A Visual Language For Algorithm Knowledge Representation

Irina Zheliazkova, Galina Atanasova

Abstract: *Task-oriented design environments (TODEs) are an alternative of domain-oriented environments that support both self-directed learning and individualized planned teaching. One of the key problems of the design and implementation a TODE for algorithms is definition of a visual language for algorithm knowledge representation. The paper presents an exhaustive definition of such a language.*

Keywords: *task-oriented design environment, algorithm, language, internal, visual*

1. INTRODUCTION

Lifelong learning has emerged as one of the main challenges for the worldwide information society of the future and has been given considerable international attention by the European Community and by UNESCO. A lifelong learning perspective implies that schools and universities need to prepare learners to engage in self-directed learning (SDL), e. g. what they will do in their professional and private lives outsides of the classroom [3]. Till now Domain-oriented Design Environments (DODEs) and Task-Oriented Design Environments (TODEs) are two main alternatives of SDL. Both DODEs and TODEs are designed as opened systems with economy of educational knowledge. The main differences in comparing TODEs and DODEs are:

- TODEs are based on generation of programs in a task-oriented language;
- They can be used not only for SDL but also for individual planned teaching;
- The users of TODEs can be not only the developers but also learners and teachers.

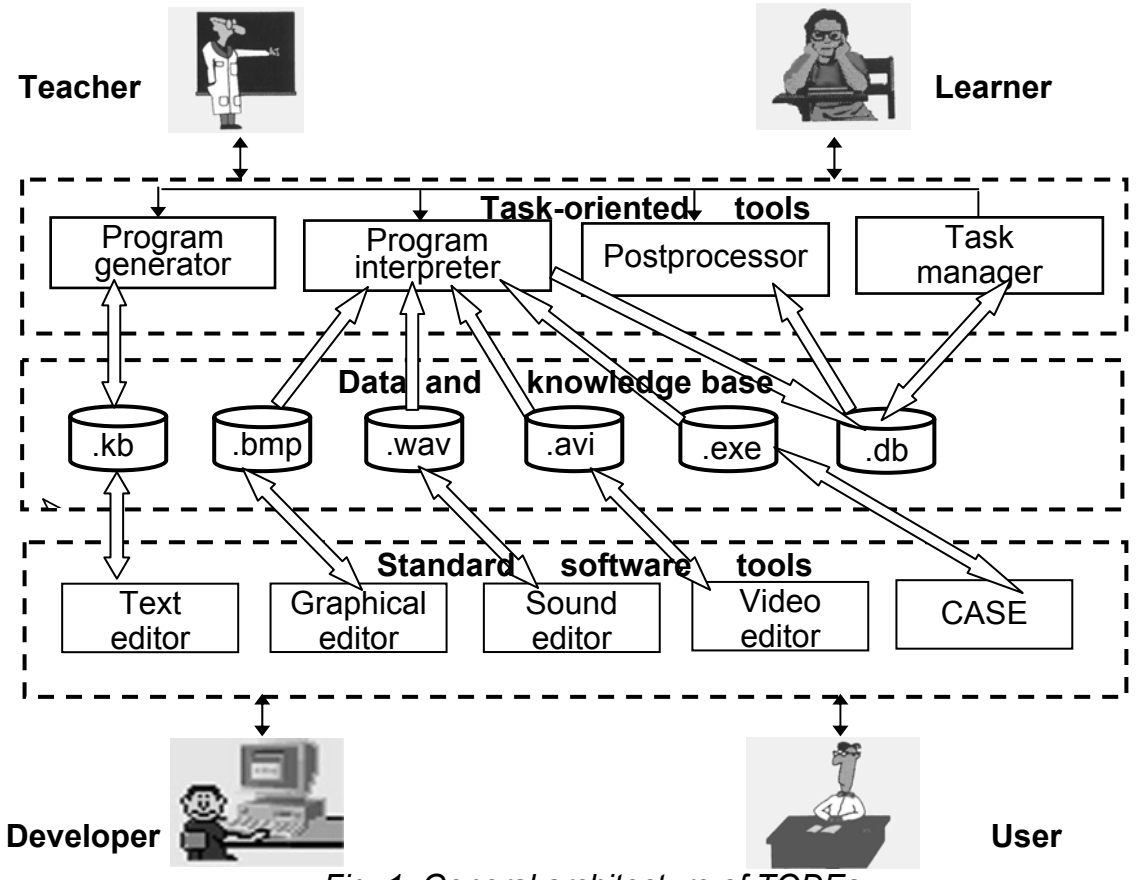

Fig. 1. General architecture of TODEs

2. GENERAL ARCHITECTURE OF TODEs

This architecture (fig. 1) is domain-independent and oriented to a large class of tasks. The graphics and multimedia necessary for the task performance by the learner are preliminary prepared by the teacher with standard tools, e.g. graphical editor, video editor, sound editor, text editor, creating the corresponding file types. Data base files (*.db*) could also be created by the teacher. In a TODE for training in dynamic system, for example, they can store the parameter dimensions or the values of the measured parameters of a real system [4]. Through a very interactive and friendly user interface the program generator creates a program representing the learner's knowledge about a given training task in an internal language. This syntactically and semantically correct program is stored in a standard text file with an extension *.kb*. The program execution is ensured by the program interpreter which extracts the learner's knowledge and skills for the task performance in the form of keyboard/mouse operations and saves his/her program in a *.kb* file. The post-processor compares the teacher's and learner's programs and executes different standard procedures for graphical representation and additional processing of the results. In case of a TODE for training in dynamic systems the list of these procedures includes: (a) graphical representation of a functional dependence, (b) tabular representation of a functional dependence, (c) graphical representation of a set of functional dependencies on a common co-ordinate system, (d) evaluation of the character and duration of transient processes, (e) evaluation of model validity. By means of the task manager the teacher plans the teaching, e.g. sets the sequence of tasks, teching goal, and the limitations of the teaching process. There are two alternatives for planning: a static plan for a group of learners and a dynamic plan for an individual learner. After the task performance the learner's model is refreshed adding the number of the task and the learner assessment in a data base file (*.db*).

The current state of the computer-aided teaching and learning algorithms is exhaustively discussed in a survey [5]. One of the key problem of design and implementation a TODE for algorithms is definition of a visual language for algorithm knowledge representation. At this stage we suppose to use for this TODE the background proposed by Zheliazkova, 1995 and discussed in [1].

This paper presents an exhaustive definition of such a language. The rest of the paper is organized as follows. Section three gives a short characteristics of the language and the tree with its keywords. More detail information about the syntax and semantics of the language is given in the next section. Section four presents the general structure of the teacher's programs. An example program modelling a classical algorithm is given in the fifth section. The last section summarizes the most attractive features of the proposed language and the authors' intentions in the near future.

3. THE TREE WITH THE LANGUAGE KEYWORDS

The internal language can be referred to visual very high-level descriptive (non-procedural) languages. Visual language means that all source of the program is generated by the generator. Very high-level language means that the program is treated in the terms of the interface components and in knowledge blocks and atoms. The descriptive language means that the program does not contain the procedures for knowledge processing. The above-said contrasts with the general-purpose visual high-level procedural languages such as Delphi and Visual C++ where only a part of the program source is generated mainly concerning the user interface and the rest part has to be created using object-oriented programming.

