NETWORKED SUBMISSION AND ASSESSMENT

Mike Joy, Pui-Shan Chan, Michael Luck
Department of Computer Science
University of Warwick
Coventry CV4 7AL
M.S.Joy@warwick.ac.uk, Pui-Shan.Chan@dcs.warwick.ac.uk, Michael.Luck@dcs.warwick.ac.uk
http://www.dcs.warwick.ac.uk/cobalt/

ABSTRACT

In this paper, we describe the new online submission and assessment software in use at Warwick, a 3-tier client-server architecture with enhanced functionality and security features, and discuss its evolution from the BOSS software which was used before. We concentrate on the changing requirements for such software brought about by Internet availability, and on the enhanced functionality afforded by the use of new technologies.

Keywords

Networked online submission assessment

1. INTRODUCTION

In response to the problems caused by ever-increasing numbers of students we have previously developed an automated system for online submission and assessment of programming assignments [4]. After several years of deployment in various guises, we have gained a maturity of understanding of the process of automated submission and assessment, and a better appreciation of the potential improvements and future directions.

In this paper, we briefly review the original BOSS system, and outline some of its limitations. We then describe its successor, Boss2, and discuss the changes.

2. BOSS

As described elsewhere [4], the BOSS system is intended to perform several distinct tasks within a single overarching framework, and is aimed at course management rather than instruction: it supports the submission, testing and marking of programming assignments, and also enables feedback to be provided to students. Thus it does not impact on the *delivery* of instructional materials, which is a separate problem, and in this respect differs from other related tools such as CourseMaster [5] (and its functionally similar predecessor Ceilidh [1] which we have discussed in more detail elsewhere [4]). The structure of the system reflects the conceptual division of the software into three core modules that can be treated as largely independent components which address the submission of assignments, their testing and their marking, as follows.

- The *submission module* allows students to submit a piece of coursework, and handles the task of copying that coursework to a secure location where it can be accessed subsequently.
- The *testing module* runs and tests a single student program against one data set, and reports success or failure according to the given expected output.
- The *marking module* assists a lecturer in marking a collection of coursework (after the submitted programs have been run and tested again several sets of data).

Each of these components provides information through the user interface that is processed and managed by lower level utilities, which access the central file store through careful and secure techniques.

3. LIMITATIONS IN BOSS

The BOSS system has been in use now for several years and, though successfully fulfilling its rôle during that period, we have become aware of a number of limitations with the BOSS software that might be addressed. In this section we describe these constraints, grouped under the broad headings of *input and output, operating* *environments*, and *institutional procedures*. We consider each in turn below.

3.1 Input and Output

A major difficulty is the lack of support *for graphical user interfaces* (GUIs) in student programs. Currently, normal input and output for programs are handled via text files, leaving little room for GUI designs. Commercial software packages have benefited enormously from the introduction of the windowing environment, most end-user applications are designed to be used interactively, and the GUI is becoming an essential part of any computer software. In addition, new languages, such as Java, which are object-oriented, allow for sophisticated I/O. Restricting students to text-based I/O is therefore now undesirable.

Another major problem is the *handling of standard input and output* when performing automatic tests on submitted programs. The BOSS software runs under the UNIX operating system and, as mentioned above, the normal program interface is handled via ASCII text files. This is not ideal for automatic testing purposes, since a sizeable number of programs will fail the tests because of unexpected non-printing control characters generated by the programs. Programs written in C and Pascal are particular susceptible to this problem. Even simple typographical errors, such as incorrect use of whitespace and capitalisation, can cause automatic tests to fail.

Furthermore, when *comparing the output* from a program against the expected output, utilities such as `diff` are used to perform the check, but these utilities do not handle certain types of files well, especially if control characters are present. When a marker is presented with the output from a student's program which appears to fail an automatic test, in order to ascertain whether it in fact was a genuine failure, or whether it should be awarded partial or full credit, the marker needs to understand the output from the comparing utility. The authors are not aware of any such tool which produces easy-to-understand output, and a marker must therefore learn about the utility, which may well involve significant time and effort.

