The Multi-courses Tutoring System Design

Goran Šimić

E-mail: gshimic@eunet.yu

The Military educational center for signal, computer science and electronic warfare, Veljka Lukića Kurjaka 1, 11000 Belgrade, Serbia and Montenegro

Abstract. This paper describes architecture, design, and implementation of Multitutor, a Web-based environment for the development the e-learning courses and for the use of them by the students. Multitutor is designed as a Web-classroom client-server system, ontologically founded, and is built using modern intelligent and Web-related technologies. Experience with Multitutor so far shows that both teachers and learners have positive feelings about using them as a support tool for creating and learning different courses. In order to illustrate the design of the system, this paper presents some details of Multitutor architecture.

1. Introduction

The use of current Internet technology to support learning in the classroom is recently getting much easier and much more feasible than it used to be. If a network of computers or workstations is available in the classroom, it is easy to install and use Apache, Orion, Tomcat, or another Web server on a dedicated server machine. These servers are able to distribute HTML pages generated statically or dynamically by an educational application. Client computers/workstations should only have an Internet browser. Hardware and software requirements for the client machines are minimal. The standardized ontological structure of different knowledge (courses) in the area of education exists today.

Today, there are menu ITS[1], for different education tasks: the learning of the DBMS design (ER modeling, normalization), the problem solving (in programming and using some algorithms), conversation exercises (in the tasks of the public relationship), the maintenance courses, the language learning, etc. All of them are domain specialized systems.

The programmers or domain experts design the most ITS systems. Therefore many ITS need some pedagogical and didactical aspects. One solution of this problem is to add the course creating functionality for the teachers. There are

[1] ITS – Intelligent tutoring systems

many different authoring tools. Also our work is related to an authoring tool named Multitutor.

Multitutor is a Web-based framework designed both - for the teacher and for the students. This system enables the teachers to develop tutoring systems for any course. The teacher has to define the chapters, the lessons and the tests. The teacher also has to specify the destination of the course materials (the chapters learning HTML pages). Multitutor is designed to contain an unlimited number of courses, and course packages. The teacher has full responsibility for the overall course design. Multitutor enables the students to learn the specified course, tests them after the learning phase and gives them the recommendations *what to learn for a better score*. The course system navigates the student through the learning space on different ways, based on the historical and actual student results stored in the student model.

2. Related works

As said above, Multitutor has two parts: the teacher side application and the student side application. The teacher side represents the tool for the course development. Therefore this part of the system is designed as an authoring tool.

Generally there are two types of authoring tools [1]. The first is the *easy for the teacher* type and the other one is *easy for the programmer* type. The first type is focused on the description of the course, individual student and pedagogical strategies and techniques. The second type prefers the domain presentation and the learning strategies.

The *Multiutor* is designed to provide the teachers an easy way to create the course. As in the REDEEM tool [2], the teacher can describe the course contents by using the course metamodel. This means that the course would be decomposed on the learning units as the chapters and lessons. These units can be semantically linked in the course body. Like a REDEEM, Multitutor contains the test editor, with possibility of creating the unlimited number of test sets. The tests would be proposed for different levels of learning. The KB[2] contains only the course metadata entered through the course description editor. Multitutor supposes that the authors create courses by some other tools. Also, the system isn't designed to be used over the Web.

The teacher side also contains the administration functionality. The maintaining of teachers and students data is the main administration task. The Multitutor administration module is realized as in the learning management systems WebCT [3]. The administrator maintains the teachers and students

[2] KB - Knowledge base

authentication data. Also the teacher has grants to edit his courses data and view the results of his students. The teacher can create the student groups; attach the students to the groups giving them permission to use the courses. All of the administration tasks are executed on the server side. The users data are well structured by using XML [4] format, and are readable via the Web (using the standard browser).

The student side of application is concrete ITS shell that provides the use of the courses for the students. This way we try to explore the architectural solutions of the Web based ITS. Extensive discussion on categorization of such architectures by Alpert et al. [5], and Mitrović and Hausler [6] was our starting point. They have found out that architectures of many Web-based ITS are either *centralized* (the application server performs all tutoring functions), or *replicated* (the entire tutor resides in a Java applet that needs to be downloaded and is executed on the student's machine), or *distributed* (tutoring functions are distributed between the client and the server). Each category has some advantages and some disadvantages, described elsewhere. For example, Johnson et al. discuss feasibility of client-side tutoring deployed in their pedagogical agent called Adele [7].

