An Infrastructure for Web-Based Computer-Assisted Learning

MIKE JOY, BORIS MUZYKANTSKII, SIMON RAWLES, and MICHAEL EVANS
University of Warwick

We describe an initiative under way at Warwick to provide a technical foundation for computer-aided learning and computer-assisted assessment tools, which allows a rich dialogue sensitive to individual students' response patterns. The system distinguishes between dialogues for individual problems and the linking of problems. This enables a subject specialist to craft individual questions and units of learning material, while allowing the order of presentation of the units to depend on a student's interaction history. We develop a markup language, specified as an XML DTD, to enable the storage of data for use in such systems.

Categories and Subject Descriptors: K.3.1 [**Computers and Education**]: Computer Uses in Education—*Computer-assisted instruction (CAI)*; H.3.2 [**Information Storage and Retrieval**]: Information Storage—*File organization*

General Terms: Human Factors, Languages

Additional Key Words and Phrases: XML, markup, DTD, CAL, CAA

1. INTRODUCTION

Computer-assisted assessment (CAA) and computer-aided learning (CAL) have become important components of many university degree courses, and bring benefits to institutions, to instructors, and to individual students [Miller 1990]. Many CAA/CAL tools have been developed, both commercial products and software created and maintained in universities.

Early CAL tools were designed for particular platforms such as Microsoft Windows, but more recent systems tend to be Web-based. Most of these tools appear similar, offering teaching staff the opportunity to author educational material, including questions of a variety of types (multiple-choice, text insertion, etc.). Students then access that material and answer the questions, and the students' answers are made available to teaching staff.

The tools vary in a number of key areas:

—the user interfaces range from very basic HTML to displays with sophisticated graphics and user-interaction facilities;

Authors' address: Mike Joy, Department of Computer Science, University of Warwick, Coventry, CV4 7AL, UK.

—the tools run on a variety of different platforms, and may or may not be Web-based;
—different technologies are used by the tools, and include, for example, Java Servlets, XML (Extensible Markup Language), and DHTML (Dynamic HTML).

These differences, however, are superficial, and hide a fundamental weakness common to all the tools we have encountered. The delivery of questions to students is a *static* activity. The presentation and ordering of dialogues does not, in general, depend on an individual student's previous answers. The tools make little use, if any, of data stored about an individual student's progress. In educational terms, their use is currently restricted to *knowledge* and *comprehension*, the lowest two levels of difficulty in Bloom's taxonomy [Bloom and Krathwohl 1996].

In this article, we discuss an initiative under way at Warwick to provide a technical foundation on which CAA tools may be written which allow a rich dialogue sensitive to individual students' response patterns. We use XML to enable the storage of data that can be used by CAA/CAL tools, which distinguishes between dialogues for individual problems and the linking together of problems. This enables a subject specialist to craft individual questions and units of learning material, while allowing the order of presentation of the units to depend on a student's interaction history.

2. CAA TOOLS TODAY

There is a wide variety of software CAA tools currently available, some of which are specialized within subject areas, but most can be used for any subject area.

Delivery technologies vary, though the highly pervasive World Wide Web is a popular choice. Some products use instead custom-implemented programs [Learning Technology and Research Group 2001; EQL International 2001; Question Mark 2001] or are based on interactive authoring tools such as Macromedia's Authorware [Macromedia 2001] (in the case of TRIADS (Tripartite Interactive Assessment Delivery System) [McKenzie 1998]).

The majority of tools are based on a simple question-and-answer dialogue, though a small number are concerned with entirely different areas of assessment, such as computer programming [Joy and Luck 1998; Learning Technology and Research Group 2001] or diagram-based assessment [Tsintsifas 2001; Clyde Virtual University 2001]. Multiple choice questions are almost universally provided. Other common question types are multiple-response and fill-in-the-blank questions.

Bloom's taxonomy [Bloom and Krathwohl 1996] characterizes problems by six levels (knowledge, comprehension, application, analysis, synthesis, and evaluation) which are in the order of the level of cognitive ability needed to answer. Though many of the tools are able to test the lower three levels of ability, the teacher often needs to be skilled in question setting and design in order to test the higher levels. Indeed, it may be the case that computer-based

tools are inherently inappropriate for these tasks. The inability of computers to assess tasks exercising higher-level, more general skills [Perkin 1999] means CAA software as it stands appears unable to process this type of answer beyond simple techniques such as pattern-matching of input.

The provision for groupwork is lacking, though some interesting systems have been developed based on peer assessment, for example OASYS (On-line Assessment SYStem) [Bhalerao and Ward 2001].

Adaptive testing can be defined as a method of assessment where the selection of later questions depends on the responses to earlier questions [Perkin 1999]. It can be applied in the assessment-as-learning setting as a powerful feedback reinforcement mechanism, for example by repeating questions from a student's weaker topics or dynamically adjusting difficulty to stretch all students. Adaptive testing is an under-represented concept in both CAA tools and the standards that specify CAA question formats. A small minority of tools [Kryterion 2001; Dalziel and Gazzard 1998] allow for conditional questioning, in which the outcome of simple conditional tests, based on ranges of scores previously attained, affects the selection of the next section. As discussed later, the IMS standards do not (at the time of writing) cover adaptive testing and the conditional selection mechanisms required for it.

