



SCIENCE AND ENGINEERING DATABASES IN AN OPEN-SOURCE SOFTWARE WORLD

By Norman Chonacky and Dante Choi

STORING AND ORGANIZING SCIENTIFIC AND ENGINEERING INFORMATION IS A NATURAL APPLICATION FOR DATABASES.

BUT DATABASE DESIGNS AND SYSTEMS HAVE CONVENTIONALLY BEEN DRIVEN BY THE BUSINESS-APPLICATION MARKET, ESPECIALLY

by large businesses. The character of database software has tended to follow business drivers—inventory control, personnel services, and accounting—which are readily definable in advance and mutually understood by system users. Scientists and engineers in research environments have been left to develop, or possibly to adapt, business systems for their own uses, such as compiling individual researchers' measurement data sets prior to analysis and archiving critically analyzed data.

But the Internet is changing the way business operates. As e-businesses' transactions increasingly consist of searching for products and receiving current prices for those products via the Web, modern databases are increasingly wedded to Web applications that can perform these functions. Significantly, the aggregate market for database software—especially that which can be married to a Web front end—has driven down the costs, and arguably increased the variety of database systems. For example, because the demands of competitive marketing for Web-shoppers' attention require that Web pages be "entertaining," first animation and now dynamic page creation have become drivers for Web-served, database systems. Without

database-driven content on a dynamic Web site, Amazon.com could not display personalized product recommendations when a visitor logs in, nor could Washingtonpost.com let readers create custom homepages featuring news of personal interest. The commercial sector has explored many uses for Web applications that underscore the value of delivering customized, pertinent information tailored to specific, repeat end users.

It seems natural that such functionality and applications might be useful in collaborative research. In multidisciplinary scientific research projects, for example, they could be used to present individuals' contributions to other engineers, scientists, and government program managers who might find different parts and representations of that information understandable and relevant. Variations and mismatches in disciplines, viewpoints, and responsibilities are the issues that plague information management in cross-disciplinary research collaborations. Spurred by the availability of Web-based technologies, serious-minded software system designers and engineers in research labs are now working to address these issues, many using and contributing to open-

source software. This is how we were first drawn to working in open source, as well.

Dynamic Approach to Customized Presentations

One of us (Norman Chonacky) works in the field of environmental sciences, which (like many of its sister engineering fields) frequently brings personnel from multiple disciplines together in project collaborations, thus providing a stalking ground for cross-disciplinary issues. As a case in point, the recent workshop on Earth Systems Questions in Experimental Climate Change Science included participants representing a formidable combination of disciplines, viewpoints, and interests.

In creating a Web site to display its proceedings—presentations, notes from break-out sessions, audios of group discussions, videos of plenary talks, and so on—to the participants, we were confronted with the task of accommodating the conflicting priorities and perspectives the participants manifest at the workshop itself. We wanted to arrange links to the files according to the participants' differing priorities, and to embed those links in explanatory narratives tailored to different categories of participants.

An answer to the challenge of organizing the presentations lay in making the Web site dynamic. Though it might be instructive to view it in conjunction with this review, unfortunately the site is not currently publicly available. I must point out, however, that those who managed the workshop

were scientists and engineers, not media moguls or professional software engineers. We faced several other constraints, as well.

Being academics, for example, we did not have an extensive infrastructure to marshal to solve our site-design problems. Moreover, the challenge of communicating across disciplines is an ongoing, unsolved information-science research problem: we couldn't have specified what kind of design we wanted, even if we'd had database-support personnel. Finally, we needed to provide Web access to objects that were in many media formats that went beyond the relatively narrow group the proprietary database software we chose could accommodate. We therefore found that we had to make this system ourselves.

Enter an open-source software solution to building dynamic Web sites on top of databases. In our case, we planned to use the database to hold document-object URLs and metadata describing the documents to which they linked, rather than the documents themselves. Nonetheless, the design principles and problems we encountered could apply equally well to databases that contain the content itself. Because this is a technology review, we will not discuss the database's structural details or Web page designs. Instead, we will focus on system implementation options available within the open-source software environment.

