Object-Oriented Knowledge Primitives for Intelligent Tutoring Shell

Ljubomir Jerinic
University of Novi Sad, Institute of Mathematics
Trg Dositeja Obradovica 4, 21000 Novi Sad, Yugoslavia
Email: jerinic@uns.ns.ac.yu

Vladan Devedzic
University of Belgrade, FON - School of Business Administration
Jove Ilica 154, 11000 Belgrade, Yugoslavia
Email: devedzic@etf.bg.ac.yu

Abstract - The difficulty of designing and developing intelligent tutoring systems (ITSs) has caused a recent increase in the interest of the AI researchers in realisation some new approaches in that field. On the other hand, the educational communities have become also interested in taking an active role in designing educational software. However, the advancement of AI methods and techniques makes understanding of ITSs more difficult, so that the teachers are less and less prepared to accept and understand these systems. The paper describes an object-oriented model of ITSs shell, called EduSof, in which the enduser (teacher) could alone make their own ITSs lessons. The model enables the developing more flexible software environments for building ITSs, significantly increasing their reusability. The processes of computer-based tutoring and learning based on this model are much closer to human-based instruction. The model can be easily extended to cover the needs of particular tutoring systems.

1. Introduction

Intelligent Tutoring Systems (ITSs) [1, 3, 4, 8, 9] are used for computer-based instruction of students in a particular domain. They have the possibilities of presenting the appropriate contents to the user (student), monitoring his/her reactions and the learning process, generating problems and tasks for the user in order to check his/her knowledge, etc. Relying on a particular model of each student, the control mechanism of an ITS can suit the tutoring process to each individual user.

The intensive development of the methods of knowledge engineering and reasoning in the last two decades has provided the possibility that education can be delivered by the computer. The main goal of that approach is the complete individualisation of the educational process. The research on ITSs has produced the technology to do this task within the computer program. But, as computers have entered teaching practice, teachers show an increasing interest in educational software systems. For these teachers, however, ITSs is the most difficult one in terms of grasping the total aspects. Of course, they have a chance to read articles about ITSs, but what they can understand through the articles is
confined to the abstract level of comprehension. Experience in using ITSs directly through a system that makes transparent the inner mechanisms of ITSs would be a powerful tool to help them in comprehending it.

As research in ITSs continues to produce more refined systems, the gap between the research community and educational community continues to become wider. The educators' understanding, and usage of these research results has been much slower than the research progress. As this theory-application gap becomes wider, it becomes more difficult for educators to participate in ITSs research and application, and the result of the research becomes increasingly academic and unconnected to the pragmatic aspects of teaching and learning.

Traditional ITSs concentrate on the fields (domains) [10, 12] they are supposed to present, hence their control mechanisms are often domain-dependent. More recent ITSs pay more attention to generic problems and concepts of the tutoring process, trying to separate architectural, methodological, and control issues from the domain knowledge as much as possible. This was also one of the main ideas behind the ITS shell (environment) called EduSof [7], designed to allow fast prototyping of ITSs in different domains.

However, the original design of EduSof's mechanisms for representing domain and control knowledge has proved to be fragile in maintenance and further development. Therefore, a new version of EduSof has been designed, considering a new model of ITSs we developed using object-oriented approaches. It is called GET-BITS (GEneric Tools for Building ITSs), and is essentially a specific extension of a more general, recently developed model of knowledge bases, called OBOA (OBject-Oriented Abstraction) [6]. The new version of EduSof is much more flexible and reusable than the first one.

2. The architecture of knowledge in EduSof

The original goal of developing the EduSof system was to design a conceptual framework for representing objects, events, responses, reactions, and relationships used in tutoring and learning. The system is featured by an interactive graphical environment in which the teacher can manipulate objects at different levels of abstraction. These objects and relationships between them are used for defining, designing and creating a lesson. The intended users of EduSof are educators and instructional experts, not programmers. The framework is domain independent, and includes mechanisms for representing domain knowledge and control information, thus generating a particular ITS. These mechanisms are responsible for dynamically customising the machine's responses.

In EduSof, a teaching sequence is organised as a list of semantically connected elements - Lessons. They consist of a list of Concepts. Concepts, as basic components of a teaching sequence, can be expressed by means of Text, Picture, Voice and/or Simulation. For each Concept, a list of Questions is given, used for testing the acquired knowledge connected to that Concept. For each Question, a list of Alternatives is given, as possible answers to it, or the right Answer. For each Alternative, a Right Answer is provided. Also the information is given about What Next the pupil will learn, in case that a right or wrong answer is chosen.

