

# The Friendly Intelligent Tutoring Environment Teacher's Approach

Ljubomir Jerinic and Vladan Devedzic

## Introduction

The advancement of using the Artificial Intelligence (AI) methods and techniques in design Intelligent Tutoring Systems (ITSs) makes understanding them more difficult, so that teachers are less and less prepared to accept these systems. As a result, the gap between researchers in the field of ITSs and the educational community is constantly widening. While ITSs are becoming more common and proving to be increasingly effective, each one must still be built from scratch at a significant cost. Also present ITSs need quite big development environments, huge computing resources and, in consequence, are expensive and hardly portable to personal computers. This paper describes our efforts toward developing uniform data, explanation and control structures that can be used by a wide circle of authors who are involved in building ITSs (e.g., domain experts, teachers, curriculum developers, etc.) that is, the model of the ITSs framework, the **GET-BITS** model.

## Learning with Computers – the ITSs Paradigm

When first introduced more than 10 years ago [Psootka *et al.*, 1988, Woolf, 1992] Intelligent Tutoring Systems (ITSs)<sup>1</sup> were announced and avowed as the future of education and training. Unfortunately, despite of the success of some ITSs (Shute's Smithtown and Air Force, Clancey's GIDEON and

NEOMYCIN, Wolff's Meno Tutor, Anderson's LISP tutor, Nwana's FITS, Wagner's SCHOLAR, Johanson's PROUST, etc., gave promising and hopeful beginning), ITSs have not yet seen general acceptance. Now, 10 years later, the ITSs community is still talking about the promise of this technology while searching for the leverage that will encourage its widespread adoption and classroom use. Much has to do with the complexities involved in the definition and design of ITSs applications, as well as the paradigmatic changes required of training and education organizations in the way they practice instructional design in order to realize this new kind of education paradigm [Clancey, 1996].

## The Teacher's Role

As research continues to produce more refined and prosperous ITSs, the gap between the research and educational communities continues to become wider [Murray and Woolf, 1990]. The educators' understanding and usage of research results have been much slower than the research progress. As this theory-application gap becomes wider, it simply becomes more and more hard and complex for educators to participate in ITSs research and application. As a result of this tendency, the ITSs research becomes increasingly academic and unconnected to the pragmatic aspects of teaching and learning. Also, educational programs are different among distinct countries, and even regions in the same country. The required knowledge for learning some lesson in a particular domain is not the same even in across a single country (the north - south problems, country - town, etc.). Teachers want to have an active role in the design and eventual update and improvement of the intelligent tutor lessons. For these teachers, however, it is most difficult to fully

<sup>1</sup>. Intelligent Tutoring Systems, Intelligent Learning Environment, Knowledge-based Tutors, Intelligent Computer Assisted Instruction, are all, more or less, the synonyms for the using the computers in the process of teaching and learning by the aid of the methods and techniques of Artificial Intelligence.

understand ITSs. Of course, they have a chance to read articles about ITSs, but what and how they can understand through the articles is restricted, confined and limited to the abstract level of comprehension and perhaps understanding the AI mechanisms and techniques used in common ITSs design. Experience in using ITSs directly through a system that could render transparent the inner mechanisms of ITSs and help teachers to make their own lessons would be a powerful tool to help them in understanding and comprehending this new paradigm of teaching by computers.

### Significant Cost of Developing ITSs

As modern intelligent tutoring systems are becoming more common and proving to be increasingly effective, each ITS developer is discovering:

- the significant cost of realization,
- the big development team, and
- the large computer resources needed to use ordinary ITSs.

Conversely, software reusability is defined as the process of creating new applications using previously developed software. The primary goal of this technique is to improve both the quality and the productivity of software [Frakes, 1994, Lim, 1994], and to reduce the development cost. The main reason for taking into account reusability issues in the ITSs field is due to the fact that building ITSs needs quite big development environments. The implementation of an average ITS requires huge computing resources, lots of money, and lots of time. Furthermore, the results are expensive and hardly portable to personal computers. Taking into account that prototypes are built incrementally through successive enhancements and refinements, the time and cost of development is largely reduced if existing knowledge is reused.

### Goals of GET-BITS Model

In view of the above trends and research issues, the goal of designing the *EduSof* system – the ITSs environment, was to construct a conceptual framework or tool for representing the objects, events, responses, reactions, and relationships involved in tutoring. Also, the aim was to build a highly usable knowledge acquisition interface for rapid prototyping and easy creation, modification, deletion, and testing of both teaching concepts and tutorial strategies. Finally, our intention was to produce a system that enables teachers to build their own lessons without any computer or AI specialist. The ITSs environment works on ordinary PCs or similar hardware and on different platforms; the final goal was to create a system with which the production of the ITSs systems will be cost-effective. Our approach in resolving the above goals, based on the **GET-BITS** (*Generic Tools for Building ITS*) model of intelligent tutors is described in this paper [Devedzic and Jerinic, 1997].

### Computers in Education

In one class there is seven year old George. He somehow learns to solve the equation  $5+X=8$ . On some tests he has the

following problem to solve: "Peter has some apples. Andrea gives to Peter 5 apples more, and then Peter has eight apples. How many apples had Peter before he met Andrea?" George failed, because he does not understand that problem and he cannot connect the equations with that problem. It seems that George needs a tutor.

Andrea, the fourteen year old youth, learning her last geography lesson about Novi Sad, the capital town of Vojvodina, has the following problem. She learned that Belgrade is placed on  $44^{\circ}48'$  of north geographic width and  $20^{\circ}28'$  east geographic longitude. But, she forgot how to read geographic width and longitude on the geographical map. She has nobody to ask about that. It seems that she needs a tutor too.

There are many Georges and Andreas out in the world that have problems in many different academic subjects. There are not enough tutors for every George and Andrea. Now imagine **Algy** and **Geo** who can give George and Andrea the tutoring they need. They are not typical mathematics or geography tutors. They are computer programs.

