Architecture of a System for Interactive Training and Testing in Algorithms and Data Structures

Miloš Milivojević, Đorđe Đurđević, and Milo Tomašević

Abstract — **In this paper we propose the architecture of a new software system for self-directed interactive learning and assessment in the domain of algorithms and data structures. The system extends Visual Simulator of Algorithms (VSA), a tool previously developed at the School of Electrical Engineering, University of Belgrade. The system provides control of trainee's independence level, flexible input data set assignment and configurable automatic assessment of test-taker's actions. The paper describes the architecture of the proposed system and gives relevant implementation details.**

Key words — **automatic assessment, interactive training, knowledge testing.**

I. Introduction

SOFTWARE systems for training and assessment represent valuable means in education. On one hand, they provide repetition of lessons (i.e. training procedures) to the extent necessary to the trainee. On the other hand, after completing the training cycle, they provide knowledge verification, with optional automatic assessment. The use of such systems in these activities reduces the average time the instructor has to devote to persons intended to be trained or tested. Furthermore, the instructor may not even have to be present at the time the activity takes place, allowing him to spare more time for the activities that require greater attention. Also, advanced systems provide tools for test results analysis, which can help the instructor with important guidelines about how to adjust the contents of lessons and tests.

The most common systems for training and knowledge testing (*virtual learning environments*) [1] are based on questions with answers that the subject has to type in or predefined answers. However, this approach is suited only for questions in domains where simple knowledge reproduction is satisfactory, since it ignores the procedure of getting the answer. Moreover, the approach using predefined answers allows the subject to guess the right answer.

The previously discussed approach to giving the answers is not appropriate for the tasks that are common in the domain of algorithms and data structures. In this domain, knowledge about a specific algorithm, as well as its correct application, actually represent the answer. Consequently, a subject that has little knowledge about a specific algorithm and a subject that makes an error in the last algorithm step would be assessed in the same way.

In this paper we propose the architecture of a system for self-training and testing in the domains where the procedure of solving a problem is of great importance. Although the system is primarily developed as a software tool for improving the quality of the course in Algorithms and Data structures at the School of Electrical Engineering, University of Belgrade, the proposed architecture allows a wider range of applications.

The rest of the paper is organized as follows. In Section 2 we present the problems that an automated system for training and testing has to solve. We also give a brief overview of the existing systems and point out some of their undesirable traits which the proposed architecture intends to eliminate. In Section 3 we discuss the desired functionalities of the proposed system and present the most important architecture details. In Section 4 we give some implementation details and system prototype structure, with a brief overview of the employed technologies. In Section 5, we draw the conclusions and point out some directions for further research.

II. Problems and Existing Solutions

In this Section we give an overview of the problems in the domain of automated training and knowledge testing and give a brief overview of existing solutions.

The use of software systems for training and knowledge testing in education has several desirable implications:

- training is conducted without or with minimal involvement of the instructor, who can redistribute his time and effort to activities that require more attention,
- training can be adjusted to the knowledge level of the trainee, while the trainee can repeat the training procedure as many times as required in order to master the lesson,
- testing results are available immediately after the test is finished, independently of the number of subjects,
- training can be carried out at distance, which effectively reduces the need for physical presence (for both subjects and instructors) and increases the time of learning material availability.

The existing systems for knowledge testing, with the most important representatives given in [2], traditionally consist of a set of questions, prepared in advance, to which the subject being tested gives the answers by selecting one (or sometimes more) of the offered answers or by typing

Miloš M. Milivojević is a student enrolled in a Master's programme at the Department of Computer Science and Informatics, School of Electrical Engineering, University of Belgrade (e-mail: milivojevic.milos@gmail.com).

