A task-oriented environment for teaching and learning algorithms — uml project and its implementation

S. Nenkov, H. Iliev

Algorithms are frequently taught procedural knowledge units in computer and humanitary education. Both design and implementation issues of a Task Oriented Environment for Constructing Algorithms are discussed in the paper. Its project mainly from the teacher's point of view is presented by means of use case, activity, and sequence diagrams. A subject-independent architecture is proposed consisting of standard and specialized tools, task base and students' models integrated in a data base. Implementation of the teacher's tool prototype in DELPHI 7.0 programming environment is described and illustrated by means of screenshots.

Introduction

The algorithm theory and practice is an old and important branch in the computer and humanitary sciences. Algorithms (computational or for decision making support) are referred to as abstract procedural knowledge units, describing the step-by-step solving a given class of problems. An effective and efficient computational algorithm leads to a reliable and effective program implementation [1],[3]. Description of an algorithm knowledge unit presents in an encoded, compressed, and understandable form the semantics of the corresponding data processing. Its implementation in the computer memory by means of frames, semantic networks, rules, and even their combinations is called algorithm knowledge representation.

Like other abstract procedural units such as structural schemes, Petri nets, state machines, and so on, the algorithms have their own statics and dynamics, taught in different styles: textual (natural description, pseudo-code, and so on), graphical (flowcharts, activity diagrams, and so on), tables for decision making, and even in a mixed style. First the static, e.g. the structure of the flowcharts has to be taught as more simple then its dynamics, e.g. interpretation. Empirical studies with different systems for algorithm visualization and animation [2] have confirmed that they enhance the angorithm skills acquisition due to learner's activity, friendly interface, interactive solving, color coding, intelligent support, and so on. The main requirements for intelligent teaching in the area of algorithms are formulated by Robling & Naps [4]. They are developers of DAPHNIS cited as the first intelligent language-independent system with a declarative method for visualization with algorithm animation, based on the data stream tracing.

For several years Zheliazkova's research group at the Rousse University has been working on development, implementation, and studying an intelligent and adaptive Task-Oriented Environment for Teaching and Learning Algorithms [6]. The individual user categories are: administrator, task author, instructor, learning and examined student.

The present paper focuses on the project and implementation of a Task-Oriented Environment for Teaching and learning Algorithms (TOETLA). It is organized as follows. The project is presented in the next section by three types of UML diagrams (use case, activity, and sequence). The third section deals with the TOETLA architecture, which is platform-, algorithm- and language-independent. The implementation in the DELPHI programming environment and user interface with several screenshots are presented in the next section. The conclusion outlines the paper contributions and the authors' intentions for the near future.

UML project of the environment

The multi-user use case diagram of the TOETLA is given on fig. 1. It shows the functional requirements of the different user categories with the following priority, e. g. system administrator, task author, instructor, learning and examined students. After login with user name and password, the environment gives the user rights to operate the environment specific for the corresponding category. The task author is allowed to create, change, delete, and assess a task for flowchart constructing. A task description includes: informal formulation, key directives for the instructor's intervention, author's performance as an expert, and the computed values of the task pedagogical parameters. The instructor is responsible for planning, organization, and monitoring a test-like exercise of a subgroup of students. Under the instructor's pedagogical knowledge the learning student can see the author's task and relevant lecture material in the form of a context-dependent help. The examined student can see only the task formulation and parameters to perform flowchart constructing tasks for a fixed time.