3.2 Operating Environments

As most of the commonly used programming tools are available on both PC and UNIX platforms, students now have the capacity to develop their programs under several environments. At Warwick, students are encouraged to use and to get exposed to a wide range of such programming tools. However, while students often use PCs, the Department of Computer Science at Warwick, does not support PC use, instead concentrating its resources on UNIX. As a direct consequence, many students will transfer files from a PC to the departmental UNIX machines when they submit their work. There is a definite and yet subtle difference between the two platforms concerning *text files*. A text file created in a DOS/Windows environment will contain non-printing characters, causing the automatic tests to fail under the UNIX environment. Due to the seemingly transparent nature of computing networks, many students overlook this issue unknowingly.

The BOSS software does not have any built-in network capability, and is an application running on a UNIX machine connected to the campus data network. Students would have to logon to the UNIX system in order to access the BOSS software. Many students may feel uncomfortable if they only have *limited exposure to the UNIX platform*, and since many users of BOSS are not Computer Science majors this is not an uncommon problem.

For a variety of technical reasons, it has not been possible to port BOSS to all platforms available to students, and in particular provision of the software under Windows was not an option. One of the major reasons for this was simply the *complexity of multiple installations* of the software on a network supporting a variety of processors (Sparc, Intel, R5000) and UNIX versions (Solaris, Linux, IRIX), and subtle differences in the behaviour of "standard" system utilities. BOSS is not built upon the client/server architecture and has no built-in network capability, and it would have required a complete rewrite of large parts of the software in order to migrate BOSS to another platform.

The *security of BOSS* relies on the security of a simple networked UNIX machine, and is not scalable to a distributed implementation. Differences in the file protection and other security mechanisms afforded by different operating systems were important factors, as we were only prepared to deploy the software on machines on which we had sufficient confidence that our software and stored files would not be vulnerable either to malicious attack or to accidental system failure.

3.3 Institutional Procedures

Within BOSS we make a distinction between *markers* and *moderators* for those staff users involved in the marking process, on a "per module" basis. This has worked fine for most modules, but occasionally a more sophisticated and configurable model of staff duties and privileges is needed.

Logical representations of the institution processes such as submission policies and assignment marking schemes are heavily dependent on the UNIX directory and file structure. Information is held in text files placed under a fixed and predefined directory structure, effectively forcing a *tree structure on the BOSS data storage*. Thus modules subdivide into assignments, which subdivide into exercises. Data retrieval is done via traversing the relevant directories systematically. This strong coupling of data representations and the software implementation does not accurately model the actual presentation of a module, which may have a more complex interrelationship between the various assessment units.

4. Boss2

With the advent of client/server concept and architecture, and enabling technologies such as the Java language, we see a great opportunity for the BOSS software to be developed further. The use of the client/server architecture will offer a flexible structure which enables BOSS to evolve more gracefully. The new version of BOSS, Boss2, is rewritten using the Java language which boosts its portability, robustness and ease of integration with other computer software and systems.

4.1 Architecture

The traditional client/server architecture (2-tier) approach usually involves two computers communicating with one another. There is a tendency for the client programs to take on more processing tasks and hence become bigger (fatter) over time, and this has a knock-on effect on the performance and maintainability of the software. Recently, there has been a strong movement shifting from the 2-tier approach towards a 3-tier architecture. This architecture utilises enabling technologies such as Remote Method Invocation (RMI) or CORBA to produce a middle layer between the client and the server, and a further interface between the server and the data storage.

The new BOSS software is built upon this 3-tier architecture which includes clients and servers, both written in Java, and a relational database, MySql. Client/server communication is handled using RMI, and Java DataBase Connectivity (JDBC) provides an interface between the servers and the relational database.

With this approach, the clients just provide a user interface whereas the servers contain all the complicated application logic including handling student assignment submissions, automatic testing and assignment marking. All information ranging from student registration to assignment marking schemes is held in a MySql database which can be assessed via the JDBC link from the servers.

The main advantage of employing this architecture is to introduce some form of flexibility into the software. The client/server interface is clearly specified, allowing a simple object model of the processes to be implemented in the servers. Similarly, the interface to the relational database permits the system developer a clear understanding of the underlying storage mechanism. The software needs to be highly robust and configurable in order to cope with different demands and parameters for modules, and the model we have implemented meets this requirement. With this architecture, attention and development effort can be targeted to a particular area more efficiently, and enables a variety of software development activities to be carried out separately, thus helping to speed up development time. New development and changes to different areas can be introduced with ease as the impact of modifications can be contained, helping BOSS to evolve and mature in a graceful manner.

4.2 System overview

As shown in Figure 1, Boss2 composes of several distinct components.

