Published in International Journal of Online Engineering (iJOE) 4, issue 2, 17-23, 2008
which should be used for any reference to this work

NeOS: Neuchâtel Online System

H. Sturzrehm, F. Aubert, P. Kropf and R. Corfu
University of Neuchâtel, Switzerland

***Abstract*—Most online courses include training assignments and offer various forms of tests to evaluate the students' performance. Some courses, in particular computer sciences courses, require elaborate hands-on training experiences. The *NeOS Framework* presented in this paper proposes an e-learning tool developed supporting in particular hands-on training through programming exercises. The user locally develops his code in an editor-like user interface and then tests his program code on a well secured remote machine. In the case of programming exercises, the *NeOS Framework* supports computer aided evaluation as well as the cross check of solutions submitted. Furthermore, it offers several security measures at the system level by securing in particular the lab machines from being affected by attacks from the outside, and at the usage level by reducing the cheating opportunities.**

***Index Terms*—Virtual Lab, Computer Aided Evaluation, Cross Check.**

I. Introduction

Due to the sustained advances in computer and networking technologies, Internet services and networked applications have become widely available and largely deployed. Leveraging from these developments, e-learning has become more and more important in modern education. With the use of e-learning systems, it is no more necessary for students and teachers to stay at the same place at the same time, and the course contents and objectives may be easily customized and adapted to individual needs and goals. Furthermore, the world wide knowledge is growing everyday, and is, most importantly, widely and rapidly available. Virtual classrooms and e-learning in general heavily support the lifelong learning process necessary to participate in the steep and continuous increase of knowledge.

Since the year 2000, the Swiss Virtual Campus [1] initiative promotes learning over the Internet at the Swiss institutions of higher education. Currently, there are 82 courses online, covering a wide range of domains. Among these courses is the Operating System Laboratory *OSLab* [2] shortly presented in this paper.

Several companies are offering a large variety of online e-learning platforms. The WebCT platform (Web Course Tools) [3] from Blackboard has been chosen for the Swiss Virtual Campus (SVC) online courses. In order to simplify the access to the SVC courses, a unique nation-wide user management has been deployed. All Swiss universities and other institution of higher education are members of SWITCHaai [4], the Swiss wide Authentication and Authorization Infrastructure. This federated identity management enables course providers to use identity information across security domains including multiple universities (Fig. 1). Furthermore, it deals with issues such as interoperability, liability, security, privacy and trust. This large scale identity management allows students to attend SVC courses at any university in Switzerland.

A. OSLab

The Operating System Laboratory, *OSLab,* is an online course to teach students the principles of computer operating systems. *OSLab* is using a constructionist and problem-oriented learning approach. We thus focus on hands-on training experience of the students. The course is designed for university undergraduate students and can be used as an integral coursework or as a complement to traditional lectures.