Đorđe M. Đurđević and Milo V. Tomašević, School of Electrical Engineering, University of Belgrade, Bulevar kralja Aleksandra 73, 11120 Beograd, Srbija; (tel: 381-11-3218-385; e-mail: zorz@etf.bg.ac.rs, mvt@etf.bg.ac.rs).

in the correct answer in form of a short text. Sometimes parameterization is present in the questions, which allows the verification of the level of understanding, instead of simple knowledge reproduction. An example of parameterization is given in [3]. However, such a knowledge verification method induces several important problems. Firstly, it is inadequate to assess the tasks that require the knowledge of an algorithm because, in case of an incorrect answer, it cannot take into account the level of algorithm conversance demonstrated by the subject. Also, in case of a correct answer, it is impossible to tell whether the subject actually knows the answer, guessed the answer or found the answer by making several errors (that somehow cancelled each other). Secondly, limited parameterization implies that the instructor has to create a larger number of similar questions on the same topic. Finally, when creating a question, the instructor also has to give the correct answer. This greatly reduces the possibility of training, as the subject can only rely on the materials (questions) provided by the instructor. It also significantly reduces the possibility of self-testing, when the subject defines and solves the problem on his own. The motivation for designing the proposed architecture of a system for training originates from the fact that the existing publicly known systems do not provide an adequate solution for the abovementioned problems.

It is very desirable for an educational system to allow autonomous interactive *passive*, *controlled,* and *self-directed* training. Passive training implies that the subject observes an automated procedure of algorithm execution, in steps, with the possibility to control the execution speed and to jump (possibly back) to a specific execution step. Controlled training implies that the subject tries to solve the problem by manually indicating execution steps, while being given hints that help deducing the next step. Self-directed training implies that no help is given to the subject. Also, except for the passive training, a warning message is issued as soon as an error in the execution steps given by the subject is detected. Essentially, testing is the same as self-directed training, but with no warning messages.

In the remainder of this section, we give a brief overview of previously mentioned systems that support QTI standard [4]. As indicated in [5], QTI defines a data model that is used for representation of questions, answers, assessments, results and aggregation of assessment units in order to create tests. The specification is designed to support simple questions and test materials as well as complex ones. Data sharing between different assessment systems is possible, due to the XMS schema implementation.

Moodle (Modular Object-Oriented Dynamic Learning Environment) [6] is a free, PHP based, *open-source* web application for e-learning. Since April 2011, Moodle's user data base contains more than 54000 verified sites, that serve more than 42 million users in more than 4.5 million courses. It is mainly used by educational institutions. *Dokeos* [7] is another *open-source* web e-learning application, also developed in PHP. Currently, it serves more than 3 million users worldwide. It is used in educational institutions, multinational companies, health institutions and state administrations. *Sakai Project* [8] is a free, *community source* application written in Java, intended for teaching, research and collaboration. It originates from the cooperation of several US universities. Currently, it is used in more than 160 educational institutions by about 200000 users. *OLAT* [9] is a free, *open-source* web application written in Java. Currently there are 150 verified installations, with more than 170 000 users in 8 000 courses.

To the best of our knowledge, none of the mentioned systems provides all the functionalities that the desired system should have, such as controlled or self-directed training and testing in the domains where the procedure for solving a problem is of great importance. The authors are not aware that the architecture of such a system is publicly available. Consequently, it represented the motivation for developing a system for autonomous interactive passive, controlled or self-directed training and knowledge testing in the domains where the knowledge of procedures is valuated with a configurable method. It should be noted that the sequence of activities that represent the correct answer does not have to be unique.

III. System Description and Architecture

In this Section we present the architecture and capabilities of the system for assessment and training in the field of algorithms and data structures, which is the topic of this paper.

In essence, the proposed system provides a means of conducting a part of practical training as an interactive computer session where the simulator, which is used by students, acts as a personal trainer. The obligation of teachers is to provide the students with a set of problems over which they have to learn and test their knowledge, and also to provide them with the test used for assessing the acquired knowledge. Student's work is recorded and can give teachers the insight into segments of working material that the students tend to adopt at a slower pace, which is particularly important in the learning mode. In the knowledge testing mode, test access is provided only to those students who belong to a group of users for whom this test is intended. The scoring of students' work is done by comparing the given responses to the expected ones, where the system comes up with the expected response by itself by executing the given algorithm in accordance with the selected input parameters. Based on the recorded score, the grade is given by a software grading component, selected by the teacher.

