of standards, our method allows implementing ITSs that can be downloaded from repositories of Learning Objects and loaded into different educational platforms.

ACKNOWLEDGMENTS

This work was supported by the Portuguese Foundation for Science and Technology (FCT): through individual grant SFRH / BD / 66225 / 2009 and project PEst-OE/EEI/LA0021/2011 (pluriannual INESC-ID research grant).

REFERENCES