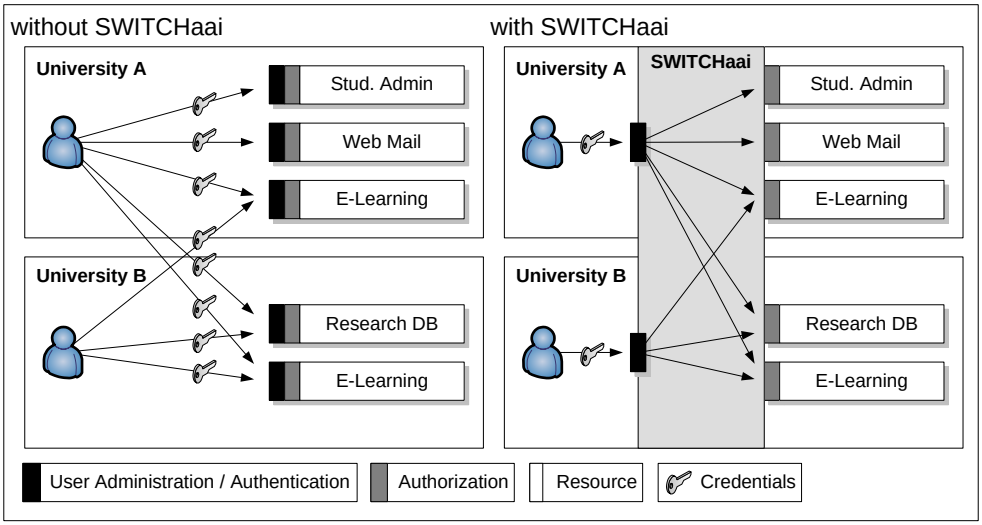

Figure 1. Features of SWITCHaai

The course shows a modular structure where each module covers one particular topic in a self-contained manner. The tutor responsible for the course can select modules according to particular learning objectives and needs. Furthermore, the flexible structure easily allows adding new modules to the course. Prior to the *OSLab* we have developed a similar e-learning course called *VITELS* (Virtual Internet and Telecommunications Laboratory of Switzerland) [5]. Experience with the use of *VITELS* revealed the need for a unified architecture for the hands-on assignments. As it is proper to courses in computer science, assignments typically include programming tasks or results obtained from executing programs. Solutions to assignments can thus be well-defined. In contrast to the evaluation of free-text solutions requiring interpretation for assessing them, automatic computer aided evaluation may be introduced. Taking into account these particularities, the following requirements have been identified within the *OSLab* project for the development of the *NEOS* (*Neuchâtel Online System*) *Framework*:

- **Compatibility with the WebCT based VITELS course.** Integration into the deployed course work architecture and existing course.
- **Authenticity.** Decrease of the possibilities to cheat.
- **Computer aided evaluation.** Automated evaluation of hands-on tasks in a way the tutor only needs to verify the (automated) evaluation results.
- **Flexibility.** Platform independence of the evaluation tool and generic customizable deployment.
- **Ease of evaluation.** Support for a quick cross check of the solutions.

II. Design and Implementation

The presentation of the *NeOS Framework* design and implementation follows the order of the above requirements list. First, we describe the architecture which is compatible with the *VITELS* course and WebCT. Then, our authenticity solution and the computer aided evaluation are discussed. At the end of this section, we finally present the implementation of the *NeOS* tools.

The framework consists of three principal components: (1) the *NeOS Frontend* applet, (2) the *NeOS Backend* server, and (3) the *NeOS Tutor Tool* that provides the interface for evaluating the assignments.

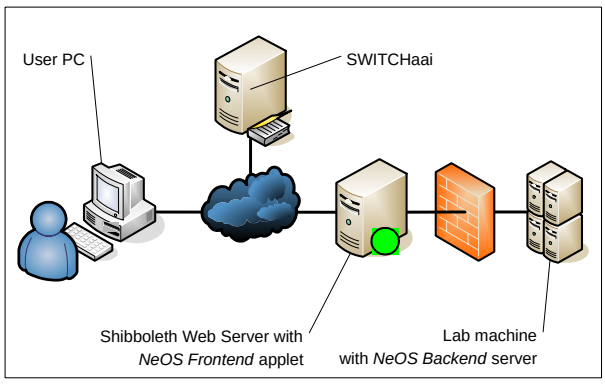

Figure 2. Architecture for *NeOS*

A. Architecture

The *VITELS* course as well as the *OSLab* course development is a collaborative effort including several partners. The hands-on assignment environment had been individually developed by the partners in the case of *VITELS*. As a consequence, a great variety of methods and technologies had been used for the hands-on tasks. Some modules of the course used Java applets, others had the need of a direct connection to the lab computers only accessible with a Java implemented SSH console, and still others used Java applications running with the Java Web Start utility. However, all have in common the deployment on an Apache Tomcat Web server [6] which used a Shibboleth [7] called plug-in to support the authentication architecture of SWITCHaai.

When planning the new *OSLab* modules, we realized that, in this course, most of the hands-on assignments would include programming tasks, which could not be supported by the existing implementations from the former course. On one hand, there are simulators which need some program fragment (e.g. a user written Java method) as input. These simulators are commonly too large to be efficiently loaded via the Internet for local execution, which calls for other approaches. On the other hand, students shall develop Java programs to be tested on the distant lab machines. As these machines are firewall protected, the students needed to upload them through a secure SSH connection. While this is certainly a viable solution, it does not prevent students from uploading and executing malicious codes.

Taking into account these constraints, two new components have been added to the existing system architecture as shown in Fig. 2: the *NeOS Frontend* applet and the *NeOS Backend* server. Both components are implemented in Java [8]. The *NeOS Frontend* applet serves as user interface for the students and is deployed on the Web server already used in *VITELS*. Within this applet, the students may program and then submit their implementation to the *NeOS Backend* server, which is located on one of the lab machines. On this computer the (Java) program code of the student is executed in a sandbox or a simulator is launched together with the student's program fragment.