From above examples the need for some kind of computer tutors (i.e., computer programs) is obvious. According to [Woolf, 1987] the average success of testing students that are training with traditional talking or class (the teacher is talking to the group of students from 50 to 200) method is between 50 and 60 from 100 points. When a teacher with the same teaching strategy uses a diagnostics test (to improve and modify his approach to teaching that subject), the results are better, i.e., about 84 from 100. But, when the students have individualized training the results are 98 from 100. Of course, in our schooling system, according to reality, it is almost impossible to have individualized teaching and learning. The only way to override that educational crisis is to find out how computers could teach.

### The Beginning – Computer Assisted Instruction

From very early in the computer age, researchers have hypothesized the computer's enormous potential for assisting and even coaching the education process. Early research efforts were restricted by a number of factors:

- the lack of effective and powerful hardware,
- the deficiency of appropriate and suitable software for developing computer-based representations of domain and pedagogical knowledge,
- the inadequacy of the methodology and instructional theories to use the computer's potential, and
- the inability of researchers to develop a unified framework for computer-based instruction and learning.

Computers have been used in education for over 20 years [O'Shea, 1982]. Computer Aided Instruction - CAI<sup>2</sup> systems were the first such systems deployed and designed as an attempt to teach using computers. In these systems, the instruction was not individualized to the learner's needs. The

decisions about how to move through the lesson were script-like, such as:

- if the current screen is read, proceed to the next screen, or
- if the current question is answered correctly, go on with the next question; otherwise go to question 33 or,
- if the current picture is shown, proceed with next topic, or
- if the current task is done correctly, proceed with next topic; otherwise go back and show the current topic.

These actions were buried into the source code, meaning that any changes to some noticed disadvantage were impossible without the programmers. Analogously, the learner's abilities and previous knowledge were not taken into account; the same teaching material is presented to everybody in the same way, and there is no difference between different classrooms. The required knowledge for learning some lesson in some domain is not the same even in one country itself (the north - south problems, country - town, etc.), but CAI systems do not allow different approaches. Also those systems do not inspire or motivate the learners and they do not identify common or specific errors or misunderstandings.

We could say that individually pieced instruction in computer programs and frame-based computer aided instruction comprised early attempts of using the computer in education. Even though these systems had some success for some types of learning, generally they did not prosper because their learning environments were too monotonous and simple and their ability to adapt was limited to branching between static screens. Recent technological advances in computer hardware, software and multi-media systems, by themselves have been insufficient to realize the dreams of a computer-driven revolution in educational practice. Significant benefits will start to accrue when computer-based technologies are systematically incorporated into a system of tools that support thinking and learning activities.

### Intelligent Tutoring Systems

Although CAI systems may be somewhat effective in helping learners, they do not provide the same kind of individualized attention that a student would receive from a human tutor. For a computer-based educational system to provide such attention, it must reason about the domain and the learner.

This has prompted research in the field of intelligent tutoring systems - ITSs. ITSs offer considerable flexibility in presentation of material and a greater ability to respond to distinctive, idiosyncratic and characteristic student needs. These systems achieve their "intelligence" by representing pedagogical decisions about how to teach as well as information about the learner. This allows for greater flexibility and ver-

<sup>2</sup>. The profusion of acronyms like CAI, CAL, CBE, CBL, CMI, etc. (where C stands for Computer, A - Aided or Assisted, B - Based, M - Managed, I - Instruction, L - Learning and E - Education), describe the variety of the names for early usage of computers in education. All these names, more or less, described a similar way of using the computers for education purposes.

satility by altering the system's interactions with the pupil or student.

We could say that ITSs are computer-based instructional systems that have separate data bases, or knowledge bases, for instructional content (specifying what to teach) and for teaching strategies (specifying how to teach), and they attempt to use inferences about a student's mastery of topics to dynamically adapt instruction. ITSs design is founded on two fundamental assumptions about learning. First, that individualized instruction by a competent tutor is far superior to classroom-style learning because both the content and style of the instruction can be continuously adapted to best meet the needs of the situation. Second, that students learn better in situations which more closely approximate the situations in which they will use their knowledge, i.e. they "learn by doing," learn via their mistakes, and learn by constructing knowledge in a very individualized way. ITSs use techniques that allow automated instruction to come closer to the ideal by more closely simulating realistic situations and by incorporating computational models (knowledge bases) of the content, the teaching process, and the student's learning state.

In the last half-decade, ITSs have moved out of the lab and into classrooms and workplaces where some have proven to be highly effective as learning aids. These systems have been shown to be highly effective at increasing students' performance and motivation too. For example, students using the LISP tutor [Anderson and Reiser, 1985, Anderson, 1990] completed programming exercises in 30% less time than those receiving traditional classroom instruction and scored 43% higher on the final exam. In another example, students working with an Air Force electronics trouble-shooting tutor for only 20 hours gained proficiency equivalent to that of trainees with 40 months of on-the-job experience [Lesgold *et al.*, 1990]. Also, students using Smithtown, an ITS for economics, performed equally well as students taking a traditional economics course, but required half as much time covering the material.

### ITSs Tools

Intelligent tutoring systems are becoming more common and are proving to be increasingly effective in their basic goal of individualizing the learning process. But, as we point out, the fact is that the definition and implementation of a typical ITS needs a quite large development environment and this process is very time consuming, i.e. the accomplishments of an average ITSs require huge computing resources, lot of money and time. Furthermore, the results are expensive and hardly portable to personal computers and as we know PCs are most prevalent in average schools.

Traditional ITSs are concentrated on the domain knowledge they are supposed to present and teach; 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 domain knowledge as much as possible. In other words, there are interactive and inte-

grated development tools for building ITSs, i.e. for helping the developer plug in some domain knowledge and test the prototype system and then gradually and incrementally develop the final system. Such integrated tools are often referred to as *shells* (e.g., ITS shells, or ILE shells, or IES shells, etc.), which usually require a knowledge engineer in order to be fully used, or *authoring tools*, which can be also used by human instructors who do not necessarily have knowledge engineering experience. Still more recent ITSs are developed for collaborative learning, and often use Internet/WWW/agents technology in order to provide comfortable, user-oriented distributed learning and teaching facilities [Brusilovsky *et al.*, 1997].