Basic Site Components

In the balance of this article, we assume that you, as we did, need to learn about the relationships among the pieces necessary for building such a system, and how to choose them. We examine the nomenclature and functions of, as well as selection criteria for, these pieces, beginning with the

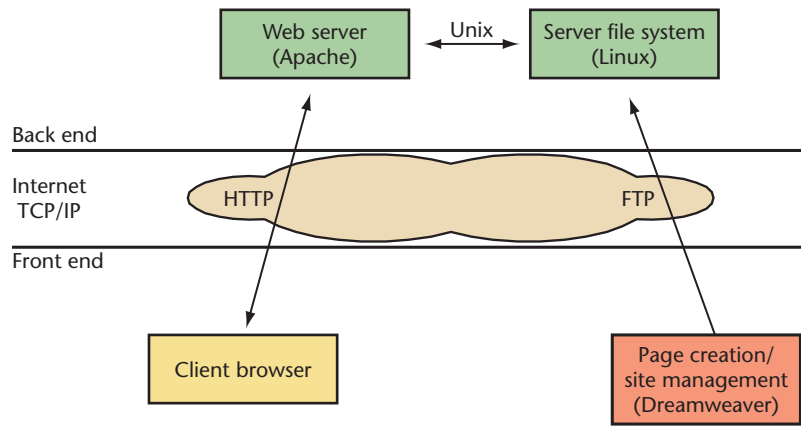


Figure 1. Client-server system schematic for a “static” Web site. Open-source products are depicted in green, freeware or shareware products are yellow, and the proprietary HTML editor is shown in orange.

basic Web server.

First, “server” is invariably used to refer both to the hardware (strictly speaking, the *host* machine) that executes a given service function, and to the software application whose execution performs that service. There often is confusion over this distinction in discussions between information technology experts and nonexperts. On the other hand, there is somewhat less confusion in making the client-server distinction. In the current context, the “server” is the application on the host machine, and the “client” is the corresponding application on the user’s machine that communicates with the server application on the host machine. For example, Apache is a Web page server application that operates on Unix host servers. The Apache server provides Web pages to Netscape client browsers. In our system, we used Apache on a Macintosh OS X server host.

In its simplest (static) configuration, the Web server application is a continuously executing, resident background process. It is a *receiver* of HTTP requests from client browsers, directed to the server application through the host’s operating system communications port #80, listening to the Internet. It is a *processor* for arguments accompanying such requests. A

normal argument for an HTTP request contains a *pathname*—relative to a system-designated home directory—to an HTML file in the host’s file system. The Web server process responds to the request by sending this file to the client browser on the user’s machine that generated it, for display on the user’s machine. Figure 1 schematizes this system, and includes another component that we have not mentioned yet: an HTML editor for creating pages and managing the files containing such pages on the host.

The HTML editor is optional in the sense that we could create Web pages of HTML code with a simple text editor, and we could manage these files by using some self-discipline to keep track of where they are in the host’s file system. However, such editors greatly simplify the process of creating Web pages of the aesthetic quality that everyone now expects, and their associated file-management features promote site integrity. Features such as “code by example” and site-management tools that discourage, if not eliminate, broken links make buying a product like this practically a no-brainer. For our workshop site, we selected a proprietary product, Macromedia’s Dreamweaver, which appears shaded in orange in Figure 1.

