High-Level Architecture for the Intelligent Book

Intelligent Book Project
MIT Project for Mathematics and Computation
Cambridge Computer Laboratory
Summer 2003

The goal of the Intelligent Book project is to construct web-based educational resources that understand principles and strategies pertinent to their subject matter, and use these to pose and answer questions, to fill in details, and to accumulate new examples as suggested by the conversations with its readers. The project's immediate objective is to develop programs to support class exercises in the analysis and design of electronic circuits for use at Cambridge University and at MIT.

This memo provides a high-level view of the architecture of the program and the main modules—details of the individual modules are given elsewhere. This architecture is still a draft, and we expect it to change as the project evolves.

1 Basic architectural framework

Figure 1 shows the overall framework of the intelligent book system. Students use browsers on client machines to connect to a central web server. Each client machine runs a web browser that can handle HTML and Javascript. In addition to the browser, there may be client-side Java applets that perform specialized functions. In general, the browser owns the screen real-estate, which it either handles itself (e.g., with HTML) or delegates ownership of specific screen areas to the applets.

The shaded rectangle in the figure represents the client-server interface. All communication across the interface with the browser is via HTTP. The Java applets may use specialized protocols, but all client-server protocols are built on HTTP and XML-RPC.[1]

The server-side application is written primarily in Scheme and talks to the web server (Apache) via `modlisp`, which is an Apache module designed for communicating with Lisp.[2] The Scheme application servicing the client uses a data base to save any necessary user state from transaction to transaction. This data base (PostgreSQL in our implementation) is also shared among all users of the system. As the number of clients expands, we expect to be running multiple Scheme processes on the server, and eventually, multiple servers.

Scheme pages and XDOC

The server-side Scheme application consists of several modules. There are specialized modules that handle circuit design (see section 2.1). There is also a general module called

[1] We may have to back off from the requirement to use HTTP, and allow some of the Java applet protocols to use persistent connections. Our current thinking is to try not to do that. We do, however, want always to use XML-RPC and avoid ad hoc or language-specific serialization schemes.

[2] Modlisp is an open-source implemention by Fractal Concept. See `www.fractalconcept.com`.