We also studied architectures of a number of specific Web-based ITS, in order to find out about their characteristics and to relate them to our idea of Multiututor as a Web-based ITS shell. The systems whose architectures we found most inspirational in designing Multitutor include ActiveMath [8], VALIENT [9], and ILESA [10].

Theoretical work of Brusilovsky [11], as well as ELM-ART [12] system for learning programming in LISP cover a number of important issues related to adaptivity of Web-based learning environments, such as providing adaptive navigation support to the learner, links annotation, and adaptive curriculum sequencing. Based on the functionalities of the ELMART system, Multiututor offers different possibilities of navigation support. The student navigation is changeable and is related to the learning (session) phase and the state of the student model (the student results).

3. Multitutor architecture

The system architecture is very similar to architecture of other ITS environments [13] and can be represented as in Figure 1. The following sections further describe some of its details. We opted for a centralized architecture, since Multitutor had to support the idea of a Web classroom, with a centralized repository of student models and simultaneously support more students. Students and teachers work in a real or in a virtual classroom; in both cases, students learn individually and the Web technology connects the server and the client sides.

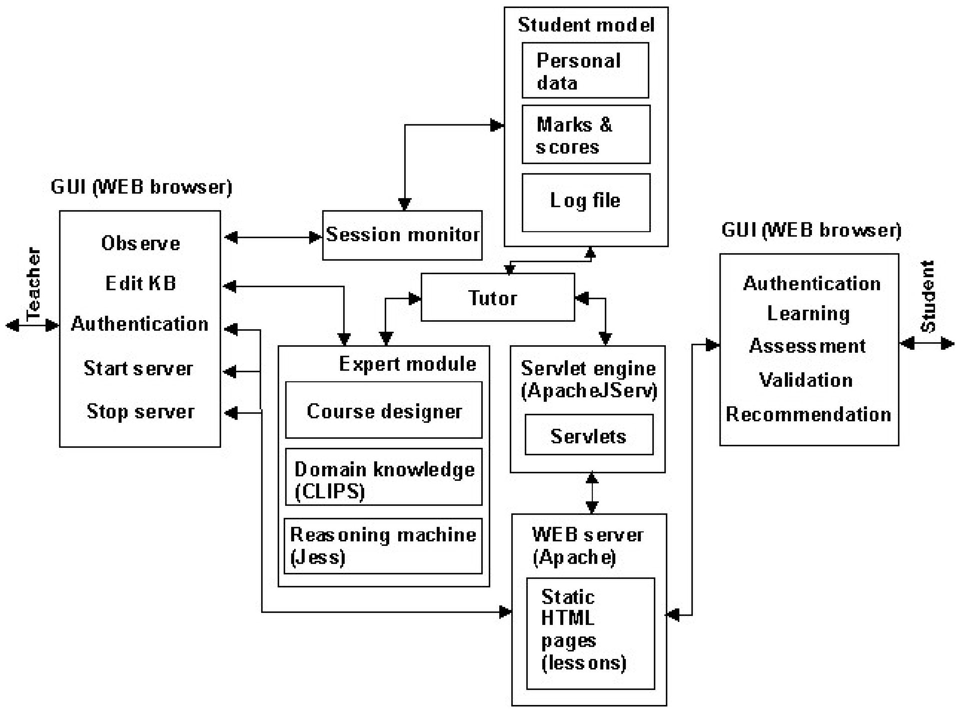


Fig.1. Multitutor's overall architecture

The system has modular nature. The *Student model* contains all students' data. The domain knowledge and the system reasoning are sited in the *Expert module*. This module is enhanced with the authoring tool named *Course designer*. The administration tools are implemented in the *Session monitor*. This component also provides the observing student results to the teacher.

The *Tutor module* is the kernel of the system. The tutor coordinates the processing in all other components of the system architecture. Some pedagogical aspects are incorporated in this module. The student interaction with the system is also through the tutor module.

The front end of the system consists of three entities. The servlet engine and the Web server are sited on the server platform and the Internet browser on the client platform. The changing of the system behavior is expressed through dynamically created HTML pages by the servlet machine. The Web server has to deliver these pages to the students.

The learning materials are represented as statically HTML pages that are copied on the Web server repository in the course creation phase. Not one part of the system limits of the multimedia support.

