Prototype Model of Tutoring System for Programming

Tonći Dadić, Slavomir Stankov, Marko Rosić
Faculty of Natural Science, Mathematics and Education, University of Split
Teslina 12, 21000 Split
{tonci.dadic|slavomir.stankov|marko.rosic}@pmfst.hr

Abstract. *Computerized tutor for programming learning helps students to understand program constructs, and syntax of specific programming language. Also, it helps to improve problem-solving skill, and ability to evaluate program solution. In this paper we propose a model concept and architecture prototype of Tutoring System for Programming. It is based on our age-long research and development of the Tutor–Expert System, a model of hypermedia authoring shell for building intelligent tutoring systems. Paper focuses on student-system dialogue, error classes in student's program and mechanism to detect correctness of student's program.*

Keywords. Programming teaching, Intelligent Tutoring Systems, TEx-Sys model.

1. Introduction

Term computer literacy is closely related to learning about computers, and had caused many debates and contrary stands in a past. What is really computer literacy? Is it learning about computer or learning how to use a computer? Authors agree that computer literacy is as important as ability to read and write. In accordance with certain statements, a computer literacy is incomplete without cognition about computer programming skills and capabilities (*www.libertybasic.com*). According to Eisenberg and Johnson [6] term computer literacy includes: (i) a knowledge about basic procedures and terminology, as well as, necessary rules about maintaining computer equipment, (ii) a knowledge about using computer-based instruction, (iii) a knowledge about technology influence on carriers, society and culture, and finally (iv) computer programming.

In this paper we particularly emphasize the importance of computer programming. We believe that students have to acquire computer programming knowledge, skills and capabilities already in primary education. Educational contents that would enable this process, begin with problem solving formalism (in individual and group work), and proceed with programming paradigm using some appropriate programming languages as LOGO, Basic or Pascal. Our consideration is directed toward goals presented in Croatian Educational National Standard (CENS) for elementary school computer science. This standard was carried out in Republic of Croatia during 2004-2005. Committee for developing CENS for elementary school computer science [4] proposes that in this domain, among all, assumptions for acquiring problem-solving and programming competency, using, so called, "turtle" programming language or some procedural programming language, have to be realized. Besides, long experience of working with our students enrolled in basic computer programming courses (Introduction to computing and Programming 1), undoubtedly imply many difficulties that have occurred due to late beginning of learning computer programming. Sometimes teachers wonder which programming language to use as a starting point. However, these dilemmas do not concern us only. Developed western countries consider these issues too. Our opinion is that choice of programming language is not the most important. More relevant are an early start of computer programming during elementary education, mastering algorithm thinking manner and performing computer programs.

Motivated by this problem, we have started developing of prototype model of computerized tutor for learning and teaching programming. In this paper we propose a concept of the model, as well as, the architecture prototype. We hope that application of this model can greatly improve process of learning and teaching of computer programming. It should realize assumptions for students and teachers to solve mentioned problems in an individualized computer-based approach combined with traditional education. Development of the computerized tutor for

programming learning and teaching is based on age-long research and development of the Tutor–Expert System, a hypermedia authoring shell model for building intelligent tutoring systems [11]. In second chapter we state our experiences in research and development of intelligent tutoring model. Finally third chapter contains concept of prototype model of tutoring system for programming overview.

2. Background and previous work

Intelligent tutoring systems (ITS) are generation of computer systems aimed for the support and improvement of learning and teaching process in certain domain knowledge, respecting individuality of the learner as in traditional "one-to-one" tutoring. ITS show the best results in evaluating students' achievement when being compared to existing technological learning and teaching process support, both traditional and individual [7]. As the need to cover a variety of different domains have arisen since, instead of having a number of specialized ITSs for the domains of interest, "ITS generators" were developed, which are usually denoted as authoring shells (Barton, 1995). These shells can be "programmed" for a particular domain by modifying the domain knowledge thus resulting in a specific ITS. According to ITS traditional modular architecture [5] and the idea of the cybernetic model of the system [9][3] we have developed a model of intelligent authoring shell called TEx-Sys. The first implementation of this model is on-site TEx-Sys (1992-2001), and after that followed research, development and implementation system based on dynamic Web document (1999-2003, Distributed Tutor-Expert System, DTEx-Sys) [10] and system based on Web services (2003.-2005. eXtended Tutor-Expert System) (Fig. 1.) [12].

