QUIZIT: An Interactive Quiz System for WWW-based Instruction.

Lúcio C. Tinoco, Edward Fox, Roger Ehrich
Computer Science Dept.
Virginia Polytechnic Institute & State University
Blacksburg, VA 24061-0106. U.S.A.
{tinoco, fox, ehrich}@vt.edu
http://pixel.cs.vt.edu/~ltinoco/

Hugo Fuks
Computer Science Dept.
Pontificia Universidade Católica do Rio de Janeiro
Rio de Janeiro, RJ 22453900. Brazil.
hugo@inf.puc-rio.br
http://www.inf.puc-rio.br/~hugo/

Abstract

In distance learning and self-paced instruction, student evaluation with prompt feedback has always been a critical issue. Instructors and teaching assistants are often confronted with scores of e-mail messages per day from students, the need to work night and day to grade exams, and heavy workloads late in the term. This paper describes the design of an automatic evaluation system that aims to eliminate some of the workload on instructors, by providing a convenient way of authoring interactive quizzes while addressing issues of reuse, portability and immediate feedback to students. We build on the standard models of Computer-based Training (CBT) and Web-based Training (WBT) systems, showing how we can combine their best features and overcome some of their problems, keeping in mind the specific requirements of our educational environment.

KEYWORDS: Student evaluation, on-line quizzes, adaptive computer quizzes.

1. Introduction

Adequate student assessment plays a major role in the effectiveness of distance and self-paced learning courses. Even for small classes (30 students or less), instructors often feel overwhelmed by the effort required to grade tests on demand within a reasonable time frame. In addition, grading such tests is often accompanied by exchange of e-mail messages, to provide personalized guidance and remedial feedback throughout the evaluation process. It seems natural to think that if an instructor wants to provide a more efficient means of performing student evaluation on demand for a widely distributed and larger audience, reusable and adaptable tools for automatic evaluation and feedback need to be developed.

Most of today's systems for automatic student evaluation are extensions of the so called computer-based training (CBT) systems, available in commercial multimedia authoring systems like Authorware, ToolBook, and others (Nichols, 1996; Macromedia Inc., 1996; Asymetrix Inc., 1996; Logic Resources Inc., 1996). Although most of these systems tend to be very powerful, providing Graphical User Interface (GUI) based authoring languages, student database support and test adaptability, they suffer from the lack of two essential features to be successfully applicable in distance and self-paced learning in universities: portability and reusability. In particular, instructors should be able to store, update, and

retrieve tests easily in appropriate formats in large digital libraries (data independence in document representation is therefore a primary concern). Further, access to these libraries should be platform independent, and available through a network. On the other hand, students should not be concerned with the internals of different software packages, nor with the general problems of adapting to a new environment every time a new version of the network software is changed, for example.

With the upsurge of the World Wide Web (WWW) as a de-facto standard for delivering on-line course material in universities (McManus, 1995a; McManus, 1995b) , several Web-based training (WBT) systems incorporating test authoring/taking modules have begun to appear (Nichols, 1996; Wheeler, 1996). Although inherently more portable, these systems tend to be much less powerful than their CBT counterparts. Most of them use HTML, or HTML extensions as authoring languages, focusing more on presentational aspects of the tests than on their structure. Consequently, for each new HTML version, entire test banks may need to be upgraded (to comply to new presentational standards, for example) and/or rewritten, greatly affecting reuse. In addition, these languages tend to be much less powerful than CBT-type languages, in part because of their focus on presentational syntax. Tables 1 summarizes the strengths and weaknesses of CBT and WBT systems.

Strengths (CBT)	**Weaknesses (CBT)**
Use of GUI's	GUI is not always portable across platforms
Powerful authoring languages	There is no standard. Documents produced are not easily integrated with documents from other vendors.
Not only for quizzes but also for training in general	Systems tend to be very expensive.
Available on popular computer platforms	Use primarily standalone personal computers. Some systems allow networking, but all the machines have to have the software installed.

Strengths (WBT)	**Weaknesses (WBT)**
Mostly freeware or shareware	WBT systems tend to use HTML extensions, which decreases portability and reuse, as HTML itself changes.
Use the WWW to deliver instruction	In most WBT systems, authoring language is also not as powerful, and it is not possible to write adaptable quizzes.