4. The design of the knowledge base

The structuring of the KB is the start point of the system development. Multitutor is essentially a rule-based system. Its expert module contains the KB in CLIPS [14] format and the Jess[3] [15] inference engine (run by the Tutor module) that interprets the KB. Conceptually, Multitutors' courses are contained in chapters, lessons, and test sets (Figure 2). The whole system is designed using the UML modeling tools and patterns [16]. Every test set contains questions and answers (*quanda*s). These entities have a number of attributes (metadata). After the teacher creates a new course, the system stores the course data and metadata in the XML format.

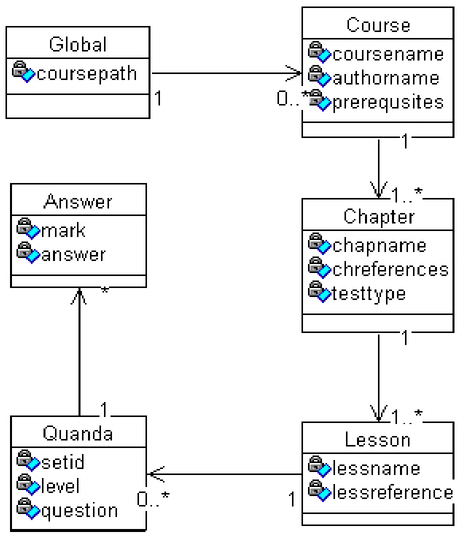


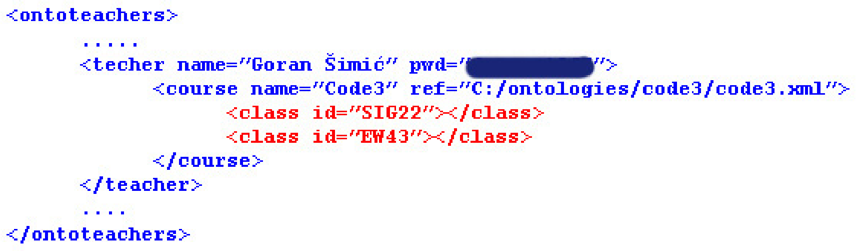
Fig.2. KB structure

The *Global* class holds the courses, groups (student classes) and course paths. There is only one global class in the system. The instances of the *Course* class are related to the global class. Every course has a name, an author and some prerequisites (related courses). If the student selects the course, the system checks if he/she passed the prerequisites. One or more instances of the *Chapter* class are related to an object of the *Course* class. The chapter has a name, references (the learning materials for different levels), a test type and one or more associated lessons. The attributes of the *Lesson*

[3] The second reason for the centralized architecture of the Multituor is the recommendation of the Jess manufacturer: Jess package is over 500KB, and mobility containers (as applets, or agents) would disturbe the system performance.

class are the lesson name and the learning material reference. The class named *Quanda* (a question and answers) represents the test sets. Each quanda is specified for appropriate level and has a unique id number. Instance of *quanda* implements a question and includes the question string and a collection of answer objects. An object of the *Answer* class contains the answer string and the appropriate mark.

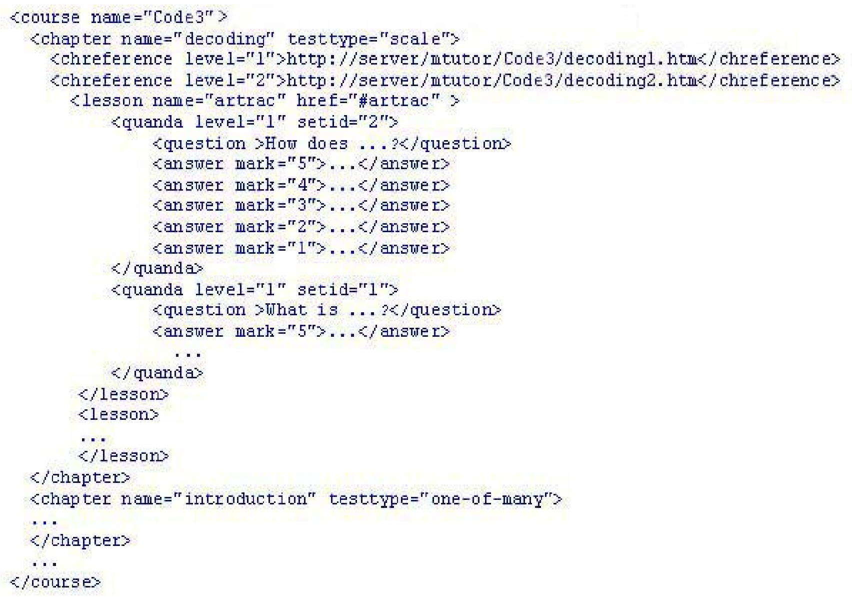
One file contains the basic system data (Figure 3). There are the teachers' names, the paths of the courses and the names of the classes (student groups). When the user (the teacher, or the student) runs the system, the first system action is the reading of the contents of this file.



```
<ontoteachers>
      .....
      <techer name="Goran Šimić" pwd="[redacted]">
            <course name="Code3" ref="C:/ontologies/code3/code3.xml">
                  <class id="SIG22"></class>
                  <class id="EW43"></class>
            </course>
      </teacher>
      ....
</ontoteachers>
```

