ADAPTATION OF A TASK-ORIENTED TRAINING ENVIRONMENT TO ITS USERS

Irina Zheliazkova, Georgi Georgiev, Rumen Kolev

Abstract: The problem of adapting teaching systems to the teacher has not been extensively covered in the specialised literature. The authors present the server-client architecture of a Task-Oriented Environment for Design of Virtual Labs (TOEDVL). The paper focuses on the computational models supporting its base of tasks (BT) and on two groups of behavioural tutor's models for planning training sessions. Detailed examples are presented.

Keywords: task, training, search, sort, adaptation, models

ACM Classification Keywords: J.1 Computer Applications, Administrative Data Processing, Education

1. Introduction

Adaptation of the teaching system to the learner's current level of domain competence, the teaching materials and the context of presenting the information is not a new idea. The learner model ensuring this adaptation was one of the main components of Intelligent Tutoring Systems, developed two decades ago. Different kinds of learner models have been exhaustively considered by *Zheliazkova & Kolev, 2004* in [1]. For the first time adaptation to another user – the teacher was considered by *Peachey & McCalla, 1986.* In [2] they determined the short-term individual learner's plan as a sequence of so called teaching operators and propose to use first order predicates for its execution. The system *EXTERN* (*Paschin and Mitin*, *1985 [3]*) also uses dynamic planning, where the duration of the teaching session and the sequence of teaching blocks are not initially defined. As a criterion for choosing the next block, the different implementations usually use one or more of the following: the volume or relative change of acquired knowledge, the time for learning, the learning speed, etc. Only in recent years the problem of adaptation of the teaching environments to their users is being seriously considered by *Jesshope at al., [4].* The arguments for this are that individual teachers use different strategies for planning the courseware presentation, level of didactic knowledge, types of assessment, scales for evaluation, rules for diagnostic of the learners' knowledge as well as ways for intervention in the learning process.

From the above discussion it follows that the training systems have to be adaptive to the author, tutor, and learner. Although intelligent, some of the well known training systems from the last two decades (*Gonzalez & Ingraham, 1994* [5]; *Chu et al., 1995* [6]; *Vasandani & Govindaraj, 1995* [7]) don't meet this requirement. The algorithm for applying the system RIDES for course development focuses on adapting the training to its author (*Fleming, 1996* [8]).

2. Architecture of a TOEDVL

Figure1 presents the architecture of a TOEDVL. In this figure the following notations are used: double arrows represent data links; single solid lines – control links; dashed lines – HTML documents transported from server to client; dash-dot-dot lines – binary data transferred between client and server. The architecture is domain and task-independent due mainly to the language for knowledge description in the training tasks. It is supposed that the user's registration/access control will be handled by a separate module administrator, also that the learner has already acquired deep structural knowledge about the simulated system by means of a similar environment for design of structural schemes [9]. The teaching material ensuring the feed-back to the learner is prepared by means of standard tools and files e.g. help editor (*.chm*), graphical editor (*.gif*, *.jpg*, *.png*), audio (*.wav*), and video (*.avi*) editors. For presenting tests to the learner, two approaches can be used. Tests can be generated and interpreted by means of specialised tools (the Test editor in Figure 1 and a test interpreter, integrated with the Program interpreter) [10]. Alternatively, they can be in HTML format, prepared by means of HTML content generator, such as *Dreamweaver*. In this case the Test interpreter (not shown in Figure 1) interprets them.

A generated program representing the user's knowledge about a given training task is extracted through visual programming by means of *the program generator.* It stores the syntactically and semantically correct program in a standard text file with an extension *.trn*. *The task manager* implements planning and execution of the sequence of

training tasks in a session (file with an extension *.ses*). This sequence is viewed as a short-term plan for the individual learners. Task execution itself is ensured by means of another tool – *the program interpreter*, which runs the simulation programs and saves the simulation results in a file with the extension *.hst*.