The communication protocol used in this architecture is depicted in Fig. 3. First, the user has to access the Web page on the Shibboleth enabled Web server where the *NeOS Frontend* applet is stored (Fig. 3, A). If he is not authenticated, he will be redirected to SWITCHaai to do so (Fig. 3, B). After successful authentication he may download the *NeOS Frontend* applet into his browser and execute it there (Fig. 3, C). When the applet initializes itself, it establishes an SSH connection via the Shibboleth Web server to the *NeOS Backend* server (Fig. 3, D) running on the lab machine. This step concludes the set-up phase and the user can now solve his task within the *NeOS Frontend* applet. If the student wants to test his solution, the *NeOS Frontend* applet submits the solution directly via the SSH tunnel to the *NeOS Backend* server (Fig. 3, E) where it will be executed. This last phase (E) can be repeated as often as the user wants to.

1) Frontend View

The previous section described the overall architecture of the NeOS. This section presents the user's view when working with the system. To start, the user must

authenticate at SWITCHaai to access the WebCT course portal where the *OSLab* course is located. From the course portal the user starts the hands-on session by clicking on a hyperlink. The *NeOS Frontend* applet (see Fig. 6 for the graphical user interface) is then started and the user can modify the program code template. When his program code is ready for processing, he submits it to the *NeOS Backend* server. The applet then receives and presents the result obtained for the task submitted. The result may be either a compilation error, an execution error, or the program output together with the automatically computed evaluation grade. In case of negative feedback (errors), the student must resubmit a modified version. When a task was completed successfully, the user may still resubmit a new improved version which might give him a better grade. Once the task will be successfully solved to the student's satisfaction, he has to copy a unique data set (see Section B. Fingerprint) to the WebCT portal. This concludes the assignment submission.

2) Backend View

The processing chain at the *NeOS Backend* server is illustrated in Fig. 4. Since we aimed at achieving high flexibility, the method invocations shown in the diagram have to be implemented by the course developer according to the requirements of the tasks to be solved. If there is no need for one of these functionalities the corresponding method can remain void. After the reception of a task submission from the *NeOS Frontend,* the server invokes the method *createEnvironment()* which is necessary, if a setup phase is required for that task, e.g., creating folders or copying additional files. In the next step, the appropriate *compile()* method for the submitted task is invoked (e.g. a Java compiler or a method to process instructions for a simulator). In case the method returns errors, these are transmitted to *the NeOS* Frontend. Otherwise, the *execute()* method is called. This method allows executing the user's executable code (obtained from the previous step), the reference executable, or a simulator with the user's input and the reference input.

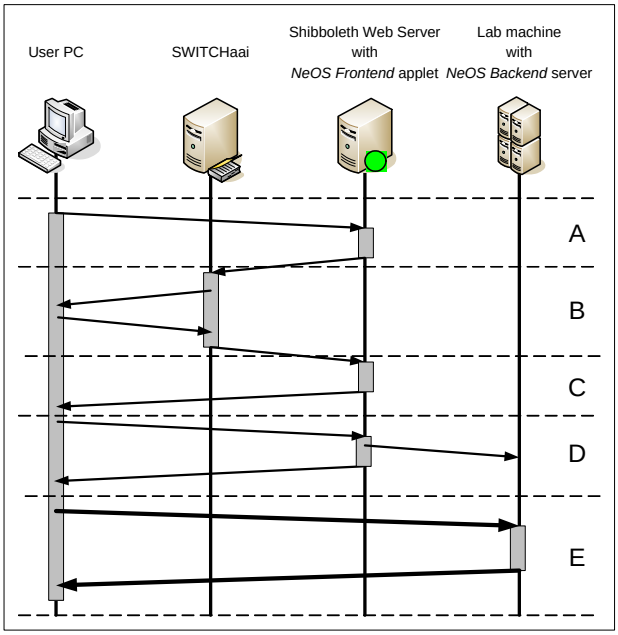

Figure 3. Communication Protocol

A more detailed description of this process including reference executables and inputs is given in II.C. When the execution returns without any error, the results are evaluated in the method *evaluate()*. This method implements the evaluation based on the principles described in II.C. If the evaluation grade reaches a certain threshold defined by the course creator, the server invokes the *encrypt()* method. The goal of this method is to create an encrypted unique data set as described in the next section.

B. Fingerprint

There are always students who cannot resist passing easily by just copying from their mates or cheating in some other way. These problems are even accentuated in an e-learning system, because of the lack of face-to-face interaction between students and tutors, the relative anonymity and the locality independence.

In the *VITELS* course, the plain text output of a program or the program code itself had to be copied into the course portal. This made it very easy to manipulate the result. In order to satisfy the requirement of authenticity, while still preserving compatibility, we devised an authenticity test in a way the user still may copy his solution into the course portal.

For this purpose, we included an RSA/DES encryption of the user data (see Fig. 5). This ensures that the students cannot change their results before copying them to the portal.