Fig.3. The global metadata that Multitutor uses on the startup

The system is ready to use after the data loading is finished. The first action of the user (teacher or student) is logging on the system. Multitutor then checks their data (names, passwords and the student class), and decides which courses the system offers to the user. After the student selects the course, the system follows the course path and opens the course XML file (Figure 4).



```
<course name="Code3">
  <chapter name="decoding" testtype="scale">
    <chreference level="1">http://server/mtutor/Code3/decoding1.htm</chreference>
    <chreference level="2">http://server/mtutor/Code3/decoding2.htm</chreference>
      <lesson name="artrac" href="#artrac" >
          <quanda level="1" setid="2">
              <question >How does ...?</question>
              <answer mark="5">...</answer>
              <answer mark="4">...</answer>
              <answer mark="3">...</answer>
              <answer mark="2">...</answer>
              <answer mark="1">...</answer>
          </quanda>
          <quanda level="1" setid="1">
              <question >What is ...?</question>
              <answer mark="5">...</answer>
                ...
          </quanda>
       </lesson>
       <lesson>
       ...
       </lesson>
  </chapter>
  <chapter name="introduction" testtype="one-of-many">
  ...
  </chapter>
  ...
</course>
```

Fig.4. The example of the course metadata

Multitutor then fills the middle layer classes with the concrete course data. After the students' logging, the student model is filed with the historical data of student results. The tutor module correlates these data with the course data and decides what chapters, level of learning, learning materials, test types and test sets should be used in the student session.

As said above the Jess shell is used as a reasoning machine. Only the CLIPS formatted files are readable for this component. Therefore, the XML/CLIPS translator is implemented in the course designer (the part of the expert module). The system translates XML data and generates the separate KB file for each chapter of the course. These serialized data are stored in the same folder as the course XML file. This way the system easily finds the appropriate KB file in the time of execution.

The KB file consists of the global variables, two templates and the rules. One template is named *Setup* and is described in the next paragraph. The other one is the chapter template. As the template represents the chapter, the pair of the template slots represents a lesson of the chapter. One slot is designed to receive the student answer, and the other is for the answer mark. The KB contains the rules, since the Jess inference engine interprets rules. There are five types of rules in the Multitutor.

The *startup* rule initiates the operation of the reasoning machine (Figure 5). There is only one start rule per chapter (KB file).

```
(defrule startup
  =>
  (bind ?*a* (assert(Setup(setnum(fetch SETNUM))(level(fetch LEVEL)))))
  ( set-reset-globals nil )
)
```

Fig.5. Startup rule

This rule hasn't a premise, only the action part. The start rule, like the listener, waits for the learning level and *setnum* to be fired. The second variable represents how many times the student passes the same chapter through the session. The reasoning machine fires a startup rule when the system sets the global variables (*SETNUM* and *LEVEL*). This event causes that the template slots are filled (on the figure the template is named *Setup* and slots are *setnum* and *level*) with the values received from the environment (from the tutor module).

The second type is the *queries-and-answers* rule (Figure 6). This rule displays questions to the student.

```
(defrule querysandanswers1
  ?fact1<-(Setup(setnum ?s)(level ?q))
  (test(eq ?s 1))
  (test(eq ?q 1))
  =>
  (store LESSONCOUNT 2)
  (store LESSON1 "decoding")
  (store QANDA1 "What is...?/answer1/answer2/.../.../...")
  (store LESSON2 "introduction")
  (store QANDA2 "...?/.../.../.../.../...")
      ....
  (bind ?*a* ?fact1)
  (set-reset-globals nil)
)
```

Fig.6. The rule that contains a test set data

The precondition of this rule is that the slots of the *Setup* template are fulfilled. The number of these rules is the same as the number of test sets for the specified chapter. The recommendation for the teacher is that the chapter should be composed of more than one test set. This prevents the student from solving the same test set two or more times during the same session. The action part of the rule sends the number of lessons in the chapter, the lesson names and related questions and answers (the composed string named *quanda*).