The computer has not been significantly exploited as an enabler of new assessment methods, rather it has been used to implement traditional assessment. Systems that use the computer's interactive nature, such as in adaptive testing and other types of guided learning, peer review systems, and so on, are few.

Computer-assisted assessment is often applied out of a desire to counter the increasing time demands on staff while still delivering an acceptable educational experience. However, the most time-intensive stage of using CAA is when it is being introduced—long-term gains in efficiency are often at a cost of short-term effort [Harvey and Mogey 1999]. The lack of adoption of standards and the subsequent lack of portability between different CAA systems leads to major difficulties in the reuse of materials.

There has been a shift toward students taking responsibility for their own learning and the teacher providing the learning environment necessary for facilitating learning [Charman 1999]. Self-evaluation, now regarded as an important life skill that should be encouraged as part of the educational process [Sambell et al. 1999], can be facilitated by a number of features in software. The students' ability to view their progress in a module acts as both an indicator of weak areas and as a motivator. Some of the software, particularly VLEs (Virtual Learning Environments), give this kind of overview, though this is as far as the provision for self-evaluation goes. Similarly, the ability to undertake practice tests (or repeating tests until a given grade is attained), is rarely supported in CAA software. Student-centered learning is also facilitated by the openness of the assessment (for example, minimal secrecy of the assessment criteria [Perkin 1999]) and the subsequent visibility of the learning process [Sambell et al. 1999; Boud 1995]. Much CAA/CAL software succeeds in this area, even if the learning process that the software encourages is often not made clear to the user.

2.1 Markup Languages in CAA

A *markup language* is a collection of textual annotations, which may be included in a document, that influence the semantics of the document. For example, HTML is a markup language used to describe how text in a Web page should be displayed. The use of markup languages is fairly common in CAA software, and has been used to provide a structured and delivery-independent approach to a number of projects. In addition to the use of IMS standards [IMS Global Learning Consortium 2001b], the WebMCQ [Dalziel and Gazzard 1998] system is partly motivated by a wider agenda for the organization of teaching materials and their reuse based on a markup language. (The Bristol Tutorial Markup Language (TML) markup-based system supported by the Netquest tool [Institute for Learning and Research Technology 2001] has been established for several years.) There appears to be little provision for definition of structure and subsequent integration in most of the software, the exception being the Virtual Learning Environments (VLEs) IMS standards and standards associated with WebMCQ [Dalziel and Gazzard 1998] that detail a multilevel hierarchy of course materials.

An industry standard model [IMS Global Learning Consortium 2001b] was developed for the representation of question and test data, facilitating the use of the data between different products. The standard has been defined by the IMS Global Learning Consortium, comprising educational, commercial, and government organization members. This standard, the Question and Test Interoperability (QTI) specification, is defined in XML [WWW Consortium 2002; xml.org 2002].

The standard itself is a public draft specification at the time of writing at version 1.2. It allows the construction of questions from a set of standard question types and the collection and scoring of responses. The standard is relatively large, with over a hundred XML elements, covering mainly the following:

—specifications for question-and-answer data and the packaging of such data;
—information about processing responses for assessment (scoring, feedback, etc.) and reporting these results;
—other metadata;
—high-level presentational instructions (though not user-interface-specific specifications); and
—the definition of external APIs for products.

The area of test delivery is defined in its own ASI Selection and Ordering Specification [IMS Global Learning Consortium 2001a], covering the selection and ordering of questions. Small sets of questions may be defined and a random set of questions chosen from each bank. These can be in order of appearance in the question bank or mixed, and questions that will always be chosen can be defined. Questions may also be defined to require other questions (*overlap inclusion*) or to remove them from being delivered (*overlap exclusion*). In its present form, the standard is limited to selection of questions based on random selection rather than as a result of previous responses, though a number of scenarios in which this takes place have been identified for future study. Included

in these are assessment methods such as *computer-adaptive testing*, in which difficulty is adjusted based on the current estimate of the examinee's ability, or simulated cases in which, for example, a medical situation is described and questions take examinees down different situations that occur based on their responses. However, it still remains that the conditional selection of questions based on outcome response is not included. The ability to define preconditions and postconditions for sections and items is also left for further study and inclusion in future versions of the standards.

2.2 CAA Innovation at Warwick

Since 1997 two Web-based tools have been developed and used at Warwick. The *Azulis* [Joy and Luck 1997] software was developed at the Department of Computer Science, initially to support programming language teaching, and in the Physics Department a similar system, known as *ATAS*, was developed to help the delivery of modules in fluid mechanics.

The structure of the original Azulis and ATAS systems is not dissimilar from the majority of tools described above. Materials are organized by topic within a course. A course consists of a set of collections of related information and questions. For each topic, there are questions at several levels of difficulty, with ordered sequences of questions within a level. A student will typically progress through the levels of a topic by answering all questions on a level and then moving to the next, but it is possible to drop through to the next level at any point.

Both systems are implemented using servers, which communicate with students via a standard Web browser, storing data on students' responses in a relational database. Students may request the server to return a document outlining their recent performance, and a tutor may similarly request statistics either for a specific user or global information for a course.