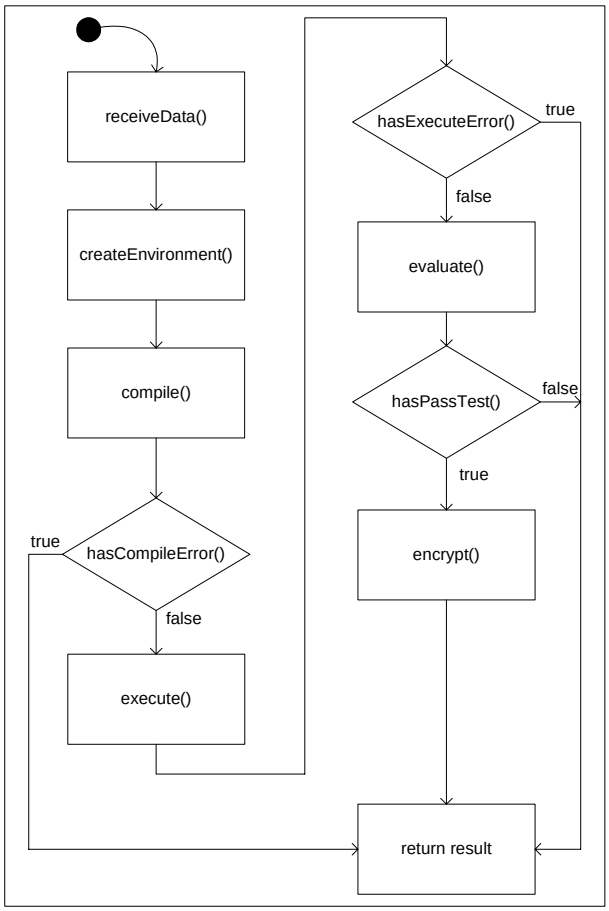

Figure 4. Process Chain of a Submission

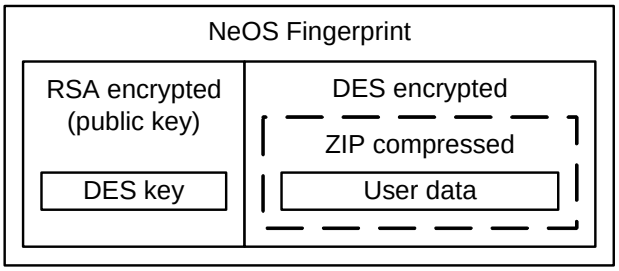

Figure 5. NeOS Fingerprint

The encrypted user data includes the user name as provided by SWITCHaai in order to create a user specific, unique fingerprint. In addition, all input data needed to reproduce the student's results are included. This enables the tutor to verify and compare the submitted solution using the computer aided evaluation described in the following section.

The user data encrypted in the Fingerprint consists of:

- **User name.** This makes the fingerprint unique for each user.
- **User modified files (program code or code fragments) and any other files generated for this test run.** This enables the tutor to reproduce the results.
- **Results of the computations.** This enables the tutor to assess the solution.
- **Computer aided evaluation grade.** The automatically generated grade assists the tutor in evaluating the submitted solution. .

For security reasons the public key has to be stored on the lab machine and the private key is only distributed to the tutors of the course. The fingerprints in the course portal can only be downloaded by the authorized tutor.

C. Computer Aided Evaluation

It is very complex to fully check the correctness of a program. We thus restrict the control to verify partial correctness, only. Partial correctness is defined here as follows: the output of the execution of a program for a certain input must be correct. A reference implementation producing correct output for the test inputs is used to compare with the output obtained from the student's solutions. According to Hoare [9] a task is correctly solved by two different programs respectively executions, if they produce equal outputs with the same input. This simple method for evaluating the correctness of the solutions submitted by the students can be formally described as follows:

$$e(R,S)=\begin{cases}1 & : \quad O_A = O_B \\ 0 & : \quad otherwise\end{cases} \tag{1}$$

with

$\{I\}R\{O_A\}$

$\{I\}S\{O_B\}$

I - Input

R - Reference Program

S - Student Program

O_A, O_B - Outputs

The output of both programs has to be significant. For this reason one must avoid that the check will only be successful for one particular input. The input for the program must therefore be randomly chosen out of a certain interval. The user can thus not manipulate his program to create a static output that would match the output of the reference program, thereby falsely resulting in a positive evaluation.

