Implementation of a Computer Algebra Based Assessment System

Contents

Introduction 2
Background 3
System Overview 3
Learning objects 7
Option structure 7
Response processing 8
Security 11
Implementation 11
Conclusions and Future Work 12
Acknowledgements 12
References 12
Implementation of a Computer Algebra Based Assessment System
Laura Naismith and Chris Sangwin
The University of Birmingham
Email: l.naismith@bham.ac.uk, c.j.sangwin@bham.ac.uk

Abstract: This article describes the design and implementation of the CABLE computer aided assessment system. Introducing a computer algebra system (CAS) to assist in marking allows instructors to design effective assessment schemes that are valid and reliable. A modular design approach allows system components, including a virtual learning environment (VLE), CAS and database, to be used for their respective strengths. Requirements for both interoperability and customisation for particular learning contexts are balanced within an easy to use, yet flexible and powerful, authoring environment. Student-provided responses are assessed objectively, with feedback that can be tailored for their particular misconceptions. The incorporation of data protection and system integrity features mean that CABLE can be used as a complete, low cost solution for computer aided assessment in undergraduate mathematics.

1. Introduction

This article describes the design and implementation of the CABLE computer aided assessment (CAA) system. CABLE, an acronym for Computer Algebra Based Learning and Evaluation, was first introduced in Naismith and Sangwin [1]. The system provides an open source, online infrastructure for use in undergraduate mathematics teaching. This includes the authoring, testing, storing, presenting and marking of lightweight and flexible mathematical learning objects.

CABLE connects a computer algebra system (CAS) with a database and a virtual learning environment (VLE). The CAS is used for the manipulation of mathematical expressions, display and response processing. CABLE uses the AXIOM CAS, although the careful design will allow other CAS systems to be substituted. The server application that reads and writes information between the CAS and the web client is written in PHP. PHP has also been used for the user interface and integration with an existing VLE. System related information, including questions, question versions and students' attempts, are stored in a MySQL database.

In the design phase particular attention has been paid to the learning objects, which are described as “CABLE questions”. By incorporating a CAS, support is provided for the following features:

- Question variables allow randomisation and mathematical processing to take place to instantiate the question for each student.
- The CAS supports the display of mathematical expressions, with a variety of output formats.
- Students are expected to provide their own answer using CAS syntax, rather than selecting from a teacher provided list, (as is the case with traditional multiple choice questions).
- The response processing makes use of computer algebra to evaluate the student’s answers.

This is not the first use of CAS, and this approach has been employed for a number of years with implementations such as AiM, reported in, for example, [2] and [3], and MacQTeX, reported in, for example, [4]. Furthermore, this project has not sought to implement a fully-fledged CAA system. Rather, it has concentrated on one question type with a view to incorporating this into existing VLE, MLE or CAA systems. In the implementation stage, open source components have been adopted and other packages have been selected and used for their particular strengths.

Section 2 provides an overview of the CABLE system architecture and design features. Specific implementation details are given in Section 3, and Section 4 presents conclusions and directions for future work.

1.1 Background

It is well known that assessment is a critical component of the learning cycle, and plays a number of roles. According to Johnstone [5], an effective assessment scheme has the following characteristics:

1. It is valid; it measures what it sets out to measure.
2. It is reliable; similar group of students produce similar results.
3. It is humane; it consists of periodic, continuous assessments, rather than a single final examination.
4. It is economical; it balances the time and resources spent on assessment against those spent on teaching.

In most undergraduate mathematics courses, weekly assignments form an integral part of the assessment cycle. While paper-based assignments are typically
reliable and humane, they may not be economical, as the marking requirements (especially for large class sizes) may divert staff attention from teaching. Plagiarism, as reported in Delius [6] for example, also calls into question the validity of the traditional paper based method of assessment.

To address the issue of economy, many institutions have turned to CAA. Depending on the particular tools used, different problems with validity may be introduced. In CAA systems that use multiple-choice (and similar) questions for assessment, students have to select from a list of teacher-provided answers, rather than provide their own answers. Problems reported with this format in Naismith and Sangwin [1] indicate that this may not be a valid measurement of a student’s current knowledge or abilities.

Incorporating a computer algebra system into CAA allows the issue of validity to be addressed directly, while, it is hoped, retaining the economy of CAA. Students’ free text answers can be tested objectively for specific properties and randomisation of question parameters can be introduced to reduce copying. Humanity is fragile: now staff do not need to mark the work of an individual it may be possible to lose track of exactly how long it actually takes a student to complete a weeks work. Examples to illustrate this will be given below. All four requirements of an effective assessment scheme can thus be met.

