THE COMPONENT BASED MODEL OF THE CONTROL KNOWLEDGE FOR INTELLIGENT TUTORING

Ljubomir Jerinic$^1$, Vladan Devedzic$^2$ and Danijela Radovic$^3$

$^1$ Institute of Mathematics, University of Novi Sad
$^2$ FON - School of Business Administration, University of Belgrade
$^3$ University of Kragujevac, Technical Faculty Cacak

Abstract

The difficulty of designing and developing more useable and cost-effective intelligent tutoring systems (ITSs) has caused the realization of some new approaches in that field, the realization of intelligent tutoring shells. Our starting point and perspective on developing ITS shell is motivated by issues of pragmatics and usability. The advancement of AI methods and techniques makes understanding of ITSs more difficult, so that the teachers are less and less prepared to accept these systems. As a result, the gap between the researchers in the field of ITSs and the educational community is constantly widening. Also, the present ITSs need quite big development environments, huge computing resources and, in consequence, are expensive and hardly portable to personal computers. The paper describes an object-oriented model of control knowledge of the ITS shell in which the end-user (teacher) could make their own ITS lessons, alone.

1. Introduction

Traditional Intelligent Tutoring Systems (ITSs) [4, 7, 8] are concentrated on the fields (domains) 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 [2, 5, 6], 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 ITS we developed using object-oriented approach. 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 [1, 3] (OBject-Oriented Abstraction). First, we introduced the control knowledge of an ITS. The control knowledge describes the ITSs problem-solving process, reasoning strategies used by the inference engine, organizational hierarchy of reasoning modules and agents, control regimes corresponding to particular steps of the problem-solving task, the methods of elaborating the answers, etc. To an extent, this knowledge complements the domain knowledge - ideally, control knowledge is an application-independent description of how the problem-solving and reasoning activities are performed during the various ITSs operation, i.e. during the learning process. The same is with the
all modern Expert Systems (ES), which have both domain and control knowledge represented in their knowledge bases explicitly.

This paper describes how control knowledge is treated by a recently developed object-oriented model of ITSs knowledge bases. The model covers all essential aspects of control knowledge, such as generic tasks, control procedures, meta-reasoning, inference paradigms, etc. All these aspects are covered in an object-oriented class hierarchy, reflecting the principles, structure and organization of most important elements of control knowledge, and showing how these elements are related to other types of knowledge. The model is based on a number of design patterns and class libraries developed in order to support building of intelligent systems. The processes of computer-based tutoring and learning based on the EduSof model are much closer to human-based instruction. The model can be easily extended to cover the needs of particular tutoring systems. The next extension to the ITSs is involving that the control knowledge of the EduSof is extended by adding the psychological type of the responds to some actions or questions of the user, and appropriate screen models. The control knowledge managed the domain knowledge according the student model and the teaching strategy. This component of the EduSof is by generalized and abstracted If-Then rules which control the domain knowledge represented with the knowledge network of semantically connected frames. That knowledge network represents the content to be taught in terms of lessons, concepts, rules, tasks, questions, actions, and examples and their interrelationships. The control knowledge according to the users action and pre-tested psychologically type of the users managed that network and the user-computer interaction.

2. Contents of the Knowledge Base for Intelligent Tutors – AI View

It is possible to think of an ITSs knowledge base as a logical entity composed of three related parts (components): the domain knowledge (i.e. a model of experts knowledge), control knowledge (i.e., a model of the teaching process and a model off the system responds), and the third part that we call explanatory knowledge (Figure 1). Domain knowledge is represented using one or more knowledge representation techniques. In the most general case, domain knowledge is a structured record of many interrelated knowledge elements. These elements describe relevant domain models and heuristics, and can vary a lot in their nature, complexity, and representation. They can be everything from simple data to instantiated knowledge primitives such as frames, rules, logical expressions, procedures, etc., and even more complex knowledge elements represented using either simple aggregations of the knowledge primitives or conceptually different techniques based on the knowledge primitives and other (but simpler) knowledge elements. Semantic nets are the best known kind of such complex knowledge elements, and are frequently used for representing deep knowledge about the problem domain.