Prototype tests on the TEx-Sys, DTEx-Sys and xTEx-Sys have been carried out with students of different chronological ages (from primary education to academic level) on designed knowledge bases and on courses educational contents established using those knowledge bases. Results of those tests are advantageous, according to surveys, and implemented and deployed software satisfies functionalities and actors' demands. Domain knowledge bases design has enabled expert environment testing, while course educational content design has enabled teacher environment testing. Roles of experts and teachers were played by teachers and students from University of Split and University of Zagreb. Student environment has been tested to determine system's usability and students' achievements in learning and teaching process. In second half of year 2004. and during 2005. we have realized 14 tests with 36 domain knowledge bases, along with 245 students from three primary schools in Split, 25 students from one high school in Šibenik and 344 students from faculty [12].

Within TEx-Sys model, in all implemented systems, knowledge is represented by semantic networks with frames [13]. Primary elements of domain knowledge are nodes and links. Nodes with frames and structural attributes are used to represent domain knowledge objects and they are semantically connected using links. Beside nodes and links, the model supports properties and frames (attributes and respective values), along with property inheritance. The model relies heavily on modern supporting technologies, such as multimedia, with the following structure attributes: picture, animation, slides, URL addresses and hypertext descriptions. We present new TEx-Sys model instance, prototype model of tutoring system for programming, in next chapter.

3. Concept of Prototype Model of Tutoring System for Programming

Paradigm of intelligent tutoring system for

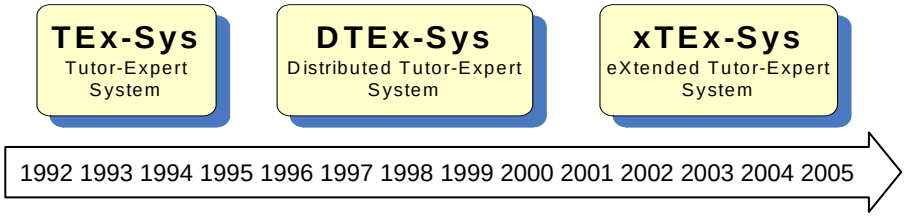

Figure 1. TEX-Sys model time line

programming teaching is based on ideal student model [1]. The ideal student finds an optimal solution for each programming task. Tutoring system tries to get programmer novice knowledge and skill, as close as possible, to the knowledge and skill of ideal student. Toward this goal tutoring system:

- Defines programming tasks.
- Determines from student's behavior what he knows and what confusions or bugs he has.
- Interrupts student's work when his choice leads to errors.
- Helps on student's demand.
- Verifies correctness of student's program.
- Estimates student's knowledge and skill and advices what he should do and when he should advance to new material.

3.1. Architecture

Although components of our system are typical for programming tutoring systems, we have designed graphical-model translators, compilers, and virtual processor of intermediate code.

Database holds program tasks, domain expert solution of these tasks, bug catalog, and history of student's learning process.

Graphical user interface is split to student-work-area and communication area. Student-work-area contains toolbox, flowchart and object-model graphical editor, source code editor, and debugging windows.

Translators are capable to build the source code from object-model and flowchart symbols, and compile source to intermediate code. They support reverse engineering as well.

Virtual processor interprets intermediate code, supports breakpoints and step by step execution. Human-tutor may customize virtual processor, so that it runs intermediate code on the way appropriate to specific course.

Student's program tester performs dynamic testing using predefined test data.