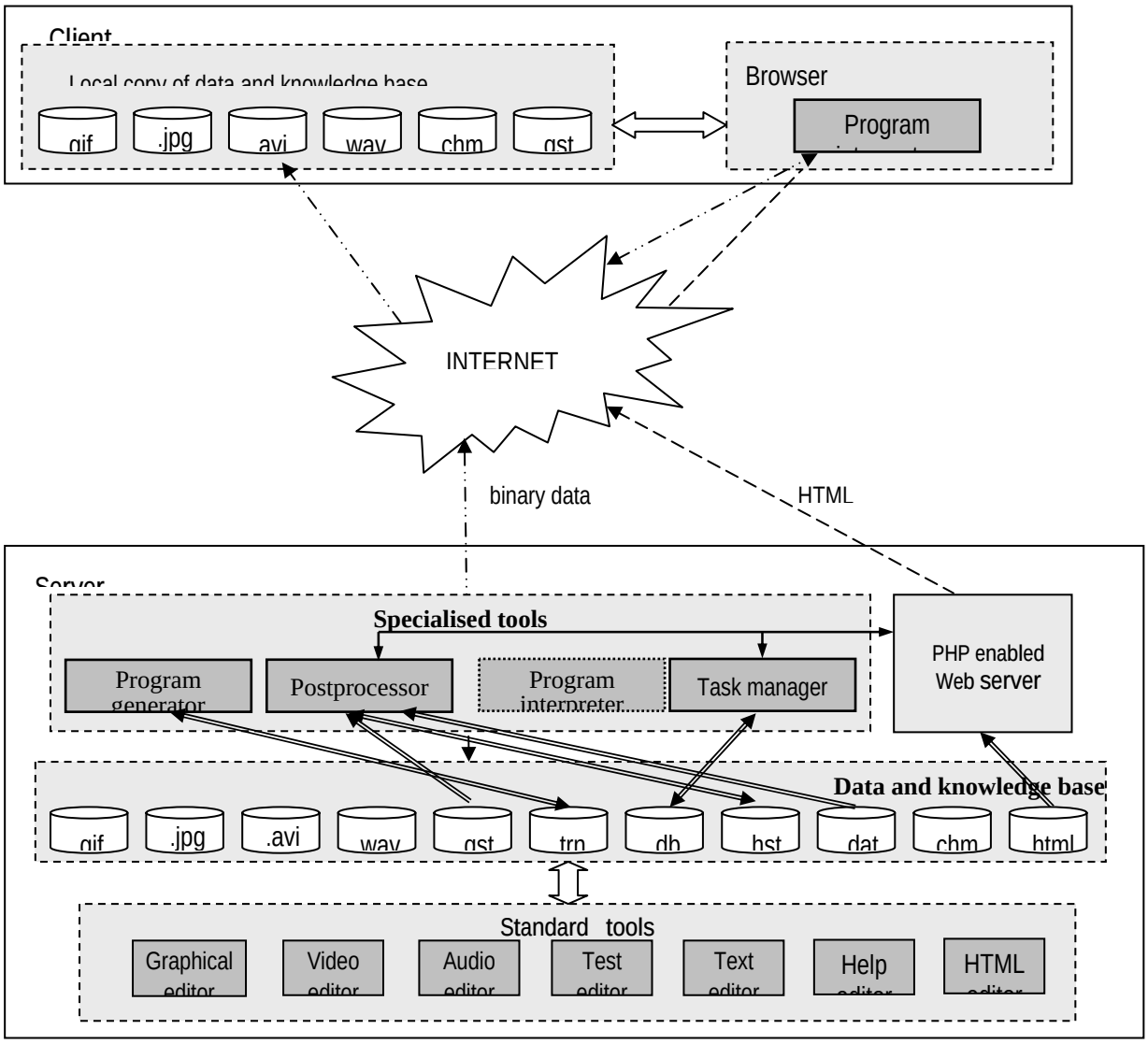

Figure 1. Architecture of TOEDVL

During the execution, the necessary operator's skills in the form of keyboard/mouse activities are added to the declarative and procedural knowledge. The computed or registered task and session parameters are stored in the base of tasks (BT), kept in a database (.db). Each task is run on the same tool once by the tutor and once by each learner. After the author completes a task and his/her *.trn* file is returned to the server, the tool updates the BT. After the learner completes a task and his/her temporal *.trn* file is returned to the server, the tool updates the learner's model, kept in the same database (.db). *The post-processor* is another specialized tool, whose purpose is to implement different standard procedures for representation and processing the simulation results. A sample list of such procedures includes the following: *(1) graphical representation of a functional dependency; (2) tabular representation of such a dependency; (3) graphical representation of a family of dependencies in a common co-ordinate system; (4) evaluation of the type and duration of a transient process; (5) evaluation of the model adequacy; (6) presentation of a test question; (7) comparison of the trainee's actions with those of the* author. Standard text files (*.dat*) are used to store the values of the traced parameters, measured in the real systems. After a given learner's session finishes the learner's session parameters, such as knowledge volume, duration, speed of learning, and so on are accumulated in the database as statistical experimental data, for the needs of an integrated system for individual planned teaching [11].

3. Language for Knowledge Description of Training Tasks