![Figure 1 The knowledge base as a composite object](image-url)
Speaking of the complexity and hierarchy of knowledge elements, it may be noted that even knowledge primitives are complex enough themselves, and their parts can be also treated as (less complex) knowledge elements. For example, the knowledge elements of a frame are its slots, its sub-frames, its demons, etc., and the knowledge elements of a rule are its If-clauses and its Then-clauses. There is a natural constraint in further dividing simpler knowledge elements. For example, an attribute-value pair has its attribute and its value, but the attribute is a simple part that cannot be further divided. Parts of a knowledge element of a certain type are either collections of other knowledge elements (e.g., the If-part, or the Left-Hand Side (LHS) of a production rule is usually a collection of If-clauses), or single elements (e.g., the attribute part of an attribute-value pair).

Knowledge elements of the domain knowledge are usually grouped to create meaningful collections. One possible criterion of defining such a collection (the one used in the EduSof model) is the type of the knowledge elements that will be included in the collection, hence we can speak of collections of rules, frames, etc., or collections of more complex knowledge elements. Each such collection is homogenous regarding the type of knowledge elements it contains. Generally, domain knowledge can be composed of n collections of knowledge elements, and there can be 0 or more collections of elements of a particular type. Note that at the analysis level we do not specify the ways the knowledge elements are linked within and across such collections.

The contents of the ITSs control knowledge are abstract, explicit, and more or less domain-independent descriptions of the way how to learn some facts during the ITSs operation. In EduSof we used two approaches in representing control knowledge: generic tasks, and the ITSs meta-knowledge, presented in the form of meta-rules. Generic tasks are abstract and typical knowledge and control structures reflecting the steps and activities performed when learning process is represented by learning-by-doing. They make it possible to represent the problem of teaching strategies explicitly in the knowledge base, using appropriate control knowledge elements to express the knowledge of how the learning by problem solving task is performed. Each generic task is featured by a specific problem type, structure, and problem solving strategy. Elementary (atomic) generic tasks, like hierarchical classification or hierarchical design by plan selection and refinement, can be used as building blocks for construction of knowledge-based systems. They can be also viewed as parts of other, more complex, but still typical expert systems tasks called complex generic tasks.

Meta-rules are similar to domain rules, but they contain strategic knowledge about the teaching/learning process, rather than domain heuristics. They enable the ITSs to examine its own domain knowledge during the operation and make decisions that actually direct further use of domain knowledge. As well as generic tasks, they also clearly separate control knowledge from domain knowledge and explicitly specify the system behavior at the task level. For example, if a certain problem can be solved either by applying a heuristic search or by decomposing it into sub-problems, meta-rules can specify what strategy to use given some facts about the problem or the nature of desired solution.

In the context of control knowledge we can also speak of knowledge elements and their collections, hierarchies, complexity, etc., only this time the contents and the meaning of the knowledge elements are different from that of the elements in the domain knowledge part of the knowledge base.

\[1\] The word "collections" is used rather than sets, lists, tables, etc., in order to avoid suggesting a particular implementation technique at the analysis phase.
Furthermore, some levels of complexity of control knowledge elements parallel those of domain knowledge elements. The same can be shown for explanatory knowledge. Therefore the most important abstraction used in the EduSof model is that of the abstract and general knowledge element as a universal concept.

The primary purpose of the part of the knowledge base that we refer to as the explanatory knowledge is to define the contents of explanations and justifications of the ITSs learning process, as well as the way they are generated. Explanatory knowledge is related to both domain and control knowledge and is often treated as a part of the other two components of knowledge bases. However, in the EduSof model it is treated as a distinct knowledge component, in which the knowledge about the learning process explanation, knowledge elements explanation, control strategies explanation, and other types of intelligent assistance is represented explicitly and is treated in its own right.

Among the knowledge elements that this component can include are: canned text associated with rules and other knowledge elements in the other two components, templates that are filled by the explanation module when required (in order to generate the full explanation text in a given situation), presentation functions needed for explanation of certain knowledge elements (e.g., some knowledge elements are best explained by using tables, others require the use of graphics, etc.), user models necessary for generating explanations in accordance to the user's knowledge level, criteria used by the explanation module when deciding about what knowledge elements to include in the explanation and what to skip, as well as what level of details must feature the explanation.

It must be stressed that apart from the explanatory knowledge, the explanation module uses extensively the knowledge from the other two parts of the knowledge base as well when generating explanations. Therefore, the explanatory knowledge may also contain explicit descriptions of explanation control strategies. These can be specified in the form of control rules like, for example: "If the explanation type is WHY, then 1/ show the current goal, and 2/ show the current domain rule instantiation, and 3/ show the meta-rule that was last applied".

