SPECIFICATION OF THE KNOWLEDGE DESCRIPTION LANGUAGES

Polina Atanasova, Irina Zheliazkova, Avram Levi

Abstract: The capabilities of existing languages for support of teaching plan activities are unsatisfied. In the paper the main features of languages for knowledge description and a methodology of its specification at task and exercise level are presented on an example language for structural knowledge description.

Keywords: domain-independent, knowledge, description, language, task, exercise.

ACM Classification Keywords: Computer and Information Science Education, Knowledge Representations of Formalisms and Methods

Introduction

During the last decade our research group is actively working on development, implementation and investigation an Intelligent Collaborative Environment for Courseware Planned Teaching (*ICECPT*). Different Task-Oriented Design Environments (*TODEs*) with a common Task Base (*TB*) are integrated in it. The purpose of a *TODE* is to support the plan/execution of a given type session (lecture/test/exercise) trough an ontology-based constructing a given type of tasks such as tested questions [1], animated concepts [4], designed algorithms [9] and schemes (*Ss*) [10], simulated systems [7], and even measuring and controlling objects [8]. The specification of a Knowledge Description Language (*KDL*) is the key problem of each *TODE*. The capabilities of the languages offered by the systems for support of collaborative plan activities are overviewed in [3] where a multi-agent logic language *MALLET* is proposed to enhance these activities.

ICIPT is close the systems specially designed for teaching and based on a domain-independent language. *TILE* language [2] is similar to the natural one while *IALMS* language [5] is procedural. In *TILE* the author (*A*) can index and annotate the course material (*CM*) using a flexible structure code language in the form of six sentences in English. By means of a flexible code query language the *CM* suitable for a given learner (*L*) is delivered using semantic search in Web. The *IALMS* language is offered only the *A* for describtion of the *CM* structural tree, the intermediate nodes of which represent logical functions and the terminal nodes active and passive blocks with *CM*. The tree is passing bottom-up to choose the blocks with highest importance for a new session.

This paper summarizing the methodology of specification of *KDL*s on an example of *KDL* is organized as follows. Section 2 gives general features of the *KDLs*, the next two sections present the sublanguages for the task and exercise level respectively. The assessment of the simulated author and several learners' subprograms is given in the section 5. The last one summarizes the most attractive features of the proposed class of languages.

General Features of KDLs

For the needs of *ICEIPT* a three-level domain-independent language called COURSESCRIPT has been specified [6] using disassembly long-term and assembly short-term planning/executing techniques. The postfix "*script*" hereinafter means that the corresponding translator converts the program files from it to *HTML* program later interpreted by a standard web browser. This language can be seen as a hierarchiy of *KDLs* (fig. 1) with the top-down methodology applied for its specification, implementation, and investigation. *KDL* of a *TODE* is *tree-agent* but consists of two task and session's sublanguages. The first one is for description of an *A*/L task knowledge

captured though ontology-based constructing a given type *KU.* Except LABSCRIPT the other sublanguages at the high level have been specified, implemented, and investigated. In this paper SCHEMESCRIPT sublanguage, e.g. for description of different kinds of *Ss* (digital, analogue, logical and even mixed) is used as an example task's sublanguage. The session's sublanguage describes the *T*'s pedagogical knowledge for planning/executing a session, including a given type tasks. These very similar sublanguages are at the intermediate level and EXERCISESCRIPT is used forward as an example session's sublanguage. Although the subprograms/program and are fully generated the *A/T* can use the task's/exercise sublanguage to increase the flexibility and productivity of teaching while the *L* cann't know them at all. A *KDL* is *internal* one used for automatically computing the task/exercise's parameters (knowledge volume, degree of prompt, degree of difficulty, and planned time for performance) as well as the coefficient of proximity of the *L*'s knowledge relatively to the A's one. The language is *visual* because it treats the user interface components (menus, buttons, icons, images, fonts, pens, and brushes) as graphical objects. A *KDL* is classified as a *very high-level* language because its keywords are similar to natural domain ones. From the *TODE* the task's sublanguage presents a set of subprograms, and the session's one a program calling them. The *A*'s subprograms are stored in text files with a fixed structure and extension (*.sch* in case of SCHEMESCRIPT). The *T's* program is stored in a text file with a fixed structure and extension (*.ecs*) for an exercise.