Hoare's logic allows us with the rule of composition (2), to cut a program into several sequential pieces P_i.

$$\frac{\{I\}P_1\{O_1\},\{O_1\}P_2\{O_2\},\ldots,\{O_{n-1}\}P_n\{O\}}{\{I\}P_1;P_2;\ldots;P_n\{O\}} \tag{2}$$

With this rule, we are able to create a fine grained evaluation of a program. When adding certain checkpoints into the program, it is possible to generate an output at these points for the comparison. By creating the median over all steps a value between 1 and 0 is obtained as shown in (3).

$$E(R,S)=\frac{\sum_{i=1}^{n} e_i(R_i,S_i)}{n} \qquad E(R,S)\in\{x \mid 0\le x\le 1, x\in\Re\} \tag{3}$$

This value can then be used for the Computer Aided Evaluation to match it to the locally used grading scale.

D. NeOS Frontend

The *NeOS Frontend* applet in the *NeOS* is the principle user interface. The screenshot in Fig. 6 shows an example with a common use case where the student develops a program fragment as input for a simulator. In order to satisfy the requirements of ease of use as well as of flexibility (i.e. the tool must be able to handle multiple diverse use cases), we decided to create a multi layer tabbing area allowing for every part to be accessible with less than three clicks. The structure of the tabs can be easily changed by a configuration file located at *NeOS Backend* server.

The first tab layer (Fig. 6, A) consists of five tabs. The program code tab on the left-hand side is the area where the user may access a text area (Fig. 6, C). This area may be editable or not depending on the access rights of the file associated with the area. The second tab layer in Fig. 6 labeled B shows an example with four tabs referencing different Java class files. Among those, only the first one is an editable Java class, while the other three are just Java interfaces which may be read but not modified.

The second first layer tab provides a console where the output from the *NeOS Backend* server will be presented. This includes all error messages (e.g. compiler error messages) not concerning the results of a simulation or the execution of the user implemented program. The grade the student would achieve for his program is also displayed in this window.

The third tab shows to the user the random input with which the user's code and the reference implementation were executed. This allows the user to assess the output of his program.

The next tab refers to the simulator output. It shows the output of both, the user specified and the reference

simulation run. Depending on the use case, it is also possible to display different output views, e.g. standard output or error output. These different views are then available through the second layer tabs. These will thus change as compared to the situation shown in Fig. 6.

The last tab in the first layer, the pass code, displays the fingerprint in the text area C as soon as the task is successfully completed. This can then be copied and pasted to the WebCT portal, thus concluding the assignment.

The tab area at the bottom of the window (Fig. 6, D) contains the send button to activate the submission and execution on the *NeOS Backend* server, a cancel button to stop the execution, the help button for further information and a reset button to restore the initial state of the applet.

E. NeOS Backend

The *NeOS Backend* server is the heart of the *NeOS* system. It consists of core and task specific modules which are loaded at startup. Thus, for each task a new instance of the *NeOS Backend* server needs to be started. We created the following interfaces to adapt the *NeOS Backend* server to the actual task in an easy way:

- **Compiler interface:** Implements compilation tasks.
- **Simulator interface:** Implements the execution of user and reference programs or the execution of the simulator with both program fragments and the random input.
- **Evaluation interface:** Implements the computer aided evaluation.

Each interface contains methods which have to be implemented depending on the actual task. Furthermore, a configuration file has to be adapted to the task context. The configuration feature allows adapting the applet, to set the path to the working directory, and to define the *NeOS Backend* server implementation to be used.

Since our *NeOS Backend* server is written in Java, all simulations should be started as an external process via the ProcessBuilder included in Java. This process starter provides the ability to start any other program or script written in any programming language. If the simulation software is written in Java, it is also possible to use the Java internal security manager to restrict the accessibility. This includes access to the hard disk or access to the network connectors. If non-Java programs are used, the executed program must be wrapped into an external sandbox.

Since a program that compiles correctly does not necessary terminate upon execution, an internal control mechanism is needed. Therefore, we implemented a possibility to automatically stop the simulation process after a certain time specified by the actual configuration. Infinite loops will thus not disturb the system. Note, that the user can always cancel an execution from within the applet

.

