A Delivery Engine for QTI Assessments

G. B. Wills, J. S. Hare, J. Kajaba, D. Argles, L. Gilbert and D. E. Millard
School of Electronics and Computer Science, University of Southampton, Southampton, UK

***Abstract*—The IMS Question and Test Interoperability (QTI) standard has had a restricted take-up, in part due to the lack of tools. This paper describes the 'ASDEL' test delivery engine, focusing upon its architecture, its relation to item authoring and item banking services, and the integration of the R2Q2 web service. The tools developed operate with a web client, as a plug-in to Moodle, or as a desktop application. The paper also reports on the load testing of the internal services and concludes that these are best represented as components. The project first developed a Java library to implement the system. This will allow other developers and researchers to build their own system or incorporate aspects of QTI they want to implement.**

***Index Terms*—E-assessment, Software tools, Question and Test Interoperability, QTI, e-learning.**

I. INTRODUCTION

At the 2006 JISC/CETIS conference, the UK assessment community confirmed that kick-starting the use of the IMS Question and Test Interoperability version 2 specifications was a high priority. The conference concluded that there needed to be a robust set of tools and services that conformed to the QTIv2 specification to facilitate this migration. It was also felt that formative assessment would be where such a system would be used by early adopters.

Formative assessment aims to provide appropriate feedback to learners, helping them gauge more accurately their understanding of the material set. It is also used as a learning activity in its own right to form understanding or knowledge. Lecturers/teachers often do not have the time to develop, set, and then mark formative assessment as much as they would like. A formative e-assessment system allows lecturers/teachers to develop and set the work once, allows the learner to take the formative test at a time and place of their convenience, possibly as often as they like, to obtain meaningful feedback, and to see how well they are progressing in their understanding of the material. McAlpine [9] also suggests that formative assessment can be used by learners to "highlight areas of further study and hence improve future performance". Draper [10] distinguishes different types of feedback, highlighting the issue that although a system may provide feedback, its level and quality is still down to the author.

E-learning assessment covers a broad range of activities involving the use of machines to support assessment, either directly (such as web-based assessment tools or tutor systems) or indirectly by supporting the processes of assessment (such as quality assurance processes for examinations). It is an important and popular area within the e-learning community [4, 1, 2]. From this broad view of e-learning assessment, the domain appears established but not mature, as traditionally there has been little agreement on standards or interoperability at the software level. Despite significant efforts by the community, many of the most popular software systems are monolithic and tightly coupled, and standards are still evolving. To address this there has been a trend towards Service-Oriented Architectures (SOA). SOAs are an attempt to modularize large complex systems in such a way that they are composed of independent software components that offer services to one another through well-defined interfaces. This supports the notion that any of the components could be 'swapped' for a better version when it becomes available. SOA frameworks are being used as a strategy for developing frameworks for e-learning [3, 5]. The e-assessment domain has been mapped and a framework constructed [11].

A leading assessment standard has emerged in Question and Test Interoperability (QTI) developed by the IMS Consortium. The QTI specification describes a data model for representing questions and tests and the reporting of results, thereby allowing the exchange of data (item, test, and results) between tools (such as authoring tools, item banks, test constructional tools, learning environments, and assessment delivery systems) [8]. Wide take-up of QTI would facilitate not only the sharing of questions and tests across institutions, but would also enable investment in the development of common tools. QTI is now in its second version (QTIv2), designed for compatibility with other IMS specifications, but despite community enthusiasm there have been few examples of QTIv2 being used, with no definitive reference implementation [6, 7]. The other problem is that no sooner has the reference implementation been finished than the specification is likely to be updated. Also people have their own views on how this should be implemented, so in this work we have also looked at how to implement the specification in such a way to stop it from becoming obsolete the moment the implementation is finished.

In this paper we first give an overview of the QTI specification and the R2Q2 project (a project for rendering and responding to questions). An overview of the architecture for the ASDEL project (a project for rendering and conducting tests) and the tools developed is then described in section 4. In section 5 we describe the rationale for first building a Java library for QTI and in section 6 we present the results of load testing the tools. In section 7 we present a discussion of this work and some conclusions.