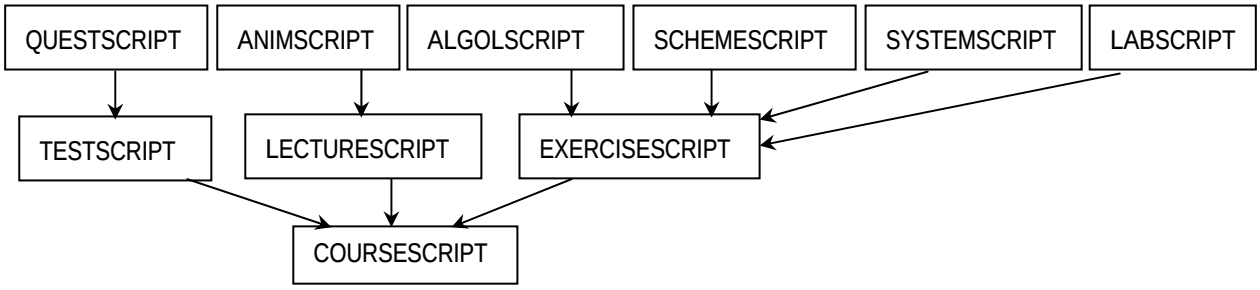


Fig.1. The hierarchy of the *KDLs*

Task's Level Sublanguage

The set of keywords for a task's sublanguage is fixed in a tree with root named with the tasks type (SCHEME on fig. 2) and terminal node with END. Symbol | (OR) outlines the differences between the *A*'s and *L*'s subprogram or the option of including key directives for the *T*'s intervention during the *L*"s performance. Then the syntax and semantics of the task sublanguage are presented by nested tables (Table1-8 in our case), each one corresponding to one syntactical construct and finishing with END (shown only in Table 1).