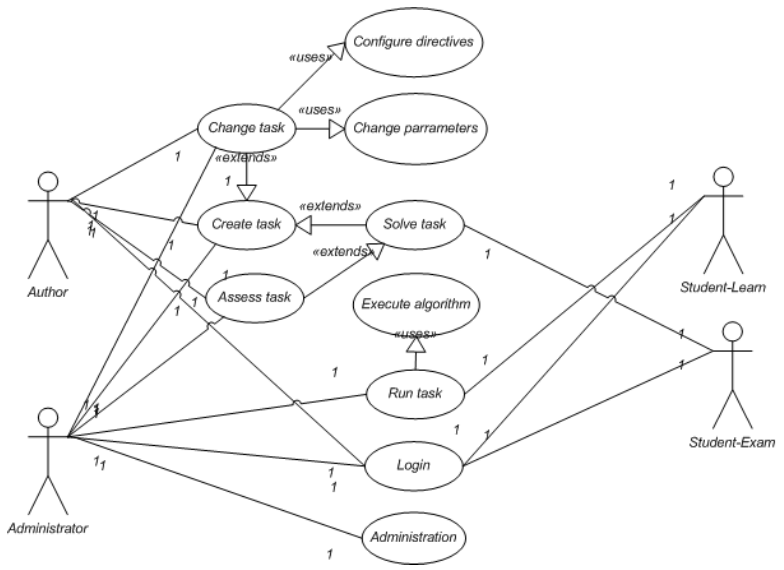


Figure 1. Use Case diagram of the TOETLA.

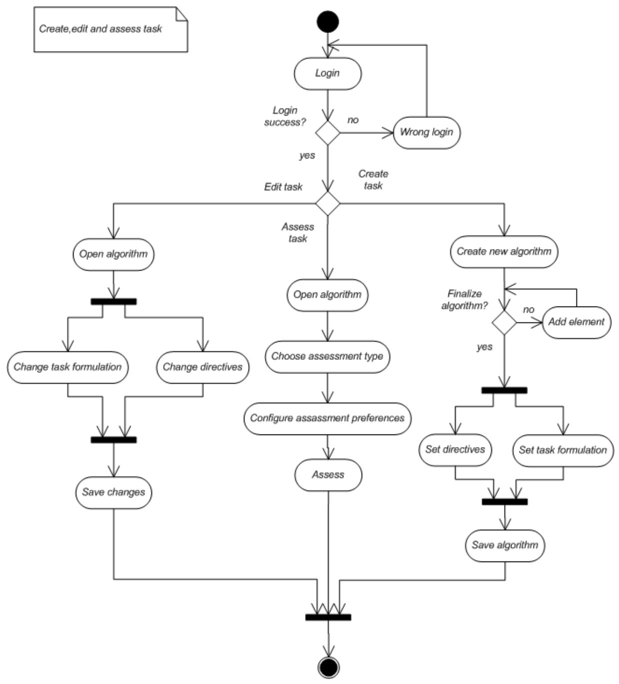


Figure 2. Activity diagram of the task author.

The task author's activity diagram is shown on fig. 2. He/she can use local for the task key directives to allow/permit the learning/examined student, for example, to print the author's performance, save the student's performance, see the author's algorithm description, and so on. Other recommendations concerns the planned time for the task performance, kind of the task assessment, assessment scale, and so on. The instructor is accessed to a homogeneous task base

(TB) in order to preview it and select appropriate tasks for each student/exercise. He/she can choose the number of its tasks, add/remove/delete an exercise task and change its parameters. The student's activity diagram for the learning mode is shown on fig. 3. The learning student is free to choose any task for flowchart construction from the exercise planned by the instructor. He/she is authorized only to open a couple of *bmp* and *.alg* file, e.g. has no rights for their edition. An .alg file presents a text file in a specialized script language for practical skills description [6]. The set of values of the input variables prepared by the author have to be chosen for passing all paths during the flowchart interpretation. The *TOETLA* offers two modes of interpretation respectively tracing to see step by step the way in which input values are transformed to their final values. The learning student also is allowed in a way similar to the author to construct his/her own task solution, to see its automatically computed parameters and time planned for the task solution. Before interpretation he/she enters input variables' values he/she wishes during construction the flowchart structural correctness is checked.

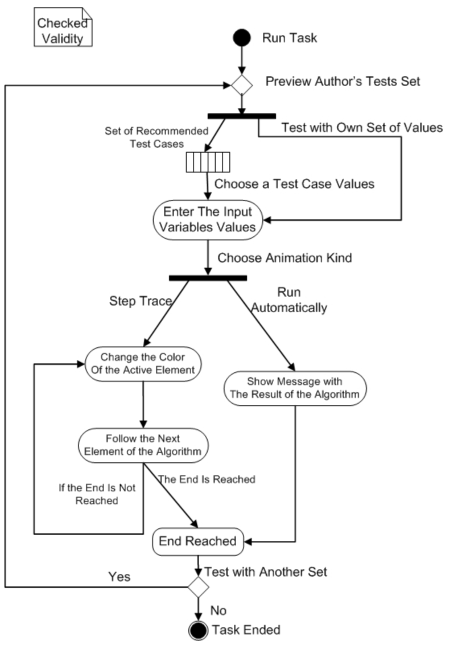

Checked Validity
Run Task
Preview Author's Tests Set
Set of Recommended Test Cases
Test with Own Set of Values
Choose a Test Case Values
Enter The Input Variables Values
Choose Animation Kind
Step Trace
Run Automatically
Change the Color Of the Active Element
Show Message with The Result of the Algorithm
Follow the Next Element of the Algorithm
If the End Is Not Reached
The End Is Reached
End Reached
Test with Another Set
Yes
No
Task Ended