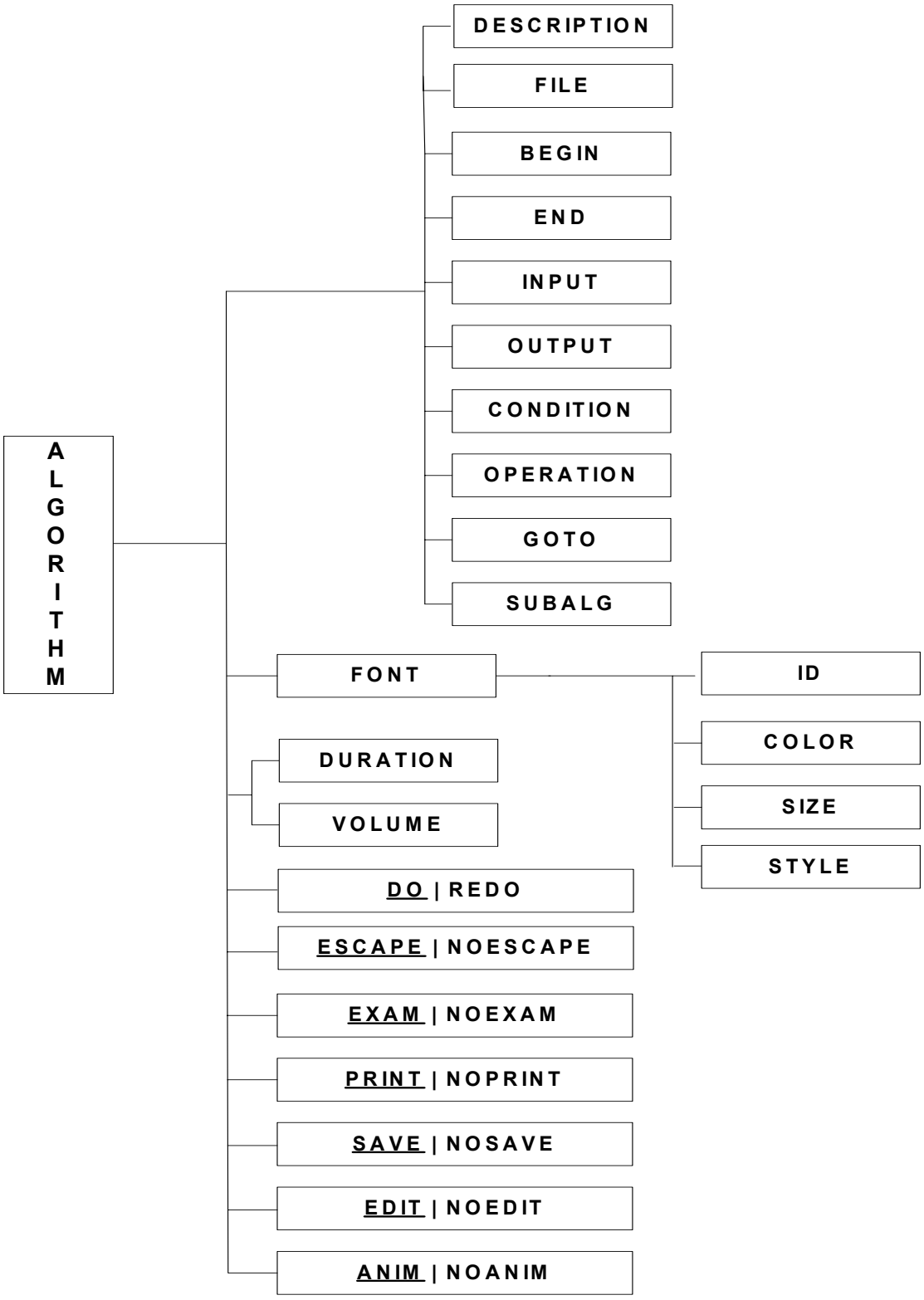

Fig. 1. The tree with the keywords of the language

4. SINTAX AND SEMANTICS OF THE LANGUAGE

The syntax and semantic of the language constructions (program, block, and font) are described respectively in tables 1 to 3, where the underlined value of the key directives means the default value.