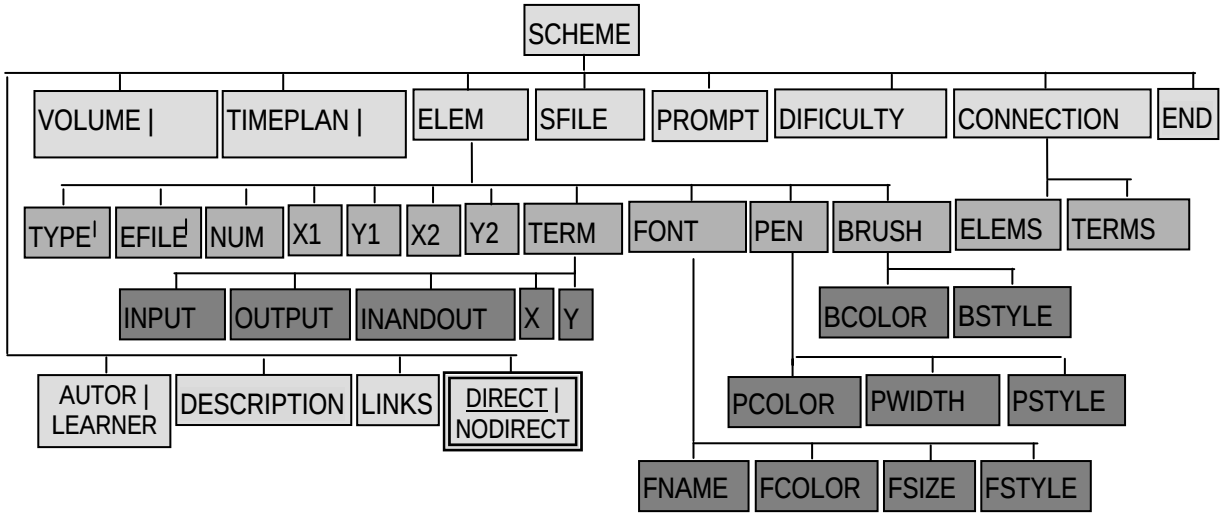


Fig.2. The example keywords tree

Table 1

SHEME <*string*>	<*string*> is the program name
DESCRIPTION <*free_text*> END	<*string*> is the subject domain name. The word END, if needed within <*free_text*>, must be escaped by a slash
VOLUME <*integer*> | PROXIMITY <*real*>	<*integer*> is the knowledge volume of the *A*'s scheme; <*real*> the coefficient of proximity for the *L*'s scheme
TIMEPLAN<*integer*>| TIMETAKEN <*integer*>	<*integer*> time planned by the *T* or spent by the *L* for scheme biulding
SFILE=<*string*>	name of graphics file, used for background
{<structure_description>}	(see Table 2)
{<directives_description>}	(see Table 6)
END	

Table 2

<structure_description>::=	
{<element_description>}	(see Table 3)
{<connection_description>*}*	(see Table 5)

Table 3

<element_description>::=	
ELEM <string>	<*string*> sets the component type (e.g. TRANS for transistor)
TYPE <*string*>	<*string*> sets the component's type within the circuit
EFILE <*string*>	<*string*> is the name of a graphics file with the component's icon
NUM <*integer*>	number of component's terminals
{<terminal_description>}	(see Table 4)

X1=<*integer1*>Y1=<*integer2*> X2=<*integer3*> Y2=<*integer4*>	coordinates of top-left and bottom-right corner of the component's window for the component visualization
[<font_description>]	(see Table 6)
[<pen_description>]	(see Table 7)
[<brush_description>	(see Table 8)

Table 4

<terminal_description>::=	
TERM <*string*> | <*integer*>	<*string*> sets the terminal name (e.g. EMITTER). An integer number can be specified instead.
INPUT | OUTPUT | INANDOUT	define the terminal's direction
X= <*integer1*> Y= <*integer2*>	screen coordinates of the terminal within the component's icon

Table 5

<connection_description>::=	
CONNECTION [<*string*>]	<*string*> sets the connection's name. Optional.
ELEMS <*string1*> <*string2*>	<*string1*>, <*string2*> the order and names of the connected components
TERMS <*integer1*> <*integer2*>	<*integer1*>, <*integer2*> the order and indexes of the element's terminals

Table 6

<font_description>::=	
FNAME <*string*>	<*string*> sets the font face name
FCOLOR <*string*>	<*string*> sets the font colour
FSIZE <*integer*>	<*integer*> sets the font size in pixels
FSTYLE <*string*>	<*string*> sets the font style, e.g. BOLD, ITALIC, etc.
X1=<*integer1*>Y1=<*integer2*> X2=<*integer3*> Y2=<*integer4*>	coordinates of top-left and bottom-right corner of the window for visualization

Table 7

<pen_description>::=	
PCOLOR <*string*>	<*string*> sets the pen colour
PSIZE <*integer*>	<*integer*> sets the pen size in pixels
PSTYLE <*string*>	<*string*> sets the pen style, e.g. SOLID, DOT, etc.

Table 8

<brush_description>::=	
BCOLOR <*string*>	<*string*> sets the brush colour
BSTYLE <*string*>	<*string*> sets the brush style, e.g. HORIZONTAL, VERTICAL, etc.