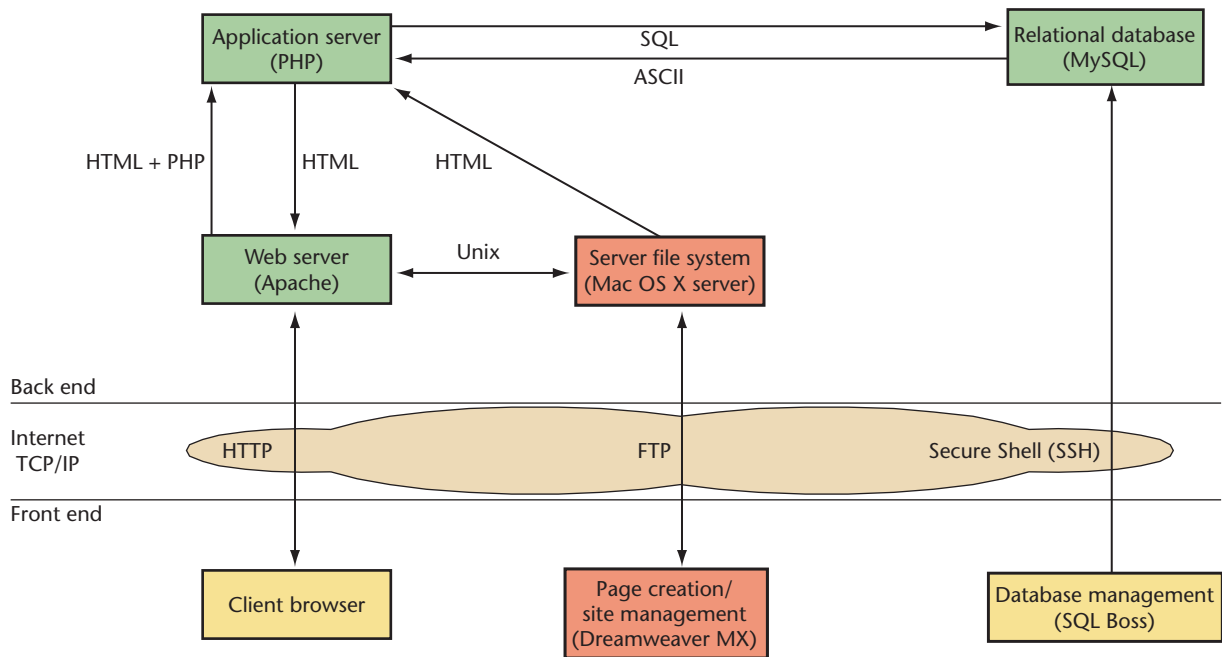


Figure 2. Dynamic client-server system. We added upper-layer components to the back end of the “static” system to achieve a dynamic Web site.

Implementing a Dynamic Back End

To allow the server to create an HTML page after receiving an HTTP request, we added more components. Figure 2 illustrates the system we used to dynamically create pages. At the outset, we only vaguely knew what goes on in the “back end” of such a system, which made it the most difficult piece to implement. Having done some investigating, we describe both the system and our component-selection process in some detail here.

Adding an *application server* to the system created another “layer” in the back end. This server’s role is to receive the Web page (template) fetched by the Web server, and then extract and execute the (PHP) commands embedded in the HTML page that the client has requested. These programmed actions can then modify the Web page—filling in a template with data from a database, for example—before the Web server sends that page to the requesting client.

We selected the PHP application server over other open-source alterna-

tives, in part because it was the one most frequently paired with the database we chose. This seemed like a good recommendation, given our relative naïveté. Another reason for the choice was that our system administrator was most familiar and comfortable with this application server.

In this approach, the host performs programmed actions triggered by the client’s request of a specially prepared “virtual” Web page that contains code to create an actual page at the moment of the request, but before the page is passed to the client. This distinctive capability characterizes a dynamic Web site. In the PHP and related embedded-code approaches, the requested HTML page is actually a template that contains layout information and symbolic links to objects that will populate the page.

The layout parameters define the page’s architecture, and the symbolic links are associated with locations within that architecture; they define the processes that determine which actual objects will be retrieved to appear on the page. These processes might be

contingent upon, and therefore assume values particular to, each instantiation of a page request.

These contingencies can be enforced by such parameters as session-state variables, such as a user’s previous requests from earlier in the session. Other contingencies can be enforced by user input from an *interface* page, which might include a form with buttons and text boxes. Developers can thus achieve customization and interactivity using these approaches. In our application, for example, we used this capability to fetch URLs stored in our conference database to create hyperlinks to the locations of those documents—in effect, flexibly providing “indirect addresses” for them.