The system consists of a server and a client side. The server side includes one application (the server), while the client side consists of three independent applications: student, instructor, and administrator application. The main reasons for having three independent client applications are to reduce the complexity of implementation, and to increase the overall system security. Given the technology selected for the development, reverse engineering of the client application

could reveal certain implementation details that would allow cheating. With three separate applications, out of which only the student application would be made publicly available, the chances for undermining the system security by potentially malicious users are substantially reduced. Component diagram showing the main parts of the system is given in Fig. 1.

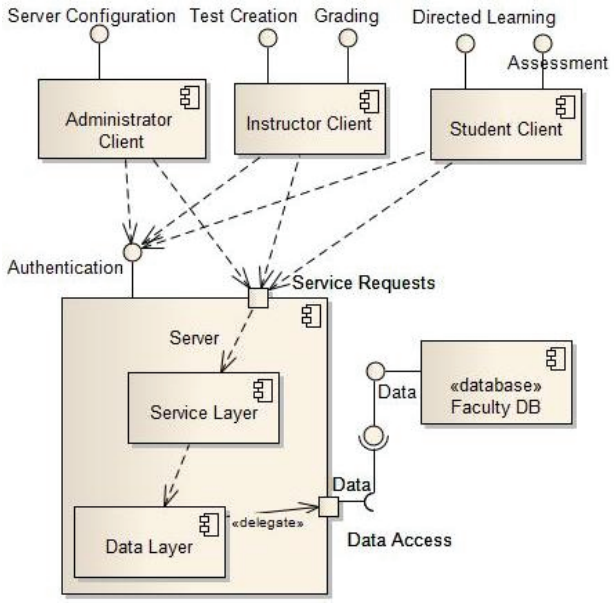


Fig. 1. A component diagram showing the main components of the system - three clients, a server and a faculty database. Client components provide server configuration, creation and grading of tests, directed learning and assessment. The server provides authentication and processing of service requests, which rely on the communication with the database.

The server application is used for authentication and authorization of the clients and for processing the client requests. In case of a prolonged inactivity of the client, it may also terminate the client session with prior sending of an adequate warning.

For an increased overall security and flexibility, system architecture has been organized into three layers: presentation layer, service layer, and data access layer. Presentation layer is responsible for displaying the graphical user interface and is implemented in the client components (see Fig. 1). Service layer defines the functionality of the system. By introducing the service layer we are bridging the gap between the presentation layer and the data access layer, which leads to better component decoupling and, subsequently, to greater system flexibility and security. Data access layer is responsible for the persistence of information through communication with the database.

The database (see Fig. 1), with which the information flow is conducted exclusively through the data access layer within the server application (see Section IV), contains a repository of questions and algorithm input parameters, a list of all currently active tests, information pertaining to student groups, data on users, statistical data, and temporary backup data of the tests currently in progress.

The administrator application is responsible for configuring the server application, as well as the organization of users (e.g. adding or deleting a teacher) and user groups (e.g. adding users to a group intended for potential "guest" users). There are two types of guest users - anonymous and authorized guests. Anonymous guests are not logged onto the system, and the only functionality available to them is passive training and an overview of public tests. Authorized guests are logged onto the system and have access to most of the student functionalities, with the exception that those functionalities apply only to public tests.