Table 9

SCHEME [<*string*>]
DESCRIPTION [<*string*>]<*free_text*> END
VOLUME <*integer*>| PROXIMITY <real>
DURATION <*integer*> | TIMETAKEN <*integer*>
SFILE=<*string*>
{<structure_description>}
[<directives_description >]
[TIMER @ | <*integer1*>:<*integer2*>:<*integer3*>]
END

<structure_description>::=
{<element_description >}
{<connection_description>} END

<element_description>::=
ELEM <*string*>
[TEXT <*string*>]
FILE <*string*>
NUM=<*integer*>
{<term_description>}
X1=<*integer1*>
Y1=<*integer2*>
X2=<*integer3*>
Y2=<*integer4*> END

<term_description>::=
TERM <*string*> | <*integer*>
INPUT |
OUTPUT |
INANDOUT
X=<*integer*1>
Y=<*integer2*> END

<connection_description>::=
CONNECTION [<*string*>]
ELEMS <*string1*><*string2*>
TERMS <*integer1*><*integer2*> END

<font_description>::=
FNAME <*string*>
FCOLOUR=<*string*>
FSIZE=<*integer*>
FSTYLE=<*string*> END

<pen_description>::=
PCOLOUR=<*string*>
PSIZE=<*integer*>
PSTYLE=<*string*> END

<brush_description>::=
BCOLOR=<*string*>
BSTYLE=<*string*> END

The Baschus-Naur notation is used to present the structure of a *KDL* subprogram. A subprogram in SCHEMESCRIPT (fig. 4) begins with the index of the *S* after SCHEME and finishes with END. The free text for the task DESCRIPTION is followed by the task parameters description. The structural knowledge itself includes sets of element (*E*) and connection (C) declaration. An *E* declaration contains the component domain name, its function and/or properties, the name of the file with the component schematic, the number of its terminals, a set of terminal (*T*) declarations, and coordinates of the upper left and lower right corners of the screen window. The attributes for a *T* declaration are the domain name (or the ordered number) of the *T*, its type (input, output or input and output), and the screen coordinates. A *C* definition includes the domain names of the two connected elements and the indices of their terminals. Font, pen, and brush descriptions are only for better *S* design.

Performing the i^{th} task by a *A*/*L* leads to generation of the subprogram tree (AT_i / LT_i). The levels of this tree correspond to the levels of nesting of the subprogram constructs, e.g. *S* for level 1, *E* and *C* for level 2, and *T* for level 3 (fig. 3). The number of the tree nodes at a given level is equal to the number of the constructs, nested in the parent construct. The nodes on the left of an *S*, *E, C* or *T* node contain domain knowledge, while the nodes on the right represent the keywords of the attributes, associated with the same construct. A terminal node on the right of an attribute represents its domain name or value. The sum of the i^{th} tree nodes (V_i) and arcs (U_i)

serves as a common measure of the task knowledge volume i.e. $q_i = |V_i| + |U_i| \approx 2|V_i|$. In case of a *S* if e_i is the number of ELEM constructs, m_i the number of CONNECTION ones, r_{ij} the number of elements, included in the j^{th} TERM construct $q_i = 2\left[8 + 8m_i + 16e_i + 9\sum_{j=1}^{N_i} r_{i,j}\right]$. The degree of prompt p_i generally meaning the part of the *A*'s knowledge available during the *L*'s task performance for a *S* constructing task $p_i = (4 + 3m_i + 8e_i + 6\sum_{i=1}^{N_i} r_{ij}) / q_i$. Let a_{ij} be the number of nodes missing in the AT_i, but present in LT_i, and b_{ij} the number of nodes present in AT_i, but missing in LT_i. The expression $c_{ij} = (q_i - a_{ij} - b_{ij}) / q_i$ is used to measure the degree of proximity of the j^{th} *L*'s knowledge to the *A*'s one. To encourage the faster *L* and reprimand the slower *L* the time correction $c_{ij}^{*} = c_{ij}(t_{1i} / t_{2i})$ is applied, where t_{1i} / t_{2i} is the *A's/L's* time and $(t_{1i} / e \le t_{2i} \le e.t_{1i})$. Let the j^{th} *L* has completed the i^{th} task for time t_i with assessment c_{ij}. Then the average planned time $\bar{t}_i$ (initially equal to t_{1i}) and degree of difficulty $\bar{d}_i$ (initially equal to 0.5) are recomputed, e.g. $\bar{t}_i = (\bar{t}_i + t_i)/2$ and $\bar{d}_i = (\bar{d}_i + c_{ij})/2$.