Although the tasks in a real lab vary greatly depending on thfe physical nature of the real objects they can be classified as (1) *calculation tasks* for a given set of output parameters, (2) *monitoring tasks for* dynamic mode of the system, (3) *investigative tasks* for static mode of the system, (4) *drawing tasks* for the transient process in the system, (5) *drawing tasks* for a set of functional dependencies, (6) *optimisation tasks* for choosing the most appropriate system according to some criteria, (7) *control tasks* to cope with abnormal situations, and (8) *diagnostic tasks* to identify failed system components.

Table 1. The common structure of programs

<session_description>::=SESSION	<task_description> ::= SYSTEM [<string>]
ORGANIZATION <string>	FILENAME <string>
DEPARMENT <string>	DESCRIPTION <memo>
TEACHER <string>	DURATION <integer>
COURSE <string>	VOLUME <integer>
TOPIC <string>	PROMPT <real>
GOAL <free text>	DIFICULTY <real>
DURATION <integer>	{<parameter description>}
VOLUME <integer>	{<dependence description>}
DIFICULTY <real>	[COLOURS <integer><integer>]
[<directives_list>]	[DISCR_STEP = <real>]
[<criteria_description>]	[{<event description>}]
{<task_list>}	[TRACE <list of parameters >END]
ENDS	[SPEED = <integer>]
<directives_list>::=	[TIMER @|<integer><integer><integer>]
ESCAPE | NOESCAPE	[{<procedural operator >}]
PRINT | NOPRINT	[{<operation >}]
SAVE | NOSAVE	END
EDIT | NOEDIT	< parameter description > ::= VAR <string>
DO | REDO	[X= <integer>]
ASSESS | NOASSESS	[Y= <integer>]
<criteria_description>::=	[WIDTH <integer>]
TYPE SUCCESS|	[COLOURS_ON]
FAILURE|	[INVISIBLE]
PERCENTAGE|	[UNITS <integer>]
SCALE	[VALUE = <real>]
CORRECTION <real>	[STEP = <real>]
MARK <string>	[LIMITS <real> | @ : <real> | @]
2-FROM: <integer1>TO:<integer2>	[NORMAL <real> | @ : <real> | @]
3-FROM: <integer1>TO:<integer2>	END
4-FROM: <integer1>TO:<integer2>	<dependence description> ::= DEPENDS <string>=<expression>
5-FROM: <integer1>TO:<integer2>	<event description > ::= IF < expression > THEN {<action1>} ELSE {<action2> }
6-FROM: <integer1>TO:<integer2>	END
END	