The special design of translators and virtual processor supports very important functionalities:

- Visualization of program execution.
- Automatic testing of student's program.
- Definition of programming language tailored up to teaching goal of specific course.

3.2. Programmer novice and ideal student

Tutoring system detects student's errors in each step of problem-solving, program coding and testing, comparing his behavior with behavior of ideal student in given context.

The ideal student approaches to the given programming task as follow [8]:

- Figures out the goal of program, i.e. result of program execution.
- Understands instructions of programming language.
- Decomposes given task to the simpler steps, so that each step can be coded with single programming language instruction.
- Describes the problem solution by program modeling symbols, for example by flowchart.
- Reworks solution in code of programming language.
- Locates and corrects syntactical errors.
- Identifies and prunes logical errors.
- Verifies that task requirement is achieved by program execution.

Real student, programmer novice, can do deviation in each of steps:

- Choose wrong solution.
- Unacceptable push down problem to next steps, for example, nesting function or procedure calls, while no one of them solves the problem at all.
- Turn to blind direction, where we mean error free direction that can never reach task goal.
- Choose error free solution, but such that it is not optimal or significantly decline from good programming style.
- Student may be in dept about appropriate solution.

System interrupts student's work on error-type sensitive manner, and estimated student knowledge, as well as his learning progress.

3.3. System interventions

System intervention is a help initialized by detected declination of teaching student knowledge from ideal student model. We distinguish interventions by error-detected-system-action time interval and by level of help provided.

Immediate interrupt is appropriate to low level lessons and poorer students.

Delayed interrupt is applicable to advanced programming tasks and best students, leaving them opportunity to identify and correct errors themselves.

Tutoring system help depends of estimated student's learning progress and may be provided (i) writing solution of current step directly to the student's work area, (ii) writing template of correct solution and requesting student to complete it, or (iii) explaining correct solution verbally.

Let us illustrate system actions solving very basic programming task step by step. System starts finding programming task appropriate to the student's knowledge grade and estimated progress of his learning process. Communication between student and tutoring system takes place on graphical user interface.

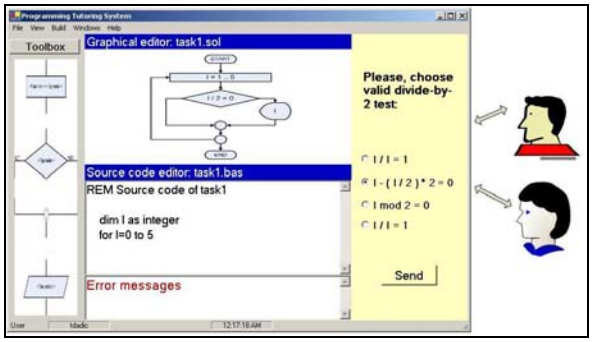

Figure 2. Tutoring system graphical interface

We are showing communication as textual dialog, where prompt *T* stays for requests of tutoring system, and *S* for student's responses.

T> Programming task: Write on standard output device even integers between 1 and 5!

T> Do you understand the task?

S> No, I don't.

System finds that understanding of terminology "standard output device" was the most frequent problem at this stage. An explanation follows immediately:

T> Standard output device is screen of your computer by default. You write results of program computation to the device using keyword print. Do you understand the programming task now?

S> Yes, I understand.

T> Please, find algorithm solving task problem and create flowchart!

Student chooses flowchart elements from toolbox, drags them to the graphical editor area constructing his problem solution, shown on Fig.3.

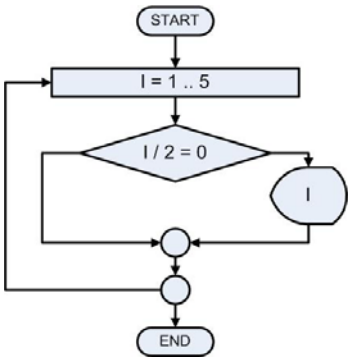

Figure 3. Student's flowchart

T> Flowchart is correct. Now write source code, mapping each flowchart element to a programming language keyword!