2. System Overview

The CABLE system is based on a modular design approach and is comprised of the following main components:

1. A VLE acts as the front end for all users.
2. The CABLE interface scripts integrate the “CABLE question” learning objects with the VLE, and provide all functions for dealing with these.
3. The CABLE server scripts provide a mechanism to connect to the CAS, and to manage a pool of CAS processes.
4. A database, stores questions and attempts. This may be the same database used by the VLE.

Figure 1 illustrates the CABLE system architecture.

An explicit goal of this project was to make minimal use of the underlying CAS and separate the functions that require a CAS from those that handle the authoring, presenting and storing of questions. Essentially, the CAS is used as a stateless collection of useful mathematical functions. The authors believe that this approach facilitates future integration with an alternate CAS and/or VLE and compliance with developing standards for question and test interoperability. All processes within the CABLE system, for which CAS is required, may be reduced to the following two operations:

**TASK A:** Evaluate a list of variables.

Given a list of assignments of the form

\[
\text{key}:=\text{value},
\]

the CAS executes the commands in value and assigns the result of this evaluation to the variable name key. The key may be used in subsequent calculations in the list. For each such execution, the CAS returns a data structure consisting of three parts:

1. A *value* form of the evaluated expression, in un-parsed CAS syntax, suitable for use by the CAS at a later stage.
2. A *display* form of the expression, suitable for display on the screen. This could be a LaTeX string, MathML, or a simple linear string format more suitable for a screen reader.
3. Any errors generated by the CAS (hopefully empty!).
Many processes in the CABLE system use this task. They include instantiating a question where, for example, the list of assignments may be the following:

\[
n := 3 + \text{rand}(3)
\]
\[
p := (x-4)^n
\]
\[
ta := \text{diff}(p,x)
\]

Here, \(n\) is assigned a value from the set \{3, 4, 5\}, and is used in subsequent calculations. The displayed form of \(p\) will be shown to the student as part of a question. The value form of \(ta\) will be used by the CAS in subsequent response processing.

Strictly speaking, it should not be necessary to use the CAS to obtain a displayed form for the student’s answer. There should be rendering of mathematics native within either the VLE, or CABLE scripts. However, the display of mathematics still causes problems, and for implementation the above approach has been found to be expedient.

**TASK B:** Apply an “AnswerTest”

This second task takes two expressions, one which is that of the student and the other that of the teacher, and applies an “AnswerTest”. The prototype test is that for algebraic equivalence. In this, one expression is subtracted from the other, and full simplification performed. If the system simplifies the expression to zero it has established algebraic equivalence between the student’s and teacher’s answer. The test includes trigonometrical and logarithmic simplifications. Other AnswerTests could be devised which distinguish between different algebraic forms of an answer (e.g. factored or unfactored form).

The result is a data structure comprising the following:

1. A displayed form of the student’s answer.
2. TRUE or FALSE, depending on the result of comparing the two expressions.
3. Feedback, to be given to the student.
4. An AnswerNote, used for statistical analysis by the teacher.

While Task B could, strictly speaking, be reduced to Task A, the authors preferred to keep these two operations separate. One reason for this is that not all AnswerTests require the use of the CAS. A string match answer test has been implemented, which uses the PHP functions instead. Regular expression matching is also implemented as an AnswerTest, which does not require CAS. Again, PHP and CAS are being used for their respective strengths.

### 2.1 Learning Objects

While the CABLE authoring system refers to individual questions, these are in reality question templates, which may contain randomised local variables that are instantiated through CAS instructions. A question is the smallest exchangeable assessment object and is analogous to an item, as described in draft version 2 of the IMS Global Question and Test Interoperability (QTI) standard [7]. These specifications adopt the terminology of item templates, variables and template processing. Just as with items, CABLE questions comprise more than the actual “question”, and include a number of fields to describe question variables, response processing, feedback and worked solutions.

For example, in the question:

Differentiate \((x - 4)^5\) with respect to \(x\),

random values could be used in place of the 4, 5, and variable \(x\). Each student can receive a similar, but distinct, question instance (QTI cloned item). Of course, attention must still be paid to the choice of random parameters, in order to prevent the generation of impossible or trivial questions [8].

