8-16-1996

Developing Groupware for the Web

Alan R. Dennis
Department of Management, Terry College of Business, University of Georgia, adennis@uga.cc.uga.edu

Sridar K. Pootheri
Department of Mathematics, University of Georgia, sridar@math.uga.edu

Developing Groupware for the Web

Alan R. Dennis
Department of Management, Terry College of Business
University of Georgia, Athens GA 30602
adennis@uga.cc.uga.edu
Sridar K. Pootheri
Department of Mathematics
University of Georgia, Athens GA 30602
sridar@math.uga.edu

Introduction

Over the past two years, the World Wide Web has enjoyed explosive growth and has become a major force in network computing. We believe the Web (both on the Internet and on internal corporate intranets based on Internet protocols) will become the preferred development environment for future network applications, more so than Windows, Macintosh or UNIX. The web will become the universal client for client-server applications.

Today, the Web is primarily one directional -- a way to publish documents, advertising information, catalogues, and so on. The future of the Web will be different. We see the Web as a highly interactive communication super highway -- a natural groupware highway, where individuals can work together to generate ideas, discuss problems, and make decisions whether they are in the same room, or halfway around the world (Fellers, Clifton, and Handley, 1995; Quek, and Tarr, 1996).

In this paper, we describe the development of a first generation web-based groupware system that enables any Web user to have immediate access to groupware (http://tcbworks.mgmt.uga.edu:8080). We expect that by the turn of the century, most organizations will establish a host of Web servers with groupware systems like this to support their many project teams and work groups. Groupware will become another "standard" office system, much like word processing, spreadsheets and e-mail that will simply be taken for granted. We begin by discussing the general design strategy for the system, the browser environment, the server environment, and the lessons learned.

Design Strategy

Most current groupware tools take either a data-centered or process-centered view in their design. Lotus Notes, for example, is data-centered; it provides an excellent repository for storing data in a highly structured database format. There are many different types of data that can be defined and many ways to view them. However, Notes is extremely limited in the types of group processes it supports; it is difficult to analyze, prioritize, and vote on the information without writing cumbersome external procedures.

Other groupware tools such as GroupSystems or VisionQuest are process-centered (e.g., see Jessup and Valacich, 1993; Nunamaker, et al., 1991). They provide many different tools that support a variety of different group processes, such as idea generation, information organizing, and voting. However, integrating data among different processes is also difficult because each process is supported by a separate tool that treats its data separately from the data in other tools (e.g., one tool for generating ideas, another for voting). This approach has two major drawbacks. First, data must constantly be moved from one tool to another (and some tools cannot even exchange data directly). Second, to accomplish most tasks, the user must master a set of different tools each with slightly different interfaces.

We took an object-oriented view to the design of our system. The system was designed as a set of objects, each with a common set of data and processes. The principal organizing object is the project. A project contains all the data and processes needed to perform most group tasks. Projects are organized in a hierarchy, so that projects can contain sub-projects, sub-sub-projects, and so on. All projects contain the knowledge to be added, deleted, modified, and moved, among other functions.

Each project in turn contains a set of topics. Topics contain many of the same properties as projects. They are also organized in a hierarchy and can be added, deleted, modified, moved, and so on. Each topic can also be rated on a set of criteria defined by the group. Topics in turn contain comments (paragraphs of text) that can be added, deleted, and modified.

Browser Environment

The first major design decision was the development environment: what Web browser should we aim to support? One option was to develop the system to run on any Web browser; that is, to develop the system for the lowest common denominator, except for the use of forms. Alternatives would have been to develop our own Web client, or to develop for one of the specialized browsers such as Netscape or Hot Java. We choose the lowest common denominator strategy, because we wanted the system to be useable by the widest number of users.

Most of the development proceeded using the lowest common denominator strategy. However, midway through the development, it became evident that there were significant differences in execution among even the small number of most commonly used browsers. For example, virtually all browsers return form information in the order in which the form fields are defined. However, the Spry Mosaic browser used by CompuServe returned information in reverse order.

This, coupled with the fact that the voting module could benefit from the use of the table feature in Netscape, made us change strategy. We choose to develop the system to run on Netscape 1.1 browsers, even though this would limit the potential users. Our goal is to stay one step behind the bleeding edge of technology.