Table 1: Strengths and Weaknesses of CBT/WBT systems

This paper describes the design and implementation of an automatic WWW-based quiz system called *QUIZIT*, currently being developed in the Virginia Tech Computer Science Department (Tinoco, 1996) , that will be used in several on-line courses available for distance and self-paced instruction. The following sections will present the technical features that distinguish *QUIZIT* from other systems. Section 2 discusses the advantages of using Standard Generalized Markup Language (SGML) for authoring and storing adaptive quizzes, and describes our design process. Section 3 explains how the different software components fit together, the pros and cons of using the WWW and the Common Gateway Interface (CGI), and how we have addressed potential security problems. Section 4 concludes with future developments and our formative evaluation methodology.

2. Using SGML to represent interactive quizzes

One of the problems with most training systems is that each of them has its own authoring environment. Although the CBT systems all tend to provide a large variety of features and graphical interface capabilities, these sometimes expensive systems are not easily portable across platforms. In a heterogeneous university setting this could be a real problem, for students may not have access to similar machines.

On the other hand, there are WBT systems that overcome this problem of portability, but with poor solutions. Although it is unquestionable that WWW browsers have the most desirable user interface in these environments, mainly because of their large and growing popularity, it is not clear that the HTML-based languages used on these systems are the most appropriate for authoring on-line quizzes. Firstly, HTML is mostly a *presentational* markup language. Although presentation is very important, when authoring specific types of documents like on-line quizzes, much of the presentational syntax carried by HTML-type languages can be omitted in exchange for better standardization (e.g., predefined styles) and higher consistency. Secondly, if we concentrate instead on *structural* or *descriptive* markup (Coombs, Renear, & DeRose, 1987), the *contents* are emphasized (as opposed to the presentation), leading to a more usable and well-defined authoring process, and better document integration with searchable digital libraries and test banks. Thirdly, as HTML evolves, every document authored in HTML may also need to change because of new style requirements, for example. As test libraries quickly build up, one could imagine the effort to translate every document. By using a stable structural markup and a compiler to generate HTML pages, only the compiler and not the whole set of quizzes needs to be changed when the target language (HTML) is modified.

Our approach in QUIZIT is to use SGML as a representation language (Bryan, 1988; Maler & Andaloussi, 1996). As a well accepted standard (ISO/IEC 8879), SGML is stable against frequent changes. Further, as the metalanguage used for the definition of HTML itself, it provides all the flexibility needed to define a content-oriented, yet powerful markup language for writing on-line adaptable quizzes. In addition, there are several authoring helpers for SGML available on the market (editors, validating parsers, etc.) for several different platforms. Alternatively, one could also write quizzes using a simple

text editor. The following subsection describes the QUIZIT markup language in greater detail, highlighting its simplicity and the features that allow the representation of adaptable quizzes.

2.1 The QUIZIT Markup Language (QML)

The following examples, taken from the QUIZIT tutorial (Figure 1a and 1b), illustrate the markup of a simple, one-page "soccer quiz". While it is not comprehensive, it contains most of the basic building blocks for authoring more general quizzes, some of which will be explained in more detail later. The WWW page generated from this is shown further on (Figure 2b).

```
<!DOCTYPE QUIZ SYSTEM "quiz.dtd">
<QUIZ ID="soccer.sgml">
<TITLE>Simple Soccer Quiz</TITLE>
<INSTRUCTION>
You should take this quiz now, and submit your answers as soon as you are done. This should take less than 90 minutes (i.e., do not spend more time than that). You can refer to your notes and the readings (textbook, soccer magazines).<P>
The honor code is in effect. All work on this quiz should be your own. Do NOT send a copy of your answers to anyone else, or look at anyone else's answers, anytime during this course.<P>
</INSTRUCTION>

<!-- This is a comment -->
<QUESTION-GROUP ID="quizAA.html">
<MCHOICE ID="q0001">
<DESCRIPTION>
Which country won the first soccer World Cup, in 1930?<P>
</DESCRIPTION>
<ANSWER VALUE="WRONG" LABEL="a">Brazil</ANSWER>
<ANSWER VALUE="WRONG" LABEL="b">Germany</ANSWER>
<ANSWER VALUE="RIGHT" POINTS="10"
     LABEL="c">Uruguay</ANSWER>

<HINT>This country has won the World Cup twice so far.</HINT>
<TRUEFALSE ID="q0002">
<DESCRIPTION>
Please assign TRUE or FALSE to each of the following questions.
</DESCRIPTION>
<ANSWER VALUE="FALSE" LABEL="a">Brazil has played in four World Cup finals</ANSWER>
<ANSWER VALUE="FALSE" LABEL="b"> The 1982 World Cup was played in Italy</ANSWER>
<ANSWER VALUE="TRUE" POINTS="10" LABEL="c">Paolo Rossi was the leading scorer of the 1982 World Cup </ANSWER>
</QUESTION-GROUP>
</QUIZ>
```

Figure 1a: SGML markup for our simple soccer quiz.