II. QTI

The IMS QTI Specification is a standard for representing questions and tests with a binding to the eXtended Markup Language (XML, developed by the W3C) to allow interchange. Each item (question) has three core elements: *ItemBody* declares the content of the

item itself, *ResponseDeclaration* declares a variable to store the student's answer, and *OutcomeVariables* declares other variables, in this case a score variable to hold the value of the result. There are 16 core item types described in version 2 of the QTI specification (QTIv2). These are:

1)	Choice	2)	Hotspot
3)	Order	4)	Select point
5)	Associate	6)	Graphic
7)	Match	8)	Graphic Order
9)	Inline Choice	10)	Graphic Associate
11)	Text Entry	12)	Graphic Gap Match
13)	Extended Text	14)	Position object
15)	Hot Text	16)	Slider

The different item types can be written with templated questions or adaptive questions, providing an author with numerous alternative methods for writing questions appropriate to the needs of the students. Templated questions include variables in their item bodies that are instantiated when a question is rendered (for example, inserting different values into the text of mathematics problems). Adaptive questions have a branching structure, and the parts that a student sees depends on their answer to previous parts of the branch. In total these allow at least sixty-four different possible combinations of question types.

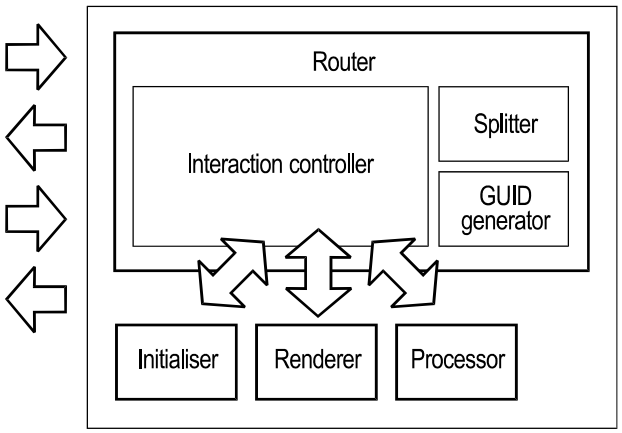

Figure 1. The R2Q2 Architecture

III. R2Q2

The R2Q2 project was an earlier project funded by JISC for rendering and responding to the 16 core item types, discussed in section 2. The R2Q2 service allows a student to view a question, answer a question, and view the feedback. The R2Q2 engine (see Figure 1) is a loosely coupled architecture comprising of three interoperable services. All the interactions with and within the R2Q2 engine are managed by an internal component called the Router.

The Router is responsible for parsing and passing the various components of the item (QTIv2) to the responsible web services. It also manages the interactions of external software with the system, and it is therefore the only component that handles state. This enables the other services to be much simpler, maintaining a loosely coupled interface but without the need to exchange large amounts of XML.

The Processor service processes the user responses and generates feedback. The Processor compares the user's answer with a set of rules and generates response variables based on those rules. The Renderer service then renders the item (and any feedback) to the user given these response variables.

IV. ASDEL

The QTI specification details how a test is to be presented to candidates, the order of the questions, the time allowed, etc. The ASDEL project built an assessment delivery engine to the IMS QTI 2.1 specifications that can be deployed as a stand-alone web application or as part of a SOA enabled Virtual Learning Environment (VLE) or portal framework.

The core components of the ASDEL system were built into a Java library called JQTI (see section 5). The JQTI library enables valid QTI assessment XML documents to be interpreted and executed. The library also provides auxiliary services like the handling of QTI content packages and the provision of valid QTI conformance profiles and reports.

The first instantiation of the library is called *Playr*, the ASDEL tool which delivers a QTI test. The AssemblerRenderingEngine part of this tool is responsible for the assembly and rendering of output (i.e. questions and associated rubric). Initially, only an XHTML renderer has been developed; however, the design of the *Playr* enables different renderers to be plugged in. Figure 2 presents the conceptual design diagram for the *Playr*.