Student has written two statements of his program, and couldn't continue on his own.

S>dim I as integer

S>for I=1 to 5

As student's time to advance to the next statement elapsed, tutoring system interrupts his work offering help:

T>Do you want automatic coding based on your flowchart?

Student accepts help, and source code is completed in editor.

```
S> dim I as integer
S> for I=1 to 5
T>     if I / 2 = 0 then
T>           print I
T>      end if
T> next I
```

Outputs of program above, coded in chosen language are: 2, 3, 4, 5.

Program is considered incorrect as system finds differences between these outputs and ideal student's program outputs. Comparing standardized source code [14] of two solutions, system locates student's if-condition mistaken. Student is asked to choose valid test:

T>Choose valid divided-by-2 test:

a) I / I = 1

*b) I – (I / 2) * 2 = 0*

c) I mod 2 = 0

d) I mod 2 = 2

Student chooses option b). Obviously, his knowledge of mod operator is poor and system advices him to study related lessons.

Finally, student corrects error and his program is found correct afterwards.

We find that classification of detected error is not trivial process. In next section we deal with classes of typical errors in students' programs.

3.4. Error classes in students' programs

As automatic identification of errors is very important and complicated tutoring system process, it is helpful to categorize possible errors.

Algorithmic error category contains incorrect problem-solving solutions of given task, let us say, thinking bugs. Unclean and hard to follow symbolic presentation belongs to this category as well.

Programming errors are caused by improper use of programming language structures. That is, correct algorithm is coded on wrong way.

Furthermore, we distinguish classes of programming errors:

- Syntax errors.
- Improper usage of data types.
- Wrong use of programming language structures to control program flow.
- Logical errors caused by coding mistakes.
- Deviation of optimal code and good programming style.

System has to detect existence of errors in student's program, identify their sources and help to prune them out. There in our tutoring system, correct ideal student's solution (identical to domain expert's solution) exists for each programming task. Errors detection and identification are achieved comparing student's solution against ideal student's solution. Syntax errors and non-optimal code are well known topics from compilers theory. Since system goal is didactic response to errors, we deal with these error classes as well.

3.5. Correctness of student's program

Correctness of student's program is verified by automatic dynamic testing. It is based on fact, that each program transforms its inputs to outputs:

$$Y = f(X), \qquad (1)$$

where X stays for set of program inputs, Y for set of outputs and f represents program itself.

Program may receive inputs from standard input device (keyboard), from file or command line parameters. In some programming languages (i.e. BASIC) inputs are read from built-in code *data* statements. It isn't uncommon practice to provide function and procedure parameters as hard coded values.

If X_S and Y_S stay for sets of inputs and outputs of student's program f_S, and X_T, Y_T for inputs and outputs of ideal student's program f_T, we can write:

$$Y_S = f_S(X_S) \qquad (2)$$

$$Y_T = f_T(X_T) \qquad (3)$$

Ideal student's program is previously tested and assumed correct. Our intermediate code processor is developed to run student's program supplied by archived (in database) ideal student's inputs. If student's program transforms ideal student's inputs, to outputs equal to ideal student's outputs, we find it correct.

$$f_S(X_T)=f_T(X_T) \rightarrow C_V(f_S), \qquad (4)$$

where $C_V(f_S)$ represents correctness of student's program.

During program execution, virtual processor saves to memory of virtual machine all inputs and outputs. So called program context switching, provides execution of student's program behind scene, using ideal student's inputs and comparison of its outputs against expected results.

In order to test in-program built functions and procedures, program inherits interface defined by task. Program has to implement interface or compilation error arises. Doing on this way, function calling names, number and data type of parameters, as well as returning types are equal in all programs inheriting the same interface. Freedom in parameters naming is left to programmers. It is easy later to call student's function or procedure with ideal student's parameters, and compare returning data values.