Figure 3. The student's actions in the trainee mode.

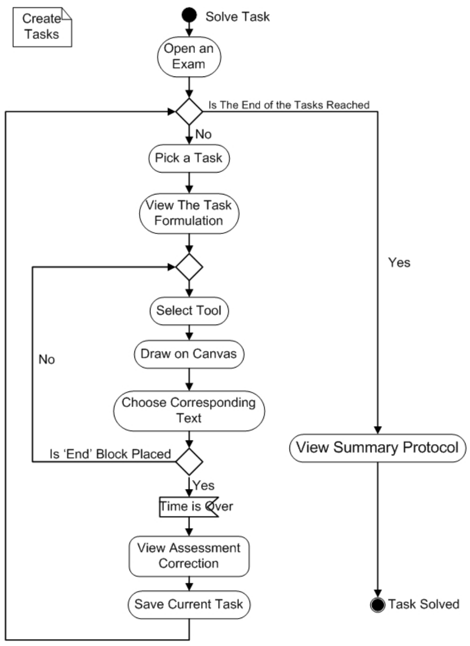


Figure 4. The student's actions in the examinee mode.

The instructor plans the exercise in accordance with the exercise goal, author's recommendations, and his/her own preferences. In order to facilitate and accelerate preparing a set of different and equivalent tasks for different students in a subgroup the instructor has to prepare .ecs file with syntax given in the following paper changing its automatically computed parameters as well as the A's key directives. After the instructor finishes the exercise preparing the exercise parameters tougher with the intervals for the assessment scale are automatically computed on the base of the tasks parameters.

The sequence diagrams on fig. 5 corresponds the situation when the author and instructor are teamworking to create and change an exercise. It also corresponds to the teamwork but in a long time process depicted as a vertical rectangle. On the top of the figure the environment's units for the process implementation are shown. The solid arrows are used for the users (author and instructor), and the dash ones for the environment units' reactions.

The architecture of the TOETLA

This architecture (fig. 6) slightly differs from architectures of other Task-Oriented Environments,

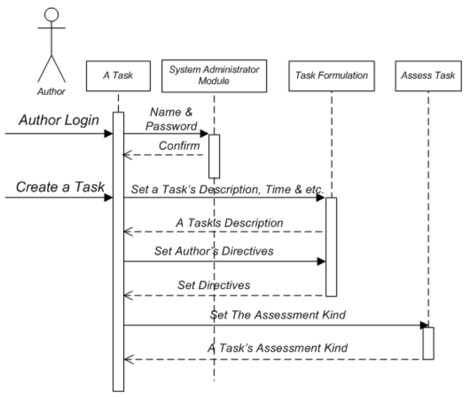


Figure 5. Author's sequence diagram.

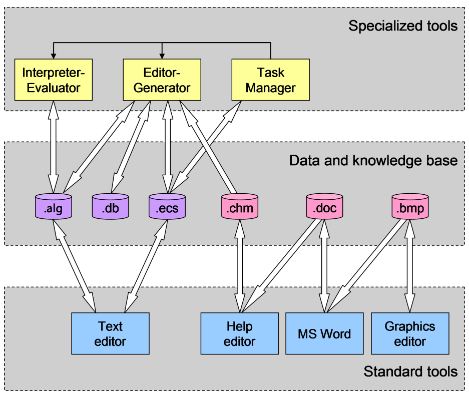


Figure 6. The architecture of the TOETLA.