The organization of the elements of the domain knowledge is as homogenous collections, K_elements (Figure 2). A K_elements object is a collection of objects of the K_element class. The K_element class describes a single knowledge element of the domain knowledge and is elaborated below. The double line from the D_knowledge class icon to the K_elements class icon, with the point close to the D_knowledge icon, means that the D_knowledge class uses the K_elements class. The 1 and n at the ends of the double line denote the cardinalities of the corresponding classes: the D_knowledge class can have only one instantiation for a particular knowledge base, and there can be n instantiations of the K_elements class.
Likewise, for each instantiation of the K_elements class, there can be n instantiations of the K_element class. K_elements is a generic (parameterized) class, that can be instantiated to generate classes denoting collections of knowledge elements of a particular type. The instantiation relationship between the corresponding classes is denoted by a dashed line, with an arrow pointing to the generic class. For the sake of simplicity and basing this description on commonly known concepts, it is assumed that some knowledge primitives belong to the second highest level in the organizational hierarchy of knowledge elements. Hence the classes Frames, Rules, etc. are instantiations of the K_elements class, and denote homogenous collections of frames, rules, etc., respectively. In the actual implementation, there can be several intermediate levels between the level of knowledge primitives and the highest (and also conceptually the most abstract) level of knowledge elements, i.e. the level of K_elements and K_element classes.

The classes in Figure 2 are designed by filling-in a standard template for class properties. As an example, some important details of the D_knowledge class design are shown here, skipping those unnecessary for the purpose of this paper:

| Name: | D_knowledge |
| Visibility: | Exported ; i.e., visible outside the enclosing class category |
| Cardinality: | 1 |
| Base classes: | - ; in general, list of class names |
| Derived classes: | - ; in general, list of class names |
| Generic parameters: | - ; makes sense only for generic classes |
| Public Interface | Operations: SetName, GetName, GetSize, Create_K_El_Collection, Get_K_El_Collection, Update_K_El_Collection, Compile,... |
| Implementation | Uses: K_elements ; in general, list of class names |
| Fields: | Name, K_El_Collection_Ptr [],... |
| Persistence: | Static ; disk files |

A D_knowledge object lets other objects (communicating with it) use its public interface functions to set and get its name, get its size, access its collections of knowledge elements, compile it, etc. It should be stressed that the model strictly applies encapsulation, as a general and extremely important principle of object-oriented design: most attributes (fields) of classes and objects used in this design cannot be accessed directly. Client objects must use the server object’s public interface functions. For example, in order to read a knowledge element from a certain collection of knowledge elements belonging to the domain knowledge part of an ITSs knowledge base, the corresponding D_knowledge object’s Get_K_El_Collection interface function is called (in other words, the Get_K_El_Collection message is sent to the D_knowledge object). It uses the appropriate private K_El_Collection_Ptr pointer and calls interface functions of the corresponding K_elements object to retrieve the desired knowledge element.

K_element (Figure 3) is an abstract base class with a number of subclasses inheriting its properties. In the Booch notation, the inheritance relationship among two classes is denoted by a solid line with an arrow pointing to the base class. Abstract base classes cannot have instantiations, so only the objects of the Rule, Frame, Slot, If-clause, Signal, and other subclasses can exist in the knowledge base. Again, we assume the classes designating the knowledge primitives mentioned are derived directly from the K_element class, skipping more complex classes of knowledge elements. The
model is open for extensions by less common (or even new) knowledge element types. The K_element class defines some useful common properties that are inherited by all types of knowledge elements, and including another knowledge element type in the hierarchy requires only type-specific properties to be defined. Second, all the elements of a knowledge base are treated the same way, regardless of what their type is and how common (standard) or specific they are. This means that all the common operations on elements of the knowledge base (like updating, searching, deleting, etc.) use the interface of the K_element class, and element-specific operations are provided through polymorphism in the corresponding subclasses.