4. Conclusion

Intelligent Tutoring Systems (ITS) have to simulate the procedure of human teacher in given context. The same procedure has to be applied to the systems teaching computer programming as well. This goal can be achieved performing syntactic and semantic analyses of a student's source program code and arising appropriate actions based on errors found.

In this paper, we present intelligent teaching model of Tutor Expert System (TEx-Sys) extended by functionalities of programming teaching. Systems derived from Tex-Sys model

facilitate teaching from arbitrary chosen learning fields. Following that approach, we have extended TEx-Sys model with capability to support teaching of from-menu- chosen programming language defined by related grammar.

5. Acknowledgements

This work has been carried out within projects 0177110 *Computational and didactical aspects of intelligent authoring tools in education*, and TP-02/0177-01 *Web oriented intelligent hypermedial authoring shell*, funded by the Ministry of Science and Technology of the Republic of Croatia.

6. References

[1] Anderson JR, Reiser BJ. The Lisp Tutor. Byte; 1985. http://act-r.psy.cmu.edu/papers/113/TheLISPTutor.pdf

[2] Barton M. Shells for Intelligent Tutoring Systems. 7th World Conference on Artificial Intelligence in Education, Washington DC, USA, August 16-19, 1995. http://www.pitt.edu/~al/aied/barton.html.

[3] Božičević J. Fundamentals of Automatic Control, Part I – System Approach and Control, 10th Edition, Zagreb: Školska knjiga; 1980 (in Croatian).

[4] Budin L. at all. Education in the Field of Information and Communication Technology in Elementary School: Croatian National Education Standard, Ministry of Science, Education and Sport, Zagreb; 2005.

[5] Burns HL, Capps CG. Foundations of Intelligent Tutoring Systems: An Introduction In: Polson MC, Richardson JJ, editors, Foundations of Intelligent Tutoring Systems, Lawrence Erlbaum Associates Publishers; 1988. p. 1-18

[6] Eisenberg, M. B., Johnson, D. (2003), Learning and Teaching Information Technology Computer Skills in Context, US Federal government, http://www.libraryinstruction.com/info-tech.html

[7] Fletcher JD. Evidence for Learning From Technology-Assisted Instruction in edition by H. F. O'Neil, Jr., R. S. Perez: Technology Applications in Education - A Learning View, Lawrence Erlbaum Ass. Publishers, Mahwab, New Jersey; 2003.

[8] Hovell K. First Computer Languages. Journal of Computing Sciences in Colleges, vol. 18, issue 4; 2003. p. 317-331

[9] Pask G. A cybernetic model of concept learning, Proceedings of 3rd, Congress International. Assoc. Cybernetics, Namur 1961; Gauthier-Villars.

[10] Rosić M., Establishing of Distance Education Systems within the Information Infrastructure, M.Sc. Thesis, Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, Croatia; 2000 (in Croatian).

[11] Stankov S., Isomorphic Model of the System as the Basis of Teaching Control Principles in an Intelligent Tutoring System, PhD Diss., Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture, University of Split, Split, Croatia; 1997. (in Croatian).

[12] Stankov S., Principal Investigating Project TP-02/0177-01 Web oriented intelligent hypermedial authoring shell, Ministry of Science and Technology of the Republic of Croatia, 2003-2005.

[13] Stankov S, Glavinić V, Rosić M. On Knowledge Representation in an Intelligent Tutoring System in Proceedings of 4th IEEE International Conference on Intelligent Engineering Systems – INES'2000, Portoroz, Slovenia; 2000. p.17-19.

[14] Xu S, Chee Y.S. Transformation-based Diagnosis of Student Programs for Programming Tutoring Systems. IEEE Transactions on Software Engineering, vol. 29, no. 4; 2003. p.360-384. http://www.ietf.org/internet-drafts/draft-ietf-avt-rtp-new-10.txt [10/31/2001]