Server Environment

We choose a Linux server environment because we believed that it could provide fast throughput under heavy load -- and because it was the only server available to us. The next problem was to find a good inexpensive text-oriented database. We chose MiniSQL, a lightweight DBMS offering a basic subset of SQL features that is at least as fast as, if not faster than, Oracle and Ingress. It includes the database engine, a terminal "monitor" program, a database administration program, a schema viewer, and a C language API. It can support up to 75 clients without any extra memory overhead regardless of the number of active clients.

Although MiniSQL has a Perl interface, we decided to write C programs. Perl is an interpreter-oriented language and therefore is more limited in functionality and several times slower than C. The system uses a series of C programs working on the SQL database to generate the HTML forms that are used to capture the user's commands and present information in response. These forms are generated on the fly in response to the user's actions. Each program in our system first generates an HTML Form by reading the parameters passed by the previous program and by accessing data from the MiniSQL database.

The Web server doesn't remember the previous states that it took reach any given screen; each transaction is a separate request. Therefore, the HTML form that is returned with each request must contain sufficient information to tell the system all choices made by the user in all previous screens, in addition to the current request. For example, to add a new comment into a topic, the HTML form must contain information on the current topic and current project, because the system has no other way of knowing in which project the user

is working. This information is passed along from program to form and form to program as a hidden value. Thus each form contains the information needed to process the current request, and also the system's memory of who the user is and where the user was.

Debugging in such an environment can be difficult. Our programs produce HTML documents which are displayed by web browsers. Any error in a program will result in a badly formed HTML document that cannot be read by the browser, causing the browser to display an Error 500. One cannot use regular C debugging tools to locate the mistakes in these programs since the inputs come from a HTML form from the web server, not the normal input process.

The decision to use an object-oriented design approach meant that many of the software modules could be reused. For example, the code to add, delete, modify, and move topics is virtually identical to the code for projects. Once the program modules for the projects were complete, it was simple to reuse most of it for the topics.

Lessons Learned

The first major lesson we learned was that it is possible to develop a major application system using the Web. The use of a Web browser such as Netscape as the client environment offers some distinct advantages and disadvantages over traditional system development. First, because the Web browser provides so many built in functions, development proceeds much faster than it would in a traditional development environment. Traditionally, 80% of any application is the user interface. With Web-based development, much of the interface is automatically provided by the browser; you only need to specify what functions to use. It is very much like building with children's blocks -- or objects. Interface coding is reduced to about 40% of the application, saving a significant amount of programming time. The disadvantage is that you are limited to the capabilities of the browser. For example, traditional interface concepts such as drag-and-drop are not (yet) available, so you can't use them.

A second major lesson deals with user expectations. Most users have been quite enthusiastic about the software. It is a novel Web application and also provides clear value to their work groups and project teams. However, some users have been less than enthusiastic. Rather than viewing the system as a Web application, they see it as a regular desktop application and hold it to the same standards. They expect to use drag-and-drop to move items and use pull-down menus with multiple windows. Until the next generation of browsers become more common, we will not be able to satisfy them. Finally, we learned that it is important to test the system on many computers and browsers. Netscape, for example, produces slightly different displays for the same HTML form when running on UNIX, Macintosh, or Windows. And each version of Netscape has its own unique bugs.

Conclusion

We believe that we are the edge of a revolution in system development. We expect that within two years, most companies will realize that the Web is not only as a means of electronic publishing, but can be a full-scale system development environment. Rather than developing systems for Windows, UNIX, or Macintosh, companies will begin developing for the Web. Even systems that are designed solely for internal use (and do not access the Internet) will use Web browsers as the client. Of course, the Web will need to mature, and there will still be the need for traditional applications, but more and more applications that require networking will use the Web, rather than proprietary operating systems and networks. Those who get to the Web first, learn its intricacies, and push its limits will have a distinct competitive advantage.

References

Fellers, J.W., Clifton, A. and Handley, H. (1995) Using the Internet to Provide Support for Distributed Interactions, in Proceedings of the Twenty-Eighth Annual Hawaii International Conference on System Sciences, 52-60.

Jessup, L.M. and Valacich, J.S. (eds.) (1993), Group Support Systems: New Perspectives, Macmillan Publishing Company

Nunamaker, Jr., J.F., Dennis, A.R., Valacich, J.S., Vogel, D.R., and George, J.F. (1991) Electronic Meeting Systems to Support Group Work. Communications of the ACM, 34 (7): 40-61.

Quek, F. and Tarr, I. (1996) An Example of the Use of the WWW as a tool and environment for Research Collaboration, in IFIP Working Group 8.4 Conference Proceeding (ed. Glasson, B. et al), April, Arizona