The EduSof system is organised in three separate modules. The creating module Teacher, serve for designing lessons by the teacher of the subject, and the module Learn is
a kind of interpreter of the knowledge introduced by the **Teacher** module, used by pupils while mastering the subject-matter and getting the pertinent knowledge. The third module can be used to test students for marking.

There are **six** modules in **EduSof**’s architecture: **Expert** (containing domain knowledge that an instructor enters), **Student** (student model), **Tutor** (knowledge of teaching strategies and pedagogical knowledge), **Diagnostics** (evaluation of student’s knowledge), **Explanation** (strategies for explanation of hard topics on student’s request), and **Interface** (user interface). These modules together make **EduSof** capable of handling basic knowledge (lessons, text, pictures, etc.), additional information (for helping and improving the use of basic knowledge), learning mechanisms and student’s reactions.

At the current time we can consider authoring systems or ITSs shells [8, 9] in two different classes. Firstly there are the commercial systems like Authorware, ToolBook, Icon Author and Smarttext. These systems have a number of advantages over programming languages such as help in lesson planning, and built-in widgets which make instructional presentation easier. What they lack is any AI. All the adaptive behaviour has to be programmed in by an experienced user. The other class of systems are the different research tools, such as IDE (Russell et al, 1988), ID Expert (Merrill, 1989), KAFITS (Murray and Woolf, 1992), COCA (Major and Reichgelt, 1992), GTE (Van Marcke, 1992), RIDES (Munroe et al., 1994), Byte-sized Tutor (Bonar et al., 1986) and EON (Murray 1996). These systems come with knowledge base structure and interpreters much like and expert system shell. Their difficulty is knowing how much flexibility to offer to the teacher and how to make that acquisition process as easy as possible. These systems have not made it out of the lab or seen wide use in multiple domains, and according to T. Murray that happens because of one or more of these factors:

- They are based on a specific instructional approach;
- The domains of application are limited because the system was modelled from an existing task-specific intelligent tutor and generalised to similar domains;
- They are too complex because they are based primarily on theoretical concerns or AI techniques;
- They provide tools for structuring and using knowledge, but not for creating appealing student interfaces or learning environments.

In general, these ITSs, though some are fairly powerful and general, were not designed with significant user input, and do not address the practical issues encountered when educators actually use these systems (with the exception of COCA, which underwent some user testing).

Our try to solve these problems with authoring tools and ITSs shells is that instead a component architecture for ITSs shells (like the most of these shells) we used the object-oriented approach in designing new version of **EduSof** shell. Also, the previous version of **EduSof** suffered from some deficiencies, which we tried to overcome in the new version. First, different kinds of knowledge in its modules were designed separately, although all of them conceptually had much in common. Second, decomposing the system functionally to the above modules made it hard to make additional changes and extensions when they were needed. Adding new knowledge representation techniques when needed required substantial changes in several modules. Finally, whenever there was a need for a change, not much of the relevant software could be used without any change.
3. Object-oriented knowledge primitives for intelligent tutoring

First we describe a unified abstraction of different knowledge representation techniques and different models of action and maintain the knowledge in knowledge bases in general. That techniques (of a unified abstraction) are used while designing the shell for design intelligent tutoring lessons. The abstraction is derived and realised by applying object-oriented approach. The motivation for defining such an abstraction was to provide a unified description and representation of different elements in knowledge bases and more general knowledge access methods for use in interactions between an knowledge base and the other modules of ITSs at runtime, as well as in interactions between different modules of integrated ITSs building tools at development time.

As a result, an object-oriented model of knowledge bases is developed and is called OBOA. It covers both designs of knowledge bases and communication between the knowledge base and other ITSs modules. However, the OBOA model sets only general guidelines for developing and using of object-oriented knowledge bases. It is open for fine-tuning and adaptation to particular applications. The ultimate practical goal of developing the OBOA model was to use it as the basis for building a class library of knowledge representation and knowledge access tools and techniques.

The background for developing the OBOA model was several different ideas coming from the fields of knowledge and data modelling, representation and management. Gruber and Cohen [5] have described a hierarchy of knowledge representation techniques and tools, starting from most primitive ones to quite complex knowledge structures. Also, some ideas from the field of object-oriented databases [13] and object-oriented applications [2] were also adopted in the OBOA model.