The third rule type is a *receive-answer* rule (Figure 7). This rule collects the students' input when he is answering questions. The receive-answer rule is similar to the startup rule. There is only one receive-answer rule per chapter (KB file). The receive-answer rule sets the template slots with the student answers.

```
( defrule receiveanswer
  =>
  (bind ?*u* (assert(decoding(f1b(fetch ANSWER1))(ax25(fetch ANSWER2))..)))
  (set-reset-globals nil)
)
```

Fig.7. The rule that receives the students' answers

The next type in the sequence are the *working* rules. These rules evaluate the students' answers (Figure 8). In the IF clause the inference machine checks the slots values. In the case of pattern matching, one of the working rules is fired per every student answer. The rule firing sets the value of the mark slot of chapter template. This means that every lesson in the chapter is represented with the one question during the test.

```
( defrule validate_f1b114
  ?fact1<-(decoding(f1b "One chanel freq.shift keying")(of1b nil))
  ?fact2<-(Setup(setnum ?s)(level ?q))
  (test(eq ?s 1))
  (test(eq ?q 1))
  =>
  ( modify ?fact1 (of1b 4))
  ( bind ?*u* ?fact1 )
  ( set-reset-globals nil )
)
```

Fig.8. The example of the working rules

The last rule in the sequence is the *goal* rule. This rule calculates the students' final score (Figure 9).

```
( defrule goal_rule
  ?fact<-(decoding(of1b ?x)(oax25 ?o)(odecoding ?f))
  (test(neq ?x nil))
  (test(neq ?o nil))
  (test(eq ?f nil))
  =>
  (bind ?f (round(/(+ ?x ?o)?*i*)))
  (bind ?*u* ?fact )
  (set-reset-globals nil )
  (store MARKNUMBER (+ ?*i* 1))
  (store 01 ?x )
  (store 02 ?o )
  (store 03 ?f )
)
```

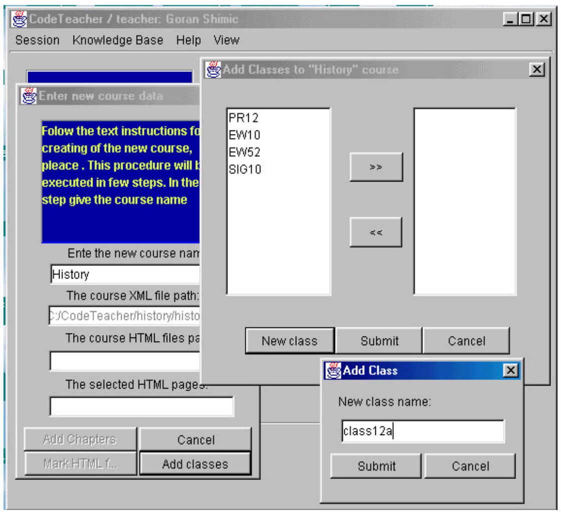
Fig.9. The goal rule

The goal rule is fired when the inference engine assesses every student answer. There is one goal rule in one KB file. While the previous rule receives or processes the input data, the goal rule sends the results of inference to the environment (the tutor module). The system receives the results and updates the student model. This affects on the future system behavior against the student.

5. The new course designing

The system requires that the learning materials should be created in the HTML format. As in the REEDEM tool [2], Multitutor expects from the teacher to create the learning contents with some other tools (visual HTML editors like *FrontPage*, or some text editor like MSWord that is possible to save the document in the HTML format).When the teacher starts to design a new course, he has to link these materials to the course. Our recommendation is to create at least one HTML page per course chapter.

At the begining of the new course design, the system activates the course wizard. In the first dialog (Figure 10a), the teacher specifies the name of the new course. In the next step the teacher associates the classes to the new course.