Fig. 5. Author's sequence diagram Fig. 6. The architecture of the TOETLA developed by Zheliazkova's research group. The architecture consists of standard editors and specialized tools supporting a common TB integrated in the environment's data base. The standard editors are four: a word processor (e.g. MS WORD), a text editor (e.g. Notepad), a graphics editor (e.g. Paint), and a help editor (e. g. MS HTML Help Workshop). The lecture material in the Word document is preliminary prepared by the course lecturer and converted to an .hlp file with a context-dependent help. The demonstrated schemes, diagrams, and so on in the form of images (.bmp, jpg, gif, and so on) are prepared by means of the graphics editor and then imported into the document.

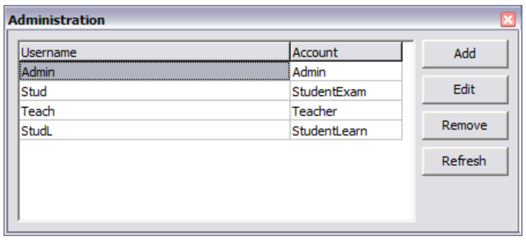
The specialized tools are three, called respectively program-generator, interpreter-evaluator, and task manager. The author, instructor, and student operate these tools though a highly interactive and intuitive user interface. By means of the first tool the author's and students' structural knowledge is extracted and the couple of files are generated. An .alg file in the AlgolScript language is seen as physically separated subprogram describing the author's structural knowledge for a given flowchart. Additionally, the tool extracts the instructor to his/her pedagogical knowledge for a given exercise and stored in a standard .ecs file in the ExerciseScript language. Both files can be open and edited by means of the standard text editor to avoid the slow process of editing-generating if needed equivalent tasks for different students.

The interpreter-evaluator parses the .alg script to compute automatically the task parameters (knowledge volume, degree of difficulty, planned time, and so on) and stored in the TB. The tool also provides diagnostics of the student's knowledge refreshing his/her short-term relational model, also integrated in the DB. After a constructing task is performed by the student, the tool analyses his/her results relatively to the author's ones.

Besides the task sorting and presentation of their formulations, task manager interprets the underlying in the .ecs file local teaching strategy, e.g. the short-term plan for performing the exercise. More precisely the tool interprets the key directives, fulfills the missing and corrects the wrong knowledge, as well as refresh the student's model. After a session finishes the parameters of the exercise (knowledge volume, time undertaken, rate of learning, and degree of difficulty) are accumulated in the TB as statistical parameters.

Implementation the WINDOWS-based Prototype

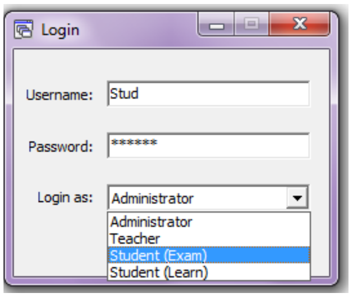
For implementation of the WINDOWS-based environment DELPHI programming environment had been preferred over other ones such as VISUAL BASIC and VISUAL $C++$. Zheliazkova's group experience in implementation of windows-based task-oriented environments has shown that Delphi programming allows creating such kinds of applications easy and faster from one or two programmers. Several reasons can be pointed out for this choice, namely: its visual



Administration

Username	Account
Admin	Admin
Stud	StudentExam
Teach	Teacher
StudL	StudentLearn

Figure 7. The window for the administrator.



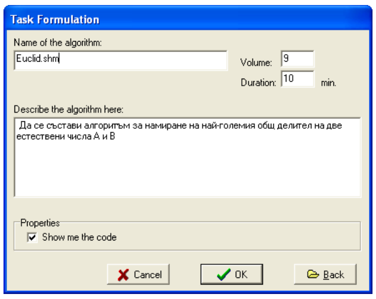
Login

Username: Stud

Password: ******

Login as: Administrator

Figure 8. The window for user login.



Task Formulation

Name of the algorithm: Euclid.shm

Volume: 9

Duration: 10 min.

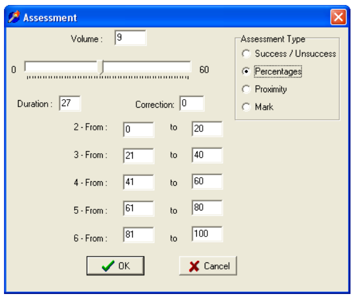
Describe the algorithm here:

Да се състави алгоритъм за намиране на най-големия общ делител на две естествени числа A и B

Properties

[✓] Show me the code

Figure 9. The window for the administrator.