The Azulis and ATAS systems were combined at an early stage, and the new system (known as *AtWeb*) was deployed and tested on different undergraduate courses attended by several hundred students in total. The response has been generally favorable, and we are confident that this framework is appropriate, easy to maintain, and is extensible in the future.

3. ATWEB

Static collections of simple questions, though valuable, are only the first stage in the design of an online CAL resource which will successfully provide the benefits identified by Miller [1990] and allow the inclusion of features not provided by other CAA/CAL tools. We wished to develop AtWeb into such a tool, where an individual student's dialogue could be sensitive to that student's response patterns, and in the following sections we discuss the technical foundations for AtWeb that support that functionality.

3.1 Problems and Roadmaps

A fundamental decision was to split AtWeb into two interrelated but conceptually separate parts.

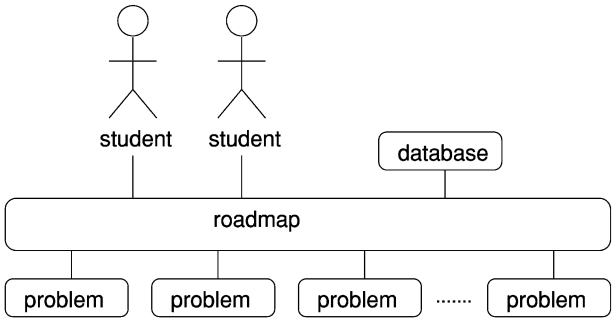

Fig. 1. Roadmap and problems.

First of all, we identified an entity, which we named a *problem*, a crafted dialogue that might be written by a "subject specialist" who would be familiar with the type of dialogue necessary to enhance the student's understanding of a very small and very specific topic. Second, we identified the need to link problems together, so that a package of learning or assessment material might be constructed using multiple problems. In this *roadmap* the content of the package and the order of presentation of the problems in the package would potentially depend on the students' interaction history. These are illustrated in Figure 1.

A *problem* is an individual question or set of questions that the student has to answer. It can be considered as a single task that a student must accomplish, and can be authored independently of other problems a student might be presented with. Although a problem may contain a sequence of interactions with a student, the interactions themselves are deterministic, and their logical interconnections are completely determined within the problem. Therefore the appearance of a problem to a student will be essentially identical, irrespective of that student's answers to other problems.

A *roadmap* is a logic that links individual *problems*, in order that each problem may be presented to a student in an order possibly determined by external data. For example, a record of a student's responses to problems may be stored in a database, and a roadmap might access that database in order to ascertain a suitable problem which that student would be presented with next.

We envisage a (large) bank of *problems* being available to AtWeb, together with a variety of external tools (databases, agents, etc.), and that a variety of roadmaps would be written for different courses. Thus each student's experience of the learning process would be individualized, and many of the problems would be shared by different roadmaps.

3.2 XML

An early decision was made that the markup for AtWeb should be coded as XML. We therefore designed a framework for representing the concepts and the logic of AtWeb that could be represented as an XML DTD (Document Type Definition). The system needed to support arbitrarily complex logic, for example a dialogue that asks progressively harder questions when a student is answering correctly, and gives useful feedback when errors are made.

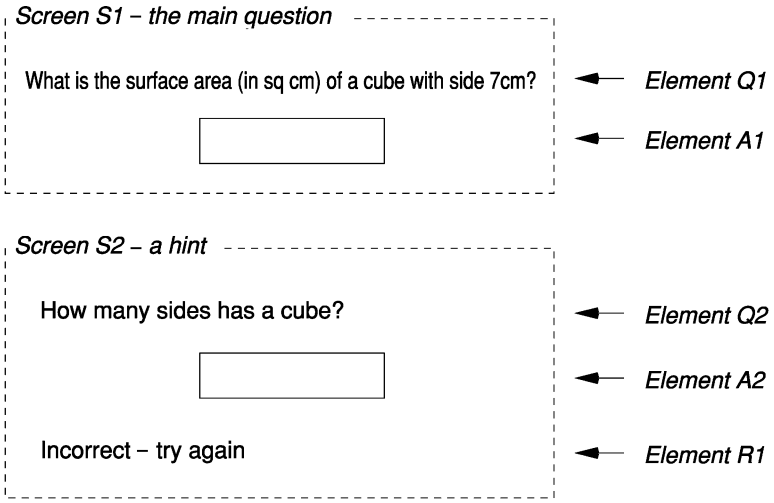

Fig. 2. Example problem.

In the following sections we include fragments of the XML DTDs, although knowledge of the syntax of XML DTDs is not essential in order to understand the sections. References to AtWeb concepts are rendered in *italics*, and references to XML elements are in `fixed-width font`.

4. THE PROBLEM SPECIFICATION

A *problem* consists of one or more *elements*, one or more *screens*, and a *logic*. The general idea is that a student will, at any given time while accessing the problem, see a document consisting of a collection of elements—this is referred to as a *screen*. The *logic* specifies the order in which the screens are presented to the student, dependent on the responses to previous questions (each of which is contained in a screen).