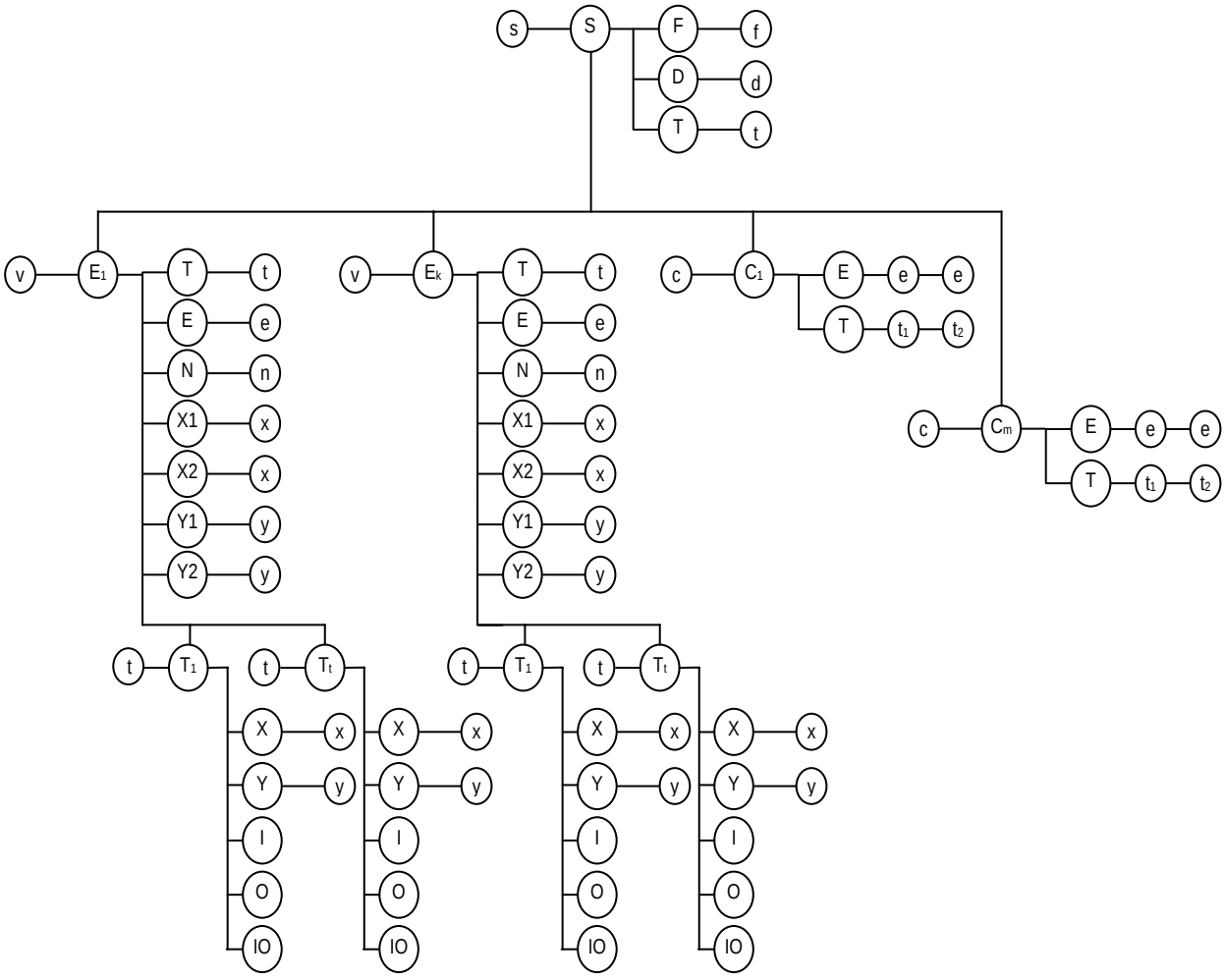


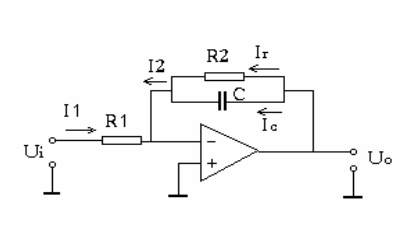
Fig. 3 The example subprogram tree

Table 10

SCHEME integrator DESCRIPTION Build the analogue inverting integrator END VOLUME 76 PROMPT ? TIMEPLAN 10 DIFICULTY 0.5 ELEM RESISTOR TEXT R1 FILE RES.GIF NUM=2 TERM 0 INPUTANDOUTPUT X=0 Y=5 END TERM 1 INPUTANDOUTPUT X=40 Y=5 END X1=20 Y1=40 X2=60 Y2=50 END FONT FNAME ACADEMY CYRILLIC FCOLOR BLACK FSIZE 9 STYLE ITALIC BOLD END BRUSH BCOLOR GREEN BSTYLE SOLID END PEN PCOLOR BLACK PSIZE 1 PSTYLE SOLID END	ELEM RESISTOR TEXT R2 FILE RES.GIF NUM=2…END FONT… END BRUSH… END PEN… END ELEM OPAMP TEXT A1 FILE OPAMP.GIF NUM=3…. END PEN… END BRUSH… END ELEM CAPACITOR TEXT C FILE CAP.GIF NUM=2… END FONT END BRUSH… END ELEM TIE TEXT T1 FILE TIE.GIF NUM=1… END BRUSH… END PEN… END ELEM TIE TEXT T2 FILE TIE.GIF NUM=1… END BRUSH… END PEN … END ELEM GND TEXT GND FILE GND.GIF NUM=1… END CONNECTION ELEMS R1 T1 TERMS 1 0 END CONNECTION ELEMS T1 A1 TERMS 0 0 END… END END

The *A*'s and several *L's* tasks are simulated to serve as a test bed for the *TODE* implementation. The analog circuit inverter integrator had been chosen for this purpose (table 11). A's subprobram (Table 10) and three learner's subprograms are prepared in *Notepad* , which task's parameters values are computed manually. If *S L1* is topologically equivalent to the *SA* and *L1* is faster than the *A TODE* has to estimate this performance with c* > 1. If the *S L2* has one redundant element and one missing connection is constructing for the planned time the *TODE* has to assess him/her with *c**< 1.00. The *S L3* has to be scored even lower if one element and one connection are missing and the time taken is more than the planned one.