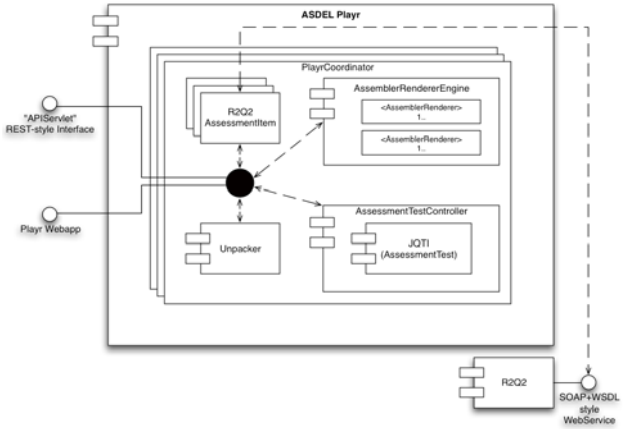

Figure 2. Architecture for the Assessment Delivery System

The ASDEL project integrated with the other projects in the JISC Capital Programme on item banking (Cambridge: 'Minibix') and item authoring (Kingston: 'AQuRate') to provide a demonstrator (see Figure 3). Together the three projects provide an end-to-end service: AQuRate allows item authoring, which are stored in MiniBix. A test incorporates these items and is played through the ASDEL *playr*. Regular workshops between the projects ensured this happened. These projects can be found at www.qtitools.org.

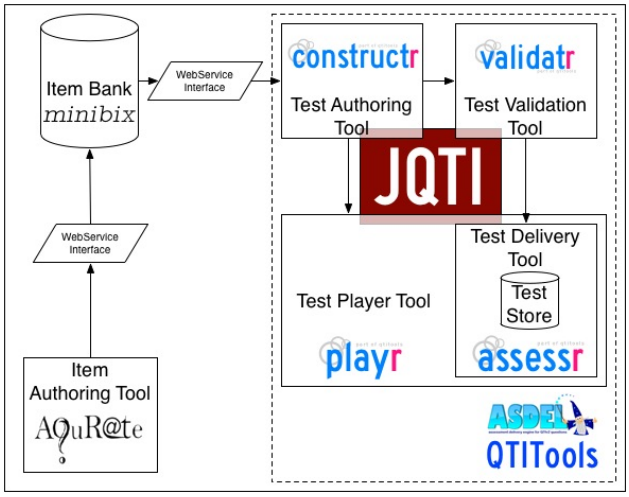

Figure 3. Integration of the ASDEL, AQuRate Item Authoring (Kingston) and MiniBix, Item Banking (Cambridge)

The test player only rendered the test, while the rendering of the questions and the response to these questions was the responsibility of the R2Q2 web service. During the design and implementation of ASDEL a number of small problems were identified in R2Q2 that needed to be fixed. Firstly, the default R2Q2 render renders full xhtml pages rather than rendering fragments—ASDEL requires fragments so that it can append various elements of rubric and other textual information about the test before and after the question. The output from ASDEL also needs to be in the form of a fragment so that it can be integrated with a VLE or portal framework. The second problem with R2Q2 was due to the way it always rendered feedback that was included in an item (at the correct time of course)—the problem is that the QTI assessment specification allows the delivery engine to control whether or not an individual item should render feedback.

Further ASDEL tools were developed. The *validatr* tool provides the validation of a QTI test and also gives indications of any errors in the QTI document. Similar to an integrated design environment for writing program code, *validatr* also allows experienced users to correct the XML of the test. The *validatr* has a visual front end that allows users to visualize the structure of the test and the different paths students can take through the assessment (see Figure 4).

The test player tool only delivers the test, so the *assessr* tool manages the test for the lecturer or teacher. Lecturers can upload a class list from a spreadsheet, schedule the test, put embargos on the release of the test information, etc.