The user interface layers handles three separate activities: submission, testing and marking. Each activity will have its own interface in order to acquire data from the users as well as to present responses from the servers.

The server layer consists of a set of services to facilitate the three main activities, both externally and internally. As expected, the Submission, Testing and Marking components are the core functions provided by the server. The Security component provides functions to authenticate users and to manage sessions between each invocation of Boss2. A channel for the markers and moderators to communicate with the students is provided by the Feedback component. A Plagiarism Detection component [3] helps to detect unethical submissions, and the Configuration component provides functions to adjust the Boss2 settings. Finally, a Miscellaneous component provides general support for the Boss2 software. The Java packages which form the server software mirror these components.

The data store layer is composed of a file store and a database. The file store holds the actual submission files (in compressed form) from students as well as test data sets and expected results for automatic testing.

The interface and server layers are designed to be placed on different machines to enhance software flexibility, and the two layers are linked via a computer network. For performance and security reasons, the data store layer is placed on the same machine as the server layer. The RMI connection between the client interfaces and the servers is encrypted using the Secure Sockets Layer (SSL) protocol.

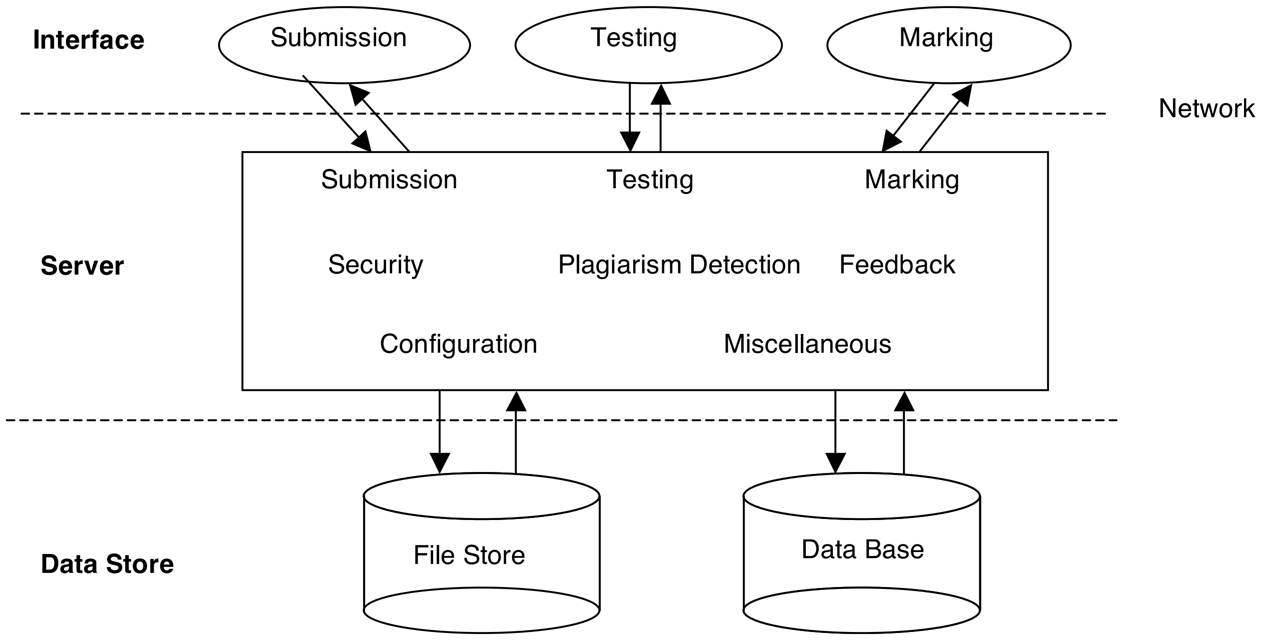

Figure 1: the 3-tier architecture for Boss2

4.3 Enhancements

Our successor to BOSS, Boss2, differs from its predecessor in a variety of ways.

Boss2 is implemented using Java, a very conscious design choice that supports several desirable features. First, Java brings with it the general benefits of using object-oriented technology with the associated improvements to modularity and abstraction. Second, since Java embraces web-based technology through its network APIs, and its use to program applets, it can provide a solution that is equally suited to use as a standalone application or as a web-enabled system. In principle, this allows the software to be integrated with other web-based functionality and to be used from across the Internet. Third, Java's platform independent model allows for deployment on all hardware currently available to our students, regardless of concerns with underlying operating systems. Finally, Java provides a wealth of resources for a range of relevant purposes, including database access and security, so that a coherent software solution can be developed.

