The Marking System for CourseMaster

Colin Higgins, Pavlos Symeonidis and Athanasios Tsintsifas
LTR group,
School of Computer Science and Information Technology,
The University of Nottingham,
United Kingdom
cah@cs.nott.ac.uk, pxs@cs.nott.ac.uk

Abstract
CourseMaster (CM) is a Computer Based Assessment (CBA) system. This paper describes the motivation and aims for developing CM’s Marking System. It also explains the architectural forces and design decisions that have been established in order to engineer the Marking System. The Marking System adheres to the rigid specifications of the initial CM’s design, which are: reliability, coherency, security, feedback-richness, extensibility and customisability. The above notions and the features that CM’s Marking System provides are examined in detail. Concrete implementation issues are also discussed with conclusions on usability and extensibility observations.

Keywords: courseware; automatic assessment; teaching; distant learning

1. Introduction and Motivation
CM is a Computer Based Assessment (CBA) system. In [1] we have described the problems and limitations of CM’s predecessor, the Ceilidh system [2]. CM has been developed in order to overcome Ceilidh’s limitations, by providing the requirements for a reliable, robust, secure, user friendly and flexible CBA system.

In order to accommodate these requirements, the CM system was build around the concept of separate systems. There are seven systems in CM: the Login, Ceilidh, Course, Submission, Marking, Archiving and Auditing systems. The focus of this paper is on the Marking System, which performs the marking of student work, and returns marks and feedback to the students.

Section 2 of this paper describes the details of the functionality provided of CM’s Marking System. Section 3 focuses on the architectural forces that had to be considered prior to design and implementation. Section 4 discusses the design of the Marking System and its extensible nature, while section 5 expounds on selected implementation-specific issues. Finally, section 6 presents an overall evaluation of the Marking System.

2. Available Functionality of the Marking System
CM provides for the marking of Java and C++ programs, Object Oriented (OO) languages and design, flowcharts and logical circuits. CM supports automatic assessment of courseware that requires the students to:

- Read the question of the exercise. The question accurately describes the specifications that a student’s solution has to comply with, and the formats of any input and output files.
- Obtain a skeleton solution along with header files and/or testing tools. At the discretion of the teacher, the skeleton may be empty or contain a partial solution.
- Submit their solution for assessment and receive marks and feedback.

CM permits the students to resubmit their solution as many times as the exercise developer allows.

2.1 Available Marking Tools
The assessment of CM exercises is carried out by configuring a set of metric tools that perform various quality checks upon submission of students’ solutions. As table 1 illustrates, available metric tools include typographic layout, dynamic execution, program features, flowchart marking, OO design marking and logic circuit marking.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typographic Tool</td>
<td>Checks the sources of the students’ solutions for typo-graphic layout</td>
</tr>
<tr>
<td>Dynamic Tool</td>
<td>Tests the solutions’ behaviour by running them against test data</td>
</tr>
<tr>
<td>Feature Tool</td>
<td>Inspects for features specific to the exercise being marked</td>
</tr>
<tr>
<td>Flowchart Tool</td>
<td>Converts the students’ flowcharts to BASIC programs, that can be subsequently marked with the use of the Dynamic Tool</td>
</tr>
<tr>
<td>OO Tool</td>
<td>Checks the students’ OO diagrams for OO analysis and design</td>
</tr>
<tr>
<td>CircuitSim Tool</td>
<td>Simulates the students’ logical Circuits</td>
</tr>
</tbody>
</table>

Table 1: CM’s marking tools

For programming work, the Marking System uses the Typographic, Dynamic and Feature Tools. The rest of the tools are used to assess student solutions in their respective domains.

The Typographic Tool checks the program layout, indentation, choice and length of identifiers and usage of comments. All typography parameters can be customised on a per exercise basis.
The Dynamic Tool runs the students’ programs with various sets of test data. This verifies whether the students’ programs satisfy the stated specification. The Marking System allows a wide range of customisations concerning the flow, control and order in which the dynamic tests are executed.