The *assessr* tool sends a token and a URL for the test to each student who can log into the *playr* using the token and take the test. The *assessr* allows the lecturer to see which test they have set, who has taken them, and which tests are shared with someone else (see Figure 5).

Constructr is an extremely lightweight test construction tool. This is distinguished from item authoring since it simply allows a lecturer to select a pool of questions from an item bank and put them into a basic test.

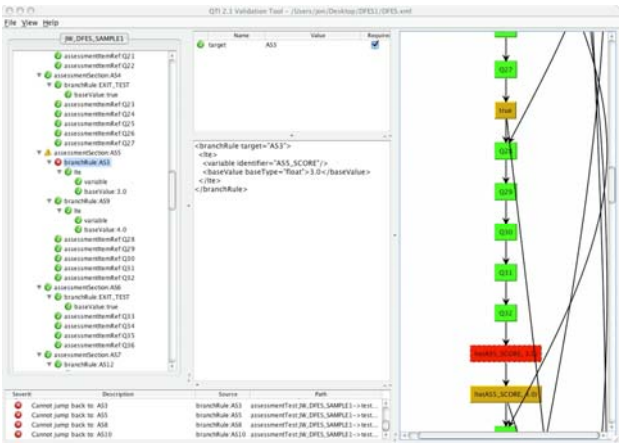

Figure 4. *Validatr* screenshot

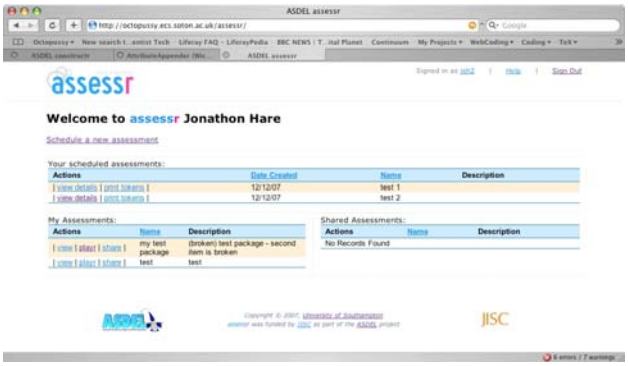

Figure 5. *Assessr* main screen

During the development of the project, a number of modifications and fixes were made to the R2Q2 tool in order to facilitate its interoperability with the ASDEL toolset. The biggest changes were the development of a better web service interface, improvements to enable finer control over the rendering, and the addition of basic MathML support.

We have also developed a version of the *playr* tool called *PlayrDE*, which can be downloaded and used from the desktop without an Internet connection. It provides a simple online QTI validation tool which enables a validation report to be generated for an uploaded content package or assessment XML file. Finally, we have developed an activity module plug-in for the Moodle VLE that replicates all of the functionality of *assessr*, but directly integrated with Moodle.

V. JQTI: WHY BUILD A LIBRARY FIRST

In this section we reflect on some of the issues and factors that needed to be considered in implementing a software library for the QTI specification.

The core of the ASDEL software is a library we call JQTI. JQTI is essentially an interpreter for IMS QTI v2.1 XML. QTI XML is rather unlike most XML documents, as it contains instructions as well as data. These instructions determine how tests and items are presented, processed, and evaluated. The QTI specification defines a programming language that happens to be expressed in the form of an XML document. For the ASDEL project, JQTI implements all of the parts relevant to the AssessmentTest class, although we hope in the future to

add the remaining (AssessmentItem) classes and to retrofit JQTI into R2Q2.

In implementing JQTI we considered two options; we could either use a binding technology such as JAXB or Castor to bind the QTI XML schema to a set of automatically generated Java classes, or we could write the whole library from scratch, using a DOM parser to parse the XML. XML binding technologies work well for binding to XML containing data, but are problematic when the XML contains instructions that need to be evaluated. In this case, every automatically generated class would have to be manually modified to have a 'behavior' added to it so that it could be evaluated. Another problem with binding to the XML schema is that the schema is not nearly as expressive as the full QTI specification document—it is possible to have an XML document that validates against the schema, but is not valid QTI. It is for these reasons that we used "custom classes + DOM parser" in our implementation of JQTI.