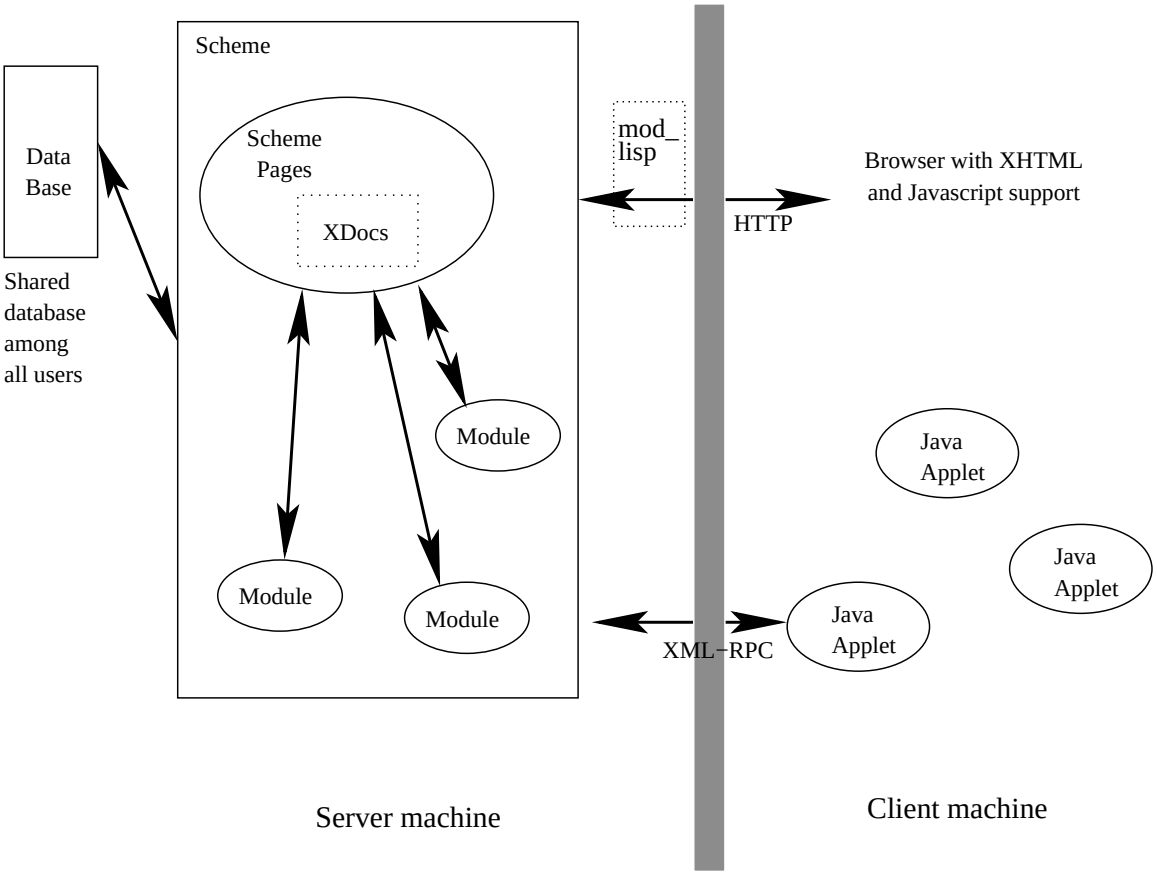

Figure 1: High-level architecture of the intelligent book system showing how functionality is partitioned between the server and the clients.

Scheme pages.[3]

Scheme pages is a module that processes descriptions of web pages, generating HTML to be presented by the web server (via modlisp). The language in which these pages are described is called *XDOC*. XDOC is being jointly developed with Chris Terman, who is using it for MIT's Computer Architecture class (6.004). XDOC will also be used to support the next version of MIT's Artificial Intelligence class (6.034), and eventually, MIT's introductory software course (6.001). The goal here is for all these courses to adopt a consistent page description language.[4]

XDOC is an XML-based language with several kinds of features:

[3]We need a better name for Scheme Pages. Hal suggests SSP (Scheme server pages), in analogy with ASP and JSP.

[4]The XDOC language is meant to be consistent across courses, but we expect that different courses will use different implementations of XDOC. Scheme pages is one implementation. In In 6.004 uses and XDOC implementation in Python.

- General-purpose layout and interface elements (such as radio buttons and menus), described by appropriate tags. In some cases these are just HTML tags, but others may be more elaborate. This tag set is extensible—tags are translated into HTML or Javascript, which is then sent to the browser.[5]

- Special-purpose interface elements for education (such as a multiple-choice or fill-in-the-blank questions).

- General methods for specifying how elements combine to form page layouts, and how layouts combine to form more complex layouts.

XDOC pages need not be static—the page definitions can include escapes into Scheme to provide dynamic generation of page elements.[6]

2 Architecture of the circuit system

The circuit design system follows the basic architecture described in section 1. Figure 2 shows the server-side Scheme modules for the system.

2.1 Server-side modeling components

The part of the system to the left of the dotted line is UI-independent. The main components are:

- **Scmutils:** Scmutils is an extensive algebraic manipulation system implemented in MIT/GNU Scheme. It extends Scheme's generic arithmetic operators to include symbolic algebra, and it also contains numerous packages and utilities for symbolic and numeric mathematical operations.

- The constraint propagation and truth maintenance system runs on top of Scmutils (although the actual dependence on Scmutils is currently very minimal). It integrates constraint propagation and truth maintenance though a mechanism that can attach condition nodes to each constraint. By manipulating these condition nodes, the TMS interface can cause certain constraints to be asserted or withdrawn.

  For example, in an electrical circuit application, there may be several different models of a transistor, each one with its own $v - i$ characteristic represented as a constraint that relates voltage and current. Asserting all of these constraints simultaneously would lead to contradictions. The TMS permits the different assumptions for the different models to be individually asserted and withdrawn under user control.

[5] Hal advocates developing a system like the one Microsoft adopted in ASP.NET, which interposes an abstraction layer between the ASP.NET inteface elements (called "controllers" in ASP-speak) and ordinary HTML.

[6] Even for dynamically generated pages, the intent is to design XDOC so that all page descriptions can be processed into HTML that is then sent to the browser.

