Get-Bits Model of Intelligent Tutoring System for Learning Introductory Programming

Ljubomir Jerinic

Institute of Mathematics, University of Novi Sad
Trg Dositeja Obradovica 4, 21000 Novi Sad, Yugoslavia
Fax: +381-11- 350458; Tel: +381-11-58888
Email: jerinic@uns.ns.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 realization of some new approaches in that field. Issues of pragmatics and usability motivate our starting point and perspective on developing of ITSs tool called GET-BITS. Considering commercially available and widely used authoring systems for traditional computer-based teaching, we try to give the next step, the next paradigm shift that is needed to enable some of the ITSs advantages. It was developed while trying to redesign a previously developed ITSs called EduSof. The model enables the developing of more flexible software environment for building of the ITSs, significantly increasing their reusability, and the model can be easily extended to cover the needs of particular tutoring systems. In this paper an overview of using the GET-BITS model in designing of ITSs for learning of basic programming is given. The aim in realization of LeaPas system, intelligent tutor realized by using EduSof shell, is to help beginners in programming in solution defining, learning language construction while using adequate examples, as well as in implementation and testing of his own programs. Intelligent tutoring system LeaPas does not only give reports on syntax, semantic and conceptual user errors. It also tries to understand users point of view while designing solution of given problem, to help, to give advice and accept original ideas, i.e. self-learning is possible. Inter-code representation of knowledge of basic programming enables pupil to direct creation of his solution. One mental model is given, the way of thinking, for better problem solving in basic programming.

Introduction

The endeavors to design Intelligent Tutoring Systems (ITSs)\(^1\) [Ohlsson, 1987], [Woolf, 1988] and [Woolf, 1992] have opened a new research direction in the application computers for educational purposes. These systems are capable to help solving difficult problems in the process of knowledge transfer, i.e. the teaching and learning. They enable an almost full individualization of the teaching process, as well as the student's advancement in the stages of acquiring knowledge and application of the new knowledge according to their capabilities, aspirations, previous knowledge and the like. Also, ITSs stimulate the use of computers in education as a new education tool and controller of the teaching process, and opened new approaches for the use of computers in education.

When first introduced more than 10 years ago [Psotka et al., 1988, Frasson, 1988], ITSs 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, Wolf’s Meno Tutor, Anderson’s LISP tutor, Nwana’s FITS, Wagner’s SCHOLAR, Johanson’s PROUST, etc gave promising and auspicious beginning), ITSs have not see the general acceptance yet. 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 wide spread adoption. Much of this has to do with the complexities involved in definition and design ITSs applications, as well as the paradigmatic changes required of training and education organizations in the way they practice instructional design in order to accomplish this authoring [Clancey, 1996].

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 have 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. Also, the educational programs are different among the distinct countries. Then, 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.). The teachers want to have the active role in design and eventually update and improvements of intelligent tutors. For these teachers, however, ITSs is the

\(^1\) 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.
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 could transparent the inner mechanisms of ITSs and help them to made their own lessons, would be a powerful tool to help them in understanding and comprehending this new paradigm of teaching by computers.

While intelligent tutoring systems are becoming more common and proving to be increasingly effective; each one must still be built from scratch at a significant cost, the big developing team and the large computer resources. Contrariwise, the 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]. The main reason for taking into account reusability issues in the ITSs field is due to the fact that to build ITSs needs quite big development environments. The implementations of an average ITSs require huge computing resources, lot of money and 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.

In view of the above trends and research issues, the goal of designing the EduSof system was to construct a conceptual framework 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 which works on ordinary PC or similarly hardware, on different platforms and with which the production of the ITSs systems will be cost-effective. Our approach in resolving the above goals is definition and realization of the:

- switch between the old paradigm based on the ITSs shells and ITSs closed systems;
- methodology of the designing and implementation the knowledge bases needed for tutoring and learning;
- representation of the objects, events, responses, reactions, and relationships involved in tutoring and learning, etc.