The instructor application is responsible for the creation and maintenance of student groups (primarily related to the courses), question creation, putting the tests together, publication of the tests, and also for reviewing the usage statistics and grading of the test scores. A question is defined by selecting an algorithm from the desired area and setting the algorithm's input parameters (either by selecting them from the repository or by creating new values). The input parameters include the initial state of the data structure over which the algorithm is executed, and also the parameters that define the behavior of the algorithm itself. For instance, for sorting algorithms, a relevant parameter would be the sorting order (ascending or descending). It should be noted that the system also supports the traditional way of defining the questions (e.g., in the form of multiple choice textual questions), but that is not the subject of this paper. A test consists of any number of questions. When adding questions to the test, the teacher is allowed to choose questions from the repository of existing questions, or to define a new one. For each question, the system keeps the account of the tests in which it has been used, giving the teacher an insight into the usage frequency of any particular question. When publishing the test, the teacher designates the student group to which the test pertains, the duration of the test, the test activation date or the duration of its availability, and the type of the test (for student evaluation purposes or for independent practice). The teacher defines the grading method of student test scores by selecting one of the available configurable grading modules. The test is considered published after the first user gains access to it (the first fetch from the repository). A published test is automatically "locked" and it is not possible to make any subsequent changes to it. The teacher can invalidate the test, which prevents the download of the test by the students. Also, it is possible to clone a test, which allows the teacher to make similar tests, with minor changes to their contents and purposes (to whom they are intended and in what manner).

The student application is designed for training and assessment, and it provides interactive work (the display and execution of tests) in the graphical user interface which employs the application developed in [12] and [13] as its basis. After the authentication, the student application retrieves a list of available tests from the

server, out of which the student may choose one to solve. The student application automatically downloads all data necessary for running the test, including the implementation of algorithms. It should be noted that the application can be run independently, without connecting to the server application. The application is designed to run in two modes: training and assessment. As noted in Section II, training can be passive, controlled or self-directed. Testing can be either self-testing or graded testing. Self-testing is different from graded testing in that its duration is not time restricted. In this mode, the student progressively solves the test, question by question, without assistance, and then, at the end of the test, he is shown the test results in the form of a percentage match between his actions and the required ones, and the number of errors made, grouped by their severity.

The component in charge of determining the percentage match between the user's actions and the required ones is the algorithm implementation, specified during question creation, and the component in charge of assessing the error severity is the implementation of data structures over which the algorithm is executed. The test scores are stored into the database, where they are available to the teachers responsible for their grading.

IV. Implementation Details

In this Section, we present the details of implementation of the prototype application that was used to validate the system architecture proposed in this paper, and outlines the technology used.

Connection of the client applications to the server is based on a built-in library of the Java programming language, intended for the remote method invocations (RMI). RMI has a convenient feature that allows the communication between two classes located on different computers (and whose methods are, therefore, executed in different Java Virtual Machines) to be performed transparently, in the same manner as if the classes were on a single computer.

For database access and data manipulation we used *Hibernate* (version 3.6.4), the popular open-source software framework for object-relational mapping. It is an improvement over the traditional approach primarily in that it allows easy and straightforward mapping of the database model (relational model) into the object model (classes that are used by the system), with the possibility of writing queries programmatically, using framework's own classes for expressing different query criteria.

Integration issues, security and transaction management are implemented using *Spring Framework* (version 3.0.5), an open-source application framework for the Java platform. The *Spring Framework* comprises several modules that provide a range of services, out of which the following are of interest in this paper: inversion of control, aspect-oriented programming, database access, transaction management, remote method calls, authentication and authorization and testing.

Java SE2 1.6 and NetBeans (version 7.0), an open-source integrated development environment for the Java platform, were used for the development of the prototype application.

Fig. 2 shows a simplified UML class diagram comprising key classes of the server application. *Server* interface provides a communication API that must be respected in the implementation of the server and client classes. *ServerImpl* class implements the *Server* interface. Its role is to authenticate and authorize the client applications, and provide them access to the functionality through the service classes.