Authoring tools or shells can speed ITSs development and reduce costs by providing essential infrastructure along with a conceptual framework within which to approach an educational problem. However, a commitment to a given tool or shell entails other commitments that may or may not be suitable for a given application. For example, a shell or tool may be appropriate for only certain kinds of task domains.

Shells and tools<sup>3</sup> are commercially available for traditional CAI and multimedia-based training, but these systems lack the sophistication required to build intelligent tutors. Commercial authoring systems excel in giving the instructional designer tools to produce visually appealing and interactive screens, but behind the presentation screens is a shallow and insignificant representation of content and pedagogy. Youngblut [Youngblut, 1995] says there are too few ITSs to make informed design decisions about ITSs shells and authoring tools. There is certainly a grain of truth to this, but it is also true that so few ITSs exist for evaluation and generalization because they are so difficult and expensive to build.

Do we know enough about ITSs to build authoring tools or shells? The answer is probably “NO”. So, we must continue to explore the various approaches to ITSs authoring, and develop suitable metrics to evaluate and measure value of their success. Appropriate metrics for success include:

- the cost effectiveness of using the tools to build ITSs, i.e. the reduction of the price of the final product, the ITSs;
- the ease and skill with which the tools can be used, i.e. who could build the ITSs with that tool and how;
- the variety of ITSs that a tool can be used to build, i.e. the different subjects, level of education, diversity of teaching styles, student models, etc.; and
- the depth or sophistication, refinement and lack of simplicity of the ITSs the tools can produce.

It is still too early to evaluate the overall effectiveness of ITSs authoring tools, since most of the current tools and shells are different research tools, such as:

- **PIXIE** [Sleeman, 87],
- **Byte-sized Tutor** [Bonar, 1988],
- **IDE** [Russell *et al.*, 1988],
- **ID Expert** [Merrill, 1989],
- **KAFITS** [Murray and Woolf, 1990],
- **COCA** [Major and Reichgelt 1992],
- **GTE** [Van Marcke, 1992],
- **RIDES** [Munroe *et al.*, 1994],
- **EON** [Murray, 1997], etc.

These systems come with knowledge base structure and interpreters much like an expert system shell. The difficulty is knowing how much flexibility to offer to the teacher and how to make the knowledge acquisition process as easy as possible. These systems have not made it out of the lab or seen wide use in multiple domains because of one or more of these factors:

- They are too complicated and complex because they are based primarily on theoretical concerns or AI techniques;
- Using them still needs a big development team and huge computer resources, so the product is still expensive;
- These shells and tools are very inappropriate for changing their elements (they simply copy the philosophy of expert systems shells), i.e. the definition of the elements are buried in the source code, so some changes could require the complete redesign of the shell;
- Most of them are based on a specific instructional approach;
- The domains of application are limited because the system was modeled from an existing task-specific intelligent tutor and generalized to similar domains; and
- They provide tools for structuring and using knowledge, but not for creating appealing student interfaces or learning environments.

In general, these ITSs tools and shells are fairly powerful and general, but they 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, KAFITS and EON, which underwent some user testing).

### The GET-BITS Approach

Instead a traditional expert systems shells approach (like most of the above shells and tools), we attempt to solve these problems with ITSs, authoring tools and ITSs shells using the object-oriented approach combined with component based architecture. We use that approach in designing the new version of *EduSof* system [Jerinic and Devedzic, 1996]. 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

<sup>3</sup>. In the rest of the paper, to avoid misconceptions and different terms, we refer to ITSs authoring tools and/or shells as tools.

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.

The ITSs models used in most of these systems, shells, and authoring tools, as well as the corresponding knowledge models, differ only to a limited extent. However, the design methodologies employed vary a lot, and sometimes even remain blurred for the sake of system functionality alone. On the other hand, using a shell for developing an ITS brings more systematic design, but can also become a limiting factor if the shell doesn't support a certain knowledge representation technique or design strategy that may be needed in a particular system. Many researchers have also noted that current ITSs are usually built from scratch.

#### What Would be Nice to Have When we Develop ITSs?

It would be very nice if:

- we could *easily assemble* our ITSs, shells, authoring shells, agents, etc., *from existing and pre-tested pieces of software*, without the need to develop and implement them from scratch;
- we could have our shells and toolkits offering us *only the tools and options that we really need*; we don't want our shells and toolkits to lack a tool or an option that we really need in a given project, but we also do not need a whole bunch of unnecessary tools and options from them;
- we could *easily replace* any piece of software in an ITSs by a similar (and possibly new) one, without any serious harm to the rest of the system; this would allow us, for example, to experiment with several versions of our system, each one having a certain functionality implemented in a different way (i.e., by a different piece of software);
- in order to develop a new piece of software that we find out to be necessary in our project (and it happens frequently) we could *have some other piece of software to start with*; the other piece of software should, of course, be logically and functionally similar to the desired one whenever it is possible;
- we could *automatically refresh and update* the repository time after time, putting in it some new pieces and deleting some pieces of software that are no longer needed, based on the functionality and use-statistics information;
- *access* to the repository could be as easy as possible, e.g. through the Internet or an Intranet; in other words, if we could easily get, put, organize, and update software in a remote repository.

In short, it would be very nice if we could concentrate more on *design* of ITSs, and automate their implementation and maintenance as much as possible. It would be very nice if we could dedicate our work mostly to cognitive aspects, learning and teaching issues, and effectiveness of ITSs, while having most of the software changes done quickly.