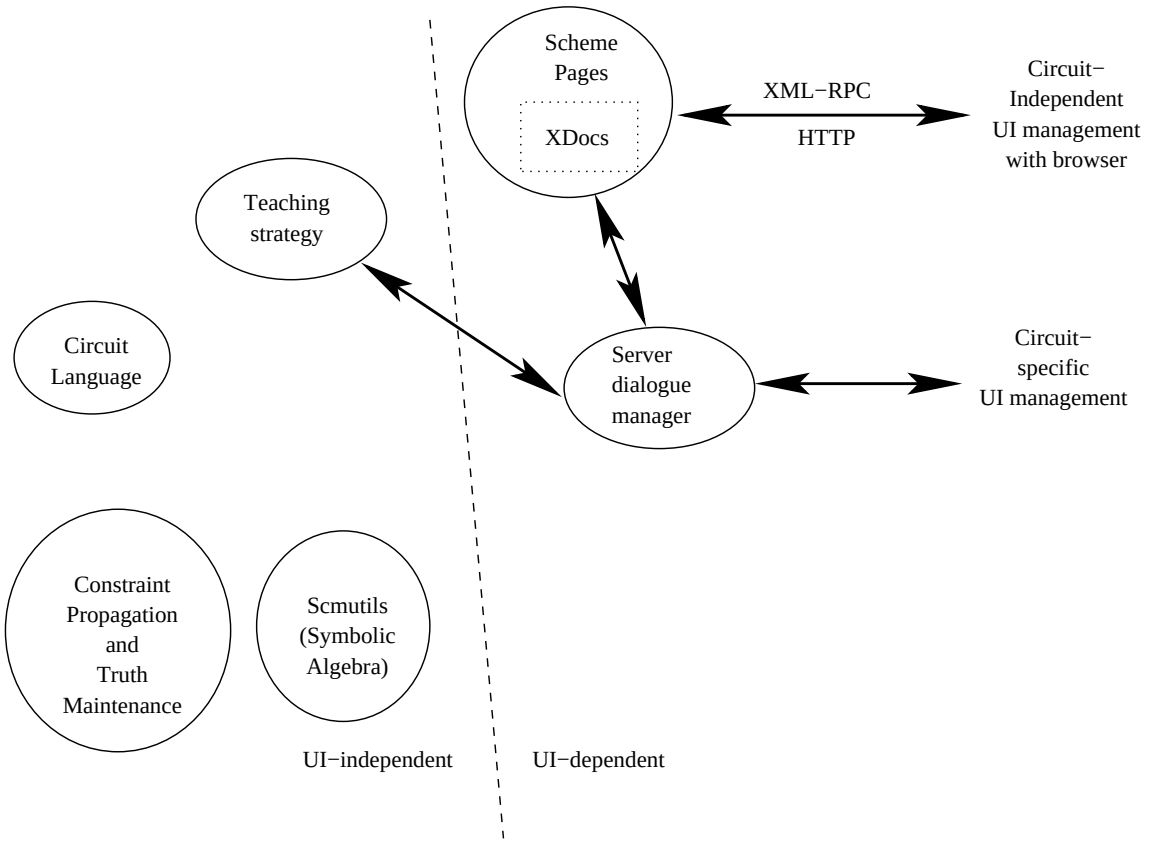

Figure 2: Major server-side modules for the circuit application. All of these are written in Scheme. Modules to the left of the dotted line deal with circuit modeling, reasoning, and teaching strategy, independent of user interface. Modules to the right deal with the UI.

- The circuit language takes circuit descriptions, expressed in terms of parts and their interconnections, and instantiates circuit models, represented as constraint networks with accompanying TMS nodes.[7]

- The teaching strategy module controls the interaction with students in circuit analysis problems. It is based on the circuit language and constraint propagator. The hope is that the module can generate useful interactions largely through general heuristics, and without a lot of circuit-specific knowledge. For example, if the student asks for a suggestion about what to do next in an analysis task, the program could suggest finding the value of a variable that (according to the constraint propagator) can be deduced in a small number of steps from values that the student already knows.[8]

[7]See `http://swissnet.ai.mit.edu/projects/intelligent-book/circuit-language.pdf` for documentation on the circuit language.

[8]See `http://swissnet.ai.mit.edu/projects/intelligent-book/teaching-strategy` for information on the teaching strategy module.

2.2 Server-side interface components

The dotted line in figure 2 separates the components that deal with modeling and reasoning from the components that handle the interface. The link between these two parts of the system is the communication between the teaching strategy module and the *server dialogue manager*.

The dialogue manager serves as the abstraction barrier between the interface-dependent and interface-independent parts of the system. For example, if the teaching module wants to ask the student to set the value of some parameter it would call the `ask-student-to-set-value` procedure, which is part of the dialogue manager. A rudimentary implementation of the manager could simply print a message on the screen:

```
Please set the value of a component
```

and wait for the student to type a reply.[9]

A more mature version of the manager would interpret the operations specified by the teacher module, breaking these down into two sorts of elements:

1. General interface elements that can be handled by the browser and handled via XDOC.
2. Circuit-specific elements that are handled by the client applets.