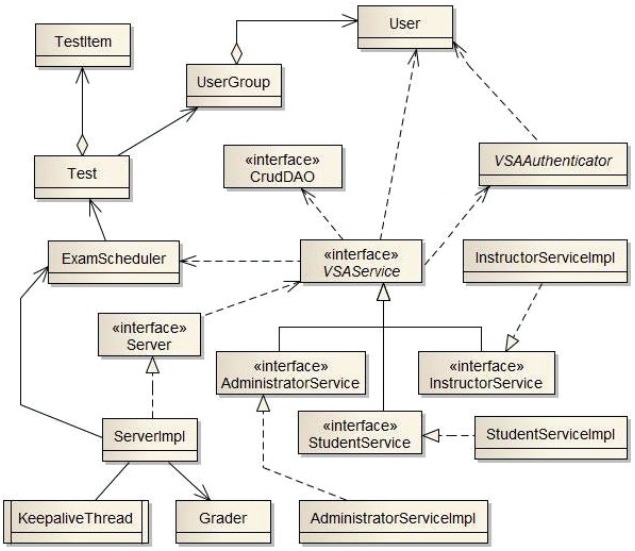


Fig. 2. Simplified diagram showing the core classes of the server application. *VSAService* class and interface hierarchy form the service layer of the application, while the remaining classes either represent the abstractions of the domain or serve to integrate the service API with the rest of the system.

System functionalities are implemented in the class hierarchy consisting either of the direct implementations of the *VSAService* interface or of the classes that implement its inheritors (classes *AdministratorServiceImpl*, *InstructorServiceImpl* and *StudentServiceImpl*), and which the server application instantiates after successful authentication according to the type of the client application. Each of these classes implements the appropriate interface - *AdministratorService*, *InstructorService* and *StudentService* - which provides an API for remote communication with client applications. The aforementioned classes and interfaces form the server application's service layer (see Fig. 1). Each of the client applications has access to a precisely defined set of functionality through its *VSAService* implementation. *StudentService* interface provides the functionalities required by students, such as fetching the desired test from the database or entering the respondent's answers into the database. On the other hand, *InstructorService* interface provides functionality required by teachers - creating questions and placing them in the question repository, creating and publishing tests, grading of test scores, etc.

The role of *KeepaliveThread*, which is an active class, is to take into account the duration of client applications' inactivity periods and, if necessary, to terminate client sessions.

Each of the classes that implement some of the *Service* interfaces (*AdministratorService*, *InstructorService*, *StudentService*) uses the realizations of the *CrudDAO* interface to persist their data and to communicate with the database. *CrudDAO* interface provides a minimal set of methods that all of its implementation must follow and which are related to the basic database operations - create, read, enter and delete (often abbreviated CRUD). Hierarchy of classes that implement *CrudDAO* interface constitutes the data access layer (see Fig. 1). For mapping of the classes of the object domain into the tables of the relational domain we used the previously mentioned object-relational mapping framework, *Hibernate*.

Fig. 3 shows a simplified UML class diagram of the core classes of the client applications. *Client* interface provides a basic API for communication with the server, while the class that implements it, *ClientImpl*, implements the behavior common to all the client applications - authentication and authorization, configuration of communication parameters (e.g. IP address of the server), etc. Student, instructor and administrator client applications are implemented as classes *StudentClientImpl*, *InstructorClientImpl* and *AdministratorClientImpl*, respectively. As can be seen in the diagram, each of the client applications uses the appropriate implementation of *VSAService* interfaces.

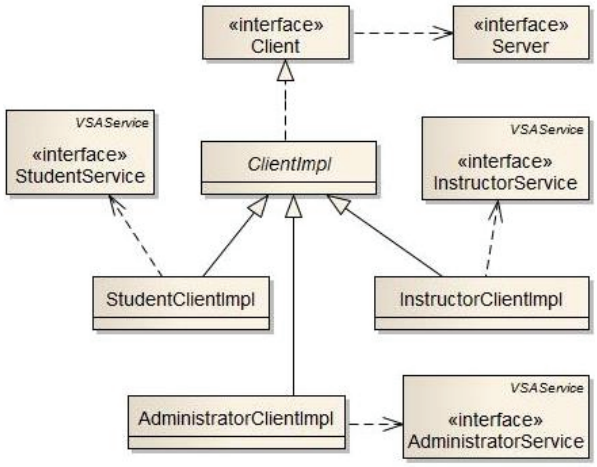


Fig. 3. Simplified diagram of the core classes of the client applications. Client interface provides the communication API, *ClientImpl* implements the functionality shared by all the clients, and client applications use the proper implementation *VSAService* interfaces to achieve functionality.