#### The Architecture of ITSs

Traditional ITSs [Woolf, 1992] consist of an **Expert Module**, a **Domain knowledge module**, a **Student model**, **Tutor** or **Pedagogical module**, a **Diagnostic** or **Misconception module** and an **Interface** or **Communication module** (Fig. 1). We extended these ITSs concepts and their components with an **Explanation module** [Jerinic and Devedzic, 1997].

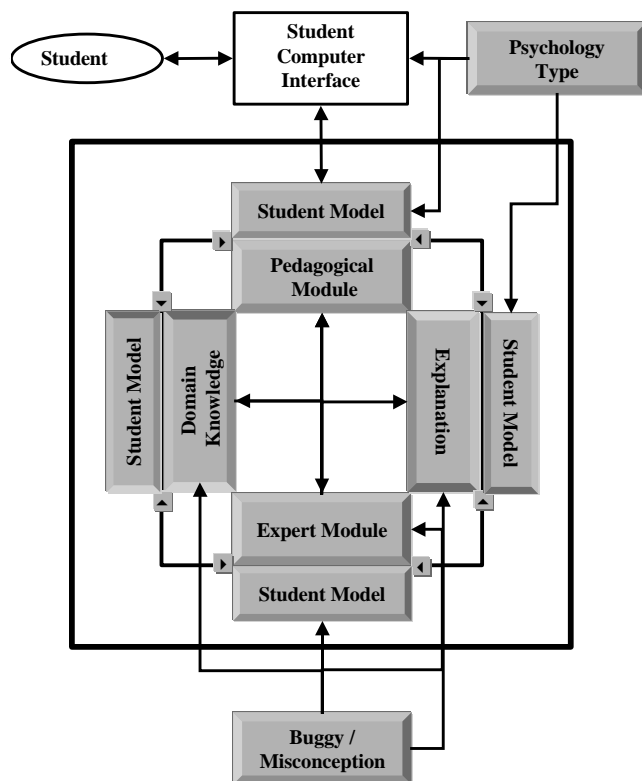


Fig. 1. The ITSs Architecture – GET-BITS approach

The **domain knowledge** module contains the information that the tutor is teaching, and is the most important since, without it, there would be nothing to teach the student. Generally, it requires significant knowledge engineering to represent a domain so that other parts of the tutor can access it. One related research issue is how to represent knowledge so that it easily scales up to larger domains. Another open question is how to represent domain knowledge other than facts and procedures, such as concepts and mental models. This module entails a general problem in AI: How do we represent domain knowledge? In **GET-BITS**, the domain knowledge expert module is represented with a knowledge network of semantically connected frames. That knowledge network represents the content to be taught in terms of lessons, concepts, rules, tasks, questions, actions, examples and

their interrelationships. It can be thought of as more than just a network of facts. The knowledge network in **GET-BITS** could be coupled to a domain expert, or it could be a collection of text, pictures, simulation procedures, and/or voice.

The **expert model** is similar to the domain knowledge in that it must contain the information being taught to the learner. However, it is more than just a representation of the data; it is a model of how someone skilled in a particular domain represents the knowledge. Most commonly, this takes the form of a runtime, or executive, expert model, i.e. one that is capable of solving problems in the domain. By using an expert model, the tutor can compare the learner's solution to the expert's solution, pinpointing the places where the learner had difficulties. The expert module consists of the domain knowledge to be tutored. This component of the ITSs is assigned the task of generating problems and evaluating the correctness of the student's solutions. In **GET-BITS**, the expert module is represented with *control knowledge* and with the concept of *transactions*.

The **interface module** is the mechanism by which the student and the tutor communicate. No communication can exist without this component, and therefore it has been a part of every ITS developed to date. Presenting this module as a component of the ITSs architecture reflects the current view of the importance of a well-designed user interface to any interactive program. The general goal of the user interface is to use the available devices, usually keyboard, mouse, monitor, etc., to display to the student the necessary information, and use these devices to obtain students' responses. Interactions with the learner, including the dialogue and the screen layouts, are controlled by this component. How should the material be presented to the student in the most effective way? This component has not been researched as much as the others, but there has been some promising work in this area. In **GET-BITS** we use a menu-driven, icon-managed, and easy-to-use user interface.

The **tutor module** is responsible for deciding how and when the domain knowledge is to be presented to the student. This component provides a model of the teaching process. For example, information about when to review, when to present a new topic, and which topic to present is controlled by the pedagogical module. As mentioned earlier, the student model is used as input to this component, so the pedagogical decisions reflect the differing needs of each student. Because the tutor needs to know information about the student to be able to make these decisions, the tutoring module relies heavily on the student-modeling module. In **GET-BITS** the tutor module consists of various teaching strategies, the rules that decide what should be taught to the student given the current state of the student model. This is realized by parameterizing a semantic network of frames, with the combination of If-Then-Action rules, as the part of the control knowledge.

The **error or diagnostic module** in **GET-BITS** consists of the rules used to identify student misunderstandings, update

the student model, manage the teaching rules if necessary, and store common student mistakes and errors.

The **student model** stores information that is specific to each individual learner. At a minimum, this model records how well a student is performing on the material being taught. A possible addition to this is to record misconceptions as well. Since the purpose of the student model is to provide data for the pedagogical module of the system, all of the information gathered should be able to be used by the tutor. In the **GET-BITS** model we extend the student model with the psychology type,<sup>4</sup> i.e., the relevant knowledge about an individual student concerning the flow of information in an ITSs that is referred to as *student behavior*. As a high level component, it can be seen as an aggregation of two other components: Student history and a student model. Although some authors discuss whether student history should be naturally included in the domain knowledge or it should exist as a separate module, that problem is not of high importance in the **GET-BITS** model. The student behavior component and its parts look the same regardless of the part of the knowledge base in which they are included; their relations and communication with other components are always the same.

#### The **GET-BITS** model of knowledge bases

An important area in modern intelligent software technology is that of integrating multiple knowledge sources, knowledge types, reasoning paradigms, inference models, etc. In a general case, at the architectural level an ITS is a complex system with its knowledge and problem-solving activities distributed to several modules. **GET-BITS** acts as the ITSs shell allows that several knowledge sources operate together in the learning process.

