INTELLIGENT TUTORING AND SUPERVISED PROBLEM SOLVING IN THE BROWSER

William Billingsley, Peter Robinson, Mark Ashdown
University of Cambridge
William Gates Building, 15 JJ Thomson Avenue, Cambridge CB3 0FD, UK
William.Billingsley@cl.cam.ac.uk

Chris Hanson
Massachusetts Institute of Technology
The Stata Center, Building 32, 32 Vassar Street, Cambridge MA 02139, USA
cph@zurich.csail.mit.edu

ABSTRACT

Web-based intelligent tutoring systems are becoming increasingly sophisticated. One approach to the resulting complexity is to separate the application into a front-end applet providing the user interface and a back-end server managing the tutorial material. This makes it easier to integrate more complex questions and simple short answer questions in a single script. However, it is difficult to produce a tutorial client that can use a simple HTML page and also support rich interaction and "supervised" problem solving with a server-side tutor. We describe a system we have built which has these properties, and discuss our experiences using web standard methods to implement it. We also describe various places where we have needed to deviate from established web standards out of technical necessity, and evaluate the system's usefulness as a tutor. This paper describes joint work with Hal Abelson and Gerald Sussman from the Massachusetts Institute of Technology.

KEYWORDS

Intelligent Tutoring, DOM, XML, Java.

1. INTRODUCTION

Students who are tutored individually or in small groups have been shown to perform better in tests than those who receive classroom teaching alone (Bloom 1984), but this teacher-intensive tutoring is often expensive to provide. Intelligent Tutoring Systems (ITSs) seek to mitigate this problem by providing automatic tutoring in a specialised subject area. Typically, an ITS gives the student a problem to solve and either analyses the student's final submitted answer or monitors the student's individual solution steps and actions as he or she makes them.

Recently, there has been much interest in deploying ITSs on the Web to give students remote access to them and so that the ITS server can collect data on student performance and progress. For ITSs where the student has to enter a textual answer and send it to the server for evaluation, this can simply be done with static HTML forms, as with SQL-Tutor Web (Mitrovic & Hausler 2000) which teaches students to write SQL database queries. Where the student must answer the question graphically, for example by drawing a graph, a Java applet is often used to help the student draw the answer, and then compose a data message to be sent to the server. SIETTE (Guzman et al 2002) and JellyFish (Scott & Stone 1998), which can both be used to ask short graphical questions in a variety of subjects, use these kinds of applet.

However this style of interaction, where the student creates an answer, submits it, and then receives feedback on it, has disadvantages for tutoring. A number of successful standalone ITSs are interested in the individual actions the student is making to create an answer, and provide immediate feedback on those actions. For example, in the Andes physics tutor (Conati et al 2002), students are asked to draw force vector diagrams, but Andes will mark vectors right or wrong as soon as they are drawn rather waiting for the whole diagram. One can equally imagine a tutor that would automatically correct a vector if it was nearly right. To do this kind of supervised problem solving on the Web, ITS writers have generally had to move away from HTML browsers, and implement dedicated client graphical user interfaces for their systems which then communicate with an application server over HTTP. Algebrain (Alpert et al 1999), which teaches basic algebra, and Belvedere (Suthers et al 2001), which teaches argument skills and allows students to work collaboratively with each other, are examples of this.

This move from browsers to dedicated GUIs is unfortunate because browser clients do offer a number of advantages:

1. They allow a question from a complex ITS to be integrated seamlessly into a set of questions from a simpler ITS.
2. The layout of the tutor and the components it contains can be scripted from the server at run-time.
3. Because the layout is scripted, it is easier to introduce new kinds of question and component into the client, and adapt the tutor for different subjects.
4. External on-line resources and materials can be dynamically integrated into the tutor.

The first point is especially important because it is very common to implement tutors for short answer and multiple-choice questions using HTML forms and a variety of server-side scripting. The server script passes the student answers into a small checking routine that is written in a specialised language for the subject matter. Alice Interactive Mathematics (Klai et al 2000) uses the Maple maths system for answer checking; MIT's 6.001 Tutor (Lozano-Perez 2000) for Scheme programming uses checking routines that are also written in Scheme. These less intelligent short answer tutors are very easy to write questions for. A question for a more complex ITS that still uses a browser client could be incorporated into a short answer test simply by sending the appropriate page. The student would not need to be aware of the change in tutoring system at the back end, and it would even be possible for the script of the short answer tutor to choose which questions from the more complex ITS would be asked and which ITS components would be delivered to the browser.