We called the special purpose language developed for task knowledge description *SystemScript*. It can be classified as *internal visual very high-level mark-up* language. In Table 1 a common structure of the generated training session's program and task's subprograms is presented using the Backus-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; **{ }** enclose a construction, which can be repeated; **[ ]** enclose a construction which is not mandatory; **< >** enclose the name of a syntactical construction, which is not yet defined. In addition to administrative data, parameters of the session and the sequence of the training tasks, the program in *SystemScript* includes six global key directives. They are meant to allow the tutor to intervene during the learner's

performance. Their meanings allow or disallow the learner to: redo the task performance (DO|REDO); give up task performance (ESCAPE| NOESCAPE); printing the *.trn* file (PRINT|NOPRINT); saving this file (SAVE|NOSAVE), editing the file (EDIT|NOEDIT), assessing the learner's performance (ASSESS|NOASSESS). When assessment is allowed, an additional block for criteria description is added to the tutor's program. The tutor can choose between four types of the learner's assessment, e.g. SUCCESS/FAILURE, PERCENTAGE, MARK or PROXIMITY. In the last case the intervals of the traditional mark scale have to be pointed out. In such a way the adaptation of the environment to tutor's preferences are ensured.

In a task's subprogram three kinds of knowledge are embedded, namely: declarative, procedural, and operational. *Declarative knowledge* is represented in the form of parameter, functional, and conditional blocks. *Procedural knowledge* can be seen as a sequence of procedural operators for performing a given task in a standard manner after declarative knowledge comparison. The example list of operators 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*. A multiple-choice question could be presented to test whether the learner observing the results of the procedural operator has made the right conclusion. *Operational skills* reflect the way of using declarative and procedural knowledge in order to cope with abnormal situations in pseudo-real time. An author's program for an investigative, control or diagnostic task requires specific learner's actions during its interpretation. Some of the other language constructions serve for flexible control of the simulation process. More detailed information about the language syntax and semantics for task knowledge description can be found in [12].

Table 2. A Sample BT

id	*kw*	*k*	*q*	*p*	*d*	*t*
Task1	simulation, static state, working characteristics, DC motor	2	197	0,36	0,50	9
Task2	examining, transient process, DC motor, DC generator	1	201	0,54	0,50	9
Task3	optimisation, criterion, choose, DC machine, internal parameters, characteristics	5	230	0,47	0,50	9
Task4	exploring, dependencies, effect, external parameters, rotational frequency, relation	3	198	0,36	0,50	9
Task5	monitoring, control, pseudo real time, maintain, rotational frequency	4	236	0,60	0,50	9
	Training session parameters (K, Q, P, D, T):	0.62	1062	0.47	0.50	45

4. Supporting the Base of Tasks

For each task in the training session the *BT* includes the following parameters: identifier (*ID*), set of keywords (*kw*), kind (*k*), knowledge volume (*q*), degree of prompt (*p*), degree of difficulty (*d*) as well as the time expected for its completion (*t*). The analogical parameters of the training session are denoted with corresponding upper letters.

For better illustration of the models proposed in this and next section, Table 2 presents a sample *BT* concerning the well-known system of a DC motor, DC generator and a mechanical connection between them. Some of the parameters have constant values calculated in accordance with the presented formulae, and others have statistical values with initial values shown.

Let Z be the maximal number of the different kinds of tasks, *y(j)* be equal to 1, if the j^{th} kind of task is present in the training session, and to 0, if it is absent. Then the degree of training session variety can be computed as $K = (\sum_{j=1}^{z} y(i)) / Z$. The concept of program tree and the correspondences between its terms and the graph terms was introduced by the authors in [6]. The task performing by the author or learner is viewed as filling in the program tree nodes with keywords, data types, and attributes' values (names, numbers, and text). The numbers of the nodes *V(i)* and links *U(i)* of the tree can serve as a precise and sensitive measure of knowledge volume within the text nodes dimensions, i.e. $q(i)=|V(i)|+|U(i)|\approx 2|V(i)|$.