A problem also has two attributes, an identification name and a description. The XML defining a problem is

```
<!ELEMENT problem (elem+,screen+,logic)>
<!ATTLIST problem name ID #REQUIRED>
<!ATTLIST problem description CDATA #IMPLIED>
```

4.1 Example Problem

Figure 2 presents a simple problem where a hint may be given to the student. The example problem contains several elements and two screens.

Student is initially presented with screen S1. If the answer is incorrect, the student is repeatedly presented with screen S2 until he has answered the hint question correctly, and is then presented with screen S1 for a final time. The response R1 is visible on second and subsequent presentations of S2 only.

If the student answers correctly the first time, a mark of 100 is awarded, if he requires a hint and then answers correctly, a mark of 50 is given; otherwise

0. Before we detail the logic in Section 4.5, we define some terms to assist us with that definition.

4.2 Element

An *element* is the basic component of a problem. It can be thought of as an atomic interaction, or a display of text/images. During a dialogue with the system, a student is presented with a sequence of HTML documents, each one composed of several elements. If the element represents a dialogue with the student, a *value* may be returned, and each element is rendered as a fragment of valid HTML (possibly empty). An element may or may not be displayed, and if not is used to control the logic. For example, a *text gap element* has as value the string that is the text typed into the gap. A *Boolean element* has the value *true* or *false*. Both would typically be rendered with the HTML `INPUT` tag.

It is possible that within a given problem an element may be displayed multiple times, for example when a student is re-attempting a part of the problem he had previously answered incorrectly.

The name used in our DTDs is `elem` (rather than `element`) to avoid a name conflict in certain XML parsers. The XML for an *element* takes the following form:

```
<!ELEMENT elem (html|mchoice|textgap|boolean|setting)>
<!ATTLIST elem name ID #REQUIRED>
<!ATTLIST elem value CDATA #IMPLIED>
<!ATTLIST elem hide (yes|no) "no">
```

We now present the *element* types individually.

4.2.1 *HTML.* The *HTML* element is the simplest of the interaction elements, and requires a block of `PCDATA` (Parsable Character Data)—an HTML string. It is the only element type that describes an interaction which does not require student input and returns no value. The XML is trivial:

```
<!ELEMENT html (#PCDATA)>
```

4.2.2 *Multiple Choice.* A *multiple choice element* contains two or more *choices*, together with attributes that indicate whether the choices should be displayed randomly, and whether more than one choice may be selected. The "semirandom" display means that the choices are randomly displayed except the last one, which is always displayed last (for example, when an "other" choice is included). Each choice contains the actual option value itself, which in XML is stored as the child of the `choice` XML element, that is, between the `<choice>` and `</choice>` tags. It also has an attribute that is the name value needed later in the logic block when specifying the matching component. The chosen selection is the value returned.

```
<!ELEMENT mchoice (choice,choice+)>
<!ATTLIST mchoice multianswers (yes|no) "no">
<!ATTLIST mchoice order (default|random|semirandom) "default">
```

```
<!ELEMENT choice (#PCDATA)>
<!ATTLIST choice value ID #REQUIRED>
```

4.2.3 *Text Gap.* A *text gap* element allows a student to type in a text string. The `size` attribute specifies the number of characters available for the student to type in. A child of the `textgap` XML element is text, which will be displayed as the default when the element is initially displayed. The value returned is the test string entered.

```
<!ELEMENT textgap (#PCDATA)>
<!ATTLIST textgap size CDATA "40">
```

4.2.4 *Boolean.* A *Boolean* element allows a student to select between the values *true* and *false*.

```
<!ELEMENT truefalse EMPTY>
<!ATTLIST truefalse (true|false) "true">
```

4.2.5 *Setting.* The *setting* element is used simply to store a value, which can be set and interrogated by the program logic. For instance, suppose a given problem contains a hint, the student may or may not have accessed, this element can be used to store that information so that the problem's logic can respond to that knowledge. A `setting` element has no HTML rendering.

```
<!ELEMENT setting (#PCDATA)>
```

4.2.6 *Pseudoelements.* There are two pseudoelements, `current_screen` and `mark`, used in the problem logic that do not have XML representation in the problem.

Any problem has exactly one `current_screen` element and one or zero `mark` elements. These elements are used in the problem logic to set the screen that needs to be displayed next and to accumulate the total mark for the problem. They are never part of any screen. The value of the `current_screen` element is the id of the screen presented to the student, and the value of the `mark` element is the mark recorded for the student's attempt to terminate the problem.

4.3 Screen

A screen is an ordered collection of elements, each of which is either *visible* (and will be displayed to the student) or *hidden* (and will be processed, but not displayed). An example hidden element might be a hint that is not displayed the first time a student encountered it, but would be the second time the student attempted the same problem.