The implementation of the system presented in this section relies on the tool called Visual Simulator of Algorithms (VSA). This tool is the result of a series of theses and final papers of undergraduate students at the School of Electrical Engineering in Belgrade [10] - [14]. VSA's presentation layer is used as the basis for the new system's presentation layer. Because of this, in this section, we will briefly describe the core of the VSA tool.

VSA core consists of three key basic classes and their specializations: the *Algorithm* class, which defines the algorithm itself, i.e. the execution procedure; the *ObservableVariable* class, which is the base class of all the data structure classes used by algorithms, and *Action* class, which is an abstraction of actions taken by the algorithm upon its data structures. The *Action* class contains all relevant information required for defining the participants in the action, on one hand, and the state of the used data structures after the execution of the action, on the other hand. For example, in the case of swapping of two array elements, an instance of the *Action* class that represents this step in the algorithm will contain an identifier of the array whose elements exchange values, a copy of the array before the swap, and the array positions (indices) of the elements that swap their values.

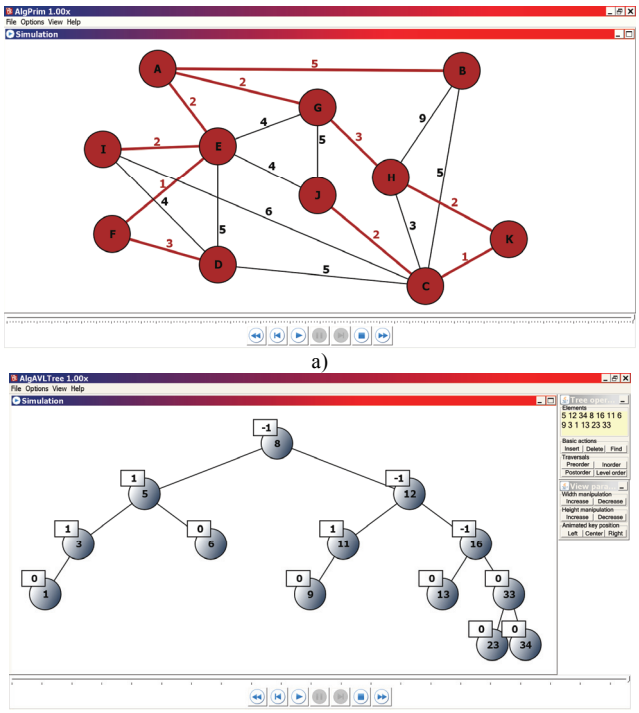
Executing the algorithm over the given data structures will result in a list of appropriate actions that are subsequently interpreted by the presentation layer to form an adequate visual representation. Data structure abstraction class, *ObservableVariable*, in addition to representing structures themselves, also provides utility methods that are abstractions of the basic operations on the corresponding structure and are also responsible for generating the aforementioned list of actions.

The crucial part of the presentation layer is represented by hierarchies of two basic classes - the view class (*AbstractView*) and the viewer class (*AbstractViewer*). The view class defines graphical representation of components used to visualize the algorithm, while the viewer class determines what, when and where to draw. The view class is vital for realizing display flexibility and represents an abstraction of the way of viewing the process of executing an algorithm. This class defines a standard interface for processing actions that are generated by an algorithm, as well as methods for displaying abstract data structures over which an algorithm associated with the view is executing. The view class is in charge of the list of actions formed as a result of executing an algorithm, which is done by iterating through the list of actions and displaying the state of the used data structures after each step of the algorithm, possibly with animation of the action that led to this state.

The viewer class (*AbstractViewer*) is an active class that collaborates with the view class so that it provides the necessary context for drawing. Its responsibility is to call the appropriate methods of the view class in a timely manner. Fig. 4 shows examples of using the VSA tool for the visualization of some algorithms on graphs and binary trees.