Let *s(i)* be the number of elements included in the construction SYSTEM, *n(I,j)* – the number of elements included in the j^{th} VAR construction, o_i – the total number of operators in the j^{th} IF-THEN-ELSE construction. If *N(i)* is the number of parameters, *M(i)* – the number of the elements, *k(I,m)* – the number of terminals of element *m*, *Q(i)* – the number of connections, *L(i)* – the number of traced parameters, *P(i)* – the number of functional dependencies, *R(i)* – the number of conditional operators, *I(I,r)* – the number of operators in the THEN-part of the r^{th} IF-THEN-ELSE operator, *j(I,r)* – the number of operators in the ELSE-part of the r^{th} IF-THEN-ELSE operator. The nodes to the left of an *S*, *V*, *E, C* or *T* node contain information associated with the subject domain, while the nodes to the right represent the keywords of the attributes associated with the same construction. Then,

$$q(i)=2\left[\begin{array}{l}2+2s(i)+2N(i)+2\sum_{j=1}^{N(i)}n(i,j)+18M(i)+\\ 7\sum_{ij-1}^{M(i)}t(i,j)+6Q(i)+L(i)+P(i)+3R(i)+\sum_{i=1}^{R(i)}o(i,j)\end{array}\right]$$

Having *N* tasks planned for the training session, the total volume of acquired knowledge would be $Q=\sum_{i=1}^{N}q(i)$. Let $PT_1(i)$ and $PT_2(i)$ be respectively the author and learner's program tree. Let *a(I,j)* be the number of nodes missing in $PT_1(i)$ but present in $PT_2(i)$, *b(I,j)* – the number of nodes present in $PT_1(i)$ but missing in $PT_2(i)$. The following formula *c(I,j) = (q(i) – a(I,j) – b(I,j)) / q(i)* can serve as a precise and sensitive measure of the degree of proximity between the performance of I^{th} task by the j^{th} learner and that of the author. This parameter also varies between 0 and 1. The formula $c(I,j)^*=c(I,j).(\Delta t_1/\Delta t_2)$ that takes into account the learner's time for task performance Δt_2 relative to the author's Δt_1 presents time correction of *c(I,j)*. So by means of the environment parameter *e*, where $(\Delta t_1/e)\le\Delta t_2\le e\Delta t_1$, the learner who is faster than the author could be encouraged while the slower one could be reprimanded. The calculated parameter and the registered time for performance stored in the learner's *.trn* program present the main part of the third-level learner's model [5].The degree of environment prompt determines what part of the author's knowledge is available to the learner when he/she is performing the task. As it is assumed that the domain names are fixed by the author and the keywords by the environment

$$p(i)=\frac{1+s(i)+N(i)+\sum_{i=1}^{N(i)}n(i,j)+9M(i)+4\sum_{j=1}^{M(i)}t(i,j)+3Q(i)}{q(i)}$$

Now if *M* is the number of trainees who attempted to solve the I^{th} task, the formula for the calculation of the task difficulty becomes precise and simple: $d(i)=\sum_{j=1}^{M}c(i,j)/M$. The degree of difficulty of the session is calculated as the average, i.e. $D=(\sum_{i=1}^{N}d(i))/N$, where *N* is the number of tasks included in the session. Initially *d(i)* = 0.5, for every task, consequently *D*=0.5. In practice, the expected completion time for a task is calculated as the average, i.e. $\bar{t}_i=(\bar{t}_i+t_i)/2$ after each task execution by a learner. Similarly, the average time for completing the session is calculated as $\bar{T}=(\bar{T}+T)/2$ after each session completes. Having an initial session duration set by the tutor as *T*, the initial values of *t(i)* are taken to be the same and equal to *T/N*.

5. Planning of the Training Session

5.1. Search-based Models

Dominant model: This model is used to exclude the least appropriate tasks having extreme values. For example, given the criterion *(q=max) AND (p=max*), task 5 in Table 2 will be excluded.

Restrictive model: Used for reducing the set of tasks to a subset by imposing limits on certain parameters. For example, given the criterion *(p>0,5) AND (q>200*) tasks 2 and 5 will be selected.

Keywords: This model uses the *kw* field as a limiting factor. When more than one keyword is given, a logical *AND* is performed on them. The criterion for example, must be (*kw*="characteristics" AND "DC"), which will yield tasks 1 and 3.

Logical formula: This is a more general model than the preceding one, as it allows a combination of AND, OR, NOT operations as well as brackets. For example, in order to find all tasks, containing the terms "characteristics" but not containing "DC", the criterion will be (*kw*="characteristics" AND NOT "DC"), which will yield task 4.