When a student views a problem, the first screen the student sees is the *initial* screen. It is expected that each problem normally contains a unique *initial screen*, although multiple initial screens are possible. The elements in a screen are displayed in the order they appear in the list. The XML for a screen is simple:

```
<!ELEMENT screen (elem+)>
<!ATTLIST screen name ID #REQUIRED>
```

Table I. Problem Elements

element	type	content
Q1	HTML	Main question
A1	textgap	student's answer to question Q1
Q2	HTML	Hint question
A2	textgap	student's answer to hint Q2
R1	HTML	Message displayed if student gets A2 wrong
hint_given	setting	flag set if the hint screen is accessed

Table II. Problem Screens

screen	elements	comment
S1	Q1, A1	initial screen with the question
S2	Q2, A2, R1, hint_given	hint screen

Table III. Snapshots

element	start value	during hint	end value
A1	(empty)	500	294
A2	(empty)	(empty)	6
hint_given	no	yes	yes
current_screen	S1	S2	S1
mark	(empty)	(empty)	50

The example previously displayed in Figure 2 is composed of two screens each containing several elements, and detailed in Tables I and II.

4.4 Snapshot

A *snapshot* is the set of values of all elements, including the `current_screen` and `mark` pseudoelements. Thus, at any moment during the dialogue with a student, the elements will each have a value, set either by the the problem logic or by the student—and the set of all those values uniquely specify the state of the problem at that time.

Taking the example in Figure 2 above, Table III details the values of the elements (snapshots) at the start of the dialogue, at the start of the dialogue with the hint screen, and at the end when the student has successfully entered the correct answer.

4.5 Logic

The *logic* specifies how a student is allowed to move between the screens, and allows the dynamic interactivity with the student to occur. It consists of a set of *initial snapshots*, a set of mappings between snapshots, and conditions for problem termination.

A logic is composed of two subcomponents, *matches* and *initial snapshots*. Further subcomponents *jumps*, *setvalues*, and *terminations* are available but not directly as children of the logic component.

An *initial snapshot* is a start configuration which a student may be presented with on entry to a problem. We expect that only a small number of such

initial snapshots exist for a given problem, and often only one. Multiple initial snapshots might exist if, for example, a problem were presented differently to defined categories of student, such as "novice" or "experienced."

A *match* is the equivalent conditional statement, and allows us to define the presentation of the problem to a student based on the student's interaction history *while attempting the current problem*. The XML for a logic is

```
<!ELEMENT logic (initial_snapshot|match)+>
```

4.5.1 *Element Modification.* The value of an element can be initialized and changed with `setvalue`:

```
<!ELEMENT setvalue EMPTY>
<!ATTLIST setvalue elem ID #REQUIRED>
<!ATTLIST setvalue attribute CDATA DEFAULT "value">
<!ATTLIST setvalue arg CDATA #IMPLIED>
```

The `elem` is the ID of an element; the `attribute` the name of an attribute for the element (or one of its children); and `arg` an argument passed to that attribute. For example, to hide element `a` we might have:

```
<setvalue elem="a" attribute="hide" arg="yes">
```

The elements referred to may be in any screen, or in no screen at all.

4.5.2 *Initial Snapshot.* An *initial snapshot* is a snapshot that describes how a problem should appear to the student upon entry to it. It contains a reference to the screen to be displayed, and optionally `setvalue` children to initialize the elements.

```
<!ELEMENT initial_snapshot (setvalue)*>
<!ATTLIST initial_snapshot screen_id ID #REQUIRED>
```

4.5.3 *Match.* A *match* returns a Boolean value as follows. A specific element value is checked to see whether the value of the response "matches" a given "answer." If the two are considered a match, the enclosing block of commands is processed; otherwise the block is skipped. The match checks short-circuits; that is, if a match is successful, all the remaining XML elements that are siblings of the match are not evaluated. For example, the following XML fragment:

```
<match elem="A2" arg="6">
  <jump>
    <setvalue elem="current_screen" arg="S1"/>
  </jump>
</match>
<jump>
  <setvalue elem="current_screen" arg="S2"/>
</jump>
```

is equivalent to the if-then-else statement *if (A2==6) then jump_to_screen(S1) else jump_to_screen(S2)*. The attributes are the `elem`, the ID of the element against which, the match is being made, and `attribute` is the name of the

element's attribute (which defaults to value if omitted). The value for that attribute is arg, against which the check is made and method describes *how* the check is to be performed. The method might be, for example, case-sensitive (for a case-sensitive textual match) or exists if the existence of the attribute is all that is required.

```
<!ELEMENT match (match|jump|terminate)+>
<!ATTLIST match elem ID #REQUIRED>
<!ATTLIST match attribute CDATA DEFAULT "value">
<!ATTLIST match method CDATA DEFAULT "equal">
<!ATTLIST match arg CDATA #IMPLIED>
```

4.5.4 *Jump.* A *jump* changes the snapshot by changing values of individual elements. It can only be used inside a match element and has no attributes.

```
<!ELEMENT jump (setvalue)*>
```

4.5.5 *Termination.* Specifies the end of the problem. The mark for the attempt is the value of the mark pseudoelement. We do require this tag; and do not rely on identifying the end of the possible matches, since this simplifies the structure of the XML DTD.

```
<!ELEMENT terminate (setvalue)*>
```

4.5.6 *Full Logic Example.* A single "if" statement is implemented as a tree of match instructions, each of which checks the value of an individual element.

```
<match elem="a" ...>
  <match elem="b" ...>
     <match elem="c" ...>
```

The above is equivalent to the condition that a snapshot has a particular state of element a, AND a particular state of element b and particular state of element c.