Table 11



	q	c	t	c*
Scheme of the author (*SA*)	76	-	10	-
Scheme of the first learner (*L1*)	76	1.0	8	1.25
Scheme of the second learner (*L2*)	77	0.8	10	0.8
Scheme of the third learner (*L3*)	71	0.6	12	0.5

The Session's Sublanguage

In a TODE the assembly technique is applied to plan an exercise session. So, if N is the number of the tasks included in the exercise its knowledge volume $Q = \sum_{i=1}^{N} q_i$ and degree of prompt $\bar{P} = (\sum_{i=1}^{N} p_i) / N$. If M is the number of the learners performing the exercise, its planned time (initially $\bar{T} = \sum_{i=1}^{N} t_{1i}$) and degree of difficulty (initially $\bar{D} = (\sum_{i=1}^{M} d_i) / M = 0.5$) are recomputed as $\bar{T} = (\bar{T} + \sum_{i=1}^{N} t_{2i}) / 2$ and $\bar{D} = (\bar{D} + (\sum_{j=1}^{M} c_{ij}) / M) / 2$.

Except these parameters for adaptation to a given *T* preferences a program in *KDL* contains key directives to allow or disallow: giving up of the task (ESCAPE|NOESCAPE); printing of the scheme description (PRINT|NOPRINT); saving the it in a file (SAVE|NOSAVE), assessing the learner's task (ASSESS|NOASSESS), accessing content-dependent *CM* (HELP|NOHELP), showing the clock (TIMER|NOTIMER). The above underlined values are default ones. When assessment is allowed, the criteria description constuct is added, and the *T* has to choose among four types of the assessment (SUCCESS|FAIL, PERCENT, PROXIMITY or MARK). In the latter case the score intervals of the scale are added.