5.2. Sort-based models

Significance model: The criteria are sorted by significance and the tasks are compared with the most significant one. The task, which outclasses the rest according to the first criterion, is selected. If more than one task has the same value, the next in significance criterion is used and so on. For example, having the following order of criteria: *1) d=max; 2) p=min; 3) q=max*, task 4 in Table 2 will be selected, but only after applying the third criterion.

Weighted model: Depending on the parameters' significance, each is assigned a weight. Then selection is based on the following formula: $A(i) = \sum_{i=1}^{n} w(k).p(i,k)$, where: *A(i)* – rating of the *I*th task; *w(k)* – weight of the *k*th parameter; *p(I,k)* – quantitative evaluation of the *I*th task for the *k*th parameter; *n* – the number of evaluated parameters. Supposing for a certain tutor can assign *t* a weight of 5, the weight of *p* can be 10, while those of *q* 0,01. Then the formula for the weighted sum of the *I*th task becomes: $A_i = 5*t_i + 10*Cp_i + 0{,}01*Q_i$, *(I=1..5).* In accordance with the calculated values of A_i , the tasks in the BT are ordered thus: 1, 4, 2, 3, 5.

Ideal model: Based on the idea that it is not mandatory that the significance of each parameter will grow as its quantitative evaluation. Generally, the selection criterion is: $D(i) = \sum_{i=0}^{n} w(k).|\ p(i,k) - v(i,k)\ |$, where the additional notations used are: D_i – degree of tutor's dissatisfaction with the *I*th task; *v(I,k)*– the ideal value of the *I*th task for the *k*th parameter. The lower the value of D_i, the higher the evaluation of the *I*th task. If it turns out that all parameters match their ideal values, then |*p(I,k)-v(I,k)*|=0, and consequently D_i=0. For example, keeping weights from the previous example and ideal values for a given T of *t*=5 and *p*=0, the tasks will be ordered thus: 3, 2, 5, 4, 1.

6. The Current State of the TOEDVL

Only a part of the above-described ideas have been implemented in the current prototype of the TOEDVL. The program generator is designed to run locally as a standalone WINDOWS application. Due to the need of portability and independency of the quality of the network connection status, the interpreter is implemented by means of ActiveX controls. Currently, we are in the process of implementing the Task Manager that allows easily switching between different sets of keywords, i.e. using different spoken languages.

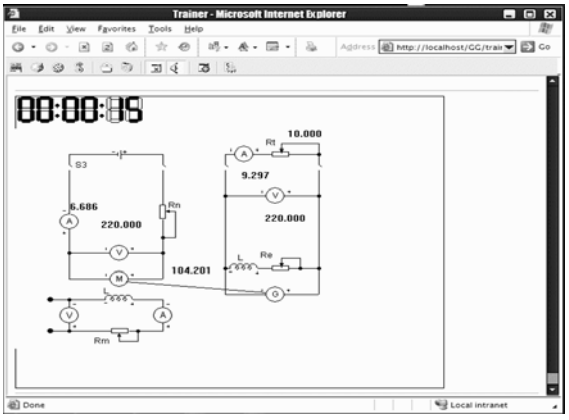

Figure 2. The main window of the program interpreter

A learner's session starts by opening a Web page; in this way the learner contacts the Task manager. After authorization, the learner is presented with the sequence of tasks planned in the *.ses* file. The program interpreter, if not present at the client computer, is downloaded from the server. As it starts interpreting the current *.trn* program, it downloads any additional files, needed for running the current task on the learner's computer. The monitored values of control points appear in the corresponding windows of scheme's bitmap (Figure 2). During execution they are refreshed depending on the model and user interactions. A simulator's clock in the toolbar indicates the elapsed time from the beginning of the simulation. The interpreter-evaluator periodically updates the learner's history file on the server. This file keeps track of all parameter changes, whether due to changes in the status of the modelled lab object, or to trainee's interactions with the model.

