Software for Formative Assessment of Programming Exercises

Uta Priss, Nils Jensen, Oliver Rod

Ostfalia University of Applied Sciences
www.upriss.org.uk, {n.jensen,ol.rod}@ostfalia.de

Abstract. This paper discusses an architecture for combining learning network and assessment systems (LNAS) with computer-based assessment systems (CBA). The advantage of such a combination is to support reuse and sharing of e-learning materials, in particular for automated formative assessment of programming exercises which are very labour-intensive to create. An architecture for such a combined tool is presented together with its requirements and the implementation and testing of a prototype of such a tool.

1 Introduction

Even though there are numerous e-learning tools available to support computer science teaching, it can still be difficult to find a single tool that provides a number of desired features. We are therefore proposing an architecture which facilitates the integration of different e-learning tools for the purpose of enhancing formative assessment of programming exercises. We start this paper with an explanation of the background of the problem followed by a description of the architecture and the methods we are employing. Although we are currently building an implementation of such a system using the Lon-Capa[1] and Praktomat[2] software, the discussion in this paper will be at an abstract level independently of which specific tools are used.

Existing e-learning tools used in computer science education can be roughly classified into three groups. First there are virtual learning environments (VLE), or also called course management systems (CMS), open course ware, or learning platforms. These tools provide course content, student groups, timetables, some forms of assessment (for example multiple choice questions) and often additional communication tools (such as email, chat and forums). Examples of these systems are WebCT, Blackboard, Moodle, StudIP and Ilias. Because such tools do not usually provide specific functions for assessing computer science exercises, these tools are not of interest for this paper.

A second group of tools can be called learning network and assessment systems (LNAS). These systems may also have some or all of the VLE features but are more focussed on exercises which can be automatically assessed and shared by lecturers across courses and institutions. Such systems either provide their own repository of exercises which can be searched by lecturers from participating institutions (such as Lon-Capa (Kortemeyer, 2009)). Or they provide materials modelled as reusable Learning Objects

[1] http://www.lon-capa.org
[2] https://github.com/danielkleinert/Praktomat

(using, for example, the LOM or SCORM standards) which can be shared across different types of tools. As long as VLE systems do not provide support for large repositories of exercises, they cannot fully replace LNAS systems. The exercises can be of a variety of formats including quizzes, fill-in-the-blank and short answer questions but they tend to employ mainly simple automated assessment algorithms, for example, identifying selections from a fixed set of possible answers (radio buttons or drop-down menus) or performing simple text comparison tasks. Additional feedback can be entered manually by a lecturer.

A third group of tools are computer-based assessment systems (CBA), which are also sometimes called "submission systems"[3]. An overview of such tools is provided by Rongas et al. (2004). These tools allow students to upload code which is then automatically assessed. CBA tools are usually much smaller in scale than VLE or LNAS because they are used for individual courses and thus have no need for course management apart from enrolling a group of students. The automatic assessment of the exercises is much more complex than in the case of LNAS tools. CBA tools tend to incorporate features that are commonly used in software development such as compilers, testing and style checking tools, version control and online submission systems. The usefulness and features of such tools are discussed, for example, by Zeller (2000), Spacco, et al. (2005), and Demuth & Weigel (2009).

Considering that there are many tools which are already available, it is surprising that many computer science lecturers still write their own CBA tools instead of using existing tools[4]. It seems that because there are many different types of programming languages, exercises and teaching styles, a single existing tool might not support all desired functions. Therefore it is desirable to have tools that can be integrated with other tools and provide for reuse of functionality across tools instead of replicating features that already exist in one tool in another tool. For CBA tools it is especially important to facilitate exporting and sharing of exercises because creating assessments for programming exercises is very labour-intensive. It is not cost-effective to use exercises only once. Therefore a combination of LNAS and CBA tools is very desirable.

2 Proposed architecture

In this paper we are proposing a generic integration of LNAS and CBA tools which hopefully will lead to tools with more features, more support, larger user groups and large repositories of sharable content. If lecturers are able to reuse tools and content, then presumably they will spend less time on course development and subsequently have more time for face-to-face support of their students. Furthermore using a CBA tool means that lecturers do not have to spend time helping students with simple problems which can be solved via automated feedback from the CBA. Instead lecturers can help students with more complicated problems that require help from a human tutor or with conceptually challenging questions that require discussion.

[3] http://www.dcs.warwick.ac.uk/boss/

[4] The authors counted on average 2 instructor-written CBA tools in each CS department they know personally.

A similar project to ours is described by Gotel et al. (2007), but their system is based on WebWork which uses a program generating macro language. This is in contrast to our approach which does not require lecturers to learn a macro language because lecturers can use testing approaches they are already familiar with (for example unit tests) in addition to some standard tests provided via web-based menus.