CodeTeacher / teacher: Goran Shimic

Enter new course data

Folow the text instructions fo
creating of the new course,
pleace . This procedure will b
executed in few steps. In the
step give the course name

Ente the new course nam: History

The course XML file path: C:/CodeTeacher/history/histo

The course HTML files pa:

The selected HTML pages:

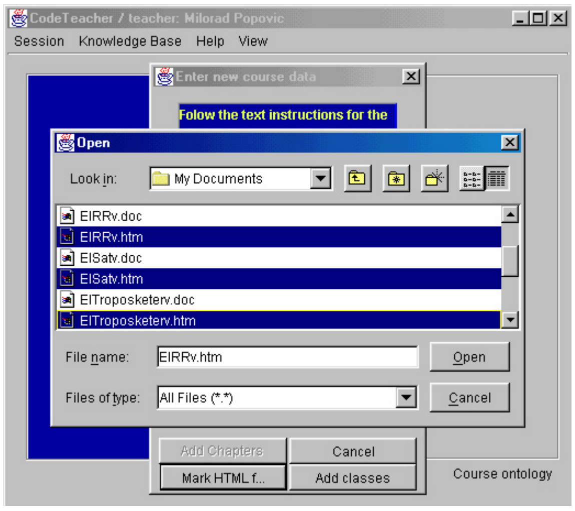
Add Classes to "History" course

PR12
EW10
EW52
SIG10

Add Class

New class name: class12a

a) Connecting the classes to the new course


CodeTeacher / teacher: Milorad Popovic

Enter new course data

Folow the text instructions for the

Open

Look in: My Documents

EIRRv.doc
EIRRv.htm
EISatv.doc
EISatv.htm
EITroposketerv.doc
EITroposketerv.htm

File name: EIRRv.htm

Files of type: All Files (*.*)

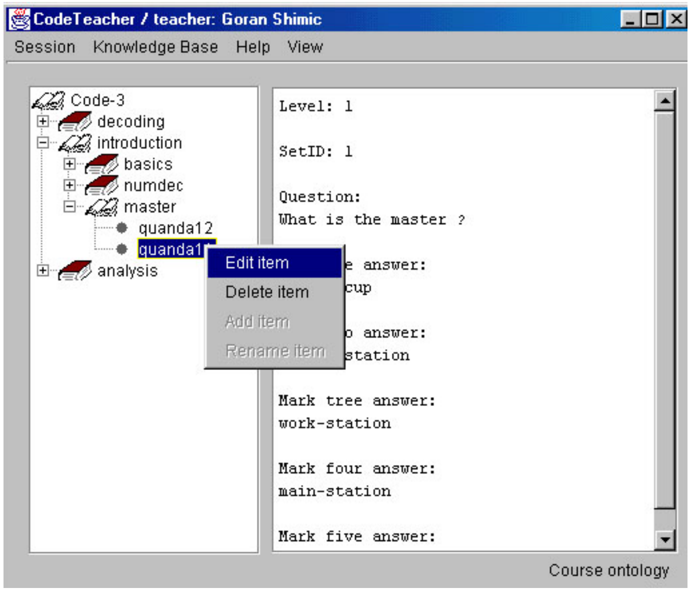
b) Connecting the learning materials to the new course

Fig.10. The teacher interface for the course design

The learning materials represent the other initial data of the new course. In the new dialog the teacher selects all HTML pages that are designed for the course. Multitutor copes these files on the new created course folder at the Web server repository.

The teacher links these pages with the chapters in the next phase of the course design. Then the course wizard directs the teacher to specify the chapters. After the teacher enters the chapter name and test type (that will be implemented in the chapter tests), he has to select one page from the list. The system saves this selection as the chapter reference. The same action occurs when the teacher creates the chapter lessons. This way, the page bookmarks fill the list of the lesson references. After the chapter material (page) is selected, Multitutor extracts all bookmarks from this page. The bookmark list is presented on the new lesson dialog. The last phase of the course design is the defining of the test sets.

Multitutor visualizes the course by the tree structure (Figure 11). There is the course global data in the root. The course branches on the chapters, chapters on the lessons, and *quandas* are the leaves of the tree.



```
Level: 1

SetID: 1

Question:
What is the master ?

e answer:
cup

o answer:
station

Mark tree answer:
work-station

Mark four answer:
main-station

Mark five answer:
```

Fig.11. The visualized structure of the course

On the right pane the system shows selected item's data that can be edited. The teacher opens the appropriate dialog by selecting an item from the popup menu.