### 2.2 Option structure

In any CAA system there is a tension between system flexibility and simplicity of question authoring. At one end of this spectrum, it should be possible in each question to alter every system feature, from the question stem, answer and number of marks available right through to the fonts used, the teacher’s email address and the URL for a help page. At the other end of the spectrum, questions need to be authored with a maximum of efficiency. Really, the only compulsory fields are:

1. the question stem: i.e. the actual “question” posed to the student, and
2. the answer.

It should be possible to author a viable question by only entering this information, with sensible default values for other options being taken from a context.

The present solution to this problem is a cascading options structure. Options are global variables that the questions and quizzes (QTI assessments) rely upon.

Examples of currently implemented options include display method, number of marks and penalty value for repeat attempts. The AnswerTest used is also an
option, with the default being algebraic equivalence. The options structure appears very flexible, and hence useful. For example, an option allowing or forbidding decimal expressions for fractions could also be implemented. If allowed, another option could set the required level of accuracy in decimal places or significant figures.

Option values are set at the system level and can be changed at the quiz level, overriding the system level values, or at the question level, overriding system and quiz level values. When authoring a question or quiz, instructors can choose to use the system level option values (referred to within the user interface as the default values) and create new questions and quizzes quickly and consistently. Allowing instructors to change options provides maximum flexibility and control. Changing a quiz level option will affect all questions in the quiz that have assumed the default value.

It is envisaged that the current options structure could be extended in a number of directions. This of course includes adding other options as described above, or allowing students to choose some options themselves and override those set by the system. Examples of these might include the display format and input method preferred (e.g., text entry or equation editor). While this latter extension has not been implemented, the current options structure would make this sort of flexibility relatively straightforward to add.

The options allow a single CABLE question to behave in quite different ways when used for different student groups, with different options set. Note however, the option structure developed in this project is at odds with the IMS QTI view of an assessment item as being self-contained: CABLE questions may assume much from the quiz context, including for example the default AnswerTest to be used. This will make interoperability with systems other than CABLE difficult, unless questions are assumed to take the contextual values when exported.

2.3 Response Processing

Response processing is the mechanism that evaluates a student’s response, and assigns a numerical mark and written feedback. Currently, CABLE requires students to enter their answers in free text format using CAS syntax. In CABLE the numerical mark and penalty scheme used by the AiM CAA system has been replicated. A validation method checks the answer for syntax errors before it is submitted to the CAS for marking. Students are not penalised for syntax errors, as with AiM, and can also choose to use the validation method to ensure that the CAS is interpreting their answers as intended before submitting them for marking. The actual response processing procedure, however, is quite different from AiM.

One problem with AiM, and other existing systems, is the difficulty inherent in writing questions that provide extensive response processing and feedback. Often the question author becomes a computer programmer. This project has sought to overcome this problem by developing a conceptual response processing flow chart, as part of the following response processing procedure.

The response processing procedure is evoked when the student enters their answer and asks for it to be marked. First, a list of variables is generated in the following order:

1. instantiated versions of the question variables;
2. the student’s answer, assigned to a variable name;
3. answer variables, which may depend on either of the previous variables.

This list is evaluated in order by the CAS, just as described in Task A. Note that the answer variables may depend on both the question variables and the student’s answer. The student’s answer may be processed mathematically, allowing it to, for example, be differentiated or manipulated algebraically.

Next, the response processing flow chart is traversed. Each node of the chart evaluates an AnswerTest, and depending on the result of this either the TRUE or FALSE branch is executed. The two objects compared are nominally known as the student’s answer and teacher’s answer, although of course they can be any expressions, including any of the variables. Each branch then does the following:

1. Adjusts the mark, and penalty.
2. Provides feedback to the student.
3. Adjusts the AnswerNote (used for statistical purposes).
4. Goes to a nominated node.

No node may be executed more than once, preventing infinite loops.

This mechanism allows a flow-chart structure to be developed to implement complex branching of marking schemes. The question author is required to enter only the expressions to be compared, nominate an AnswerTest, and write the feedback. A minimum amount of CAS specific code is thus required. The question author is also not writing complex if then else statements in code.
As an example, consider the following question.

Give an example of a quadratic \( p(x) \), with roots at \( x=1 \) and \( x=3 \).

One response processing flow chart could be comprised of the following nodes. Note that here, marks have not been proposed, although partial credit can be assigned in CABLE for each TRUE/FALSE branch of each node. Partial credit may consist of either an absolute mark, e.g. “mark is 0.5”, or an arithmetic operation on the existing mark, for example, “add 0.25 marks”. They have been omitted here, since the decision taken for the exact mark weighting assigned to each node is not the important feature of this example.