Any particular teaching strategy operation could be handled by elements of one type or the other, or by combinations of elements of both types. Including the server-side dialogue manager as an abstraction layer permits us to change these decisions without affecting the teaching module itself, as XDOC and the circuit-specific applets evolve.

2.3 Client-side components

Figure 3 shows the main components of the client architecture for the Intelligent Book circuit application.

The major component here is a Java applet called *Jade*. Jade is a circuit-diagram editor and circuit simulator written for use in MIT's computer architecture class (6.004), and is currently being modified both for the Intelligent Book and for new 6.004 applications. Jade was designed to be a stand-alone Java application. We are developing a *Jade API* that permits Jade to be run as a Java applet and exposes the basic interface operations for program manipulation.

The Jade-specific APIs are used by a library of *circuit-specific UI components* that is designed to be an extensible library of basic UI elements for use by the circuit program.

This library is used by a *client-side dialogue manager* to implement the operations that have been prescribed by the server-side dialogue manager. Neither the server-side nor the client-side dialogue manager has yet been designed, so we're not sure about how

[9] This rudimentary implementation, which does not use a client-server architecture at all, is what we are currently using to debug the modeling and reasoning portions of the Intelligent Book.

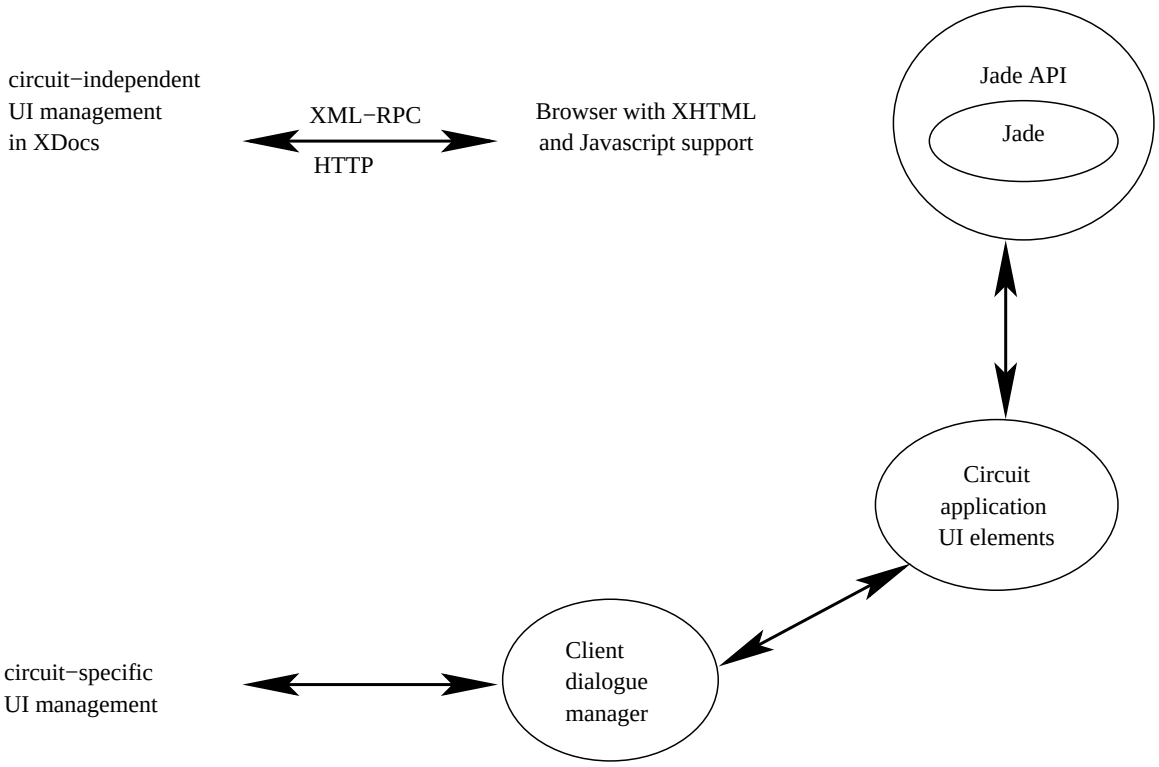

Figure 3: Client-side architecture for the circuit program.

functionality is partitioned between them. But in general, the server-side manager will deal with the abstract circuit, while the client-side manager will deal with the actual geometric circuit layout.

As an example of how an operation might proceed through these layers of abstraction, the teaching strategy module might issue an operation such as "suggest that the student provide values for the resistance of R_{C} or R_{E}. This would be interpreted by the server-side dialogue manager as "make the client display a message and highlight R_{C} and R_{E} on the diagram. This would in turn be realized by the client-side dialogue manager as "display a message and highlight the circuit elements at (x, y) position $(30, 67)$ and $(30, 20)$. This would be implemented in terms of calls to the UI application library, some of which would use the Jade API.