Similarly, it also becomes easy to incorporate a browser-based ITS into an on-line teaching portal.

2. THE INTELLIGENT BOOK TUTOR

We have developed an ITS which supports supervised problem solving and co-operative work, where the client is a collection of components that can be shown in the browser. Our tutor, the Intelligent Book tutor, is a homework system that can take first year undergraduate students through example problems and past exam questions in computer science and electronics topics. Its focus is on longer process questions, where the student is engaged in a chain of reasoning and an AI module on the server follows and debugs that reasoning. For this reason, it was important that the tutor should fit in well with server-side scripting tutors that could ask shorter questions more simply, but should still engage in a rich enough interaction to be able to debug the student's thinking at a low level.

The need to support more than one teaching topic means that the client has to support the introduction of new kinds of graphical component, to show different kinds of diagrams. Because the server is driven from a teaching script, it is also possible to interface to different AI systems on a per-question basis, or to use multiple AI systems for one question.

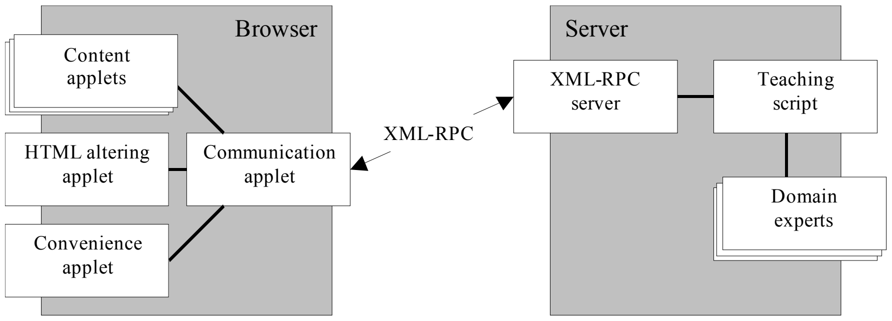
2.1 Architecture

An outline of the architecture is shown in Figure 1.

Communication between the client and the server is over XML-RPC. On the client, an invisible Java applet handles communication with the server. The Java applet security model does not allow an applet to listen for XML-RPC requests without maintaining a permanent connection to its server, which would be awkward in an ITS, introducing the need to deal with reconnects and timeouts. To work around this, whenever the client makes an XML-RPC call to the server, it expects the server's response data to be an array of XML-RPC calls that it wishes to make on the client in return.

Content applets are used by the student to construct an answer in a graphical way. For example, for electronics questions there is a single content applet: a circuit editor. A screenshot of this is shown Figure 2, together with a screenshot showing a content applet for discrete maths questions.

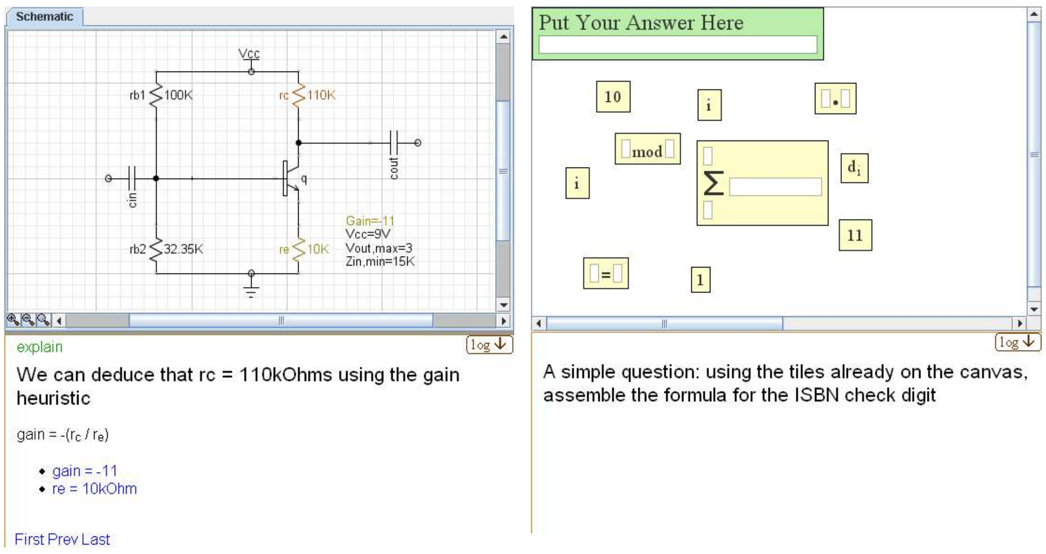
Figure 1. The high level architecture of the Intelligent Book tutor



The HTML applets control changes to sections of the HTML page. Because the content applets contain state, page reloads would cause them to need to re-fetch their state from the server. It is therefore more useful to alter the HTML on the page dynamically for the life of the question.