After the course is designed, the student can use it. He/She logs on the system, customizes the interface, selects the course and chapter, and starts the learning. The system limits the student selections based on previous sessions and results. The student can navigate through the pages and learn. The learning materials of the higher level aren't available to the student.

Learning a chapter is followed by an appropriate assessment. The student results affect the system behavior. If the student fails, he has to repeat the same chapter. There are three options for a student who completes a lesson successfully. The first one is to choose another lesson, and the process is the same as described in the previous paragraphs. The second one is to repeat the same chapter and try to get a better score. In this case, the new score overwrites the previous one. The last one is to learn the new level of the same chapter. An intelligent manifestation of the Multitutor is that the system helps the student by recommending him what to learn next.

6. Evaluation

Multitutor is used in two courses. One course is in domain of radio-communications, and other is about the human possibilities of adaptation (psychology). The learners are high school students. The students get a questionnaire before and after using the system. The first questionnaire is about their abilities and experiences in the use of computers and e-learning systems (Figure 12). About 80% of students used the computer. Most of them have experiences and use of the Internet. Usually they use the PC for entertainment.

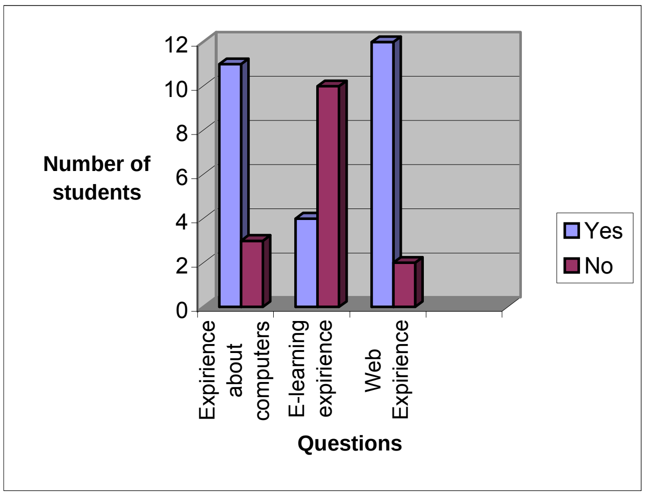


Fig.12. The results of the first questionnaire

The second questionnaire represents the students' impressions about the system. One of the questions is *What are your general impressions about the* *system*. The students typed their answers in a free form. We grouped these answers in three main categories (Figure 13).

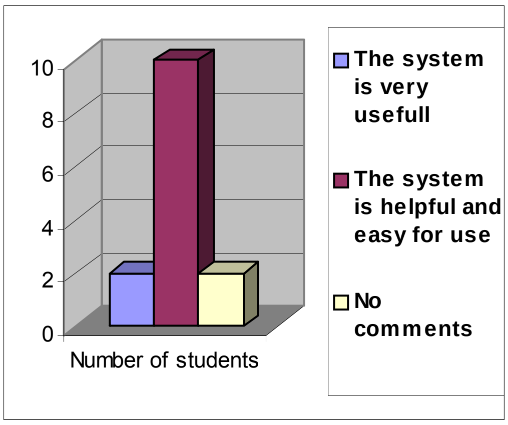


Fig.13. The student impressions about the system

The students' opinions about the system are very positive. They prefer the free navigation through the learning space and individualized learning. The students accent their active participation in the learning process. Also, they have considerable suggestions for the system improvement: more questions in the tests, more test sets and more multimedia contents in the learning materials. Multitutor has no limits for the number of test sets, questions or the learning materials contents.

7. Conclusions

The first two versions of the system have no possibility of the course design. There are only the student functionalities. The present version of Multitutor is developed and implemented using the latest technology. Big technological changes would not mean big changes in Multitutor. The changes would only affect parts of the system. The object-oriented design contributes to the system extensibility.

One of the first steps in further development of Multitutor is to make it capable of learning by itself. This means that the system should be able to interlink the teacher's input based on the keywords [17] of the learning units (course, chapters or lessons) and generate the appropriate XML file(s). This way the system should be able to construct a semantical web between different KB, and to improve reusability of the learning units. For these purposes we have to improve Multitutor ontology support. The number of formatting standards like DOM, OWL, OIL, RDF, RDFS, provides the realization of this goal.