Although the OBOA model was developed for representing knowledge in expert systems, it can be easily extended to other intelligent systems [6]. The essence of the OBOA model is a unified abstraction of different knowledge representation techniques and different models of human knowledge. All kinds of knowledge (e.g., domain, control, explanatory, etc.), as well as all types of knowledge representation formalisms (e.g., rules, frames, logic, etc.), can be viewed from the perspective of an abstract and fairly general concept of "knowledge element". Speaking in terms of object-oriented analysis and design, the OBOA model defines an abstract class K_element for representing an abstract knowledge element, no matter how complex the element is or what its purpose is. In other words, knowledge representation techniques like production rules, frames, semantic networks, etc., as well as problem solving strategies (such as generic tasks, heuristic classification,...), and higher-level concepts and agents (e.g. planners, scripts, blackboards, multiagents, etc.) can all be defined as more specific knowledge elements, simple or aggregate. No matter what and how many knowledge representation techniques are needed in a particular intelligent system, it is always possible to define a suitable hierarchy of knowledge types needed and design a corresponding class diagram starting from K_element, as the most abstract class, in the root.
As an example, Figure 1 illustrate what meaningful subclasses can be derived directly from $K_{element}$ regarding domain knowledge representation in expert systems (the classes shown do not exhaust all the possibilities). Rule and Frame classes are used to specify If-Then rules and frames. Attribute and Relation classes define attributes of more complex knowledge elements and relations that can be defined among some other knowledge elements. $K_{chunk}$ (Knowledge chunk) class objects can be used as slots for Frame or Media objects, or If- and Then-parts in Rule objects.

Of course, the subclasses shown can be (and some of them actually are) too abstract to be used directly in a particular application. Many other subclasses may also be needed for knowledge representation in other intelligent systems or expert systems. In all such cases, additional subclasses can be derived either directly from $K_{element}$, or from its direct subclasses. It is a matter of design of a particular intelligent system to define such additional subclasses accordingly.

4. The OBOA model of knowledge primitives

In this section the usage of the OBOA model in the knowledge representation of the knowledge primitives of EduSof is presented. That knowledge representation is concerning some particular and concrete type of knowledge elements that are necessary for the realisation of the Expert module in any ITS shell. One of the most important elements of knowledge primitives for designing the intelligent tutoring lesson is the model of Lesson, and this is basic class needed for modelling the learning process. Any lesson is consisted of one or more issues, modelled by the class Topic in the OBOA model. We assume that the student must learn these issues, during mastering that lesson. The basic attributes of the lesson are: the name (Name), current topic (CurrentTopic, the issue or the problem introduced, defined and/or explained in the present moment), the task that is currently solved, the level of prerequisites for the student (StudentLevel), and so on.

Applying the principles of the OBOA model, we can define appropriate class hierarchies for developing ITSs. That's what GET-BITS is about. It also starts from the concept of knowledge element, and derives meaningful subclasses that are needed for building a wide range of ITSs. However, classes for knowledge representation are not the only tools needed to build an object-oriented ITS. Apart from knowledge of various kinds, in each such ITS there must also exist some control objects that functionally connect the system's modules, handle messages, control each session with the system, monitor student's reactions, etc. In other words, such objects provide control and handle dynamics of ITSs.
GET-BITS also specifies classes of these control objects. Due to space limitations, we describe only the most important classes specified in GET-BITS for representing various kinds knowledge in ITSs. They are illustrated in Figure 2.

Obviously, many key classes are derived from the Frame class (Lesson and Topic - lessons and topics the user learns; TQ - an abstract class used to derive the classes representing questions the student has to answer, tasks and problems the system generates for him/her, etc.; Explanation - explanations the system generates on request for various classes of users (end-users and system developers), as well as topic- and concept-oriented explanations). Additional attributes of these classes make them semantically different and more specific than ordinary frames, although they are actually implemented as more specific frames. The names of the other classes are easily interpreted.

Once all of the important classes for knowledge representation (and also for specifying control objects) are implemented within a class library, it is a straightforward task to use them for designing and implementing an ITS shell like EduSof. That is the main practical goal of GET-BITS: letting ITS shell developers "get bits" of software they need, suit them to their own design needs without starting from scratch, yet retaining control over further development of their tools and making them more reusable.

5. Application of EduSof

The EduSof shell is used in developing FORLAN, an ITS in the domain of formal languages. The idea of the FORLAN project is to develop software that supports systematic introduction of students into the system's domain, in accordance with both the logical structure of the domain and individual background knowledge and learning capabilities of each student. The system is discussed here only from the EduSof perspective. The Expert module contains all of the domain-dependent knowledge:

- the concepts, topics, facts and domain heuristics the student has to learn;
- a database of examples used to illustrate the domain concepts, topics, etc.; and
- the pedagogical structure of the domain.