Fig. 1 shows our proposed architecture for integrating CBA and LNAS tools. The idea is that users (both students and teachers) interact with the system via the web interfaces of an LNAS tool. The reason for this is that LNAS tools usually have more sophisticated user interfaces than CBA tools. In particular, LNAS tools tend to provide full functionality for course content, communication and course management. CBA tools are then connected to the LNAS tools in order to evaluate student submitted code. Different CBA tools can be used for different types of programming languages (for example one tool could be used for object-oriented languages and another tool for database languages) or a single CBA tool can be used for a variety of tests. Because the data exchange between the LNAS and one or more CBA tools is reasonably complex we propose to have a dedicated server (Proforma - Programming exercises Formative Assessment) using a REST interface and an XML format for the exchange. This is not meant to imply that the Local Proforma Server must be physically implemented as a separate tool on a separate machine but logically the Proforma Server forms a separate unit. Any LNAS or CBA tool that supports the XML format can be integrated in this manner.

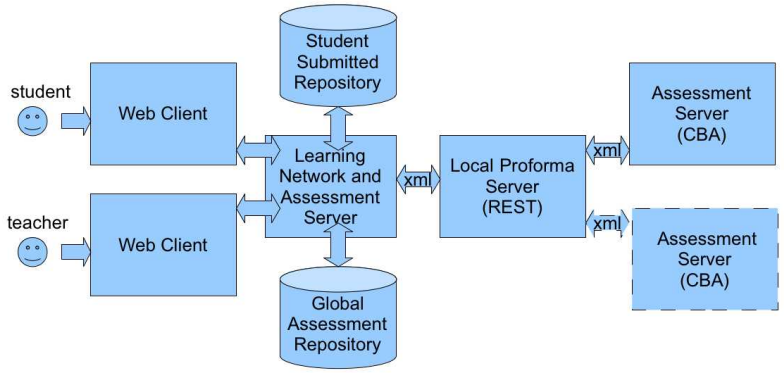

Fig. 1. Architecture for combining LNAS and CBA tools

The Proforma Server facilitates the information exchange between the LNAS and CBA servers. At a minimum the LNAS server has to provide a session ID, user ID and the exercise data. The CBA tool then returns the session ID, user ID and the evaluation data. In general the information exchange is fairly complicated. We think that it is important that the XML exchange format is supported by a number of existing LNAS and CBA tools to reach a critical mass for widespread acceptance. We are currently discussing such a format with programmers from a variety of tools to negotiate an exchange format that is widely supported. Therefore we do not discuss the format further in this paper, but instead focus on our requirements analysis and on the testing of our prototype.

3 Requirements analysis

We conducted a requirements analysis by studying relevant literature and by interviewing lecturers of programming courses. Apart from basic requirements for the user interface and the testing functionality, we identified several more challenging requirements which are more complex and some of which may not even be easily solvable with currently existing technology. The more interesting challenges are:

- Security: the data exchange must protect systems security and data security. This is an open-ended problem and will require the system to be continuously monitored and updated. Furthermore, standard tools will need to be employed to secure the web interface against common threats (such as cross-site scripting and SQL injection attacks) and to check the student submitted code for hacking attempts.
- Parameterised exercises: LNAS tools often parameterise exercises so that plagiarism becomes more difficult because different students get slightly different exercises. This is challenging for programming exercises because the tests need to be parametrised as well. Zeller (2000) describes a solution for this problem by using a macroprocessor that generates exercises with tests. It will be difficult to implement this in a manner that is exchangeable across LNAS and CBA tools.
- Detailed feedback and hints: the CBA tools might provide more detailed feedback than just "pass/fail", release partial feedback in form of hints or release different feedback at different times (at code submission time, hand-in deadline or when the results are published) which needs to be sent to the LNAS server. In general it can be challenging to format such automatically generated feedback (which contains debugging output from different tools and instructor-written comments) in a manner that is optimally useful for students[5].
- Usability: some LNAS and CBA tools have a steep learning curve for lecturers. Combining tools must not make it more difficult for lecturers or students than it already is otherwise they will refuse to try out new tools.

[5] An anonymous reviewer of this paper commented that students do not appear to read automatically generated feedback. We suspect that timing and comprehensibility may be crucial. Students will want to receive feedback that helps them with a problem while they are struggling with it. Automatically generated debugging information is not always very comprehensible to inexperienced programmers.

– Long-term support and benefits of sharing: lecturers will only be willing to use tools if they are ensured that they are available for a long-term. Apart from releasing tools as open-source, it is desired to connect to existing technologies and standards in a manner that ensures long-term support. The additional effort that is required for creating exercises for the tools must be balanced by the advantage of being able to share exercises among lecturers and by long-term usage.