#### Knowledge base contents

It is possible to think of an ITS's 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 the third part that we call *explanatory knowledge*.

#### Domain Knowledge

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

<sup>4</sup> As is shown in Fig. 1, the part of the ITSs in **GET-BITS** denoted as the **psychology type** has influence on the other parts of the ITSs, especially on the communication module.

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.

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 [Devedzic and Jerinic 1997] are usually grouped to create meaningful *collections*<sup>5</sup>. One possible criterion for defining such a collection (the one used in the **GET-BITS** model) is the type of 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.

### Control Knowledge

The contents of the ITS's control knowledge [Jerinic *et al.*, 1998] are abstract, explicit, and more or less domain-independent descriptions of the way to learn some facts during the ITS's operation. The **GET-BITS** model defines five levels of control in a problem solving process, Fig. 2. Appropriate control knowledge is associated with each level, and should be specified explicitly in the knowledge base. Doing so guarantees maximum flexibility and maximum clarity in both the knowledge base development and the ITSs operation. On the other hand, it requires an extensive set of reasoning procedures and tools to be available to the system developer (i.e., to be implemented and integrated with the shell used for the ITSs development).

The **GET-BITS** model defines a base (and abstract) class and a set of concrete subclasses describing objects that perform control at each of the five levels. By instantiating these classes with the appropriate control knowledge (e.g.,

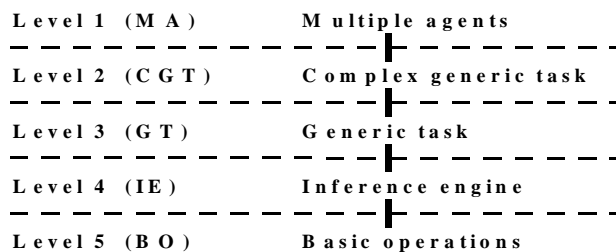


Fig. 2. The levels of control knowledge in the **GET-BITS** model

attribute values, particular methods, parameters, etc.), the ITSs developer defines control objects, i.e., specifies the control knowledge part of the ITSs knowledge base.

### Explanatory Knowledge

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 ITS's learning process, as well as the way they are generated. Explanatory knowledge is related to both domain and control knowledge, and often that knowledge is treated as a part of the other two components of knowledge bases. However, in the **GET-BITS** model it is treated as a distinct knowledge component in its own right in which the knowledge about the learning process explanation, knowledge elements explanation, control strategies explanation, and other types of intelligent assistance are represented explicitly.

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 with the user's knowledge level;
- *criteria* used by the explanation module when deciding what knowledge elements to include in the explanation and what to skip, as well as what level of detail to provide in the explanation.

It must be stressed that in addition to explanatory knowledge, the explanation module uses extensively the knowledge from the other two parts of the knowledge base 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 such as:

<sup>5</sup>. The word "collections" is used rather than sets, lists, tables, etc., in order to avoid suggesting a particular implementation technique at the analysis phase.

**IF**

The explanation type is **WHY**

**THEN**

Show the current goal, and  
Show the current domain rule instantiation, and  
Show the meta-rule that was last applied.

The **GET-BITS** model takes into account the fact that in today's ITSs the explanation module is often implemented as a user-oriented, rule-based system. It uses all the available information from the domain knowledge base (contents of lessons, objectives, topics, etc.), as well as from the student model, in order to answer the student's questions and provide desired explanations. Specifically, its components support:

- determining the contents of the answer/explanation,
- deciding upon the explanation presentation style, depending on the question itself and on the relevant attributes of the student model,
- selecting the knowledge model that provides the easiest explanation in cases when multiple knowledge models can be used, and
- composing the explanation and ordering its statements in a coherent, reasonable and comprehensible way.

The abstract class **PQ** describes common elements of the **Problem** and **Question** classes. It defines the fields *WhyPtr*, *HowPtr*, *WhatPtr*, etc. With these fields lesson creators can store explanations related to a problem and/or a question. The instances of the *Problem* and *Question* class are presented to the student during the learning process.

The **GET-BITS** model distinguishes among several kinds of explanations that can be presented to ITSs users. Some of them are:

- explanations required from students when checking their knowledge - the *StudentExplanation* class;
- explanations presented to the ITS developers - the *DeveloperExplanation* class;
- explanations concerned with explaining the system's functioning - the *SystemExplanation* class;
- explanations of various concepts or topics - the *ConceptExplanation* and *TopicExplanation* classes, etc.

In generating explanations, **GET-BITS** based ITSs can use knowledge from various kinds of knowledge elements (rules, frames, knowledge chunks, etc.). Regarding the explanations generated as results of the student's request for explanation, **GET-BITS** defines various types of explanations according to the type of question the student puts: *WhyExplanation*, *HowExplanation* and *WhatExplanation*, corresponding to the *Why*, *How* and *What* requests.

#### **Knowledge primitives in the GET-BITS Model**

The knowledge that student must learn could be presented as a tree, where each node is a piece of domain knowledge. A parent-child link in the tree implies that in order for the stu-

dent 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 AI - Implementation View**

The main object classes are the **Lessons** and the way they are presented. Units of knowledge that can be taught, repeated, summarized, etc., represent **Lessons**. 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., which point to presentations. A presentation is an array of questions or tasks that represent expository or inquisitor interactions with the student. It is 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. A presentation also contains the possibilities for responding to the pupil, such as hints, congratulations, elaborations of the answer, etc. That knowledge representation concerns some particular and concrete type of knowledge element that is necessary for the realization of the **Expert** module in the ITSs shell. Any lesson consists of one or more issues, modeled by the class **Topic**. The basic attributes of the lesson are: the name, current topic, 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, and so on.