The pedagogical structure of the domain is considered a part of the domain knowledge rather than a part of the tutor module, as well as in [11]. In FORLAN, pedagogical structure of the domain is defined as a set of directed graphs showing explicitly precedence relationships of knowledge units within each lesson and among the topics of different lessons.

FORLAN always operates in one of the following three modes of operation: teaching, examination, and consulting. It is actually the EduSof's Tutor module that operates in one of these three modes. FORLAN's Explanation module tightly co-operates with the Tutor module in the consulting mode, in order to answer the student's questions and provide desired explanations. Student model in EduSof and in FORLAN is an object of a class derived from the corresponding GET-BITS class.

The following example illustrates how GET-BITS has been used in designing EduSof's classes in order to adapt the shell to support development of FORLAN. A lesson in FORLAN is a meaningful subset of concepts, topics, facts and domain heuristics. These items in a lesson are closely coupled but they can refer to items in other lessons. Some important attributes of each FORLAN's lesson are sets of objectives and goals, sets of
topics, concepts, facts, theorems, etc. taught in that lesson, a set of the corresponding teaching rules, and a set of associated problems (tests, questions and exercises). The Lesson class, as it is specified in GET-BITS and included in the current version of EduSof, supports most of the above attributes. However, when structuring the domain knowledge for implementing it in FORLAN, it turned out that many lessons could be better organised if the Lesson class had some additional features. Therefore a new class, T-Lesson, has been designed and included in the version of EduSof which is used for the development of FORLAN. The T-Lesson class supports using theorems in presenting a lesson and fine-tuning the presentation by showing/hiding theorem proofs, lemmas and corollaries (this is controlled by dedicated Boolean flags):

<table>
<thead>
<tr>
<th>Name:</th>
<th>T-Lesson</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base class:</td>
<td>Lesson</td>
</tr>
<tr>
<td>Derived classes:</td>
<td>FirstLesson, LastLesson, HardLesson, EasyLesson</td>
</tr>
<tr>
<td>Interface Operations:</td>
<td>SetTheorem, GetTheorem, DeleteTheorem, CreateTheoremCollection, GetTopicCollection, ..., SetSkipProofs_Flag, SetSkipLC_Flag</td>
</tr>
<tr>
<td>Implementation</td>
<td>Uses: Theorem</td>
</tr>
<tr>
<td></td>
<td>Fields: SkipProofs_Flag, SkipLC_Flag</td>
</tr>
<tr>
<td>Persistency:</td>
<td>Static; disk files</td>
</tr>
</tbody>
</table>

This example simultaneously illustrates how computer-based tutoring and learning based on the GET-BITS model can be easily adapted to closely reflect the way human-based instruction is done in a given domain and given the student’s background knowledge and goals. It is possible to control the setting of SkipProofs_Flag and SkipLC_Flag from the rules of the Tutor module. Among the other conditions and heuristics, pedagogical rules use the values of the relevant attributes of the student model in order to adapt the lesson presentation to each individual user.

6. Conclusions

The present model of intelligent tutoring systems, presented in the paper, allows for easy and natural conceptualisation and design of a wide range of ITS applications, due to its object-oriented approach. It suggests only general guidelines for developing ITSs, and is open for fine-tuning and adaptation to particular applications. ITSs developed using this model are easy to maintain and extend, and are much more reusable than other similar systems and tools.

The GET-BITS model of intelligent tutoring systems, presented in the paper, allows for easy and natural conceptualisation and design of a wide range of ITS applications, due to its object-oriented approach. It suggests only general guidelines for developing ITSs, and is open for fine-tuning and adaptation to particular applications. ITSs developed using this model are easy to maintain and extend, and are much more reusable than other similar systems and tools. The model is particularly suitable for use by ITS shell developers. Starting from a library of classes for knowledge representation and control needed in the
majority of ITSs, it is a straightforward task to design additional classes needed for a particular shell.

Further development of the GET-BITS model is concentrated on development of appropriate classes in order to support a number of different pedagogical strategies. The idea is that the student can have the possibility to select the teaching strategy from a predefined palette, thus adapting the ITS to his/her own learning preferences. Such a possibility would enable experimentation with different teaching strategies and their empirical evaluation. Another objective of further research and development of GET-BITS is support for different didactic tools which are often used in teaching.

References