A comprehensive library for handling QTI needs to perform two core operations; it needs to be able to generate QTI XML, and it needs to be able to parse/evaluate QTI XML. The library should not be responsible for the actual rendering of items/assessments, although it should provide relevant hooks to obtaining the required information needed for rendering. The reason is that the specification itself is agnostic towards how content should be rendered (even though most implementations so far have rendered XHTML).

QTI XML is more of a programming language than a data format. This means that there are some very special considerations that need to be taken into account when designing and implementing a QTI library. Perhaps the most significant of these considerations was that it is possible (and rather easy) to write a QTI XML document that is syntactically valid according to the QTI XML schema, but is not syntactically or semantically correct according to the specification. As an example of this, consider the following XML fragment:

```
<equal toleranceMode="relative">
      <baseValue baseType="float">
            1.0
      </baseValue>
      <baseValue baseType="float">
            1.0
      </baseValue>
</equal>
```

This XML will validate correctly against the QTI XML schema, but it is not valid against the specification because the element is missing the tolerance attribute that is required (the toleranceMode is relative). An example of a semantic error occurs in referring to an element (for example in the target of a branchRule) using an identifier that does not actually exist.

This consideration mandates that the library must be able both to validate the syntax of the QTI XML documents that it reads (in a more comprehensive manner than by simply validating the XML against the schema) and to assess the semantic correctness of the document. Semantic correctness is important, because the chance of any errors or exceptions being thrown during the execution of a test or item needs to be minimized, though it is impossible to check every possible error case because processing the XML will rely on user input. Checking QTI XML for semantic correctness (at least as far as possible) requires a static analysis on the XML document in order to verify that it will work correctly as it is executed.

There are two possible implementations for processing and evaluating a QTI XML document: either parse the data on a line-by-line basis and perform steps as required (for example by user response), or read in all the data and construct an object tree. The first option has the advantage of lower memory consumption, but has the disadvantages of a difficult implementation for syntactic and semantic validation (c.f. static analysis), and for moving around (i.e. backward/forward through a test).

The class hierarchy in the library also needs to be considered. The specification provides some hints as to how QTI classes are related, but is not an implementation guide. Many of the classes defined in the QTI specification are implemented in our JQTI library, though the hierarchy is often a little different—i.e. all the Java classes that are related to QTI classes inherit from a common abstract XmlObject class. Another consideration is that some classes defined in the specification are not relevant to the processing and evaluation of the XML document, and only serve as hints to the renderer (i.e. the XHTML classes). These classes usually don't need to have any concrete implementation associated with them. The QTI specification also serves as a good pointer as to the breakdown of the class structure into a suitable granularity. For example, rather than implementing all of the expression classes in a single class (there are so many of them, and some are rather complex), the specification suggests that all the expressions would be individual classes inheriting a common abstract expression class whose methods can be overridden for evaluation of the expressions.

The library requires a good testing framework, and the QTI specification forms a basis for determining the functional requirements for each class. The library implementation can make use of these for constructing a set of unit tests for individual components, as well as for determining when runtime exceptions should be thrown.

In summary, a good QTI library implementation needs to provide a set of custom classes that implements the functionality of the QTI specification (i.e. items and assessments can be run, evaluated, validated, etc), and that also binds to the XML (so that tests and items can be read in and written out). The library also needs to handle runtime errors in a systematic way through the use of exceptions, and be backed by a comprehensive test suite that validates that it conforms to the specification.

VI. LOAD TESTING

The original design for the ASDEL *playr* tool called for a number of small loosely coupled internal services communicating using SOAP, together with an external SOAP-based API. Running load tests with tens of simultaneous users showed significant failings in the quality of service the tool could provide. These were traced to numerous problems with the standard Java libraries we used for creating SOAP web services. By redeveloping the internal services as components we removed the errors, and the system worked well in

simulations with hundreds of simultaneous users. This supports the idea that small internal services are better provisioned as components and that the whole tool can be wrapped as a web service. In addition to redeveloping the internal infrastructure we also refactored the original external SOAP API into a much easier-to-use REST-style API. This facilitated the fast construction of the Moodle plug-in.