The issues that students must learn (class **Topic**) are realized with separate type of knowledge elements in the **GET-BITS** system. Any topic could be introduced with text information, graphically, and/or with the simulation of some event. Also, further explanations for that theme, or some hints could be defined. The class **Topic** in **GET-BITS** is made for specifying and defining the issues or notions needed for lesson creation.

#### **The Teachers – Users View**

The original goal of developing the **EduSof** system was to design a conceptual framework for representing objects, events, responses, reactions, and relationships in tutoring and learning. The system features an interactive graphical environment in which the teacher can manipulate objects at different levels of abstraction. These objects and the 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 customizing the machine's responses.

The power of using AI technology to build an educational software package results from the possibility of representing

abstraction and models in the computer system. This means that it is possible to encode some abstract entities such as "when the pupil is confused give more explanation," or "give topic overviews only for the beginning level," or "give this response for a right answer." The **EduSof** system uses a hybrid method of knowledge representation. This representation is based on the object model of frames, modified with a parameterizing semantic network technique and rules.

The **EduSof** system and the appropriate knowledge database are designed as an ITS shell. The structure of knowledge used in them is **dynamic**. The knowledge base is realized using a frame system, based on the **GET-BITS** model. Teachings and examining methods for lessons or knowledge are available to lesson creators (usually teachers) and are independent of the system. **EduSof** has a great capacity for representing knowledge through teaching sequences in almost all fields of educational process.

The **EduSof** system is organized in three separate modules. The creating module **Tea** (Fig. 3) serves for designing of lessons by the teacher of the subject. The second module **Lea** is a kind of interpreter of the knowledge introduced by the **Tea** module; it is used by pupils to master the subject-matter and get the pertinent knowledge. The third module **Exam** can be used to test students for marking.

In **EduSof**, a teaching sequence is organized as a list of semantically connected elements - **Lessons**. They consist of a list of **Topics** (Fig. 4). A topic, as the basic component of a teaching sequence, can be expressed by means of **Text**,

**Picture**, **Voice** and/or **Simulation**. For each **Topic**, a list of **Questions**, **Task** or **Problems** are given. This object of learning could be used for testing the acquired knowledge connected to that **Topic**. For each **Question**, **Task** or **Problem**, a list of **Alternatives**, **Explanations**, **Hints**, **More Information**, etc. are given.

For each **Topic** element, the **Control** mechanism (Fig. 5 - Connections) decides **What Next** the pupil will learn, in case a right or wrong answer is chosen, as well as based on **Explanation**, **Buggy Model** and the learning/teaching strategy.

### Discussion and Conclusion

The purpose of the proposed **GET-BITS** model of knowledge bases and knowledge base management is to make a basis for applying the ideas of object-oriented software design methodology to ITSs knowledge organization, representation, and access. It covers all-important aspects of knowledge bases. However, it is important to stress again that the **GET-BITS** model should be regarded as an open framework for developing ITSs knowledge bases, rather than as a closed set of design rules and organizational hierarchies. In fact, the model has several open ends. The first open end is the possibility of extension by including new knowledge representation techniques. Such an extension can be done extremely easily, without reorganizing the entire **GET-BITS** model. The model can also be extended to support new transaction types. This is particularly important if it is required to support some specific complex transaction.

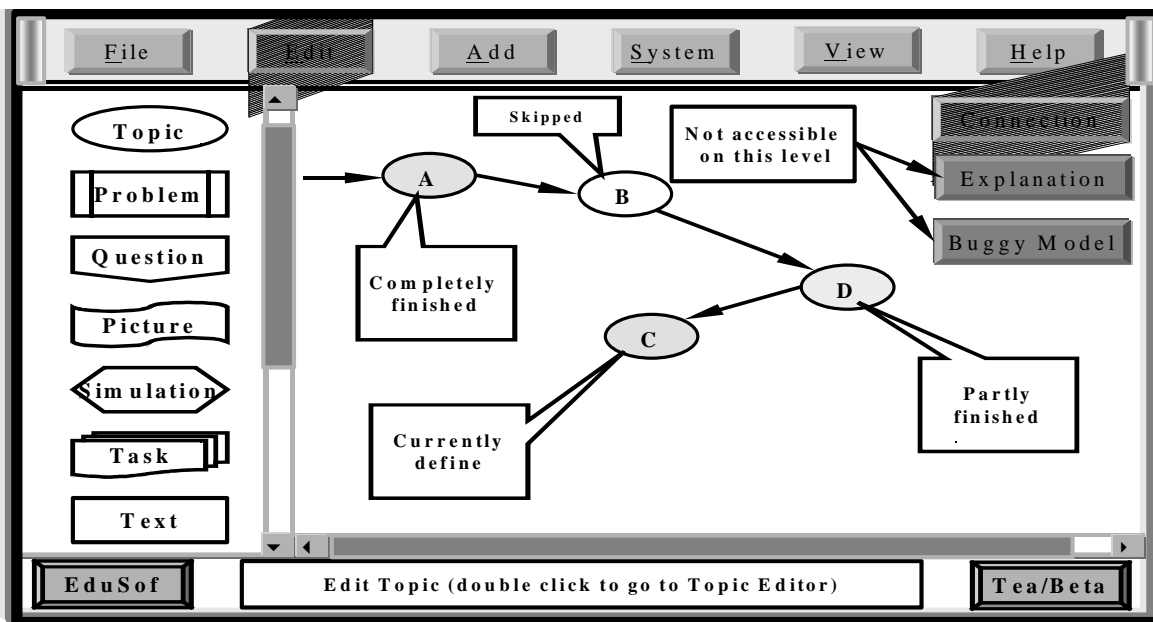


Fig. 3. The Tea module - creation of the lesson

Moreover, it is possible to make an extension to support a certain structured query language for examining knowledge bases, an analogy to SQL in relational databases.