The realization of the above goals is based on the GET-BITS (GEneric Tools for Building ITS) model [Devedzic and Jerinic, 1997], i.e. the component based object-oriented ITSs framework. The GET-BITS model is essentially a specific extension of a more general, recently developed model of knowledge bases, called OBOA (Object-Oriented Abstraction) [Jerinic and Devedzic, 1996]. In this paper one application of the new version of EduSof, for realization of ITSs for supporting the process of learning of programming is briefly described. This sort of system enables beginners to establish, design, implement and test their first programs. Realization of that intelligent system is made using inter-code technique. Tutoring system finds and gives reports on users syntax, semantic and conceptual errors. However, system tries to:

- understand pupils attitude toward solving the problem and eventually partially or completely understands and verifies the whole result of given problem in learning of the programming, as well as
- teach the elements of the programming language (Pascal) itself, and
- the basic concepts (data type, flow control) of programming.

The LeaPas system and the incorporated knowledge of learning programming are based on the great experience (more then twenty years) of the author’s in tutoring the bases and elements of programming.

**Expert Knowledge in Learning Programming**

In the expertise of knowledge for realization of learning of programming basis as well as contents of this expertise there are two cases:

- problem and formal representation required knowledge for etching of programming, and
- problem of learning, itself and acquisition of that knowledge.

Although programming (or software development) can be considered as relatively young discipline, which includes that methodology of learning of programming is not completely developed, just the learning of programming has additional difficulty in handling and use of abstract as well as computed concept and process. From one point of view, in programming defining and designing of object and program are used, which is similar to other areas that use design and construction, for example engineering. From the other point of view, the nature of these objects which are being defined and used, require strong theoretical base and acceptable programming that leads ultimately to mathematics. Although great results are achieved in programming theory, before all, in corresponding ways of program forming description, programming languages, programming transformations, predicate logic, correctness proving etc. At the present point of development of computer science we still have one practical and useful methodology which will cover the problem of construction and program realization and methodology convenient for unification of development process of the whole software.
From above mentioned reasons the problem of learning of programming is specially difficult, and from that expertise of knowledge needed for effective and correct lecture and learning of programming basics. The aim is defining of methodological aspects of programming learning, i.e. defining corresponding expert rules in knowledge realization, the abstraction technology is chosen and inter-code, i.e. realization of ideas of visual programming with communication on formal language which is subset of English language.

**Phases of Problem Solving**

Realization of problem of learning of programming in this intelligent tutoring system begins with defining of elementary problem to the programmer the beginner. After that the process of problem solving in steps is controlled and supervised, and eventual semantic or syntax errors are being taken care of. In mentioned process advises be given for more correct and effective solution and at the end the code is generated in needed programming language, in concrete case in *Pascal* programming language. Realization of so based programming learning is happening in three phases.

The first phase brings informal construction to solve given problem. Language on which the interaction between pupil and teacher (computer), during the learning, is made on very high level. This communication is very similar to the way that programmer writes informal instruction in an pseudo-language or draws data flow diagram of program execution, trying to describe to the beginners the process of solving the problem. For example, pupil can construct informal instructions (sentences):

- execute block and do it while sum is greater than 100; or
- read from the keyboard until <end> is given, etc.

After that solution analyze is made. Eventual conceptual errors are being discovered. Before going to the next phase, pupil must complete the whole development of solution in this language.

In the second phase, refined and more detail program blocks defined in the later phase, are being introduced. Introducing of half-formal programming plans does this. Plan is similar to the structure in data flow diagram and describes how given aims are transformed into real program code. In the paper of [Soloway and Ehrlich, 1985], [Navrat and Rozinajova, 1993] and [Bonar and Liffick, 1988] similar method is used. In the learning programming by using LeaPas, the array of advancements is introduced in knowledge base, gained in work experience with programmer the beginner. Usually plans have different roles as well as correlation and influence on other plans. The access to problem solving using plans enables pupils to concentrate more on solution, use corresponding plans, without barrier on syntax complexity and programming language realization (data type, corresponding operators, ways of memorization, different language limitation) which must be taken into the consideration when explicitly programming.