Quiz -- Unit AA

You should take this quiz now, and submit your answers as soon as you are done. This should take less than 90 minutes (i.e, do not spend more time than that.) You can refer to your notes and readings (textbook, articles).

The honor Code is in effect. All work on this quiz should be your own. Do NOT send a copy of your answers to anyone else, or look at anyone else's answers, anytime during this course.

This HTML code was generated by this SGML.

---

1. Which country won the first soccer World Cup, in 1930?

a)Brazil.
b)Germany.
c)Uruguay.

HINT: This country has won the World Cup twice.

---

2. Please assign TRUE or FALSE to each one of the questions.

a) Brazil has played a total of 4 World Cups finals.
TRUE FALSE

b) World Cup 1982 was in Italy.
TRUE FALSE

c) Paolo Rossi was the leading scorer of World Cup 1982.
TRUE FALSE

Please, send comments to tinoco@vt.edu

Figure 1b: HTML generated by SGML markup

The top-level element, `QUIZ,` appears in every document, and marks the beginning and end of the quiz. Each quiz also has a required ID attribute (a local system file name), which allows hypertext references to be made by other documents. The `<QUESTION-GROUP>` element, as will be further explained later, defines the group of questions the user will see in one WWW page. The HTML markup generated by the SGML code in Figure 1b is very simple, and uses the Web-based form mechanism to allow the students to enter their answers.

Currently, QUIZIT supports four different types of question: multiple-choice, true-false (both shown in the examples above), matching, and fill-in. The format is basically the same for all types of questions. HTML notation is supported also, and can be useful if one wants to refer to figures or movies inside a question's description, for example. In general, one can use HTML elements inside any textual descriptions (e.g., within `<DESCRIPTION>`, `<INSTRUCTION>, <TITLE>,` or `<ANSWER>` tags), although this is not recommended for the reasons of reusability described earlier in section 2.

Probably the most interesting feature, nevertheless, is the ability to write *adaptable* quizzes (also called 'sequential' quizzes) (Hansen, 1967), i.e., quizzes that provide remedial or progressive questions depending on the student's performance on previous quizzes. Suppose, for example, that student Lucio achieves zero points on question 1 and ten points on question 2. Maybe an instructor would want to provide some sort of remedial quiz for students like Lucio, who achieves less than 20 points on both questions. On the other hand, instructors may want to require students who achieve a higher score to take a more difficult exam next.

The way QUIZIT allows instructors to do this is by grouping questions that should be taken at the same time in one WWW page, and associating a `<REPLY>` element with it. A "reply" then can specify either a remedial quiz, a progressive quiz (what we called "harder quiz" in our Lucio example), or both. This allows a simple, yet powerful mechanism to provide adaptability in our syntax. Coming back to our

soccer example, we could add the following markup right before the </QUESTION-GROUP> tag near the end:

```
<REPLY REMEDIAL="soccereasier.html" NEXT="soccerharder.html" CUTOFF="20">
```

Here `CUTOFF` is a required attribute that specifies the branching condition for the quiz. If less than 20 points are achieved, the student will take the question group specified by the `REMEDIAL` attribute (Lucio's case). Otherwise, the question group defined in `NEXT` will be taken.

Although very simple to use, this syntax is very powerful. By combining question grouping and reply elements appropriately, one can author complex quizzes like the computerized, adaptive GRE tests (ETS 1996), for example (where you group one question per page, associating one reply element with it), or just simple one-page quizzes where no remedial or progressive actions are taken. In general, you can think of an entire quiz as a binary digraph data structure, where each node is a question-group and the one or two edges leaving it correspond to the two possible replies of the system (remedial or progressive), according to the branching value specified by the `CUTOFF` attribute. Figure 1c shows the graph structure associated with our soccer quiz, including the `CUTOFF` values for each of the nodes (pages). It is often helpful to picture this graph structure in mind when authoring more complex types of quizzes.