The presented method for building a knowledge base with explanation facilitates, suitable for the educational purpose is currently under development and the realization in the present form. In sum, the explanation part of **GET-BITS** is capable of playing the role of:

- a "research resource" that helps a student find the information he/she needs;
- choosing between alternatives when they are available;
- adding related information that motivates, enriches, and enables the student's understanding of the primary material selected;
- ordering all of this for a coherent presentation in a multimedia environment.

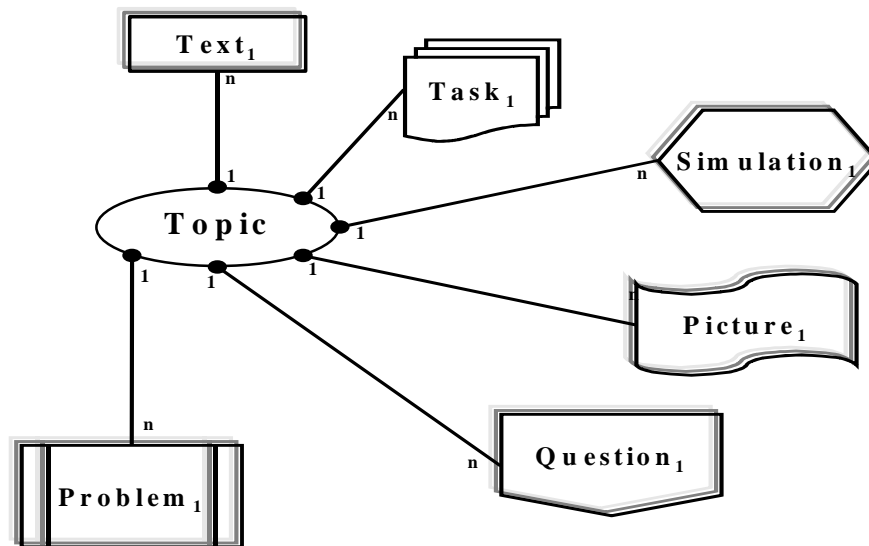


Fig. 4. The Topic element

The **GET-BITS** model of intelligent tutoring systems, presented in the paper, allows for easy and natural conceptualization and design of a wide range of ITSs 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 ITSs 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. In the current state of development we tested usage of **EduSof** in some summer schools in various subjects (chemistry, history, geography etc.). We did realize one of the goals we set down before

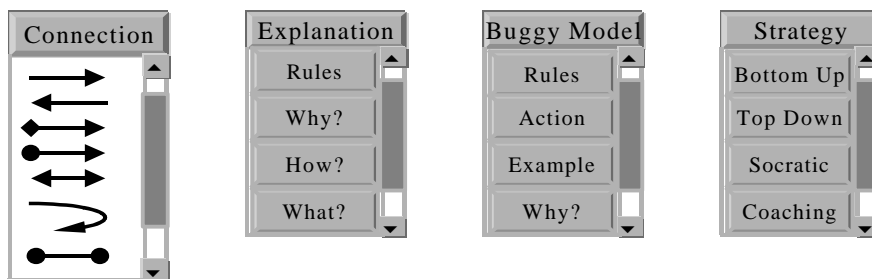


Fig. 5. The Control, Explanation, Buggy and Learning/Teaching Editor

the start of the realizing of the above concepts. The teacher's independence in design of the intelligent tutoring lessons reached almost 90% after an hour of explanation of how to use the **EduSof**. The other 10% is professional programmer time for designing simulation procedures. It is supposed that the teachers have some knowledge of using computers, a little knowledge of computer graphics, and some help of professional programmers in making simulation procedures.

In the 1997/98 school year **LeaPas**, an ITS made using **EduSof** and **GET-BITS** was used in teaching the "Introduction to programming", course in the first year of computer science studies at the University of Novi Sad. We used one group, about 20 students, and through practical exercises they used the **LeaPas** instead the classical method "chok and table, teacher explanation and try on computer alone and ask the professor for help"! At the final examination the average success of testing for students that trained with the traditional talking or class (the teacher are talking to the group of students from 50 to 200, and students practice alone with assistance on computers) method is between 40 and 60 from 100 points. When the teacher with same teaching strategy used the diagnostics test (to improve and modify his/her approach to that teaching subject), the results were better, i.e., about 74 from 100. But, when the students have individualized training with **LeaPas**, the results were 90-91 from 100.

Further development of the **GET-BITS** model, as well as the **EduSof**, are 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 ITSs 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