The described hierarchy of the VSA tool's classes plays a key role in the system proposed in this paper. The validation of the respondents' answers is based on the extensions of the *Algorithm* and the *ObservableVariable* classes. The *Algorithm* class is responsible for comparing the series of the respondent's actions with a series of valid (expected) actions. The implementation of a specific

algorithm may decide to stop the comparison of actions after the first mismatch or to attempt the recovery after the mismatch and continue trying to match further actions. This provides a more accurate assessment of respondents' knowledge. On the other hand, the *ObsevableVariable* is responsible for assessing the severity of mistakes, which is in part based on the validation of the resulting data structures. For example, if a series of student's actions led to a poorly formed structure, it indicates a serious error.



a)

b)

Fig. 4. Examples of using the VSA tool to display the execution of some algorithms, a) finding the minimum spanning tree in a given graph, b) insertion of keys into an AVL tree.

V. Conclusion

In this paper, we have presented the architecture of a software system for training and testing of knowledge in domains where the problem solving procedure is highly important, which is not the case with traditional testing systems. The proposed system provides self-directed, controlled, and passive training, allowing the users to specify the parameters and data for the algorithm execution. Consequently, the users are also able to check their knowledge and understanding in specific situations to which the instructor did not pay (enough) attention. In the test mode, the system can assess the sequence of actions specified by the user that are only partially concordant with the given algorithm, and estimate the success rate of these actions. Furthermore, this gives a better insight into knowledge and understanding demonstrated by the user.

Aside from the planned introduction of the completed system in the course Algorithms and Data Structures at the School of Electrical Engineering in Belgrade, further improvement of the system could comprise the development of the following components: performance evaluation of an algorithm (including the possibility to compare the performances of related algorithms), statistical analysis of the test results, individual progress monitoring for each student. In order to complete the system, all relevant algorithms will have to be implemented, and the test repository will have to be created, as well.

The abovementioned feature of progress monitoring could be implemented through a connection with existing virtual learning environments (such as Moodle), that already provides appropriate support. Such a connection, with a stable, flexible and widely available virtual environment, would provide several benefits, the most important being increased system rating and applicability, and centralized progress monitoring. The development of the connection module includes the implementation of the IMS QTI standard within the system, as well as the publication of key functional features of the system as web services.

References

[1] P. Dillenbourg, D. K. Schneider, P. Synteta, "Virtual Learning Environments," in *Proc. 3rd Hellenic Conference on Information & Communication Technologies in Education*, pp. 3-18, 2002.

[2] http://en.wikipedia.org/wiki/Virtual_learning_environment#List_of_some_virtual_learning_environments

[3] Moodle Question Type – Java Molecule Editor, http://moodle.org/mod/data/view.php?d=13&rid=296

[4] IMS QTI, IMS Global Learning Consortium Question & Test Interoperability Specification (QTI), 2005, http://www.imsglobal.org/question/.

[5] W. M. Davies, H. C. Davis, "Designing Assessment Tools in a Service Oriented Architecture, " In *Proc. 1st International ELeGI Conference on Advanced Technology for Enhanced Learning Napoli*, Italy, 2005.

[6] Moodle.org: open-source community-based tools for learning, http://moodle.org/

[7] dokeos - Open Source E-Learning, http://www.dokeos.com/.

[8] Sakai Project - an Open Source suite of learning, portfolio, library and project tools, http://sakaiproject.org/

[9] OLAT - Your Open Source LMS, http://www.olat.org/

[10] I. Mićanović, "Design and implementation of VSA core," bachelor thesis (in Serbian), School of Electrical Engineering, University of Belgrade, 2009.

[11] M. Bjegović, "Design of GUI and presentation layer for VSA," bachelor thesis (in Serbian), School of Electrical Engineering, University of Belgrade, 2009.

[12] D. Đurišić, "Design and implementation of advanced VSA core," bachelor thesis (in Serbian), School of Electrical Engineering, University of Belgrade, 2009.

[13] M. Milivojević, "Design and implementation of advanced graphical interface for VSA," bachelor thesis (in Serbian), School of Electrical Engineering, University of Belgrade, 2009.

[14] D. Mirosavljević, "Implementation of tree based algorithms within VSA," bachelor thesis (in Serbian), School of Electrical Engineering, University of Belgrade 2009.