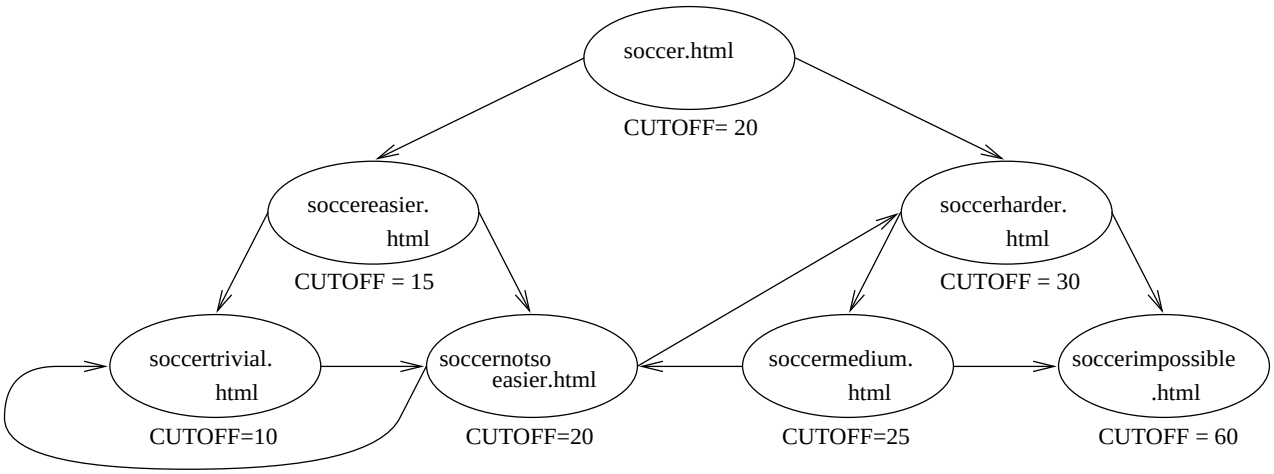

Figure 1c: Digraph structure of a soccer quiz.

The next section will explain the run-time environment of this system in more detail. We provide an overview of the architecture, and the reasons behind each particular original design feature. It is not in the scope of this paper to go into further detail on how to write and take quizzes with QUIZIT.

3. System Architecture

When designing the run-time environment for QUIZIT, we had to keep three fundamental requirements in mind:

a) *Use of the WWW to deliver quizzes*: This requirement can be explained by the dynamic nature of the university environment, where adaptive changes need to occur promptly during a course, requiring a stable networked environment and fast system feedback.

Another important issue is portability. Due to the amazing growth in popularity of the platform-independent WWW during the past few years, specially in universities, it is clear that implementing our system on any other platform would impose severe constraints on students, because of the great variety of different systems and the inherent difficulties of networking them. The Internet and the WWW seem to be the best solution for both of these issues, since information can be updated both instantly and seamlessly.

b) *Independence of authoring and run-time interfaces:* The process of authoring quizzes should be completely independent of the run-time process. In particular, as we distinguish two completely different classes of users (instructors and quiz-takers), it seems obvious to expect a high degree of decoupling between authoring and run-time modules. Unfortunately, this is not the case in most commercial systems, leading to even more portability and reusability problems, as instructors are required to author quizzes on the same machines used with the run-time processes. As we will see later in this paper, QUIZIT provides completely separate modules for authors (instructors) and quiz-takers (students).

c) *Security concerns*: Providing secure transactions between WWW clients and servers is still an open problem, specially in applications using the Common Gateway Interface (CGI). We have tried to avoid these problems by constructing an authentication layer on top of the runtime engine and by using a database model that supports locking and relatively secure transactions. It is important to stress, however, that authentication can never be conclusive since it is based solely on password validation. In fact, this is a general problem with remote authentication that instructors must be aware of when applying online quizzes in remote locations.

The next subsection explains in detail the run-time architecture of QUIZIT by walking through a typical scenario.

3.1 The Run-time Environment

We have called the set of modules that work "behind the scenes" whenever a user submits a WWW quiz page for grading *run-time environment* . Figure 2 illustrates the run-time engine, which is composed of several CGI scripts (Liu, 1994) written in C, and the databases maintained by them. The database engine used was Hughes Mini SQL server (Hughes Inc., 1996), which provides a portable, simple, free-of-charge, yet powerful application programming interface (API) that implements a subset of the SQL query language. Our WWW server is the NCSA httpd 2.4 server, running on a DEC Alpha.