Figure 6 illustrates the performance differences between the original web service-based design and the componentized design. The graph shows two sets of curves; one shows the throughput for a given number of users, whilst the other shows the error rate. Throughput is the number of requests the software is dealing with per second, and initially increases as a function of the number of users. It eventually peaks and then decreases as the server resources become exhausted (i.e. server runs out of available processing power, memory, file handles, etc). The error rate is the number of times the software fails to produce the expected outcome (for example, fails to load a page due to resource limits). In real e-assessment scenarios no errors can be tolerated, so it is useful to determine how many simultaneous users the software can support before errors will start to occur. The curves on the graph clearly show that the componentized version of the *playr* performs much better than the web service version; so much so that the number of simultaneous users can increase from about 10 to 400 before errors will start to occur. The reasons for this somewhat dramatic improvement are numerous, but are mostly related to the reduction in memory and CPU resource usage from not having to continuously encode and decode SOAP XML messages.

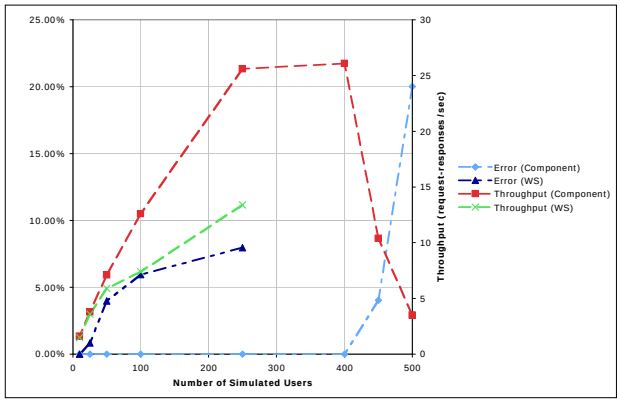

Figure 6. Performance of the original *playr* versus the improved design.

VII. CONCLUSIONS

At the 2006 JISC/CETIS conference, the UK assessment community confirmed that kick-starting the use of the IMS Question and Test Interoperability version 2 specifications was a high priority. The conference concluded that there needed to be a robust set of tools and services that conformed to the QTIv2 specification to facilitate this migration.

R2Q2 is a definitive response and rendering engine for QTIv2 questions. While it only deals with question items, it is essential to all processing of QTI questions and so would form the core component of all future systems.

In the ASDEL project we built an assessment delivery engine to the QTIv2.1 specifications. Like R2Q2, it is a web service-based system that can be deployed as a stand-alone web application or as part of a SOA-enabled VLE or portal framework. We have also built a desktop version for those who are not connected to the internet.

The engine itself cannot function alone so a small set of lightweight support tools have also been built. The engine, in combination with the tools, provides:

- Delivery of an assessment consisting of an assembly of QTI items, with the possibility that the assessment is adaptive and that the ordering of questions can depend on previous responses,
- Scheduling of assessments against users and groups,
- Rendering of tests and items using a web interface,
- Marking and feedback, and
- A web service API for retrieving assessment results.

We have provided a small set of lightweight tools that will enable a lecturer or teacher to quickly manage a formative assessment using the Web. The outcome of this project is the first open source test delivery system for tests written in the QTIv2 format. The ASDEL toolset is freely available. The toolset is designed to work as a standalone set of web based tools or on the desktop. Alternatively these tools can be integrated into a VLE (or other web based course delivery systems). We have demonstrated this by writing a plug-in for Moodle.

The library has proven to be an effective method of not only producing a reference implementation but also in solving the problem of becoming obsolete the moment it is implemented. The library will easy accommodate changes in the specification and allow people to implement a QTI solution that suits their circumstances.