<table>
<thead>
<tr>
<th>Node</th>
<th>AnswerTest</th>
<th>T/F</th>
<th>Feedback</th>
<th>Next no.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>( p ) depends on ( x )?</td>
<td>T</td>
<td>[none]</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F</td>
<td>Your answer should depend on ( x ). Please try again.</td>
<td>END</td>
</tr>
<tr>
<td>2.</td>
<td>degree(( p ))=2?</td>
<td>T</td>
<td>[none]</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F</td>
<td>Your answer should be a quadratic in ( x ), but is not.</td>
<td>3</td>
</tr>
<tr>
<td>3.</td>
<td>( p(1)=0 )?</td>
<td>T</td>
<td>[none]</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F</td>
<td>Your answer should have a root at ( x=1 ), but does not.</td>
<td>4</td>
</tr>
<tr>
<td>4.</td>
<td>( p(3)=0 )?</td>
<td>T</td>
<td>[none]</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F</td>
<td>Your answer should have a root at ( x=3 ), but does not.</td>
<td>5</td>
</tr>
<tr>
<td>5.</td>
<td>Full marks?</td>
<td>T</td>
<td>Correct answer, well done!</td>
<td>END</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F</td>
<td>Your answer is not correct, please try again.</td>
<td>END</td>
</tr>
</tbody>
</table>

Experience with other systems suggests that this will be sufficient for the vast majority of response processing tasks. Interesting discussions with the author of the Wallis system revealed the independent development of almost identical flow chart systems.

Depending on the options employed, students may be permitted to make multiple attempts at a question, incurring a penalty value each time, or may be able to request a new version of a question for further practice. This gives the students an incentive to reflect on the feedback they have received and retry the question. Previous implementations of similar systems have demonstrated that this enhances students’ learning when compared to traditional paper-based assignments [9].

2.4 Security

Security is an important consideration in the design of any assessment system. While CABLE relegates user authentication to the VLE, it is still necessary to account for possible user (student and instructor) actions that could compromise system integrity.

A CAS is a very powerful tool that may allow access to underlying operating system commands. Consequently there are two levels of parsing of any CAS expression. The bottom level is implemented by the CABLE server scripts and prevents students and instructors from accessing system commands. The top level is via the CABLE options structure and allows the teacher to specify both allowed and forbidden CAS commands for pedagogic reasons. As with other options, these can be set at the system, quiz or question level.

The CABLE server scripts listen for user requests on a single port and then delegate these requests to various CAS processes listening on separate ports. In order to prevent bypassing the main entry point and gaining direct access to the CAS, the system assigns a unique ID number to each CAS process at run time. Only requests containing this unique ID can be evaluated by the CAS.

User input may contain CAS instructions that are both allowed and syntactically correct, but require an excessive amount of processing time for the CAS to evaluate. This could constitute a denial of service attack, if performed by a malicious user. For example, a student response may contain the CAS instruction \( \text{expand}(x+y+z)^{1000} \), where expand is an allowed command. CABLE accommodates this situation by interrupting the CAS and sending an error message to the user if the CAS is still evaluating a command after a predetermined timeout value has been reached.

3. Implementation

Currently, the CABLE server and interface scripts are written in PHP 5. These are used to communicate with the Axiom CAS and a MySQL database. The
initial implementation also includes a set of web-based PHP 5 scripts in order to demonstrate the functionality of the system without reference to a specific VLE. All of the CABLE components and supporting software are available open source and are free from licensing fees.

4. Conclusions and Future Work

The CABLE system provides an open source, online infrastructure to author and execute valid, reliable, humane and economical assessments. Currently the IMS QTI Standard, version 2, does not cater for items with the sophisticated response processing which CAS enabled CAA systems such as CABLE provide. However, computer algebra marked mathematical learning objects are likely to become more common, and it is hoped experiences with CABLE, and subsequent developments, will be influential on future standards.

Documented source code will be available from the CABLE project website at http://www.cable.bham.ac.uk in November 2004.

The authors welcome collaborations with others, as users or developers, in the future.

Acknowledgements

This project is funded through the LTSN Maths, Stats and OR Network and a grant from Microsoft UK Ltd to The Centre for Educational Technology and Distance Learning (CETADL) at The University of Birmingham, United Kingdom.

References