Probably the best way to describe the sequence of actions that happen when a user ("quiz-taker") enters the system is through the use of scenarios. These provide a step by step, concrete, and comprehensive way of visualizing the actions performed by each of the run-time modules, and the interactions among them and the user. The following three-phase scenario describes a typical interaction cycle in QUIZIT's run-time system.

PHASE I: Authentication:

Lucio is finally ready to take the soccer quiz. As his browser points to the well-known QUIZIT URL, he sees an authentication page with several course options. He selects 'SOCCER 101', types his ID and password, and submits the page. Internally, the ID / password information is validated against a special database table. If there really is a Lucio with a valid password in the SOCCER 101 authentication database, a special session cookie is generated and it is passed to a CGI script called 'mkcookie'. This script reads another database table, called 'student_log', and retrieves the student record using that cookie and the ID information. A student record contains information like the last quiz taken ('quiz' field), the score on that quiz ('score' field), and the next quiz to be fetched ('next' field). Since this is the first time Lucio is taking a quiz using our system, all the fields except for ID and NEXT are blank. (The others only will be updated after we grade Lucio's answers, as we explain later.) Our CGI script then reads the NEXT field and fetches the HTML page indicated by it, called 'soccer.html', and records the current time in the time field in his record.

PHASE II: Taking the first quiz:

Lucio now sees 'soccer.html', a simple soccer quiz with several questions (Figure 1b). He spends 30 minutes taking this quiz, and then clicks on the submit button at the bottom of the page. The test is then submitted to another CGI script, that identifies the quiz, the session cookie, and retrieves the student's record in the 'student_log' database once again, along with an answer database file for that quiz. Another submission timestamp is recorded and the grading process begins. A special grader module inside the CGI script gets the right answers from the answer database file, and checks them against Lucio's answers. After adding the score for that exam, the same CGI script does lookup on a history database called 'quiz_log', to see if he has taken this quiz before (scores are not added to the final score if he has). If he hasn't taken the quiz, we then append the score, time, and quiz id into a list, and add the score for that quiz. After this grading phase is complete, we update Lucio's record once again, and check to see what the next quiz to be taken is, according to his grade. We then update that information in the database, and fetch the feedback page.

PHASE III: The Feedback Page:

Figure 3 shows a typical feedback page seen by user 'tinoco' after he submits the first soccer quiz. It contains a list of the questions missed, the final score on that quiz page, and the next quiz to be taken. We see that Lucio got all the answers right, and can therefore proceed to a next level (another quiz called "soccerharder.html"). At this point, all the table fields contain up-to-date information. It's up to Lucio to decide if he wants to proceed and take the next quiz or take a break. If he decides to proceed, another database lookup is made to retrieve and fetch the next quiz to be taken, and the whole process begins again. If Lucio decides to quit for the moment, we show a "thank you" page, and the system goes back to

its initial state. Later on, when another quiz is completed, the cumulative score will reflect the total of all quizzes completed.

The intensive use of server-side cookies plus database management routines for authenticating and updating user information leads to most of the code of systems like QUIZIT, where state information has to be securely kept throughout a session. Server-side cookies can be described as simple session identifiers that are passed by the server (via a CGI-script) to the client and vice-versa. In our case, the cookie is the unique user id generated by the authentication server for each session. Database management routines are necessary to store and retrieve student information like scores, history of quizzes taken, and so forth. We opted for MSQL because it provides a powerful API, database administration commands (including access control), and locking capabilities. Additionally, MSQL is freeware, very popular among CGI script developers, and used in many groupware and database applications (Rowe, 1996).

3.2 The SGML compiler

On the interface side, we developed a compiler that takes quiz markup and generates the answer files and HTML pages necessary for the run-time system. This compiler is completely independent, and can be run on any machine. Only the generated files have to be transferred from the machine where they were authored to the run-time server once the process is complete.

We chose to develop this compiler using Lex and Yacc (Levine, 1992). While it would be simple, and probably more efficient to write a compiler from scratch, we found that the use of these tools provides a nice way of constructing reusable compilers. If we decide later on to change our document type definition (DTD) or the target code (when HTML changes, for example), our task will be greatly simplified. Another option would be to use some validating SGML parser like “sgmls” to do part of the job for us. However, this would require the installation of yet another system on the author’s machine, which in turn increases the portability problems that we were originally trying to avoid.