Assessment

Volume : 9

0 … 60

Duration : 27   Correction: 0

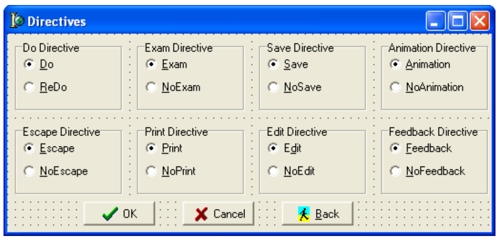
2 - From :	0	to	20
3 - From :	21	to	40
4 - From :	41	to	60
5 - From :	61	to	80
6 - From :	81	to	100

Assessment Type: Success / Unsuccess [✓ Percentages] Proximity Mark

Figure 10. The window for user login.

component library (VCL) is rich and power, visual programming is easy and intuitive, different databases are supported and so on. The screenshots presenting on fig. 7-12 confirm this statement without any comments. It is supposed that the student is login trough the user name and password given by the system administrator. The selection of the user category also is obligatory from a popup menu. The environment itself is loading pressing the button Login.

Bellow the interaction of the examined student with the main form (fig. 12) is commented. The main window of the application contains menu with commands, toolbars, client area, as well as a panel with the cursor current coordinates. The type of the task, e.g. for construction (*Construct*), interpretation (*Interprete*), and testing (*Test*) is chosen from the menu-element *Tasks*. In the first case when a new or existing task for constructing with the command *File/Open* a corresponding couple of files (.alg and .bmp) is open. The first one contains the subprogram script, and the second one — the flowchart image. With menu command *File/New* a new couple is created. In both cases a dialog form shown on fig. 9 apeared. It contains: memo field for the task formulation in a free text format, edit field for the file name, knowledge volume (*Volume*), planned/expected time (*Duration*), as well as a check-box for show/hide an additional window with the script. Three buttons of the form serve for: *Cancel, Back* and *OK* confirms the contents and move to the next form (fig. 9). Though the groups of radio-buttons the choice of one of all possible combinations of key directives it allows the author/instructor to program the virtual intervention when the student performs the current task. The scheme (DO—REDO); her redaction (EDIT—NOEDIT); suspending of her creation and construction (ESCAPE—NOESCAPE); printing the pair files (PRINT—NOPRINT); saving the files to directory or device (SAVE—NOSAVE); assessment of the proximity degree between two block schemes (ASSESS—NOASSESS). This form has the same buttons for — copying (*Copy*), deleting (*Cut*), and pasting (*Paste*) of the clipboard contents at a screen position pointed out by the cursor. The construction of the block scheme starts after returning to the main form. (fig. 3). Each command of the Tools menu item serves for drawing a given graphical primitives: *RoundRect, Rect, Romb, Para, Line, Arrow* and *Text*. For acceleration of the access in the toolbars a corresponding button is added. The command *View/Toolbars* hides/shows the toolbars. The drag and drop techniques also supports the user needs. Each command from the *Edit* menu item has a sense of copying


Directives

Do Directive: [✓ Do] ReDo
Exam Directive: [✓ Exam] NoExam
Save Directive: [✓ Save] NoSave
Animation Directive: [✓ Animation] NoAnimation
Escape Directive: [✓ Escape] NoEscape
Print Directive: [✓ Print] NoPrint
Edit Directive: [✓ Edit] NoEdit
Feedback Directive: [✓ Feedback] NoFeedback

Figure 11. The key directives.

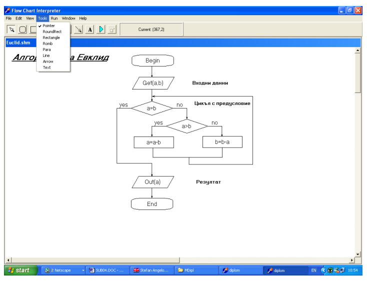


Figure 12. Euclid's algorithm flowchart.