The Feature Tool checks the students’ source code for special features that are exercise dependant. For example, an exercise set on a unit that teaches the difference between “while” and “for” statements would typically include a feature test to ensure the appropriate use of each construct.

The Flowchart Tool converts the student’s flowcharts to BASIC source code, which is then fed to the Dynamic Tool. The Dynamic Tool in effect runs the flowcharts and returns marks and feedback.

The Circuit Simulation Tool tests the students’ solutions by simulating their circuits. Any faults identified in the simulation result in marks being deducted.

The OO Tool tests the students’ diagrams for completeness, correctness, for the use of correct relationships between the various elements, for the use of classes that are not needed (or classes that should not be included) to complete the design etc.

The Circuit Simulation Tool tests the students’ solutions by simulating their circuits. Any faults identified in the simulation result in marks being deducted.

The Marking System provides the developer with the means to customise the marking of exercises by using the idea of a marking scheme. A marking scheme exists as a property for every exercise and is expressed in Java source code. For every exercise, the individual marking schemes are loaded dynamically at runtime by the Marking System. This is implemented by taking advantage of Java’s class-loading mechanism.

To author a marking scheme, the exercise developer writes a Java program that describes the actions performed by the Marking System while marking an exercise. Marking schemes can not only access CourseMaster’s state and marking tools, but are also able to use programming control structures, and additionally call any other external tools.

For example, in the case of a programming exercise, the author may instruct a marking scheme to run an external tool in order to compile the students’ code and then to perform typographic, feature and dynamic execution tests. Also, a marking scheme can contain feedback ranges. These are associations between numeric marks and feedback messages. For example, 70-100 could be “Excellent”, 50-69 could be “Good” etc.

The return of immediate feedback upon submission is a fundamental feature of CBA contributing to well known pedagogic benefits published by the CBA community [3][4].
The above forces motivated us to create a design that would encompass the Marking System. The fundamental design aims have been extensibility. New features can be added, new types of courses and exercises can be written. The original functionality of the Marking System can be modified to suit complex environments.

4. Design Review
The above forces motivated us to create a design that would allow CM’s Marking System to be:

1) Reliable and consistent.
2) Scalable (perform efficiently in large scale classes of students).
3) Easily maintainable, allowing functional extensions and diverse configurations.
4) Able to support extensive and expressive feedback.

In an effort to reorganise Ceilidh’s functionality in a more flexible way, dependencies, commonalities and variations between the available functionality were identified. Commonalities have been abstracted into class hierarchies, explicit points of extension and parameterisation were defined and dependencies were decreased by separating the various responsibilities in seven logical parts, one of which encapsulates the Marking System. The seven Systems that currently exist are shown in figure 1.

![Figure 1: A high-level view of CM’s Systems](image)

The design of a system should take into account the language in which the system will subsequently be implemented. In order to satisfy deployment requirements in various academic institutions around the globe, we decided to develop CM using Java [1] [7]. Thus, our design has taken into account the facilities that Java provides to the developer.

Java facilitates platform independence, offers an advanced exception and error handling mechanism, presents a way to dynamically unlink and reload source code, and offers a convenient mechanism for the development and deployment of distributed systems called RMI (Remote Method Invocation).

4.1 Platform Independence
Platform independence simplified the design as it eliminated the need for supporting different operating systems. The Marking System compiles students programs with external tools, enforces security, creates and manages processes that run student programs etc.

4.2 Exceptions and Errors
The exception and error handling mechanism has allowed for a class design that is clean from error handling code. Simplicity has been achieved by separating and distributing the error handling code in different classes and packages around the system.

4.3 Classloading
One of the fundamental issues that CM’s design aimed to resolve was the customisability of the Marking System. Initially, a script-like programming language was conceived that the Marking System interpreted at run-time. The language’s constructs were simple and straightforward to use, however they lacked the complexity that would allow for extensive marking customisation. After some serious consideration, it was determined that the best way to express the marking scheme is to use the Java language itself. The individual marking schemes, which are exercise-specific, are loaded dynamically at runtime by the Marking System.