Another interesting design detail in Figure 3 is the fact that certain types (classes) of knowledge elements, denoting parts of more complex elements, are at the same level of hierarchy as the corresponding aggregate elements. For example, Figure 3 suggests that If-clauses can be treated as "stand alone" knowledge elements, as well as rules. Such a design allows a particular If-clause object to be shared by several Rule objects, which is more flexible and more efficient, and can also simplify the rule compilation process. The most general class in Figure 3, the abstract K_element, is designed as follows:

<table>
<thead>
<tr>
<th>Name:</th>
<th>K_element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visibility:</td>
<td>Exported ; i.e., visible outside the enclosing class category</td>
</tr>
<tr>
<td>Cardinality:</td>
<td>n</td>
</tr>
<tr>
<td>Base classes:</td>
<td>Rule, Frame, If-clause, Then-clause, Slot,...</td>
</tr>
<tr>
<td>Derived classes:</td>
<td>Rule, Frame, If-clause, Then-clause, Slot,...</td>
</tr>
<tr>
<td>Generic parameters:</td>
<td>- ; although a base class, K_element is not a generic class</td>
</tr>
<tr>
<td>Public Interface</td>
<td>Operations: SetName, GetName, GetSize, Create_K_El, Delete_K_El, Get_K_El, Update_K_El,....</td>
</tr>
<tr>
<td>Implementation</td>
<td>Uses: K_elements, Transaction,...</td>
</tr>
<tr>
<td></td>
<td>Fields: Name, Collection_Ptr[...],....</td>
</tr>
<tr>
<td>Persistence:</td>
<td>Static ; disk files</td>
</tr>
</tbody>
</table>

3. Knowledge Primitives for Control Knowledge

The domain, i.e., the knowledge that student must learn, is essentially a tree, where each node is a piece of domain knowledge and any parent-child link in the tree implies that in order for the student...
to be able to understand the knowledge of the parent node, he must also understand the knowledge of the child node. The representation of the knowledge domain is object-oriented. The main object classes are the lessons and the way of their presentation. Lessons are represented by units of knowledge that can be taught, repeated, summarized, etc. They are categorized according to knowledge type, for example: text, picture, simulation, more examples, and so on. The concepts have pointers, including various types of prerequisite, part-of and related-misconception links to other concepts, forming the lesson network. They have pedagogical information such as summary, motivation, examples, tasks, etc., that point to presentation. The presentation or the array of questions or tasks, represent expository or inquisitor interactions with the student. They are composed of a task, such as a multiple-choice question, or problem solving exercise, and an environment for doing the task, such as a picture or simulation of some system. Presentation also contains the possibilities for responding to the pupil, such as hints, congratulations, elaboration's of the answer, etc. That knowledge representation is concerning some particular and concrete type of knowledge elements that are necessary for the realization of the Expert module in any ITS shell. Any lesson is consisted of one or more issues, modeled by the class Topic. The basic attributes of the lesson are: the name (Name), current topic (CurrentTopic, the issue or the problem introduced, defined and/or explained at the present moment), the task that is currently solved, the level of prerequisites for the student (StudentLevel), and so on.

![Figure 4 The control knowledge for the TQ class](image)

The issues that student must learn (class Topic) are realized with separate type of knowledge elements in the EduSof system. Any topic could be introduced with text information (Information), graphically (the pictures and/or the diagram - Image), and/or with the simulation of some event (Simulation). Also, the additional or further explanation (Explanation) for that theme, or some suggestions (Hints) could be defined. The class Topic in EduSof is made for specifying and defining the issues or notions needed for lesson creation.

Abstract class TQ served for description of the common elements for the two comparable and connected classes, one for the definition of tasks or assignments (class Task) and the other class for
the realization of questions or problems (class Question). That class has the fields WhyPtr, HowPtr, WhatPtr, etc. With that fields the lesson creator made their explanations for the task and/or question. The instances of that class are given to the student during the learning process. The completed definition for the classes Lesson, TQ, Topic, Explanation, etc. are given in [2., 5, 6].

The control knowledge is realized by parameterizing the actions with the combination of If-Than-Action rules. Figure 4 represents the TQ_Engine control knowledge for elaborating an answer for some task or question.

The above example of the TQ_Engine and other modules of the control knowledge are extended with psychology type of the student, which is responsible for different user interfaces in the elaborating the students answer and responding according the psychology type of the student.