If the T want to see/hide the generated script, he/she can do this choosing the command *View/Script*. The subprogram is syntactically and semantically correct, so it's editing with a standard text editor like Notepad is recommended only when the corrections are unsubstantial. For example, when different versions of one and the same flowchart he/she needs. The choice one of the commands *File/Save* and *File/Save As* stores the subprogram and the flowchart under one and the same name with different extensions. With command *Task/Interprete* the sets of the input variables are entered for algorithm interpretation. They also are added to the end of the subprogram. Before to point out the sequence of the tasks for a given exercise for a given student, the instructor has to choose the command *Tasks/Test*. As a result the dialog form (not shown here) with the edit fields: for user name, password, subject, and topic. The next form (not shown here) ensures adding, confirming, canceling, and deleting an exercise task. If necessary the T can change the criteria for assessment, e.g. the type of the scale, *SUCCESS/FAILURE, PERCENTAGE, PROXIMITY* or *MARK* number of the maximal points its duration, as well as the constant for time correction. In such way the environment adapt to the needs and preferences of the instructor.

The command *Task/Interprete* stars the flowchart interpretation with the given by the author set of the input variables values. The animation style leads to a deeper awareness of the data stream under the influence of the separated operators and to effective execution of a given operation block. The flexible control of the interpretation is done with the commands *Automatically, Manually, Step Over, Program Reset* from the menu item *Run*.

The student can choose between two alternatives for control: automatically (command *Automatically*) and manually (command *Manually*) and pressing the functional key F9). The final result of the algorithm interpretation is appeared in an additional window (fig. 8). The transition from one block to another is done repeating the command *Run/Step Over* or pressing the key *F8*. With command *Run/Program Reset* or the key combination *Ctrl+F9* the tracing is stopped. After the exercise performance or the planned time for the exercise is over the environment generates a final report. There is a possibility for the student to execute the algorithm with his/her own input data.

Conclusions and intentions

Besides algorithm-independent and intelligent the reported environment is adaptive to both author and instructor, supporting their teamwork. Its homogeneous task base can be easy extended, allows the algorithmic knowledge units reusing, that increases both author and instructor's productivity. Structural knowledge diagnostics and assessing without checking the text syntax allows teaching and learning not only computational but also decision making algorithms. The more simple interpretation of the decision making algorithms that is reduced to enter yes/no answer from the keyboard will be implemented in the near future.

The intention is also to integrate the TOETLA into a large-scale Environment for Individualized Planned Teaching different courses which web-based technology is compatibile with the word documents and excel tables. The common well developed module for administration and communication is available for all participants in the course teaching including the author of the lecture material. In such a way the compatibility with the windows-based technology will be reached, and a possibility for planning and performance of heterogeneous exercises will be ensured.

References

[1] B. Crescencio, M.J. Marcelino, A. Gomes, M. Esteves, A.J. Mendes, (2005), Integrating Educational Tools for Collaborative Computer Programming Learning, Journal of Universal Computer Science, Vol. 11, No. 9, pp. 1505-1517.

[2] Korhonen A., Malmi L., Silvasti P. (2003). TRAKLA2: a Framework for Automatically Assessed Visual Algorithm Simulation Exercises. Proceedings of the 3rd Finnish/Baltic Sea Conference on Computer Science Education, Koli, Finland, pp. 48-56.

[3] Maers B.A., Taxonomies of Visual Programming and Program Visualization, Journal of Visual Languages and Computing, Vol. 1, 1990, pp. 97-123.

[4] Robing G., Naps T.L., A test Bed for Pedagogical Requirements in Algotithm Visualization, Proceedings of the 7th Annual SIGGSE/SIGCUE Conference on Innovation and Technology in Computer Science Education (ITiCS'02), Arhus, Denmark, June, 2002.

[5] Viere F., Van De, La Simulation et l'Animation Modulaire d'Algorithmes en Langage Object. Rowdrawadoktoska. Universitte des Sciences et Technologies, Lille, Francja, 1997.

[6] Zheliazkova I., Atanasova G., Computer-Aided Teaching and Learning Algorithms, Proceedings of the 15th Annual Conference on Innovation in Education for Electrical and Information Engineering, Sofia, 2004, pp. 49-58.

Authors

Stoyan Nenkov — Student, Faculty of Electrical Engineering Electronics and Automation, Rousse University "Angel Kunchev", Rousse, Bulgaria; E-mail: *s093211@stud.uni-ruse.bg*

Hristo Iliev — bachelor degree student, Faculty of Electrical Engineering Electronics and Automation, Rousse University "Angel Kunchev", Rousse, Bulgaria; E-mail: *author1@author.com*