A convenience applet can also be included to reduce the number of calls the server will need to make on the client by automating common tasks.

Figure 2. A comparison of the interfaces for electronics and discrete mathematics questions



We can deduce that rc = 110kOhms using the gain heuristic

gain = -(r_c / r_e)

- gain = -11
- re = 10kOhm

First Prev Last

A simple question: using the tiles already on the canvas, assemble the formula for the ISBN check digit

On the server, the central point of contact is the web server, which receives the XML-RPC calls from the client and forwards them into a teaching script. This script is responsible for making the corresponding change in the server's model of the student's work and deciding how to respond. This mirrors the situation in a short answer tutor, where form requests from the browser are also passed into a teaching script, however in the XML-RPC case, the script responds by making calls to the client to show messages and make changes to the student's work, rather than by generating a new HTML page. Generally, the teaching script passes the students' work into a domain expert for analysis, and then replies to the student with details on any errors that were found. Simple heuristics for offering guidance are also encoded into the teaching script.

The server can reply to the student by making changes to the student's diagram and also by showing HTML on the page. This HTML can include JavaScript that can access the interface of the content applets, the convenience applets, and the communication applet. This allows the server to generate context-specific interaction, for example providing controls for the student to step through a long explanation from the server, to choose automatic changes to make to the diagram, or for the server to ask the student a short question.

2.2 An example question in electronics

In Figure 2, the electronics tutor shows a circuit analysis question being worked on in the browser. In this question, the student is given the circuit of a transistor amplifier and a set of specifications, and is asked to choose appropriate component values and to determine what particular voltages and currents in the circuit should be. Whenever the student sets a value in the editor, this action is sent to the server. The teaching script for this question sets this value in the domain expert's model of the circuit.

The domain expert used for this question is a constraint-based reasoning engine that can deduce values, propagate those deductions into other constraints, and provide structured explanations through backtracking (Stallman & Sussman 1977). The constraints come from the model of the circuit components. For example, the presence of a resistor in the circuit creates a constraint relating the voltage, current, and resistance in that component. If two of the variables are set, the third will be deduced, and that deduction may be used in other constraints to deduce further values.

The teaching script asks the reasoning engine whether any constraints have been broken (either because of a student setting or because of a chain of deductions from a student's setting). If they have, an HTML prompt is displayed on the client to indicate there has been a contradiction; if there has not, then the student's circuit is updated with any resultant values the reasoning engine has deduced.

The student can also ask the tutor to explain any contradictions or deduced values it has found. These explanations are animated as a series of steps. At each step, an HTML description of a deduction is shown, the relevant components are highlighted in the circuit, and links are displayed to navigate through the reasoning tree. Figure 2 shows the last step of a deduction.

In the language of ITSs, the electronics tutor is a Constraint Based tutor rather than a Model Tracing tutor because it generates feedback by looking at the current state of the student's work, and not by comparing the sequence of actions the student has performed to an ideal strategy (Mitrovic et al 2003). The advantage of this paradigm is that it is applicable regardless of what strategy the student uses to answer the question. The disadvantage is that it does not explicitly teach the student to use good strategies but only to keep out of incorrect states. Also, this encouragement to keep out of incorrect states does not match one way engineers often work: making a rough pass over the entire problem while deliberately ignoring minor errors, and then a more detailed pass to correct those errors. The script for the amplifier question works around this by encoding the two pass process into the question itself. The student is first asked to solve the problem with a naïve assumption about the transistor (that its beta is infinite) and without worrying about some of the specifications. Once this stage is completed, the other specifications are brought in and the student now finds he or she has some contradictions to fix. When those have all been corrected, the naïve assumption about the transistor is replaced by a more realistic assumption, and the student has some more errors to fix.

3. CONCLUSION

For the most part, the system uses Web standards unchanged, but there are two places where we found that these standards were not completely suitable. The first was that we had to make the communication applet poll the server for any calls it wishes to make on the client, as described in section 2.1. The second was finding a standard format for describing changes made by the student or server.