We are very conscious that any Web-based solution carries with it dangers. For example, in order to use BOSS a student must logon to a central UNIX server, and that logging on process is sufficient to verify the identity of the student (assuming, of course, that the student has taken care not to divulge their id/password, either accidentally or on purpose). However, in a networked environment, the software is placed in a vulnerable position and is prone to malicious attacks in many different ways.

A primary concern is to prevent the introduction of computer viruses and Trojan horses to the system via the student's submission. A test harness was introduced in BOSS and has been improved in Boss2, which keeps sensitive information from being probed, and the directory structure of the file storage from being discovered. Boss2 can mimic the BOSS testing paradigm, ensuring backward compatibility.

Upon the completion of a submission, an email receipt will be sent to the student immediately afterwards. It confirms that files have been copied across to the main server, together with the hash code for each individual file submitted as well as a security hash code to authenticate the email message. The hash codes are extremely sensitive to changes. It will be reflected in the sequence of the hash codes if any submission work has been tampered with. An audit trail is kept of all dialogue with submission system, and is duplicated in the database, thus making it difficult for anyone to forge a submission.

Another addition to Boss2 is the use of SSL encryption technologies to protect the network traffic from prying eyes. Together with Java RMI, we feel confident that we have developed a model which can ensure smooth and secure transactions between the clients and the servers.

More importantly, in relation to a testing process, is the replacement of the text-based UNIX solution used by BOSS, with its associated difficulties of specifying programs based on text input and output. In Boss2, by using the object-oriented paradigm, these problems can be overcome: instead of specifying text output, for example, a result object can be specified instead, which can be compared with an expected result object. However, unlike the text-based solution, methods can be easily defined by the course organiser to perform comparisons between objects, based on any criteria they choose to use, enabling a variety of degrees of strictness of matching. In addition, the scope for submission of bogus solutions is reduced, since it is not possible simply to generate unstructured textual output, but instead program code is necessary.

4.4 Future Possibilities

The original BOSS system, which has been deployed at Warwick for seven years on a variety of courses, demonstrated the applicability and value of a system for online submission and automated testing of programming assignments, and has proved the validity of the paradigm that we have used. Inevitably, but fortuitously, changes in the surrounding technological environment have provided us with the ability to enhance the functionality of BOSS over the years. Indeed, several minor adjustments have been made to the initial version including, for example, the introduction of a graphical user interface relatively early in the development (and deployment) of BOSS. Nevertheless more recent developments have given rise to much greater potential improvements, both to the overall design and structure of the system through a complete overhaul, and to some of the fundamental aspects of the system such as the testing of programs and the addition of network-enabled and security-enhanced processes. As a result we have redesigned and implemented BOSS as Boss2, which we regard as a second generation system for submission and assessment.

This redesign process has allowed us to integrate much of the code for the new system with that for our web-based CAL software [2], and we are currently investigating the feasibility of merging the two systems into a single course management utility. The possibilities for such a utility are becoming increasingly important in the light of an increased concentration on concerns of distance learning and, more recently, on the collection of different aspects of the application of computers to teaching under the banner of e-learning. In our approach, we have not reacted to the hype to develop systems specifically for such purposes, but have developed robust and effective systems on the basis of sound and extensive experience. The system we describe here can indeed be used to support networked e-learning, but it can also be used in an isolated fashion to support traditional models with enhanced quality of both pedagogy and assessment.

5. REFERENCES

[1] Benford S.D., Burke K.E. and Foxley E., A System to Teach Programming in a Quality Controlled Environment, *The Software Quality Journal* 177-197 (1993)

[2] Joy M.S. and Luck M., Computer-Assisted Learning using the Web, *Proceedings of the 5th Annual Conference on the Teaching of Computing*, Dublin, 105-108 (1997)

[3] Joy M.S. and Luck M., Plagiarism in Programming Assignments, *IEEE Transactions on Education* **42**(2) 129-133 (1999)

[4] Luck M. and Joy M.S., A Secure On-Line Submission System, *Software - Practice and Experience* **29**(8), 721-740 (1999)

[5] CourseMaster's Research Page: http://www.cs.nott.ac.uk/CourseMaster/