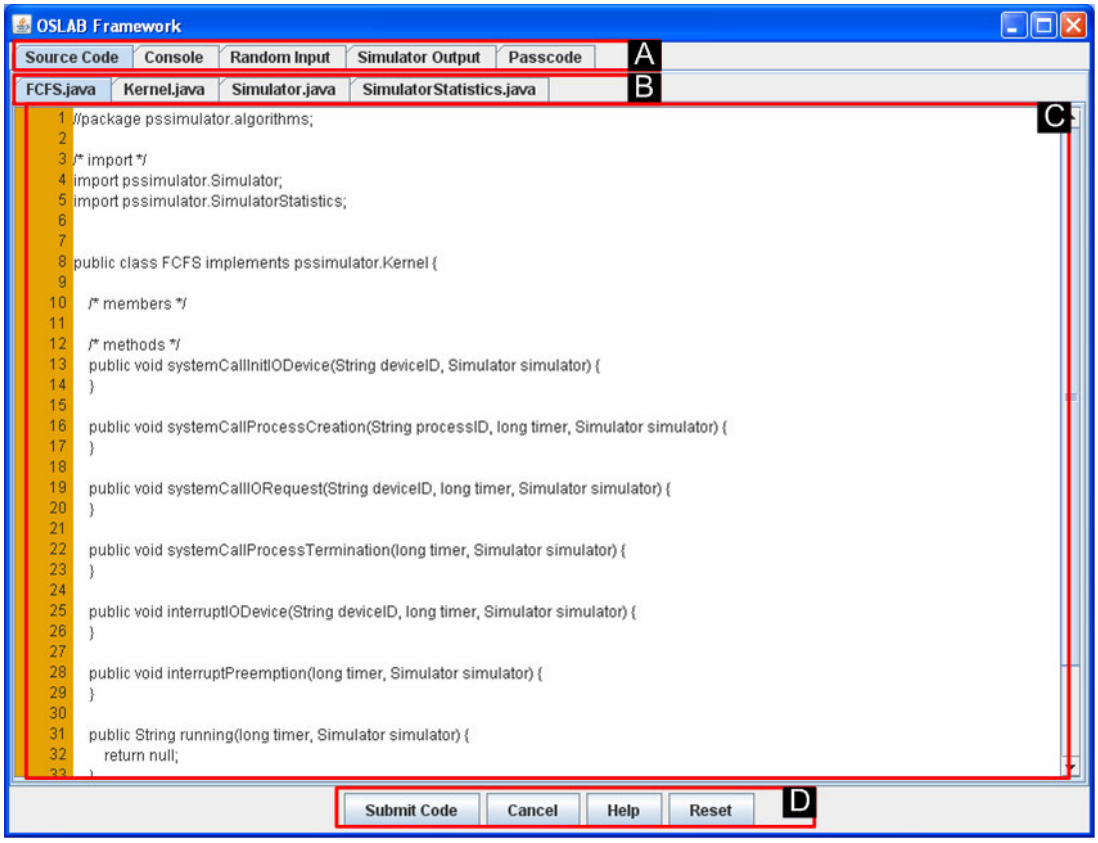

Figure 6. NeOS Frontend applet

Currently, the compiler interface supports the Java compiler. The other two interfaces concerning the simulator and the task evaluation depend on the particular tasks. A number of example implementations as well as some templates are available for these two interfaces. An implementation of the evaluation interface typically requires a method to compare the user program and the reference program outputs as described in Section II.C. As an example, if eight out of ten output values are correct, a grade of 5 on a scale of 1 to 6 might be assigned (see Fig. 7).

F. NeOS Tutor Tool

The *NeOS Tutor Tool* is intended to assist the tutor in evaluating the student's solutions. For every student, the tutor can review the program code submitted as well as the input and output data. All this data are available through the students fingerprint as described in Section II.B. Furthermore, the tool presents the results of the automatic evaluation. Fig. 7 shows an example of the *NeOS Tutor Tool* user interface for a scenario with the use of a simulator.

The results of the computer aided evaluation for a task are presented in a table (A in Fig. 7). The tutor may now proceed to the final evaluation (tutor grade) by considering the grades provided automatically, by inspecting the program code submitted, the input respectively the output data, and by checking similarities between the different solutions of all the students.

For evaluating a student's solution, the tutor has to double click on the corresponding UserID, which opens up the user window (Fig. 7, B). There, he has the possibility to grade the student and confirm it through the save button (Fig. 7, C). In the user window, all the details stored in the fingerprint can be reviewed through the tabs: the program code of the user, the test input used for the simulation and the simulation outputs (Fig. 7, D).

The different colors (or gray scales) in the UserID row (Fig. 7, E) are a hint for the tutor that there are some parts in the solution of that student which are equal to the solutions of other ones. At the moment there are two colors (gray scales), red (dark) for a complete match and yellow (light) for a partial match. The checking is done file by file against all other submitted student solutions and the reference files. More details about this comparison can be obtained through the Similarities tab available in the user window (Fig. 7, D). An example of such a detailed view for the UserID "Sabina" is given in Fig. 8. This view can be easily adapted to the tutor's needs, since it is just an implementation of an interface as in the case of the ones of the *NeOS Backend* server. So far, we just implemented a simple comparison which cuts out all non relevant parts of the program code like spaces, tabs or braces. The similarities tab displays for each file the UserID of the matches in a tree like representation. The example in Fig. 8 reveals similarity matches between Sabina's program "HelloWorld2" and the one's of Silvia etc.