Table 1

Teacher's program ::=	
ALGORITHM	
FILE <string>	
DESCRIPTION <memo>	<memo> task description
VOLUME <integer>	<integer> the knowledge volume of the algorithm
DURATION <integer>	<integer> time planned for algorithm construction in minutes
ESCAPE| NOESCAPE	Global key directive for allowing or not refusal from the algorithm construction
PRINT | NOPRINT	Global key directive for allowing or not the printing of the algorithm
SAVE | NOSAVE	Global key directive for allowing or not the saving of the algorithm
EDIT | NOEDIT	Global key directive for allowing or not the editing of the algorithm
ANIM | NOANIM	Global directive for allowing or not the animation of the algorithm
DO | REDO	Global key directive for allowing or not redo the algorithm construction

Table 2

<block description>::=	
INPUT <integer> < memo > **X1** =< integer > **Y1**=< integer > **X2** =< integer > **Y2**=< integer >	< integer > input block identifier < memo > block content <X1><Y1> the coordinates of the top left angle of the block <X2><Y2> the coordinates of the down right angle of the block
OUTPUT < integer >< memo > **X1** =< integer > **Y1**=< integer > **X2** =< integer > **Y2**=< integer >	< integer > input block identifier < memo > block content <X1><Y1> the coordinates of the top left angle of the block <X2><Y2> the coordinates of the down right angle of the block
CONDITION< integer 1>< memo > < integer 2>< integer 3> **X1** =< integer > **Y1**=< integer > **X2** =< integer > **Y2**=< integer > **FONT**	< integer 1> conditional block identifier < memo > text content of the block < integer 2> block identifier, if true < integer 3> block identifier, if false <X1><Y1> the coordinates of the top left angle of the block <X2><Y2> the coordinates of the down right angle of the block
OPERATION < integer >< memo > **FONT**	< integer > block identifier < memo > block content
BEGIN **X1** =< integer > **Y1**=< integer > **X2** =< integer > **Y2**=< integer >	begin block <X1><Y1> the coordinates of the top left angle of the block <X2><Y2> the coordinates of the down right angle of the block
END **X1** =< integer > **Y1**=< integer > **X2** =< integer > **Y2**=< integer >	end block <X1><Y1> the coordinates of the top left angle of the block <X2><Y2> the coordinates of the down right angle of the block
SUBALG < string >< integer > **X1** =< integer > **Y1**=< integer > **X2** =< integer > **Y2**=< integer >	< string > file specification with the sub-algorithm < integer > block identifier <X1><Y1> > the coordinates of the top left angle of the block <X2><Y2> the coordinates of the down right angle of the block

GOTO < integer 1> < integer 2> **X1** =< integer > **Y1**=< integer > **X2** =< integer > **Y2**=< integer >	< integer 1> block identifier < integer 2> block identifier of the destination block <X1><Y1> > the coordinates of the top left angle of the block <X2><Y2> the coordinates of the down right angle of the block

Table 3

<font description>::=	
FONT < string >	< string > font name
FCOLOR < string >	< string > font color
FSIZE < integer >	< integer > font size in pt
FSTYLE < string >	< string > font style
FX1 =< integer > **FY1**=< integer > **FX2** =< integer > **FY2**=< integer >	<FX1><FY1> the coordinates of the top left angle of the font <FX2><FY2> the coordinates of the down right angle of the font

5. GENERAL STRUCTURE OF THE TEACHER's PROGRAM

A teacher's program corresponds to one algorithm. The general structure of a teacher's program is described bellow using the Bekus-Naur notation. Here the keywords of the language are in capital bold letters and the special symbols have the meaning of: **::=** defines a syntactical construction; _ connects the words in a syntactical construction name; | divides the alternative constructions; { } encloses a repeating construction; [ ] encloses not obligatory construction; < > encloses the name of a syntactical construction, which is still not defined.

< teacher's program >::= **ALGORITHM**
FILE <string>
DESCRIPTION <memo>
VOLUME <integer>
DURATION <integer>
{<block description>}
ESCAPE | **NOESCAPE**
PRINT | **NOPRINT**
SAVE | **NOSAVE**
EDIT | **NOEDIT**
ANIM | **NOANIM**
DO|**REDO**
END
<block description>::= **BEGIN** | **END** | **INPUT** <integer><memo>| **OUTPUT** <integer>< memo >|
OPERATION <integer><memo> | **CONDITION** <integer1>< memo >
<integer2><integer3>| **SUBALG** <string><integer>| **GOTO <**integer> <memo>
X1 =<integer>
Y1=<integer>
X2 =<integer>
Y2=<integer>
[**GOTO** <integer1> <integer 2>]
<description of font>::= **FONT** <string>
FCOLOR <string>
FSIZE <string>
FSTYLE <string>
FX1 =<integer >
FY1=<integer >
FX2 =<integer >
FY2=<integer>