4 Implementation

We have implemented a prototype of the system which combines the LNAS and CBA functionality but does not yet use the full architecture as described in Fig. 1. At the moment the connection between the two tools is implemented not via a REST interface but as an IFrame as shown in Fig. 2. The data is not yet properly returned from the CBA to the LNAS but instead displayed directly on the screen for the student to read. While this setup does not allow the testing of the data exchange, it is sufficient for testing the usability and basic functionality of the system.

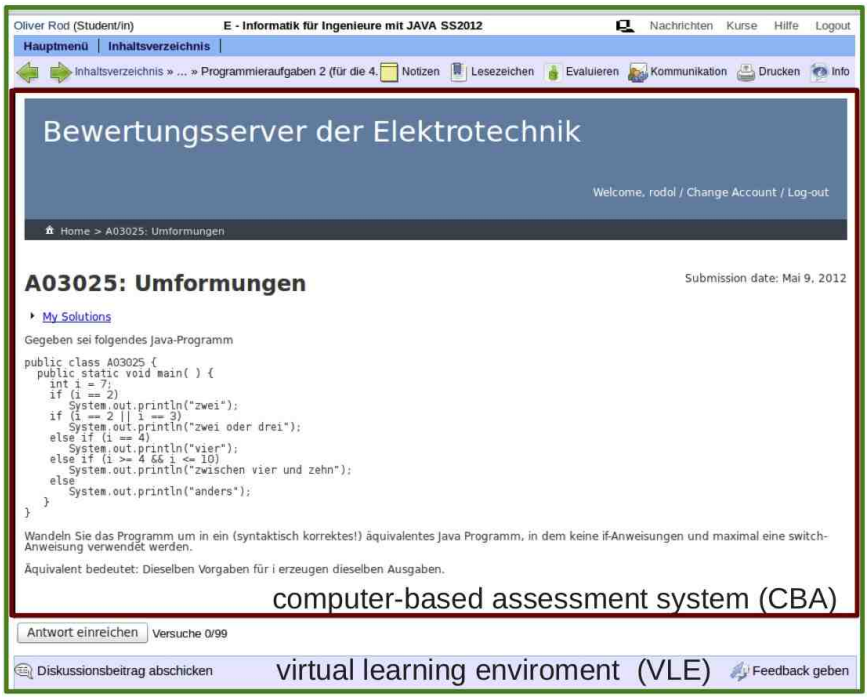

Fig. 2. Screenshot: combining LNAS/VLE and CBA

The outer frame in Fig. 2 (highlighted in green) shows the LNAS or VLE component, in this case represented by Lon-Capa. The inner frame (highlighted in red) shows

the CBA tool, in this case represented by the Praktomat. The screen in Fig. 2 is displayed after a student clicks on a link of an exercise in Lon-Capa. The login details are automatically sent from Lon-Capa to the Praktomat so that the student does not notice that a second system is involved. In order to submit code, a student clicks on the "My Solutions" link of the Praktomat. In our prototype, the interface still shows some redundant information, such as two "logout" links: one for Lon-Capa and one for the Praktomat. Furthermore, each of the tools uses a different stylesheet. This is only because of our current implementation using an IFrame. In the full version of the tool using a REST interface, there will be a uniform interface. Nevertheless, even in the current version, the students did not seem to have any problems with the interface as presented.

Once the students submit their solution, the inner frame presents feedback and results as shown in Fig. 3. In this case, the following checks of the submitted code were performed: whether it compiles (Java - Compiler); whether certain instructions are adhered to which prohibit the use of "if" and allow at most one "switch" for this exercise (Text Not Checkers); whether the code follows style conventions (CheckStyle 1 and 2); and whether the output is as expected (Blackbox Test). In this example, a student submitted a program which produced the output "zwischen vier und elf" instead of "zwischen vier und zehn". Therefore the black box test reports an error the details of which are displayed after the student clicks on the orange line "Failed! Please click here for more information".

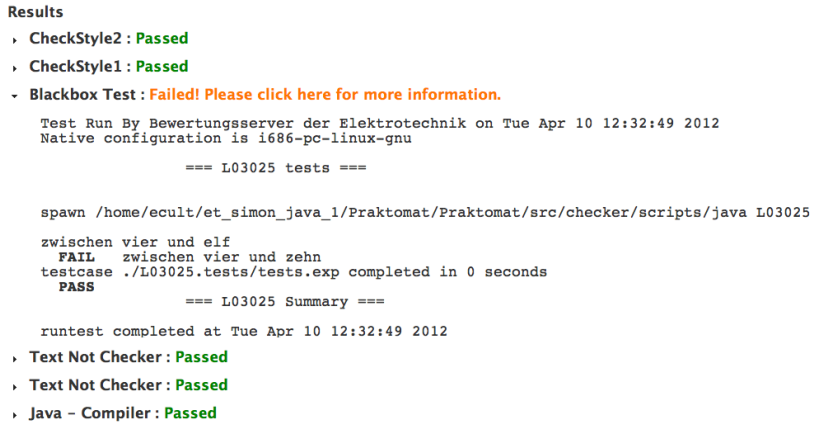

Fig. 3. Screenshot: Feedback and Results

5 Results from testing

We have tested the system with 12-16 students in two sessions of an introductory Java course. Most of the feedback we collected so far pertained to practical implementation