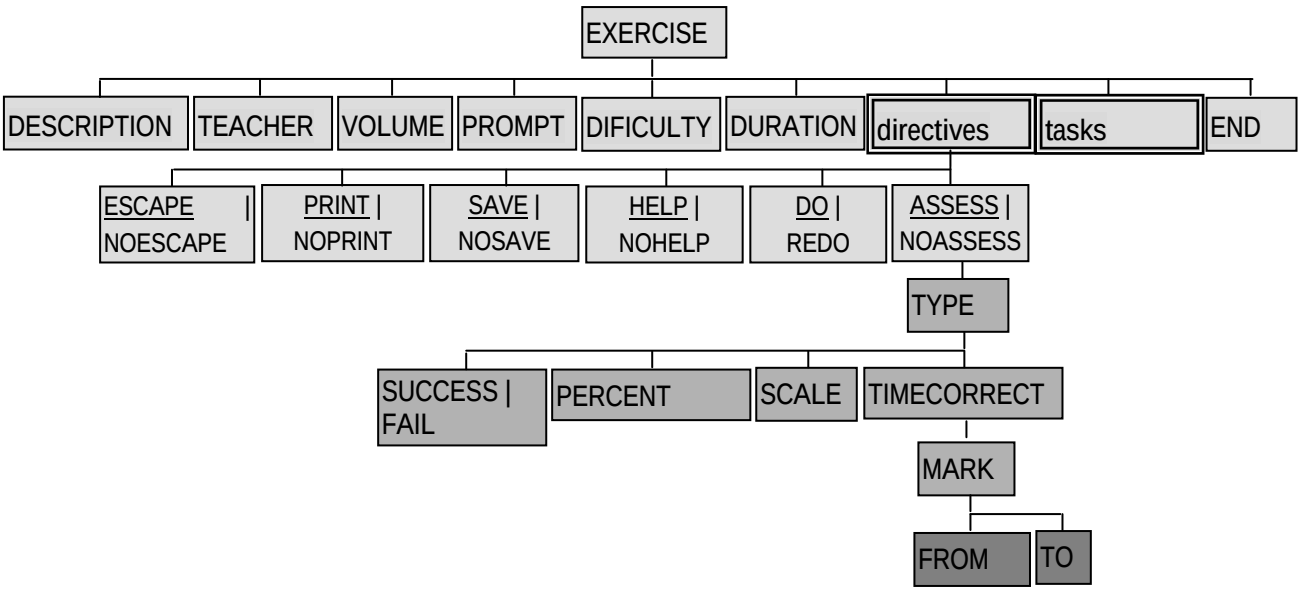


Fig. 5. The exercise keywords tree

Table 12

EXERCISE <*index*>	<*index*> is the exercise'position code in the course structure
TEACHER <*string*>	<*string*> is the teacher's name planning/monitoring the exercise
parameters_description	(see Table 13)
directives_description	(see Table 14)
END	

Table 13

<parameters_description>::=	
VOLUME <*integer*>	<*integer*> is the total knowledge volume
PROMPT > <*real*>	<*real*> is the average degree of system prompt
DIFICULTY <*real*>	<*real*> is the average degree of difficulty
TIMEPLAN <*integer*>|TIMETAKEN <*integer*>	<*integer*> is time planned/taken for the exercise performance

Table 14

< directives_description >::=	
ESCAPE | NOESCAPE	allows/disallows giving up the task constructing
PRINT | NOPRINT	allows/disallows printing the *A*'s subprogram
SAVE | NOSAVE	allows/disallows saving the *A*'s subprogram onto a disk
HEPL | NOHELP <*string*>	<*string*> links to a content-dependent *CM*, if allowed
DO | REDO <*integer*>	allows/disallows the task construting again
ASSESS | NOASSESS	allows/disallows the exercise assessment
NOTIMER | TIMER=@|<*integer1*>:<*integer2*>:<*integer3*>	the clock if shown during performance is initialized with the supplied integers (HH:MM:SS) or with the system time (@)
criteria_description	(see Table 15)