The role of plans and their definition depends on chosen programming language, because there are different programming styles (imperative, declarative). Of course, that there is a group of plans common to for couple of styles. In the sole plans is a sort of atoms, i.e. axioms that are used while programming. In this plan based programming learning it is possible to test, i.e. pseudo-execute atom. Plans are being managed by using parameterized semantics rules net, like:

- If there is use of ARRAY and INPUT then use FOR statement; or
- If there is use of STRUCTURED TYPES and there is no use of INPUT then use DYNAMIC TYPE and generate the initial values; or
- If there is use of RECORD then use WITH statement instead of ASSIGN statement.; or
- If there is use of WHY or HELP the Explain why or Give the hints, etc.

In the third phase translation of plans is being carried out. Plan base conception is translated into real code using corresponding programming language, in our example PASCAL. In this way generalized programming code can be corrected using simple editor, as well as testing using corresponding translator.

**The Use of the System**

Learning of programming in intelligent tutoring system begins with giving the problem to the pupil. This problem must belong to the beginner exercises. The tasks are being chosen that they present in simple and effective way basic concepts of imperative programming like splitting or iteration. Pupil while solving given problem goes through three described phases.

In the first phase pupil controls the array of instruction in formal language, subset of English language, use technique of modular development of program (bottom up of the other way around, depending on level of learning). Since the system is current phase realized for learning of imperative programming, block technique, step by step technique and structural programming technique is used (every block has one input and one output). In the next phase given construction are developed using plans and program is built using plan representation. In the last phase plans are being further developed and concrete program command in given programming language is generated. In current phase programming language *Pascal* is chosen.

In the further description of use, as well as this system use demonstration, suppose that following problem is given:
"Write a program which inputs from keyboard some integer number and write it in reverse order of numerals"

During the solving process a student from the main menu chooses the option marked with “Chose the concept”, which as submenus has beginning phrases: "Count", "Assign", "Output", "Continue", "Take", "Input”, “Work until”, "For”, etc. By choosing of some construction a new submenu for further work is given. For example for option "Input " submenu has "One integer", "More integers", "Array", “Record”, etc. There is also, the descriptions to the user like "What to do?", “How?”, etc. On every level there are options for help, definitions, methods, examples, additional information and/or explanation why that option is used right now. System for wrong and incorrect use of some construction explains why it is misused or warns user about possible errors, which will come up later.

After first phase, as one of possible solutions the following array of sentences is gained, i.e. pseudo-commands:

**Input ... One integer ... to Variable Number**
**Assign ... to New Variable Numerical ... One integer with module of that integer with 10**
**Output ... Numerical**
**Assign to Number ... the old Number divided with 10**
...
**Work until ... N is equal to 0**

Every from above mentioned completed or uncompleted pseudo-constructions are automatically presented on the interaction window and can be changed with adequate option. After this phase is finished tutor symbolically compares representation of this version of program with list of demands in concrete anticipated solution. This list of demands is created in the phase of lesson preparing by the teacher as the part of problem description in corresponding language.

The list of demands is usually given for more different levels. These levels are adequate to different student models for which expert considers that the student can create. The differ from exact solution, in the sense that in them not all plans must be defined, they are freed from eventual borders and the let flexible pointing. For example, one of these student models for loop explanation, enables that pupil makes loop construction by describing just the first iteration, which is followed by the “Etc…” construction. Based on expert rules system tries to conclude weather the first iteration is good, and if this is the case informs the pupil how to complete the whole expression. In contrast, if the use and definition of this first iteration is not good it gives explanation with possible consequences. If the system is unable to conclude what the pupil wanted it informs the user about that with request for further specification.