To maintain generality, the client content applets are considered to be a way of displaying and editing an arbitrary XML document. They expose a common interface so that the system can make changes to the document and highlight elements it wants to draw to the student's attention. Ideally, this interface would be a Web standard such as the Document Object Model (DOM). However, making a change using DOM is too call intensive: a number of calls are needed to traverse the DOM tree to the locations where the change is to be made, and a further set of calls are needed to make the changes. In an ITS, where the server is requesting changes to be made to a document on the client, sending each of those calls over XML-RPC would not be appropriate. XSLT, another popular way of changing XML documents, is quite large and complex, and tends to be used to transform a document into a new document, rather than to make the changes in-place. We eventually found it necessary to define our own single call per change interface for making alterations to an XML document, Simple Change Format for XML, SCFX (Billingsley & Billingsley 2004). This interface uses XPaths to identify the locations where the changes are to be made, and provides simple functions for additions, deletions, moves, and the setting of attribute and element values.

The architecture appears better suited to performing constraint based evaluation rather than model tracing evaluation. The calls the client makes to the server are generalised state-changing calls (setting values or adding structure) rather than calls that have a definite strategic meaning. This makes it easiest to use the current stat e as the primary way of analysing the answer.

We have also found it is important that there is approximately a one-to-one mapping between the elements of the data in the content applet and the representation of the data that is exposed by the domain expert. Otherwise, the server can find itself identifying problems in the student's work that it has no useful way of showing to the student. For example, in the circuit tutor, if the server wishes to explain a deduction to the student, then all the participants in that deduction must relate directly to parts of the diagram; they must be voltages, currents, and other circuit properties, rather than intermediate algebraic results. This means that if an pre-existing domain expert is to be integrated into the teaching script for a question, the content applet may need to be designed quite carefully to provide a close graphical representation of its data model.

Something that is lacking in the tutor is a student model. So far, the tutor has been solely focussed around working the student through predetermined example problems, rather than rating the student or choosing problems based on an estimation of the student's abilities. This will be addressed in future work.

ACKNOWLEDGEMENT

This paper describes joint work carried out with Hal Abelson and Gerald Sussman from the Massachusetts Institute of Technology.

REFERENCES

Alpert, R., et al, 1999. Deploying Intelligent Tutors on the Web: An Architecture and an Example. *International Journal of Artificial Intelligence in Education*, Vol. 10, pp183-197.

Billingsley, J. and Billingsley, W., 2004. The Animation of Simulations and Tutorial Clients for Online Teaching. *15th Annual Conference for the Australasian Association for Engineering Education*. Toowoomba, Australia. In press.

Bloom, B., 1984. The Two Sigma Problem: The Search for Methods of Group Instruction as Effective as One-to-One Tutoring. *Educational Researcher*, Vol. 13, Nos. 4-6.

Conati, C., et al, 2002. Using Bayesian Networks to Manage Uncertainty in Student Modeling. *User Modeling and User-Adapted Interaction*, Vol. 12, pp371-417.

Guzman, E., et al, 2002. A library for items construction in an adaptive evaluation system. *Evidence Centred Design (ECD) approach to creating diagnostic e-assessments*. San Sebastian, Spain, pp78-86.

Klai, S., et al, 2000. Using Maple and the web to grade mathematics tests. *Proceedings of the International Workshop on Advanced Learning Technologies*, Palmerston, New Zealand, pp89-92.

Lozano-Perez, T., 2000. Technologically Enhanced Education in Electrical Engineering and Computer Science. Retrieved 5 February, 2004, from http://www.swiss.csail.mit.edu/projects/icampus/eecs.html

Mitrovic, A. and Hausler, K., 2000. Porting SQL Tutor to the Web. *Proceedings of the International Workshop on Adaptive and Intelligent Web-Based Educational Systems*, Montreal, Canada, pp37-44.

Mitrovic, A., et al, 2003. A Comparative Analysis of Cognitive Tutoring and Constraint-Based Modelling. *Proceedings of the Ninth International Conference on User Modeling UM 2003*, Johnstown, USA, pp. 313-322.

Scott, N. and Stone, B., 1998. A Flexible Web-Based Tutorial System for Engineering, Maths and Science Subjects. *Global Journal of Engineering Education*, Vol. 2, No. 1, pp7-16.

Stallman, R and Sussman, G., 1977. Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis. *Artificial Intelligence*, Vol. 9, pp135-196.

Suthers, D., et al, 2001. Representational and Advisory Guidance for Students Learning Scientific Inquiry. In: Forbus, K., and Feltovich, P. *Smart machines in education: The coming revolution in educational technology*. Menlo Park, CA: AAAI/MIT Press, pp7-35.