The "jump" part of conditional c is implemented as a <jump> XML tag which has children setting the values of individual elements:

```
<jump>
   <setvalue elem="current_screen" arg="screen_id"/>
   <setvalue elem="a" .../>
   <setvalue elem="d" .../>
</jump>
```

The above will result in the snapshot with **screen_id**. All elements except a and d will keep their values, while a and d will acquire the new values specified by the <setvalue> elements.

Putting these XML fragments together, we arrive at the following:

```
<match elem="a" arg="2">
  <match elem="b" arg="3">
    <match elem="c" arg="TRUE">
      <jump screen="screen_id">
```

```
        <setvalue elem="a" arg=""/>
        <setvalue elem="d" arg="FALSE"/>
      </jump>
    </match>
  </match>
</match>
```

This will change a snapshot with an arbitrary screen, and values `a=2`, `b=3`, `c=TRUE` into the snapshot with `screen=screen_id` and values `a=""` `b=3`, `c=TRUE`, `d=FALSE`.

4.6 Example

Here is the full logic for the example problem in Section 4.1. The first entry specifies that initially the problem is presented as screen S1.

```
<initial_snapshot>
  <setvalue elem="current_screen" arg="S1"/>
</initial_snapshot>
```

If we are on screen S1, the answer is correct and no hint has been given, we set the mark to 100 and finish the problem. If the answer is correct and the hint has been viewed, we set the mark to 50 and finish. If the answer is still wrong after the hint has been viewed, we set the mark to 0 and finish. If the answer is wrong, we present the hint and set the `hint_given` setting to `yes`.

```
<match elem="A1" arg="294">
  <match elem="hint_given" arg="no">
    <terminate>
      <setvalue elem="mark" method="set" arg="100"/>
    </terminate>
  </match>
  <terminate>
    <setvalue elem="mark" method="set" arg="50"/>
  </terminate>
</match>
<match elem="hint_given" arg="no">
  <jump>
    <setvalue elem="current_screen" arg="S2"/>
    <setvalue elem="hint_given" arg="yes"/>
  </jump>
</match>
<terminate>
  <setvalue elem="mark" method="set" arg="0"/>
</terminate>
```

Screen S2 is the hint. If the answer to the hint is correct, we show the initial screen again. If the answer is wrong, we display the “Incorrect” message R1 and continue until the correct answer has been given.

```
<match elem="current_screen" arg="S2">
  <match elem="A2" arg="6">
    <jump>
      <setvalue elem="current_screen" arg="S1"/>
      <setvalue elem="A1" arg=""/>
    </jump>
  </match>
  <setvalue elem="R1" attribute="hide" arg="no"/>
  <setvalue elem="A2" arg=""/>
</match>
```

The result of the above logic is that a student is given the mark 100 if the problem is solved without the hint; mark 50 if the hint screen was displayed; and mark 0 if the answer was totally wrong.

The complete description of the problem, including the logic section, is presented in the Appendix.

5. HTTP PROTOCOL INTERACTION

The AtWeb software we have been developing is Web-based, and it is appropriate to consider how the concepts we have coded as XML data interact with such software. In particular, we must specify how the HTML representations of the data are formed.

We define a *session* as a succession of HTTP requests/responses carried out by the same user from the same browser (possibly within a reasonable time limit). At any given time during the session there is a *current snapshot* associated with a problem. Let *s_pre* be the snapshot of the problem before the HTTP request, and *s_post* the state of the problem after the HTTP request is processed but before the problem logic is applied. The current snapshot is determined according to these rules:

—Initially the snapshot is one of the initial snapshots of the problem (depending on the road map).

—An HTTP request contains information that changes the values of some of the elements in the problem, and an HTTP request can only change the values of elements that have HTML representation in the current snapshot.

—The snapshot for *s_post* has then the same screen as the snapshot for *s_pre*, but the values of problem's elements are now different (as a result of processing the HTTP request).

—The *s_response* snapshot is created by the server from the *s_post* snapshot by applying the problem logic, which may change the screen and also the value of individual elements.

—The HTTP response is generated from *s_response* snapshot by taking the elements which are part of the screen of *s_response* snapshot and concatenating the HTML representations of the visible elements. The *s_response* snapshot then becomes *s_pre* state for the next HTTP request.

5.1 Implementation of Problem Logic

The problem logic is implemented as an ordered collection of if statements:

if *snapshot satisfies some conditions* **then jump** *new snapshot*.

The first instruction that satisfies the if condition is carried out, and all the rest are ignored. The argument of the *if* would be *s_post* and the *new snapshot* is the *s_response*.

6. THE ROADMAP SPECIFICATION

While working on a course, a student who has completed a problem would then be presented with another problem to work on. This might be a single problem, or a problem chosen from a set of problems either by the student or by AtWeb. In the latter case the choice would be made using criteria specified by the course author and using data such as, for example, scores awarded to the student in previous problems.

A *roadmap* in its simplest form defines unconditional links (*jumps*) between *problems*, and this is our starting point. If a jump between problems P1 and P2 exists, then a student who has completed P1 may then be presented with P2 to attempt.

An attribute, which we have named `prefix`, indicates a "storage location" for the problems referred to, which in practice means the name of a folder or a URL where the problem is located.