Diagnostically the part of strategy used in this phase is based on comparison and finding of a student model for some problem which is being solved. Every model requires some plans made in subset of English, as well as corresponding organization of these plans. They analyze the pupils’ solution on that informal level, tutor computes how this solution satisfies requirements for every realized student model. Further learning and tutor behavior is leaded and realized by using the simplest unsatisfied plan of the most simple student model with one or more unsatisfied plans.

In the phase two further work on the specification given in the phase one is done. Variables, connections between concepts and plans are introduced etc. For example when loop is being done for input from the example in which the task is to count some numbers and calculate their sum, the following result is gained:

**Begin … Loop**
**Input X**
**Add X to Full Sum**
**Count How Numerous**
**End … Loop**

In the third phase with source to source technique translation from inter-code representation of pupils solution automatically *Pascal* program is generated. Code produced in this way can be modified with the call to little editor, translated with the appropriate compiler, and tested with the call to corresponding interpreter.

**The Examples of Task for Students**

Special attention is done in assembling and incorporating the suitable and appropriate examples for learning and testing the student’s knowledge. The examples are not the common and ordinary tasks. For example, instead to give the usual task for realizing the concept of array and the sum of some numbers, in the for of:

"Write a program which inputs from keyboard an array of integer numbers and calculate their sum"

In the *LeaPas*, the following problem (task) is given to the students:

"The fourteen young girl called Andrea, during her excursion is collecting the flowers. She gather 5 stalks of poppy, 10 lilac’s, 28 carnation’s, 17 daisy’s and 13 stalks of tulip’s. How many of flower’s stalks is Andrea pick?"

In the above example the *LeaPas* system allocate the following goals:

- To identify the “Numerous Type” to represent flowers;
- To learn that in *Pascal* the variables of numerous types could not be read from the keyboard;
• To calculate sum of the integer numbers;

In the following example, so called “the problem of mad scientist”:

“One mad scientist want to make the chemical chain, made of plutonium and lead atoms. But, if two atoms of plutonium are side by side, the chain reaction and atomic explosion will be. How many of ways the safe chain could be constructed of the length N, if the mad scientist have N atoms of lead and N atoms of plutonium?”,

the goal is to practice the recursive technique of programming. This problem is given instead the ordinary problems like:

"Write the Pascal function who realize the following mathematical function: \( F(N) = F(N-1) + 3, F(0) = 1. \)

or similar problems. We leave to the writers to recognize or solve the above problem.

The concept of “not everyday” examples in learning basic of programming and programming languages Pascal was shown very successful in the array of motivation the students. The students take active role in inventing the new interesting problems or the pre-definition the existing ones.

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, component based software design methodology to ITSs knowledge organization, representation, and access. It covers all-important aspects of knowledge bases, like their contents, knowledge representation techniques, using, updating, extending and maintaining the knowledge, etc. 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 easy, without reorganizing the entire GET-BITS model. As another open end can be easily extended to support new transaction types. This is particularly important if it is required to support some specific complex transaction. Moreover, it is possible to make an extension to support a certain structured query language for examining knowledge bases, an analogy to the SQL in the database technology.

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, explanation part of GET-BITS is capable of playing the role of:

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

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.

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 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.

In this paper the main interest is put on realization of one model for teaching of basics of programming. Inter-code technique is used, i.e. pseudo-representation of programming code in the language close to English is used. System is in the final testing phase. In the 1997/98 school year the LeaPas is used in teaching the “Introduction to programming”, course on the first year of computer science studies at the University of Novi Sad. We use one group, about 20 students, and through practical exercises they used the LeaPas instead the classical method “chock and table, teacher explanation and try on computer alone and ask the professor for help”! At the final examination the average success of testing the students that are training with the traditional talking or class (the teacher are talking to the group of students from 50 to 200, and students practice alone with assistants on computers) method is between 40 and 60 form 100 points. When the teacher with same teaching strategy use the diagnostics test (to improve and modify his/her approach to that teaching
REFERENCES