Java’s classloading mechanism has permitted the dynamically re-loading (at run-time) of selective systems and marking schemes should a change in code occur, without the need of shutting down and restarting the system.

4.4 RMI
Java’s Remote Method Invocation (RMI) mechanism was used to divide CM’s logic into separate systems. RMI is considerably more effective and easy to use than network sockets while making the network transparent to the programmer [1]. CM’s remote objects inter-communicate using a set of interfaces. It should be noted that RMI is not currently used for inter-System communication, as the physical segmentation of the Systems has not yet been necessary. However, the design caters for this segmentation thus making this future change trivial. Note that the Marking System is linked with the Submission System only.

5. Implementation Review
Thanks to a thorough design process and an effective analysis of the various architectural forces, the implementation of CM’s Marking System was relatively straightforward, but there have been some complications. There was still a need to ensure that all the various abstracted parts worked together flawlessly and reliably as originally planned.

5.1 Performance and Scalability
Good performance is indispensable to any system that has to be scalable [1]. There was some initial concern that CM would be implemented in Java, as Java is interpreted. However, with advances in hotspot compilers [5] these concerns quickly disappeared. Additionally, by using profiling and optimisation tools an increase in performance was achieved. Because the Marking System generates a new thread of execution for each student submission, should better performance be needed, a selective upgrade of the underneath platform with a faster processor (or by using SMP –
Symmetric multi-processing), will result in the Marking System running faster.

5.2 Platform Independence
Java provides many classes, methods and fields in order to achieve platform independence. For example, the programming code for creating processes and threads is the same no matter which operating system is running underneath. Another example would be the code for creating and managing files and directories. Java goes to such an extent for platform independence, that even path and file separators (";","","/"), which are different for every Operating System (OS), are treated uniformly and transparently. Also, as Java has its own Graphical User Interfaces (GUIs), namely the AWT and JFC GUIs, creating a GUI that runs on multiple operating systems is a simple and relatively fast task.

In some specific occasions the Marking System had to distinguish between the various OS’s in order to issue commands differently. For example, to execute student programs, the Marking System had to use the “Shell” under Unix and the “Command Prompt” under Windows. As Java provides a Properties class, the issue was resolved with ease. The Properties class contains methods in order to obtain, at run-time, the name of the OS, the version of the OS and the hardware architecture of the OS under which a Java program runs, amongst other things.

5.3 Reliability
As CM is written in Java, we have made heavy use of Java’s built-in exception mechanism. When an anomaly occurs, exceptions are thrown and then caught at various locations around the system. This simplifies the development process while increasing the robustness of the system. The use of exceptions also increases the readability of the code, which in turn avoids bugs.

5.3.1 Corrupted/invalid Exercise Data
In the event that the exercise that is being marked has malformed files, invalid specification and/or missing files, the Marking System will try to recover from such a situation. It will either override the attempted customised behaviour and use its default one, or abort the specific test (or subtest). In all cases, an alert message will appear on the screen during the marking, and the event will be logged.

5.3.2 Problematic Student Programs
As CM is a system to assist lecturers with the teaching of (inexperienced) students, problematic student program submissions are inevitable. The student programs might perform endless loops, they might crash, or they might hang. The Marking System always times each student’s submission and will abort the marking if such problems are encountered. The students will be notified of such an event so that they can test their code and submit again, if allowed by the exercise developer.

5.3.3 Problems of the Marking System
If, for any reason, the Marking System itself would lead the system into an unstable situation, (e.g. because of a bug), the Marking System will exit and the current submission aborted. This will not affect the stability of CM, or any other concurrent student submissions. Again, a notification will appear on the screen and the event will be logged.