The prototype of the TOEDVL has been used in the course of Intelligent Teaching Environments for design of two virtual labs in electrical systems and electronic circuits. The accumulated experience indicates that the proposed environment leads to deep understanding of the teaching material and stimulates learners' motivation and activity in their practical exercises.

7. Conclusion

The problem of adapting a task-oriented training environment to different type of users, (e.g. authors, tutors, and learners) has been discussed. The models for computation of task and session parameters are proposed to support the base of tasks by the authors. Finally, two groups of the tutor's models based respectively on search and sort in the base of tasks are considered. Comparative analysis of the learner and author's program trees produces objective and precise evaluation of the learner's knowledge, taking into account the missing/redundant elements/connections as well as the task completion time.

8. Acknowledgements

The authors wish to thank the reviewer, Prof. Douglas Harms from DePauw University, Indiana, USA for his careful reading of the paper and critical comments and suggestions for its improvement.

Bibliography

[1] I.I. Zheliazkova, R.T. Kolev, Learner Models in the Computer-Based Teaching Systems. A survey, Bulgarian Journal of Information Technologies and Control, 4, 2004, 22-28.

[2] D.R. Peachey, G.I. McCalla, Using planning techniques in intelligent tutoring systems, International Journal of Man-Machine Studies, 24, 1986, 77-98.

[3] E.H. Paschin, A.I. Mitin, Automated Teaching System EKSTERN (MSU, Moscow, 1985, in Russian).

[4] C. Jesshope, E.Heinrich, D-r Kinshuk, Technology integrated learning environments for education at distance, http://www.deanz.org.nz/jeessope.doc

[5] A.J. Gonzalez, L.R. Ingraham, Automated exercise progression in simulation-based training, International Journal System, Man, and Cybernetics,24, 1994, 862-874.

[6] R.W. Chu, G.M. Mitchell, & P.M. Jones, Using the Operator Function Model and OFMspert as the basis of an intelligent tutoring system: towards a tutor/aid paradigm for operators of supervisory control systems, Int.J. System, Man, and Cybernetics, 25, 1995, 1054-1075.

[7] V. Vasandani, T. Govindaraj, Knowledge organisation in intelligent tutoring systems for diagnostic problem solving in complex dynamic domains, Int. J. Systems, Man, and Cybernetics, 25, 1995, 1076-1096.

[8] J. Fleming, The Intelligent Computer-Assisted Training Testbeds (ICATT) Program, 1996, https://www.spider.hpc.navy.mil/html-docs/its-ida/armstrong.htm.

[9] I.I. Zheliazkova, P.L. Valkova, Computer-Aided Teaching and Learning Structural Schemes, Annual Proceedings of the Rousse University, section "Mathematics, Informatics and Physics", Rousse, Bulgaria, 2004 (accepted).

[10] I.I. Zheliazkova, R.T. Kolev, Technology of Using an Intelligent Multimedia Test Generator, Proceedings of the 15th Annual European Conference on Innovation in Education for Electrical and Information Engineering, Sofia, Bulgaria, 2004, 49-58.

[11] I.I. Zheliazkova, R.T. Kolev, A Three-Level Learner's Model for the Needs of an Integrated Environment for Individual Planned Teaching, Proceedings of the International Conference on Computer Systems and Technologies (e-Learning), Rousse, Bulgaria, 2004, IV.13-1- IV.13-6.

[12] I.I. Zheliazkova, G.T. Georgiev, Representation and processing of domain knowledge for simulation-based training systems, International Journal of Intelligent Systems, 2000, 10(3), 255-277.

Author's Information

Irina I. Zheliazkova - Dept. of Computer Systems and Technologies, Rousse University, Studentska Str.8, Rousse, Bulgaria; e-mail: irina@ecs.ru.acad.bg

Georgi T. Georgiev - Dept. of Computer Systems and Technologies, Rousse University, Studentska Str.8, Rousse, Bulgaria; e-mail: gtgeorgiev@ecs.ru.acad.bg

Rumen T. Kolev - Dept. of Computer Systems and Technologies, Rousse University, Studentska Str.8, Rousse, Bulgaria; e-mail: rkolev@ecs.ru.acad.bg