As should be obvious, the details in the above paragraph have not yet been fleshed out. What we do know is that (1) the dialogue managers must maintain a map between the abstract circuit and the geometric diagram; and (2) there will be some sort of XML-RPC protocol between server-side and client-side managers.

3 Examples

Figures 4 and 5 are preliminary sketches of the kinds of interactions we expect the system to deal with.

Figure 4 shows the student setting a particular current to 10 milliamperes, presumably in response to a question posed by the teaching module. In this interaction model, there are a couple of different screen areas. One with the circuit diagram, is delegated to Jade. Another with messages, and a third with command buttons like "Help". In our architecture, the circuit area would be controlled by Jade, while the others could be either controlled directly by the browser or by special circuit applets. We have not yet decided which model to pursue.[10]

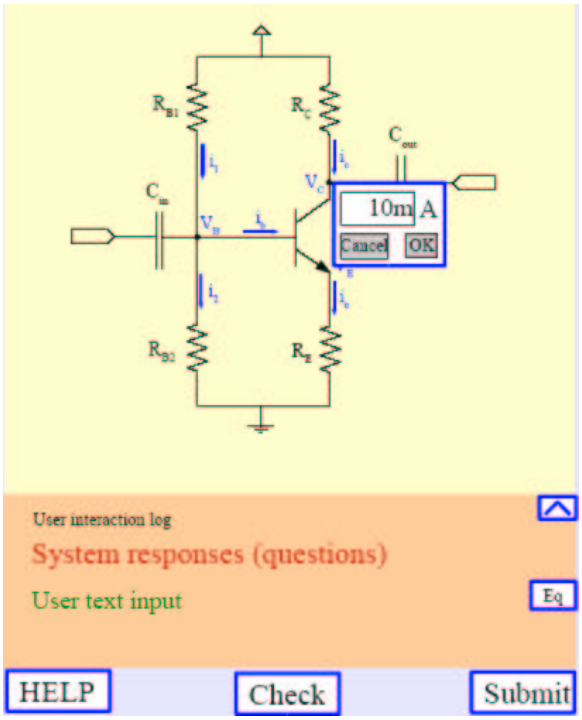

Figure 4: Example of an interaction with the intelligent book, showing the basic screen layout.

In figure 5, the system has responded to the student's value by pointing out that there is a contradiction, and asking the student which design criterion has been violated. The selection method could be handled either through browser forms and radio buttons or

[10] An advantage of delegating the latter two screen areas to applets is that it would presumably allow a more flexible mix of circuit-specific operations into the dialogue interaction without having to make a round trip to the server. On the other hand, much of the functionality we would build here would probably duplicate capabilities that are already in the browser, and isolate these interaction elements from standard means of control, such as by style sheets.

through a special circuit applet. In this case, using the browser seems more straightforward, but we have not yet made this decision.

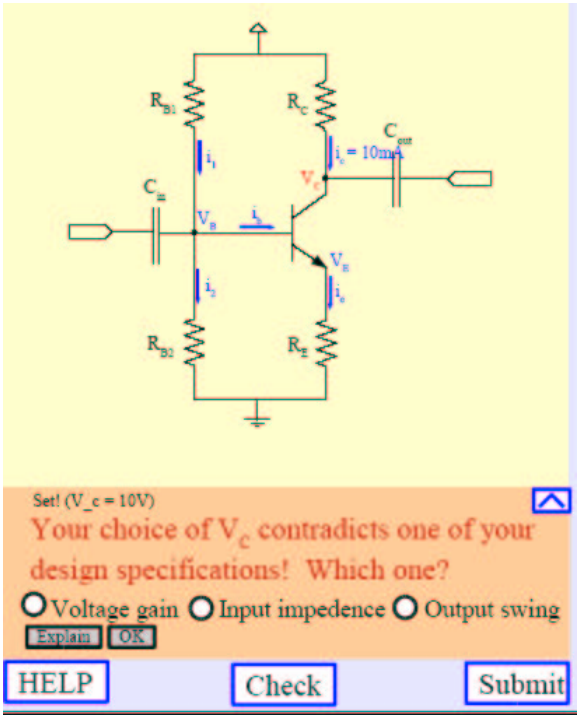

Figure 5: Intelligent book interaction: The constraint system has detected that the student's chosen value leads to a contradiction, and the teaching strategy model asks the student to say which design specification is violated.