The XML is straightforward:

```
<!ELEMENT roadmap (jump*)>
<!ATTLIST roadmap prefix CDATA #REQUIRED>
```

6.1 Jumps

A *jump* (from one problem to the next) is specified by the problem the student has just finished and the problem he could jump to next. The argument `from` has special values `initial` and `solution`, used when this roadmap has not been encountered before or a deadline has passed.

```
<!ELEMENT jump EMPTY>
<!ATTLIST jump from CDATA #REQUIRED>
<!ATTLIST jump to CDATA #REQUIRED>
```

6.2 Example

The following roadmap, pictured in Figure 3, illustrates a pathway where the student is initially given the choice of three problems—easy, intermediate and advanced. If the student selects the easy option, he must then attempt the intermediate problem Once the intermediate one has been completed, the advanced one is presented.

```
<?xml version="1.0"?>
<!DOCTYPE roadmap SYSTEM "roadmap.dtd">
<roadmap prefix="/XX0000/java/">
<jump from="initial" to="easy_problem" />
```

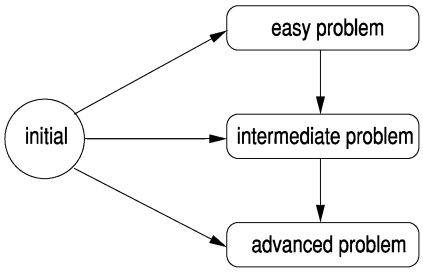

Fig. 3. Example roadmap.

```
<jump from="initial" to="intermediate_problem" />
<jump from="initial" to="advanced_problem" />
<jump from="easy_problem" to="intermediate_problem" />
<jump from="intermediate_problem" to="advanced_problem" />
</roadmap>
```

6.3 Roadmaps and External Data

At this stage, it does not seem appropriate to include further XML within the roadmap specification. The AtWeb software stores appropriate data upon entry and exit from each problem, and the software that controls navigation between problems according to the roadmap does not depend on any further data. Most courses that use AtWeb have a simple roadmap structure.

However, addition of further attributes to `jump` is not precluded, and is the subject of ongoing work.

7. COMPARISON WITH OTHER WORK

There are several notable differences between the AtWeb specification and that of IMS' QTI standard. Perhaps the most significant is in what the QTI standard defines as *dynamic selection and ordering*, the dependency of the next section or item of an assessment on the student's responses to previous sections or items. This idea is realized in the AtWeb system because the logic elements, rules that link state-to-state and section-to-section, allow a high degree of flexibility in computer response to student answers, compared to many other CAL/CAA systems.

The QTI standard includes a greater range of question types, mostly as a result of the number of different ways a multiple choice or multiple response question can be presented. QTI also includes comparatively complex question types such as order-the-items, connect-the-points, and the ability to form composite question types. Some more common question types are missing from AtWeb, notably multiple-response and short-answer questions.

Response processing (called *outcomes processing* in QTI documentation) is also defined as a separate outcomes processing specification [IMS Global Learning Consortium 2001c] and supports a number of basic scoring methods. As well as the number of right answers and the percentage of correct ones, weighted scoring and best-k-of-n are supported. Negative scores and other methods are

not supported. Scores are assigned to items, and these are used to assign scores to sections and assessments. Responses to questions cause variables to be set for each item, section, or assessment. Scoring algorithms combine with Boolean logic operators *or*, *and*, and *not* to construct complex conditions for response processing. AtWeb scores by assigning a mark by associating it with the exit point of a problem, though this score is typically a result of the logic employed in traversal from state-to-state as the student undertakes the problem.

8. CONCLUSION

We have described an initiative under way at Warwick to provide a technical foundation for computer-aided learning and computer-assisted assessment tools. The system distinguishes between dialogues for individual problems, and the linking together of problems, thus enabling CAL and CAA tools to support rich dialogues sensitive to individual students' response patterns by separating the presentation of individual problem elements and the management of their presentation. We have developed a markup language, specified as an XML DTD, to enable the storage of data for use in such systems.

We have developed a CAL tool (*AtWeb*) that uses the new framework. Learning materials written for previous CAL software at Warwick have been successfully migrated to AtWeb, and further materials are under development.