The class Control_Knowledge is designed as follows:

<table>
<thead>
<tr>
<th>Name:</th>
<th>Control_Knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visibility:</td>
<td>Exported; i.e., visible outside the enclosing class category</td>
</tr>
<tr>
<td>Cardinality:</td>
<td>n</td>
</tr>
<tr>
<td>Base classes:</td>
<td>-; in general, list of class names</td>
</tr>
<tr>
<td>Derived classes:</td>
<td>Control_Rule, Control_Frame, Control_If-clause, Control_Then-clause, Control_Slot,...</td>
</tr>
<tr>
<td>Generic parameters:</td>
<td>-; although a base class, K_element is not a generic class</td>
</tr>
</tbody>
</table>

**Public Interface**

**Operations:** SetName, GetName, GetSize, Create_Control_Knowledge, Delete_Control_Knowledge, Get_Control_Knowledge, Update_Control_Knowledge,...

**Implementation**

**Uses:** K_elements, Transaction,...

**Fields:** Name, Collection_Ptr[...]

**Demons:** TQ_Engine, Show_Questions_Ptr[...], Get_Answers_Ptr[...], Give_Encourage, Consult_Net, Question_Generator, Give_Congratulate, Calculate_the_Error, Elaborate_Answers_Ptr[...],...

**Persistence:** Static; disk files

For example, suppose that we want to design an exercise for pupils to practice how solve some tasks – examples form some Geographic lesson, i.e. test about industry and agriculture of the countries of the Europe (or from nearly all other subjects) then, according the above Figure 4 of the TQ_Engine, i.e. the procedure for elaborating the student responds of some task or question - the give feedback mechanism, and shown parameters we have the following programming code:

```plaintext
For i:=1 to NumberOfTasks
    Question_Generator[Task_i];
    Show_Question;
    Get_Answers[Answer_i];
   ...
    IF Elaborate is ON THEN Elaborate_Answers[Answer_i] WITH StudentModel of the User AND Psychology Type of the User;
```

...
IF Encourage is ON THEN Give_Encourage WITH StudentModel of the User AND Psychology Type of the User;

IF Congratulate is ON THEN Give_Congratulate WITH StudentModel of the User AND Psychology Type of the User;

... End (* For *)

... End;

Procedure Elaborate_Answer[Answer];

... IF Answer == Right THEN Consult_Net(Answer) WITH OK
ELSE IF Answer == Approximate THEN Consult_Net(Answer) WITH NOT_QUITE_OK
ELSE IF Misunderstanding is ON
    THEN Calculate_the_Error(Error) AND
    Case Error == Big
        Big: Consult_Net(Answer) WITH False
        Minor: Consult_Net(Answer) WITH Not_Too_Wrong
        Almost: Consult_Net(Answer) WITH Approximate_Wrong
    ... EndCase

end;

Procedure Consult_Net (Answer);

IF Answer == OK THEN Find(New Task [OR Question [OR Topic]])
ELSE IF Answer == NOT_QUITE_OK THEN Give_Help WITH More_Examples
ELSE IF The Answer == Too Wrong THEN Give_Help WITH Explanation_HOW
ELSE IF The Answer == False THEN
    Teach_the_Topic_Again WITH Different_Approach AND
    Give_Help WITH Explanation_HOW AND Explanation_WHY;

end;

4. Conclusion

The present model of intelligent tutoring systems, presented in the paper, allows for easy and natural conceptualization and design of a wide range of ITS applications, due to its object-oriented approach. It suggests only general guidelines for developing of ITSs, and is open for fine-tuning and adaptation to particular application. ITSs developed by 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 system, presented in the paper, allows easy and natural conceptualization and design of a wide range of ITS applications, due to its object-oriented approach. It suggests only general guidelines for developing of ITS, and is open for fine-tuning and adaptation to particular applications. 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.
The presented method for building a knowledge base suitable for the educational purpose is currently under development and the realization in the present form. In the current development state we test usage of *EduSof* in some summer schools in various subjects (chemistry, history, geography etc.). With introduced approach in building the intelligent tutors we realize the one of the goals we set down before the start of the realizing of the above concepts. The teacher's independents in design of the intelligent tutoring lessons were almost 90% after an hour of explanation how to use the *EduSof* shell. The rest 10% is used for the professional programmers time for designing the simulation procedures. It is supposed that the teacher have some knowledge of using the computers, a little knowledge in computer graphics, and some help of professional programmers' in making the simulation's procedures.

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 pre-defined 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 the support for different didactic tools which are often used in teaching.

5. References