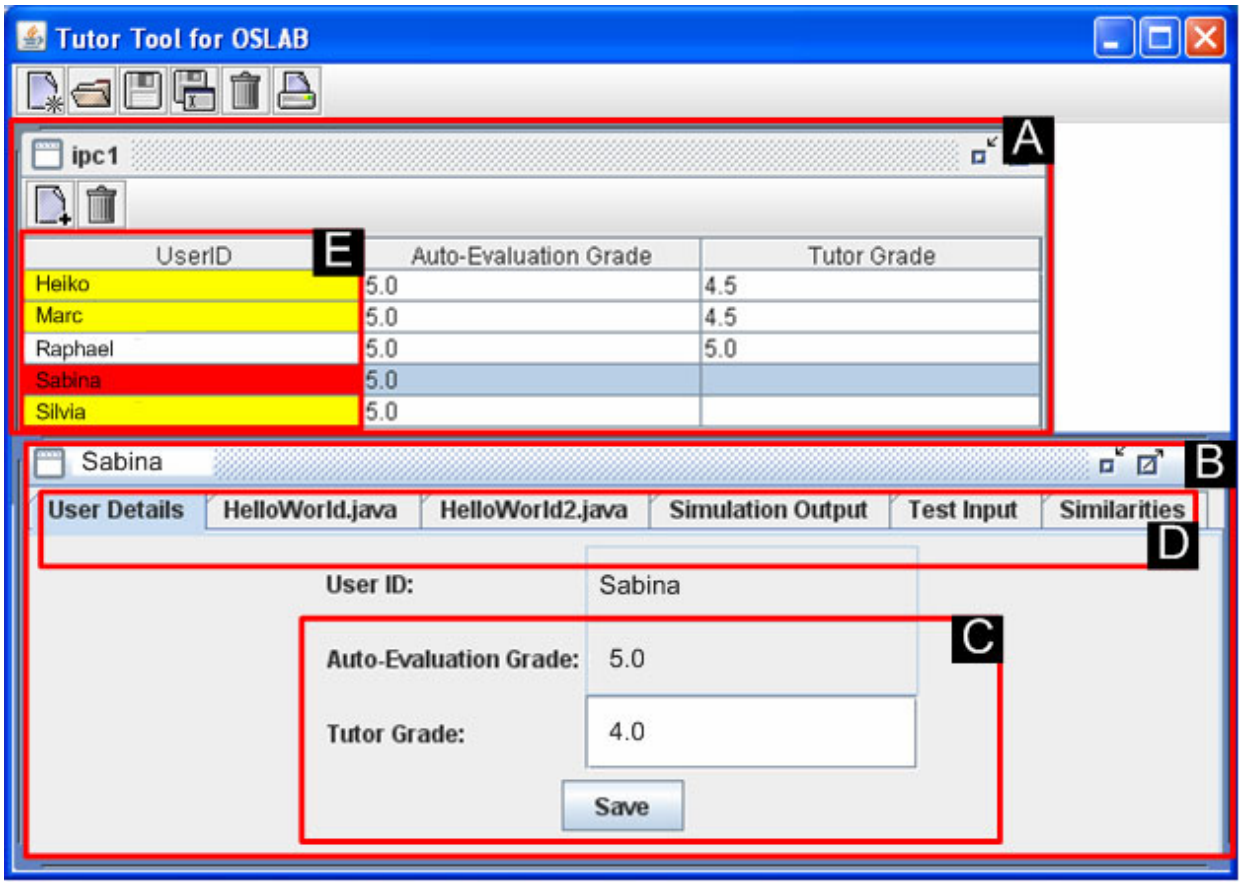

Figure 7. NeOS Tutor Tool

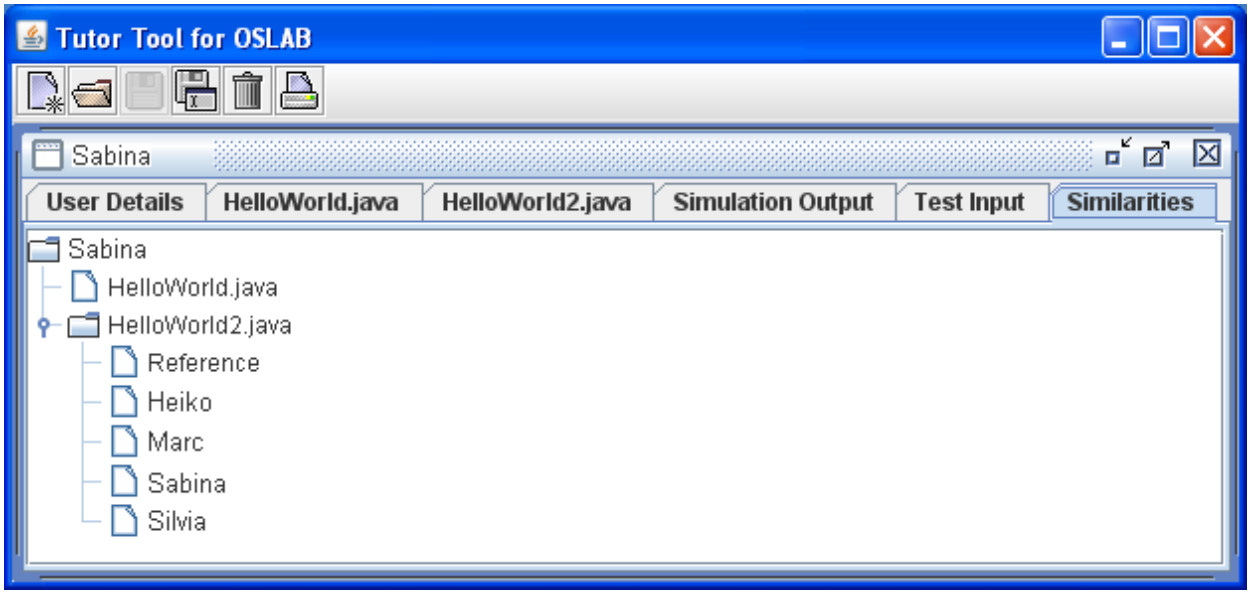

Figure 8. Similarity Tab of the NeOS Tutor Tool

III. Conclusion and Future Work

In this paper, we have presented the *NeOS Framework*, which is an e-learning tool developed to support in particular hands-on training experiences. It consists of an applet, a server and an assessment tool for the tutor, all implemented in Java. These tools allow a user to locally program in an editor-like user interface and then test his program on a well secured remote machine. The *NeOS Framework* is configured to work in the SWITCHaii and WebCT environment. However, it would be easy to adapt the *NeOS Framework* to any other e-learning environment as long as it provides a user authentication mechanism.

An advantage of the *NeOS Framework* is the computer aided evaluation for programming tasks as well as the cross check of the student's solutions for detecting plagiarism. Both features are meant to assist the tutor in his task to evaluate the student. Furthermore, the framework offers several security measures at the system level by securing the lab machines from being affected by attacks from the outside, and at the usage level by reducing the cheating possibilities.

Based on further experiences in using the *OSLab* course, we will further evaluate the system to identify which parts might be enhanced. Already in development is a single server solution, such that one instance of the *NeOS Backend* server can handle multiple tasks. There is of course plenty of room to improve the