APPENDIX: A FULL XML MARKUP FOR THE EXAMPLE PROBLEM

Here is the full XML specification for the example problem discussed in Sections 4.1 and 4.6.

```
<screen name="S1">
  <elem name="Q1">
    <html> What is the surface area (in sq cm) of a
              cube with side 7cm?" </html>
  </elem>
  <elem name="A1">
    <textgap />
  </elem>
</screen>

<screen name="S2">
  <elem name="Q2">
    <html> How many sides has a cube?
    </html>
  </elem>
  <elem name="A2">
    <textgap />
  </elem>
  <elem name="R1" hide="yes">
    <html>Incorrect - try again</html>
  </elem>
  <elem name="hint_given" value="no">
```

```
      <setting />
    </elem>
  </screen>

  <logic>
    <initial_snapshot>
      <setvalue elem="current_screen" arg="S1"/>
    </initial_snapshot>

    <match elem="current_screen" arg="S1">
      <match elem="A1" arg="294">
        <match elem="hint_given" arg="no">
          <terminate>
            <setvalue elem="mark" method="set" arg="100"/>
          </terminate>
        </match>
        <terminate>
          <setvalue elem="mark" method="set" arg="50"/>
        </terminate>
      </match>
      <match elem="hint_given" arg="no">
        <jump>
          <setvalue elem="current_screen" arg="S2"/>
          <setvalue elem="hint_given" arg="yes"/>
        </jump>
      </match>
      <terminate>
        <setvalue elem="mark" method="set" arg="0"/>
      </terminate>
    </match>

    <match elem="current_screen" arg="S2">
      <match elem="A2" arg="6">
        <jump>
          <setvalue elem="current_screen" arg="S1"/>
          <setvalue elem="A1" arg=""/>
        </jump>
      </match>
      <setvalue elem="R1" attribute="hide" arg="no"/>
      <setvalue elem="A2" arg=""/>
    </match>
  </logic>
```

ACKNOWLEDGMENTS

The authors wish to acknowledge the contributions of David Long, Marc Beavan, and Le He Ting in the development of the AtWeb software, and of Ashley Ward for detailed constructive feedback.

REFERENCES

BHALERAO, A. AND WARD, A. 2001. Towards electronically assisted peer assessment: a case study. *Alt-J—Assoc. Learning Technol. J. 9*, 1, 26–37.

BLOOM, B. AND KRATHWOHL, D. 1956. *Taxonomy of Educational Objectives: The Classification of Educational Goals, Handbook I: Cognitive Domain*. Longman.

BOUD, D. 1995. Assessment and learning: Contradictory or complimentary? In *Assessment for Learning in Higher Education*, P. Knight, Ed. Kogan, Page, 35–48.

CHARMAN, D. 1999. Issues and impacts of using computer-based assessments (CAAs) for formative assessment. In *Computer Assisted Assessment in Higher Education*, S. Brown, P. Race, and J. Bull, Eds. Kogan, Page, 85–94.

CLYDE VIRTUAL UNIVERSITY. 2001. About CVU. cvu.strath.ac.uk/admin/cvudocs/.

DALZIEL, J. AND GAZZARD, S. 1998. Assisting student learning using web-based assessment: An overview of the webmcq system. www.webmcq.com/public/pdfs/ovrview.pdf.

EQL INTERNATIONAL. 2001. EQL International Ltd. www.eql.co.uk/.

HARVEY, J. AND MOGEY, N. 1999. Pragmatic issues when integrating technology into the assessment of students. In *Computer Assisted Assessment in Higher Education*, S. Brown, P. Race, and J. Bull, Eds. Kogan, Page, 7–20.

IMS GLOBAL LEARNING CONSORTIUM. 2001a. IMS question & test interoperability: Asi selection and ordering specification public draft specification version 1.2. www.imsproject.org/question.

IMS GLOBAL LEARNING CONSORTIUM. 2001b. IMS question & test interoperability: Public draft specification version 1.2. www.imsproject.org/question.

IMS GLOBAL LEARNING CONSORTIUM. 2001c. IMS question and test interoperability: Outcomes passing specification public draft specification version 1.2. www.imsproject.org/question.

INSTITUTE FOR LEARNING AND RESEARCH TECHNOLOGY. 2001. Netquest. www.ilrt.bris.ac.uk/netquest/.

JOY, M. AND LUCK, M. 1997. Computer-assisted learning using the web. In *Proceedings of the 5th Annual Conference on the Teaching of Computing*. Centre for Teaching Computing, Dublin City University, 105–108.

JOY, M. AND LUCK, M. 1998. The BOSS system for on-line submission and assessment. *Monitor: J. CTI Centre for Comput. 10*, 27–29.

KRYTERION, D. 2001. Webassessor. www.webassessor.com/webassessor.html.

LEARNING TECHNOLOGY AND RESEARCH GROUP. 2001. Coursemaster. www.cs.nott.ac.uk/CourseMaster/.

MACROMEDIA. 2001. Authorware. www.macromedia.com/software/authorware/.

MCKENZIE, D. 1998. TRIADS (Tripartite Interactive Assessment Delivery System) manual. www.derby.ac.uk/assess/manual/tmanual.html.

MILLER, R. 1990. *Major American Higher Education Challenges in the 1990s*. Jessica Kingsly Publishers.

PERKIN, M. 1999. Validating formative and summative assessment. In *Computer Assisted Assessment in Higher Education*, S. Brown, P. Race, and J. Bull, Eds. Kogan, Page, 55–62.

QUESTION MARK. 2001. Question Mark. www.questionmark.com/.

SAMBELL, K. ET AL. 1999. Student perceptions of the learning benefits of computer-assisted assessment: A case study in electronic engineering. In *Computer Assisted Assessment in Higher Education*, S. Brown, P. Race, and J. Bull, Eds. Kogan, Page, 179–192.

TSINTSIFAS, A. 2001. Diadalos. www.cs.nott.ac.uk/azt/daidalos/.

WWW CONSORTIUM. 2002. Extensible markup language (XML). www.w3.org/XML.

XML.ORG. 2002. The XML industry portal. www.xml.org.

Received September 2002; revised March 2003; accepted April 2003