The system is scalable because there are no limitations of the number of courses, the course chapters, the chapter lessons and the test sets. Multitutor represents the domain independent authoring tool and environment that provides different services for the course composing and learning. The learning materials are created as the statically html pages, but if the teacher has spirit and certain skills, he can add the miscellaneous multimedia contents. Multitutor incorporates the links in the learning pages before they are copied on the Web repository. The links associates the learning materials with the rest of the system[4].

The second task in the future is to add the collaboration learning support. While the student is in the learning phase (before assessment), the system should be able to provide some collaboration tools like chats, mailing and whiteboard. This way the student can help other students and vice versa.
The extended functionalities complicate the data storage structure. More data require the DBMS. In other words, there would be one more layer in the system. The data translation from the DB format to the XML format is incorporated into the new DBMS. Our opinion is that in the Multitutor design the hardware requirements for the server side are minimized. Three servers (DB, Web and servlet engine) would overcharge the system. Therefore, the services of the new Multiutor version require better hardware profile. The modularity of the architecture provides us an easy way to improve the system possibilities.

8. References

1. Murray, T. Woolf, B.: Results of Encoding Knowledge with Tutor Construction Tools. In Proceedings of the Tenth National Conference on Artificial Intelligence, San Jose, CA, 17-23. (1992)
2. Major, N.: REDEEM: Creating Reusable Intelligent Courseware. In Proceedings of The International Conference on Artificial Intelligence in Education, Greer, J., ed., AACE, Charlottesville, VA, 75-82. (1995)
3. WebCT, http://www.webct.com. (2003)
4. W3C Recommendation, http://www.w3.org/TR/2001/REC-XMLschema-1-20010502/. (2003)
5. Alpert, S.R. et al: Deploying Intelligent Tutors on the Web: An Architecture and an Example. In International Journal of Artificial Intelligence in Education, Vol.10, 183-197. (1999)
6. Mitrović, A. and Hausler, K.: Porting SQL-Tutor to the Web. In Proceedings of the International Workshop on Adaptive and Intelligent Web-based Educational Systems. Montreal, Canada, 50-60. (2000)
7. Johnson, W.L. et al: Pedagogical Agents on the Web. In Proceedings of The Workshop on Web-Based ITS. San Antonio, TX, USA, electronic edition. (1998)

[4] The servlet tehnology is used to generate the student interface. The meta tags are the bridges between the servlets and HTML pages on the Web server.

8. Melis, E. et al: ActiveMath: A Generic and Adaptive Web-Based Learning Environment. In International Journal of Artificial Intelligence in Education. Vol.12, 385-407.(2001)
9. Hall, L., Gordon, A., Synergy on the Net: Integrating the Web and Intelligent Learning Environments. In Proceedings of The Workshop on Web-Based ITS. San Antonio, TX, USA, electronic edition. (1998)
10. López, J.M. et al: Design and Implementation of a Web-based Tutoring Tool for Linear Programming Problems. In Proceedings of The Workshop on Web-Based ITS. San Antonio, TX, USA, electronic edition. (1998)
11. Brusilovsky, P.: Adaptive Educational Systems on the World Wide Web. In Proceedings of The World Congress on Expert Systems Workshop on Current Trends and Applications of Artificial Intelligence in Education. Mexico City, 9-16. (1998)
12. G. Weber, Brusilovsky P.: ELM-ART: An Adaptive Versatile System for Web-Based Instruction, In International Journal of Artificial Intelligence and Education, Vol.12, 351-384. (2001)
13. Beck, J., Stern, M., Haugsjaa, E.: Applications of AI in Educationhttp://www.acm.org/crossroads/xrds3-1/aied.html. (2000)
14. CLIPS tool, http://www.ghg.net/clips/CLIPS.HTML.
15. 1. Jess – Java ES Shell, [1] http://herzberg.ca.sandia.gov/jess/
16. Larman C.: Applying UML and Patterns. NJ: Prentice-Hall, Englewood Cliffs. (1999)
17. Devedžić, V.: Understanding Ontological Engineering. Communications of the ACM, Vol. 45, 136-144. (2002)

Goran Šimic (Shimic) graduated electronic warfare on the Military Academy, Belgrade and he is a teacher on The Military Educational Center for Signal, Computer Science and Electronic Warfare. He is post-graduate student on The Faculty of Business Administration (Department of Computer Since), University of Belgrade. His current interests are in the field of Expert Systems, Web Design&Applications and Intelligent Tutoring Systems. Other fields of interest are OOA, OOD (UML) and OOP (C++, Java), XML.