- [Anderson and Reiser, 1985] Anderson, J. and Reiser, B., The LISP Tutor. *Byte*, 10, 4, 1985, 159-175.
- [Anderson, 1990] Anderson, J., Analysis of Student Performance with the LISP Tutor. In Frederiksen, N., Glaser, R., Lesgold, A.M. and Shafto, M. (Eds.), *Diagnostic Monitoring of Skill and Knowledge Acquisition*, Hillsdale, NJ: Lawrence Erlbaum, 1990.
- [Bonar, 1988] Bonar, T., Byte-sized Tutor. In J. J. Psothka, L. D. Massey and S. A. Mutter (Eds.) *Intelligent Tutoring Systems: Lesson Learned*, Lawrence Erlbaum Associates, 1988.
- [Brusilovsky et al., 1997] Brusilovsky, P., S. Ritter and E. Schwarz, Distributed Intelligent Tutoring on the Web. In du Boulay, B., Mizoguchi, R. (Eds.) *Artificial Intelligence in Education*, IOS Press, Amsterdam / OHM Ohmsha, Tokyo, 1997, 482-489.
- [Clancey, 1996] Clancey, W. J., Developing learning technology in practice. Personal correspondence, 1996.
- [Devedzic and Jerinic, 1996] Devedzic V. and Jerinic, Lj., Explanation in Intelligent Tutoring Systems. *Bulletins for Applied Mathematics*, 1196/96, 1996, 183-192.
- [Devedzic and Jerinic, 1997] Devedzic, V., Jerinic, Lj., Knowledge Representation for Intelligent Tutoring Systems: The GET-BITS Model. In: du Boulay, B., Mizoguchi, R. (Eds.) *Artificial Intelligence in Education*, IOS Press, Amsterdam / OHM Ohmsha, Tokyo, 1997, 63-70.
- [Frakes, 1994] Frakes W. B., Success Factors of Systematic Reuse. *IEEE Software* 11(5): 15-19, 1994.
- [Jericin and Devedzic, 1996] Jerinic, Lj. and Devedzic, V., An object-oriented shell for intelligent tutoring lessons. *Lecture Notes in Computer Science* Vol. 1108, 1996, 69-77.
- [Jericin and Devedzic, 1997] Jerinic, Lj. and Devedzic, V., OBOA Model of Explanation Module in Intelligent Tutoring Shell. *SIGCSE Bulletin*, Vol. 29, Number 3, September, ACM PRESS, 133-135.
- [Jericin et al., 1998] Jerinic, Lj., Devedzic, V. and Radovic D., Control Knowledge and Pedagogical Aspects of the Get-Bits Model. *Lecture Notes in Artificial Intelligence* Vol. 1416, Springer-Verlag, Berlin, 1998, 735-744.
- [Lesgold, et al., 1990] Lesgold, A., Lajoie, S., Bunzo, M. and Eggan, G., A Coached Practice Environment for an Electronics, Troubleshooting Job. In Larkin S., Chabay T. and Sheftic A. (Eds.) *Computer Assisted Instruction and Intelligent Tutoring Systems Establishing Communication and Collaboration*. Hillsdale, NJ: Erlbaum, 1990.
- [Lim, 1994] Lim W. C., Effects of Reuse on Quality, Productivity and Economics. *IEEE Software* 11(5): 23-29, 1995.
- [Major and Reichgelt 1992] Major, N. P. and Reichgelt, H., COCA - A shell for intelligent tutoring systems. In Frasson, C., Gauthier, G. and McCalla, G. I. (Eds.) *Intelligent Tutoring Systems*, LNCS 668, Springer Verlag, Berlin, 1992.
- [Merrill, 1989] Merrill, M. D., An Instructional Design Expert System. *Computer-Based Instruction*, 16: 3, 1989, 95-101.
- [Munroe et al., 1994] Munroe, A., Pizzini, Q., Towne, D., Wogulis, J., Collier, L., Authoring Procedural Training by Direct Manipulation. USC working paper WP94-3, 1994.
- [Murray and Woolf, 1990] Tom Murray, and Beverly Woolf. A Knowledge Acquisition Tool for Intelligent Computer Tutors. *SIGART Bulletin* Vol. 2, No. 2: 1-13, 1990.
- [Murray, 1997] Murray, T., Authoring Knowledge Based Tutors: Tools for Content, Instructional Strategy, Student Model, and Interface Design, Submitted to the Journal of the Learning Sciences, <http://www.cs.umass.edu/~tmurray/>, 1997.
- [O'Shea, 1982] O'Shea, T., Intelligent Systems in Education. In D. Mitche (Ed.) *Introductory Readings in Expert Systems*, Gordon and Breach Science Publishers, New York, 1982, 147-176.
- [Psothka et al., 1988] Psothka, J. J., Massey, L. D. and Mutter, S. A. (Eds.) *Intelligent Tutoring Systems: Lesson Learned*, Lawrence Erlbaum Associates, 1988.
- [Russel et al., 1988] Russell, D., Moran T. P., Jordan, D. S., The instructional design environment. In J. J. Psothka, L. D. Massey and S. A. Mutter (Eds.) *Intelligent Tutoring Systems: Lesson Learned*, Lawrence Erlbaum Associates, 1988.
- [Sleeman, 87] Sleeman, D., PIXIE: A Shell for Developing Intelligent Tutoring Systems. *Artificial Intelligence in Education*, Vol. 1, 239-265, 1987.
- [Van Marcke, 1992] Van Marcke, K., Instructional Expertise. In Frasson, C., Gauthier, G. and McCalla, G. I. (Eds.) *Intel-*

*igent Tutoring Systems*, LNCS 668, Springer Verlag, Berlin, 1992.

[Woolf, 1987] Woolf, B., Theoretical Frontiers in Building a Machine Tutor. In Kearsley, G. P. (Ed.) *Artificial Intelligence and Instruction-Application and Methods*, Addison-Wesley, Reading, 1987, 229-267.

[Woolf, 1992] Woolf, B., AI in Education. In Shapiro, S. (Ed.) *Encyclopedia of Artificial Intelligence*, John Wiley & Sons, Inc., New York, NY, 1992, 434-444.

[Youngblut, 1995] Youngblut, C., Government-Sponsored Research and Development Efforts in the Area of Intelligent Tutoring Systems: Summary Report. Inst. for Defense Analyses Paper No. P-3058, 1995.

#### **About the Authors**

Ljubomir Jerinic is lecturer and professor at the University of Novi Sad, Faculty of Science, Institute of Mathematics. His researcher interests are Artificial Intelligence, Knowledge Representation and the usage of AI methods in Education and Medicine.

Vladan Devedzic is professor at the University of Belgrade, School of Business Administration. His researcher interests are Expert Systems and the usage of AI methods in Education and Medicine.

#### **Author's Addresses**

Ljubomir Jerinic  
University of Novi Sad, Faculty of Science  
Institute of Mathematics  
Trg Dositeja Obradovica 4  
21000 Novi Sad, Vojvodina  
Yugoslavia

Email: [Jerinic@uns.ns.ac.yu](mailto:Jerinic@uns.ns.ac.yu)

Tel: +381-21-58 888

Vladan Devedzic  
FON - School of Business Administration  
University of Belgrade  
Jove Ilica 154  
11000 Belgrade  
Yugoslavia

Email: [Devedzic@galeb.etf.bg.ac.yu](mailto:Devedzic@galeb.etf.bg.ac.yu)

Tel: +381-11-237-1440