We originally set out to build the set of tools as web services that could be used together to form the ASDEL web service in another application. However load tests showed that having small, single function web services caused too much of an overhead in terms of resource allocation and limited the number of simultaneous users. This has implications for the e-learning framework, in that the granularity of the service matters when being combined to produce an effective system.

It is envisioned that lightweight suites of tools developed from ASDEL and R2Q2 will enable early adaptors of, and those researching into, e-assessment an opportunity to experiment with the alternative ways of presenting tests afforded by the QTI specification.

REFERENCES

[1] Bull, J., and McKenna, C. Blueprint for Computer Assisted Assessment. Routledge Falmer, 2004.

[2] Conole, G. and Warburton, B. "A review of computer-assisted assessment". ALT-J Research in Learning Technology, vol. 13, pp. 17-31, 2005.

[3] Olivier, B., Roberts, T., and Blinco, K. "The e-Framework for Education and Research: An Overview". DEST (Australia), JISC-CETIS (UK), www.e-framework.org, accessed July 2005.

[4] Sclater, N. and Howie K. User requirements of the ‘‘ultimate’’ online assessment engine, Computers & Education, 40, 285–306 2003.

[5] Wilson, S., Blinco, K., and Rehak, D. Service-Oriented Frameworks: Modelling the infrastructure for the next generation of e-Learning Systems. JISC, Bristol, UK 2004.

[6] APIS [Assessment Provision through Interoperable Segments] - University of Strathclyde–(eLearning Framework and Tools Strand) http://www.jisc.ac.uk/index.cfm?name=apis, accessed 30 April 2006.

[7] Assessment and Simple Sequencing Integration Services (ASSIS) – Final Report – 1.0. http://www.hull.ac.uk/esig/downloads/Final-Report-Assis.pdf, accessed 29 April 2006.

[8] IMS Global Learning Consortium, Inc. IMS Question and Test Interoperability Version 2.1 Public Draft Specification. http://www.imsglobal.org/question/index.html, accessed 9 January 2006.

[9] McAlpine, M. Principles of Assessment, Bluepaper Number 1, CAA Centre, University of Luton, February 2002.

[10] Draper, S. W. Feedback, A Technical Memo Department of Psychology, University Of Glasgow, 10 April 2005: http://www.psy.gla.ac.uk/~steve/feedback.html.

[11] Wills, G., Bailey, C., Davis, H., Gilbert, L., Howard, Y., Jeyes, S., Millard, D., Price, J., Sclater, N., Sherratt, R., Tulloch, I. and Young, R. (2007) AN E-LEARNING FRAMEWORK FOR ASSESSMENT (FREMA). In: *International CAA Conference*, 10th - 11th July 2007., Loughborough UK.

AUTHORS

G. B. Wills is a senior lecturer in the School of Electronics and Computer Science, University of Southampton, Southampton, Hampshire, SO17 1BJ, UK (e-mail: gbw@ ecs.soton.ac.uk).

J. S. Hare is a Research Fellow in the School of Electronics and Computer Science, University of Southampton, Southampton, Hampshire, SO17 1BJ, UK (e-mail: jsh2@ ecs.soton.ac.uk).

J. Kajaba is a Research Assistant in the School of Electronics and Computer Science, University of Southampton, Southampton, Hampshire, SO17 1BJ, UK (e-mail: jk2@ ecs.soton.ac.uk).

D. Argles is a lecturer in the School of Electronics and Computer Science, University of Southampton, Southampton, Hampshire, SO17 1BJ, UK (e-mail: da@ ecs.soton.ac.uk).

L. Gilbert is a lecturer in the School of Electronics and Computer Science, University of Southampton, Southampton, Hampshire, SO17 1BJ, UK (e-mail: lg3@ ecs.soton.ac.uk).

D. E. Millard is a lecturer in the School of Electronics and Computer Science, University of Southampton, Southampton, Hampshire, SO17 1BJ, UK (e-mail: dem@ ecs.soton.ac.uk).

Manuscript received 05 May 2008. This Work was funded in the UK by the Joint Information Systems Committee (JISC).