6. AN EXAMPLE PROGRAM

A program for well-known algorithm for searching the maximal element in an array of numbers [1] has been created with a standard text editor (Fig. 2). The program is intended to serve as a test-bed after the implementation of the corresponding program generator.

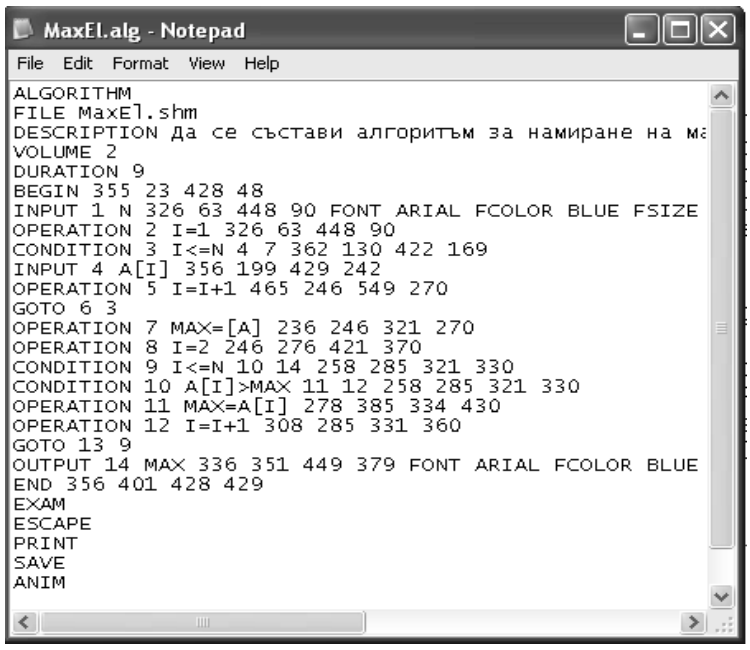

Fig. 2. The example program

7. CONCLUSIONS AND INTENTIONS

In this paper the full definition of an internal visual very high-level descriptive language for algorithm knowledge representation has been proposed. Besides these attractive features the program in this language ensures economy of educational knowledge and can be translated easily in other working languages by means of a multi-language dictionary.

In the near future we plan to design and implement the first version of an on-line program generator. The design will include the architecture of the program generator, its event-driven algorithm in pseudo-code and the algorithm knowledge representation in computer memory.

REFERENCES

1. Zheliazkova I., An Intelligent System for Teaching and Learning Algorithm, Int. J. on Computers & Education, 1995, Vol. 24, No. 2, pp. 117-125.
2. Iliev V. et al., Atanasova G., Design and Testing of an Environment for Algorithm Animation, Technical Report, University of Rousse, 2003, No. RU-PF-06.
3. Fisher G., Scharff E., Learning technologies in support self-directed learning, *Journal of Interactive Media in Education*, Vol. 98, No. 4, October, 1998, pp…,
4. Zheliazkova I. I., Georgiev G. T., Representation and processing of domain knowledge for simulation-based training systems, *Int. Journal of Intelligent Systems*, Vol. 10, No.3, pp. 255-277 (IF 0.379, 1997).
5. 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.

ABOUT THE AUTHORS

Assoc. Prof. Irina Zheliazkova, PhD, Department of Computer Systems and Technologies, University of Rousse, Phone: +359 82 888 744, e-mail: irina@ecs.ru.acad.bg

Senior Assistant Galina Atanasova, Department of Informatics and Information Technologies, University of Rousse, Phone: +359 82 888 470, e-mail: gal@nami.ru.acad.bg.