Table 15

<criteria_description >::=	
TYPE SUCCESS|FAIL| PERCENT|SCALE <*integer*>	the assessment type (success/failure, percentage, proximity or scale); <*integer*> is the number of the scale intervals
MARK <*string*> 2-FROM:<integer1TO:<integer2>… 6-FROM: <integer9>TO:<integer10>	<*string*> is the linguistic value of the interval, couple of integers means low and higher limit of the interval scores
TIMECORRECT <*real*>	<*real*> between 0 and 1 for time correction of the assessment

Conclusion

The common features of a new class of languages developed and implemented for the needs of a courseware knowledge description are given. The methodology of specificaton of a task and exercise sublanguage is presented on an example task's sublanguage for structural schemes constructing. The sublanguages offered the author, teacher, and learner are used mainly for automatically computing the task'/exercise parameters. The subprograms/program syntax correctness, reusability, and understandability is ensured through their well-structuring and fully generation.

Bibliography

[1] Andreeva M., Models and Tools for Development of an Integrated Authoring Knowledge Testing Environment, Thesis of PhD dissertation, Rousse University, 2006 (in Bulgarian).

[2] Jesshope C., Heinrich E., D-r Kinshuk, Technology Integrated Learning Environments for Education at a Distance, DEANZ Conference, 26-29 April, 2000, Dunedin, New Zealand.

[3] Fan X., Yen J., Miller M., Ioerger T. R., Volz R., MALLET – A Multi-Agent Logic Language for Encoding Teamwork, Transactions on Knowledge and Data Engineering, Vol. 18. No. 1, 2006, pp. 123-138.

[4] Stefanova S. P., Application of Didactic Petri Nets for Teaching Purpose, Thesis of PhD dissertation, Rousse, 2002

(in Bulgarian).

[5] Zabunov S., A Language for Describing the Generating Structure of the Educational Material in the Individually Adaptive Learning Management System, Intenational Conference CompSysTech, Rousse, Bulgaria, 2004, pp. V.8-1 – V.8.6.

[6] Zheliazkova I. I., A Domain-Independent Language for Courseware Knowledge Description, Proceedings of the Second National Conference on E-Leaning in the Higher Education, 14th –16th July, Kiten, Bulgaria, 2006, pp. 44-48.

[7] Zheliazkova I. I., Georgiev G. T., Representation and Processing of Domain Knowledge for Simulation-Based Training Systems, Int. J. of Intelligent Systems, 2000, Vol. 10, No.3, pp. 255-277.

[8] Zheliazkova I. I., Georgiev G. S., Minkova P., An Approach to Building Computer-Based Laboratories, Journal of Automatics and Informatics, No 4, 1997, pp. 21-26 (in Bulgarian).

[9] Zheliazkova I. I., Atanasova G., A Visual Language for Algorithm Knowledge, Proceedings of the International Conference on Computer Systems and Technologies (e-Learning), Rousse, 2004, pp. IV.24-1- IV.24-6.

[10] Zheliazkova I. I., Georgiev G. T., Valkova P. L., A Task-Oriented Environment for Structural Schemes Design, International Journal of Information Technologies and Control, Vol. 2, 2006, pp. 2-13.

Authors' Information

Polina Atanasova – PhD student, University of Rousse, Studentska street 8, Rousse 7017, Bulgaria; e-mail: valkova_99@yahoo.com

Irina Zheliazkova – Associate Professor; University of Rousse, Studentska street 8, Rousse 7017, Bulgaria; e-mail: irina@ecs.ru.acad.bg

Avram Levi – Associate Professor, Rousse University, Studentska street 8, Rousse 7017, Bulgaria; e-mail: ALevi@ecs.ru.cad.bg