5.4 Coherency
We have described the coherency requirements in section 3.2. The Marking System is fair to the students as a student’s submission is always awarded the same marks if all the tests that the Marking System carries out perform identically. However, if a student’s submission is extremely similar to somebody else’s, while it may get the same marks, this shows evidence of plagiarism. CM provides an external plagiarism detection tool that scans all the student’s programs, performs various tests and reports any plagiarisers.

5.5 Security
Both CM and its Marking System have been designed with the security aspect in mind. In this subsection, we will concentrate on the security provided by the Marking System.

Firstly, if enabled by the exercise developer, a source code level security check can be performed in order to identify potentially security problems, like “*.”, “unlink” “delete” and the usage of network sockets. The exercise developer can choose what actions are carried out next. Typically, if the tests reveal positive results, the marking is cancelled and the student is informed of the situation.

Secondly, when dynamic execution of students’ programs takes place, the Marking System uses a wrapper program that runs each student’s program in a protected sand-box. This behaviour can be enabled by configuring CM accordingly. The Marking System uses SUID/GUID when run under Unix/Linux and the “runas” command when run under Windows 2000. The wrapper program limits students’ privileges, so that their programs can read and write only into a protected file area. The students’ programs are copied into this area and are executed independently of each other. Therefore, no damage can be done onto CM or on the rest of the files on the machine that runs CM. Since there are no back doors or “cheat-codes” in CM, the system is highly secure.

5.6 Extensibility and Customisability
CM’s marking process can be extended in two major ways: by customising the exercises and by customising the Marking System itself. If a project type is supported by CM (e.g. C++ programming, flowchart design etc), all the customisations and extensions that an exercise developer may need can be achieved either by customising the exercise’s marking files and/or by customising the marking scheme. However, if a radical new project type is required (e.g. musical notation marking) then the exercise developer has to customise the Marking System itself.

5.6.1 Exercise customisations
When creating a new exercise or modifying an existing one, most of the time will be spend on writing the question and the model solution, on the editing of the exercise’s marking files and finally on the testing of the exercise. For the majority of the exercises, all that is required is the editing of text files that are associated with the aforementioned tasks.

As exercise files are ASCII files, editing these requires nothing more than a text editor. The only exception is the marking scheme file; while it only requires a text editor in order to be altered, it requires a Java compiler in order to be compiled. The choice of metrics tests to be performed, along with their parameters and values only requires simple text file editing, and does not require any alterations in the Marking System (no Java code has to be written).
5.6.2 Customising the Marking System
In real terms, if new functionality is required, it will have to be built into new marking tools. This requires design and implementation of new classes or alterations of existing classes, depending on the nature of the modifications.

As CM’s internal state can be queried within the Marking System, new marking strategies can be written that perform fairly complex tasks. For example, a marking strategy may be devised that compares the current student’s submission with all the work that the student has submitted in the past in order to establish if the student is improving or not. In the latter case, CM can suggest the student seeks help from his/her tutor in order to solve the problems he/she might have.

6. Evaluation and Conclusion
This paper has described the Marking System of CM in detail. This paper has also described the design and implementation of the Marking System. The power of marking schemes written in Java has also been reported. Also, it was shown that extensions and customisations into the Marking System are both feasible and useful, so that more functionality can be accommodated in the future.

The use of CM as a major assessment tool has propagated in the last few years to more than 20 Universities in the UK alone. The further evolution of the Marking System of CM will intensify this tendency. As a result, research is currently under way that is expected to speed up the move towards open and distant learning.

Future versions of the Marking System will include Object-Oriented Heuristics and Refactoring Strategies that will enable exercise developers and teachers to further enhance the existing CM exercises, which in turn will allow students to improve the quality of their programs.

7. Selected Bibliography


The following web pages will prove useful for more information about CM in general
http://www.cs.nott.ac.uk/~ceilidh/papers.html
http://www.cs.nott.ac.uk/CourseMaster/

The following books will give some insight and provide information on Java, OOD and Patterns