The Database

As indicated earlier, coupling dynamic page creation to a database is only one, albeit important, function for an application server. The final piece of our story—the original driver for developing our system—is this database application. A dynamic Web site is a prerequisite for database applications intended

to manage content and present it in a customized manner. The broader story, which must wait until another time, is that of interactivity that goes beyond the client to make programmed use of any resource on the host.

We first explored several Web-accessible proprietary databases, but the limited choice of standard templates any of them supports was disappointing. As an alternative to proprietary databases, we investigated MySQL, an extremely popular, free (to non-enterprise users), open-source database application. Its flexibility and cost were major considerations in our decision. A third factor we can only describe as a “philosophical bias” toward participating in a community that values openness. In the open-source context, this implies giving back from whence you have received.

MySQL is not a client-server product; users must rely on Telnet or Secure Shell (SSH) terminal connections and use SQL commands to access, create, and manipulate a MySQL database residing on a remote host. However, we discovered that several shareware programs provide a GUI environment to get you working with the database quickly. For this role, we chose the SQL Batch Object Submission System (www.sqlboss.com). Figure 2 illustrates the functional relationship among these components.

We should mention that this shift to dynamism in our Web site was not without other associated costs. Now, more than ever, we needed site-development and management tools. Fortunately for us, Macromedia was rolling out Dreamweaver MX at the time, so we did not have to learn a completely new editing and management system. MX provides a visual development environment for coding and design.

It assigns certain functions to certain

pages. For example, view pages are skeletons that contain traditional HTML tags to format the page, as well as code that adds functionality. This includes making a connection to the database, storing session variables, and sending `recordset` queries, which are requests for a subset of information drawn from the database based on criteria the programmer (and sometimes user) determines. MX also requires programmers to define a `recordset` on a page that will contain variable data. Once this is done, we can place dynamic variables, representing the data to be retrieved, anywhere on the page to serve as placeholders for variable data.

MX sped up the development process by supplying the PHP code associated with the SQL and `recordset` queries. It did not provide code for more advanced actions such as user authentication and session management for PHP, although we understand that these are in place for other scripting languages (Javascript, for example). We had to refer to outside resources for the code for these methods. In addition to being a powerful scripting and database communication tool, Dreamweaver is an excellent editor for designing Web page layouts and managing file synchronization with a remoter server.

This has been a somewhat cursory introduction to a hybrid open-source/proprietary approach to creating a dynamic Web site for modest laboratory or academic use. Although we cannot claim our solution was optimal, we are convinced that this approach—attending to the back end, as well as the front—assumes a very worthwhile posture for scientific and engineering database applications. On balance, we have not regretted our choices, and we value

the investment required to negotiate in the open-source software environment and join that community.

One of the downsides of installing the open-source software on your server is that it can require considerable learning and troubleshooting for first-timers. In our first attempt, we had the advantage of a very Linux-savvy system administrator, but the process can be somewhat daunting in an “unaccompanied” open-source software environment.

Many proprietary server systems, such as the Apple Xserver (which our research group recently acquired), now come with MySQL preinstalled. This can be a considerable advantage if you don’t want to have to learn too much about system administration. Such hybrid systems can be well suited for the modest-size research and development groups (like ours) that we expect are interested in this article.

Norman Chonacky is a senior research scientist at Columbia University. His research interests include cognitive processes in research and education, environmental sensors and the management of data derived from sensor arrays, the physico-chemical behavior of material in environmental systems, and applied optics. He received a PhD in physics from the University of Wisconsin, Madison. He is a member of the American Association of Physics Teachers (AAPT), American Physical Society (APS), and American Association for the Advancement of Science (AAAS). Contact him at chonacky@columbia.edu.

Dante Choi is a third-year undergraduate at Columbia University, studying cognitive psychology. Prior to becoming a student, she spent five years as a graphic designer developing Web content. Her career interests include user-interface and interaction design. Contact her at dc2043@columbia.edu.