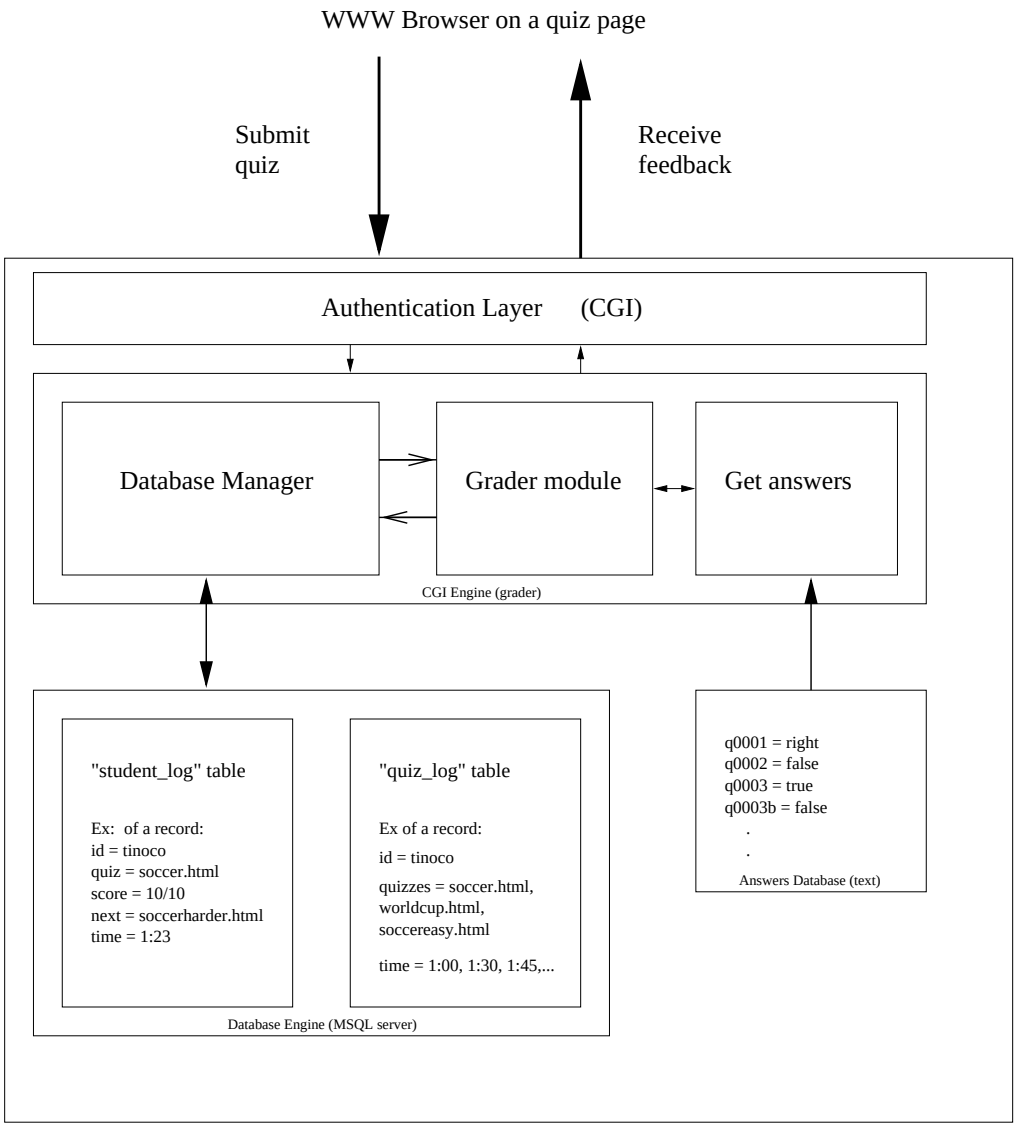

Figure 2: QUIZIT Run-time Engine

Summary of Results -- QUIZ: quizAA.html

User "tinoco"

You can proceed to the next level ...
You have taken this quiz before. Score remains unchanged.

Your total score for this quiz was

45

Your cummulative score is:

45

The next quiz to be taken is: quizAB.html

To quit and continue later, just leave the page, or kill your browser.