deficits that are easy to fix. For example, students had to make certain minor modifications to their code in order for it to compile properly on the CBA tool. Furthermore, we discovered some difficulties with respect to how the output from the students' programs is compared to the correct solution in particular with respect to matching white space and German umlauts.

Another finding was that the precision of the description of the exercises had to be increased and that the students had to get used to precise reading and verbatim implementation of the instructions. This has also previously been reported by others (Zeller, 2000) and is due to the automated comparison algorithms.

We carefully recorded the time required to convert existing exercises into the electronic format and to write all tests to be used by the CBA. We found this to be about 5-10 hours per session (consisting of 5 exercises). We hope to be able to reduce this time in the future by improving the upload mechanisms used by the LNAS and CBA tools, for example, by prefilling some of the data fields with default values and by providing semi-automated means for generating standard tests.

We believe that the automated feedback for the students currently generated by our system is only useful for small exercises where the code submitted by the students is fairly short. For longer code submissions, the students would be better advised to check their code carefully with standard development tools before they upload it to our tool. This is because standard development tools are capable of highlighting problems directly in the code whereas our tool lists errors by line numbers. As mentioned in the previous section, providing timely and comprehensible feedback is a major challenge that we still need to work on.

We also discovered a new challenge which we did not envision during the requirements analysis. There is a danger that automated tests overlook certain errors that would be picked up by a human tutor. This is because the person who implements the exercises needs to predict possible errors which is more difficult than detecting errors in student code during the marking process. This problem could be solved by using the same exercises for a number of years and manually checking of a sample of the exercises in the first instance. From a pedagogical viewpoint it would be interesting to investigate the students' attitudes, though. Does the use of an automated tool affect how carefully the students check their code before they consider it complete? Furthermore, if students are used to receiving specifically designed feedback from a CBA tool, how will they cope later in the "real world" when they are only receiving the standard support from software development environments? Maybe one would need some process of providing more feedback early in the semester and less later. Or maybe one needs a careful combination of automated and manual feedback. Or, another solution (proposed by Zeller (2000)) is to have students peer review each other's code in addition to the automated tests. Presumably these questions can only be answered if one carefully monitors the deployment and impact of such tools for a number of years.

6 Conclusion

We successfully implemented a prototype of a system that combines a learning network and assessment system and a computer-based assessment system. It demonstrates

the general feasibility of this kind of tool combination. Nevertheless, many challenging aspects are still left for future research and development. We are currently involved in discussions with developers of other tools about an XML exchange format for programming exercises that would be supported by variety of tools. If we manage to agree to such a joint exchange format, it will be straightforward to combine different e-learning tools in the manner described in this paper.

References

1. Demuth, B.; Weigel, D. (2009). *Web Based Software Modeling Exercises in Large-Scale Software Engineering Courses.* CSEET'09, Software Engineering Education and Training, p. 138-141.
2. Gotel, O.; Scharff, C; Wildenberg, A. (2007). *Extending and contributing to an open source web-based system for the assessment of programming problems.* In Proceedings of the 5th international symposium on Principles and practice of programming in Java (PPPJ '07), ACM, p. 3-12.
3. Kortemeyer, G.; Cruz, E. (2009). *LON-CAPA - An Open-Source Learning Content Management and Assessment System.* In Education and Technology for a Better World, WCCE 2009, IFIP AICT 302, Springer, p. 340-348.
4. Rongas, T.; Kaarna, A.; Kalviainen, H. (2004). *Classification of Computerized Learning Tools for Introductory Programming Courses: Learning Approach.* In Proceedings of the IEEE International Conference on Advanced Learning Technologies (ICALT '04), IEEE Computer Society, p. 678-680.
5. Spacco, J.; Strecker, J.; Hovemeyer, D.; Pugh, W. (2005). *Software repository mining with Marmoset: an automated programming project snapshot and testing system.* SIGSOFT Softw. Eng. Notes 30, 4, p. 1-5.
6. Zeller, Andreas (2000). *Making students read and review code.* Proceedings of the 5th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education, ACM, p. 89-92.