computer aided evaluation functions by applying sophisticated heuristics. We finally also note that the present framework is restricted to tasks related to programming. However, because of the openness and flexibility of the design of the *NeOS* architecture, its application in other contexts could be envisaged as well.

References

[1] Swiss Virtual Campus, http://www.virtualcampus.ch/, 2007.
[2] OSLab, http://www.oslab.ch/, 2007.
[3] Blackboard, http://www.blackboard.com/, 2007.
[4] SWITCHaai, http://www.switch.ch/aai/, 2007.
[5] VITELS, https://www.vitels.ch/, 2007.
[6] Apache Tomcat, http://tomcat.apache.org/, 2007.
[7] Shibboleth, http://shibboleth.internet2.edu/, 2007.
[8] Java, http://java.com/, 2007.
[9] C. A. R. Hoare, “An axiomatic basis for computer programming.”, Commun. ACM, 12(10):576--580, 1969.

Authors

H. Sturzrehm is with the Department of Computer Science, University of Neuchâtel, Switzerland (e-mail: heiko.sturzrehm@unine.ch).

F. Aubert is with the Department of Computer Science, University of Neuchâtel, Switzerland (e-mail: frederic.aubert@unine.ch).

P. Kropf is with the Department of Computer Science, University of Neuchâtel, Switzerland (e-mail: peter.kropf@unine.ch).

R. Corfu is with the Department of Computer Science, University of Neuchâtel, Switzerland (e-mail: randoald.corfu@unine.ch).

Manuscript received 04 April 2008. This work was supported by the Swiss Virtual Campus under Grant CVS-IP-4-019 in cooperation with the University of Bern, the University of Fribourg and the University of Distant Learning Hagen.

Published as submitted by the authors.