Figure 3: Feedback page

4. Conclusion and Future Developments

It is our primary goal to distribute the QUIZIT system to instructors in different departments on campus and in other universities. While we acknowledge that the use of multiple-choice-type exams as an assessment tool has many pitfalls (Shlechter, 1991), we think of our system as a complementary evaluation tool for instructors, specially in distance learning and self-paced education. Instructors should be aware that the suggestive name of the system itself also reflects our idea about its use, as it is envisioned as a 'quiz' tool for assessing student progress formatively only, and not summatively.

In the near future, we intend to conduct experiments with instructors and students before and during the application of QUIZIT in their classrooms. We hope to see if the use of QUIZIT reduces the grading overhead on instructors, and if the system is easily usable by students. A training plan is already in place to help instructors getting started with the authoring language (Tinoco, 1996), but further recommendations on how to apply QUIZIT in the classroom still have to be organized.

During the design of the authoring interface and the DTD, a participatory design proved to be very effective in identifying potential usability problems. It is our intent to conduct more formal usability studies in our usability labs at Virginia Tech to investigate other critical incidents in the authoring and run-time interfaces of QUIZIT.

In addition, when instructors start using QUIZIT in the future, we will be able to assess the benefits of using SGML to create large test banks, and their integration with other digital libraries. We expect great results from this structure-oriented syntax after we integrate collections of test-banks with SGML-aware retrieval systems.

5. References

Asymetrix Inc. (1996). Asymetrix's ToolBook CBT Edition. Asymetrix Inc.

Belnap, N. D. (1976). The Logic of Questions and Answers. New Haven: Yale University Press.

Bryan, M. (1988). SGML: An Author's Guide to the Standard Generalized Markup Language (5th ed.). Addison-Wesley Publishing Company., New York.

Coombs, J. H., Renear, A. H., & DeRose, S. J. (1987, November). Markup Systems and The Future of Scholarly Text Processing. Communications of the ACM, p. 933-947.

ETS Inc. (1995) Registration Bulletin/ GRE General Computer-Based Test Descriptive Booklet. Princeton, NY: Educational Testing Services Inc.

Hansen, D. N. (1967). An Investigation of Computer-Based Science Testing. In Atkinson, R. C & Wilson, H. A. (Eds.), Computer-Assisted Instruction: A Book of Readings. Academic Press Inc., New York, NY.

Hughes Inc. (1996). Mini SQL: A Lightweight Database Engine. Hughes Technologies Pty Ltd.

Lee, W. W., Mamone, R. A., & Roadman, K. H. (1995). The Computer Based Training Handbook: Assessment, Design, Development, Evaluation. Englewood Cliffs. N.J: Englewood

Levine, J. R., Mason, T., & Brown, D. (1992). Lex & Yacc (2nd ed.). Sebastopol, CA: O'Reilly & Associates, Inc.

Liu, C., Peek, J., Jones, R., Buus, B., & Nye, A. (1994). Managing Internet Information Services. Sebastopol, CA: O'Reilly & Associates, Inc.

Macromedia Inc.(1996). Authorware CBT. Macromedia.

Maler, E., & Andaloussi, J. E. (1996). Developing SGML DTDs: From Text to Model to Markup .Prentice-Hall.

McManus, T. (1995a). Delivering Instruction on the WWW. http://www.edb.utexas.edu/coe/depts/ci/it/projects/wbi/wbi.html.

McManus, T. F. (1995b). Special Considerations for Designing Internet Based Education. In D. Willis, B. Robin, & J. Willis (Eds.), Technology and Teacher Education Annual. Charlottesville, VA: University of Texas at Austin.

Milheim, W. D. (1994). Authoring-systems software for computer -based training. Englewood Cliffs, N.J. : Englewood

Nichols, G. (1996). Greg's Web Based Training Place. http://www.ucalgary.ca/~gwnichol/index.html.

Logic Resources Corp (1996). LXRTest. http://www.lxrtest.com/.

Rowe, J. (1996). Building Internet Database Servers with CGI . Indianapolis, IN: New Riders Publishing.

Shlechter, T. M. (Ed.). (1991). Problems and Promises of Computer-Based Training. Norwood, N.J.: Ablex Pub. Corp.

Tinoco, L. C. (1996). QUIZIT Tutorial. Blacksburg, VA: Virginia Tech Computer Science Dept. http://pixel.cs.vt.edu/~ltinoco/quizitdocs/

Wheeler, D. A. (1996). Mklesson user guide. http://lglwww.epfl.ch/Ada/Tutorials/Lovelace/userg.html.