The Phoenix Project: Distributed Hypermedia Authoring

M. G. Lavenant & J. A. Kruper

Biological Sciences Division Academic Computing
The University of Chicago
924 E. 57th. Street
Chicago, IL 60637-5415 USA

ABSTRACT

The Biological Sciences Division Office of Academic Computing (BSDAC) at the University of Chicago is the primary resource for support and development of instructional and research computing applications in the biomedical sciences. Driven by the goal to unify traditionally isolated teaching, research and clinical computing resources, the group has initiated a broad scale development effort known as the Phoenix Project.

The aim of the Phoenix Project is to develop an integrated academic information system providing full Internet connectivity and wide-area distributed hypermedia authoring services to the students, teachers, researchers, clinicians, and administrators who comprise BSDAC's user base. While the World-Wide-Web and its underlying data standards, HTML+ and HTTP, provide a flexible and yet powerful foundation for such a computing environment, they also present, in their current implementations, two significant limitations with respect to distributed hypermedia authoring: the lack of a user-friendly cross-platform HTML authoring tool, and rudimentary wide-area authentication/ authorization service integration. Our development effort over the past nine months has thus been twofold: to develop an effective X-windows based WYSIWYG HTML browser/editor, and to construct a prototype for integrated wide-area authentication and authorization support for HTTP service.

In this paper, we outline the design principles and application features present in the Phoenix software environment. We also suggest improvements to current WWW-based standards, and describe future directions for Phoenix development efforts. Finally, we present selected examples of how our user base is applying Phoenix utilities to further the Division's tripartite mission of advancing research, education, and patient care.

INTRODUCTION & BACKGROUND

The University of Chicago has a long and distinguished history of excellence and innovation in the areas of biological research and education. Beginning with the efforts of Robert Maynard Hutchins, and continuing with John Dewey, Joseph Schwaub, and today's current faculty, Chicago has maintained a high commitment to leadership in education. The University remains the only institution in the United States in which the basic science and medical faculties share responsibility for teaching in the biological sciences on every level, from undergraduate to postdoctoral, and Chicago is one of only a few Universities that maintains a biology core requirement for all students regardless of major.

The roughly 700 faculty members in the Division of the Biological Sciences form eleven clinical departments, six basic science departments, ten multidisciplinary academic committees, and twelve auxiliary and administrative departments. Over 200 academic courses serve a student population that includes 1,000 undergraduates, nearly 300 graduate students (affiliated with sixteen degree-granting departments and committees that comprise the Division's basic science programs), and over 435 medical students. The Division is responsible for education across the spectrum of the biological sciences -- from outreach programs for high school students and high school teachers, to the undergraduate college, to doctoral research, to medical education, to postdoctoral, residency, and continuing medical education.

It is in direct support of this community's tripartite mission of advancing research, education, and patient care that BSDAC has initiated a pioneering computing effort called the Phoenix Project.

The Phoenix Project

Computing at the University of Chicago, like many other institutions, has been "balkanized" through the creation of separate, non-overlapping computing resources for hospital administration, clinical practice, teaching, and research. As a result, individuals are unable to operate from a common desktop platform. Instead, they must master multiple operating systems running on computers that are physically separate. Thus, individuals who, for example, use personal computers extensively for analysis and presentation of research results fail to incorporate computing into other professional activities such as teaching and patient care. The goals of the Phoenix Project are to improve our ability to deliver high quality patient care, to provide instruction and training in the biomedical sciences, and to advance clinical and basic science research by integrating these disparate data sources through a common, readily accessible interface: the Phoenix Workstation.

The first tangible expression of these ambitious goals is an integrated academic information system providing full Internet connectivity and wide-area distributed hypermedia authoring services. As the next two sections detail, this system has been built upon the World Wide Web architecture, and includes two significant areas of enhancements within this framework: an effective X-Windows based WYSIWYG HTML browser/editor, and a prototype for integrated wide-area authentication and authorization support for HTTP service

THE PHOENIX HTML EDITOR

Our adoption of HTML as the defining data format for the Phoenix workstation was motivated by the format's power and flexibility -- qualities that have been amply demonstrated by the growth of the Web over the past year. While this growth is remarkable in and of itself, it is all the more so in light of the absence of a user-friendly HTML editor. Indeed, the Web author's toolbox, which currently contains conversion utilities (rtf2html, LaTeXtoHTML, ps2html) and rudimentary HTML editors (tkWWW, NextStep editor, the WYSIWYG Hypercard Stack, EMACS HTML-mode), still lacks an effective authoring tool that we can deliver to the members of our user community.

Like most providers of information and computing services, we cater to a heterogeneous group of users, both in terms of their comfort and familiarity with computers, and with respect to their installed base of hardware. Our foremost concern in defining a development strategy for the Phoenix Project has been to address this heterogeneity among our users by providing them with an editor that would be intuitive and yet full-featured, and in addition would be accessible from the three preferred operating systems on campus: Macintosh, MS DOS/WINDOWS, and UNIX.

Of the existing HTML authoring tools, Joseph Wang's tkWWW nearly satisfied our design specification. Written in the Tool Command Language (TCL) and its associated X-Windows based Tool Kit (Tk), it can be delivered to all three of our target platforms via the X-Windows service. Furthermore, it is one of the few existing editors (along with the Next editor) to support a near WYSIWYG editor interface. While it suffers from a number of significant shortcomings with respect to our particular needs (a relatively awkward user-interface, error-prone performance, difficult installation, and incomplete support of the HTML+ specification including in-line images, forms, and various text-format types), we judged it could serve as an effective springboard from which we could develop an editor tailored to our particular design objectives.

We thus built the Phoenix editor around tkWWW (version 0.9), and owe a great deal to Joseph Wang and to the considerable work he has invested in his original editor. While we have remedied many of the shortcomings we perceived in the original tkWWW design, Phoenix is still in beta release undergoing beta-testing by our Divisional community. This beta release, enhanced with the features described below, is currently running both locally on Unix machines (System V) and remotely, served to Macintosh or PC platforms running X-server software. More importantly, it is proving to be an effective Web authoring tool in the hands of our initial beta testers.

Phoenix Features

Interface enhancements

Much of our development effort has been devoted to enhancing the Phoenix editor's interface. We have pursued this objective by adhering to the general interface format of the Macintosh, which as the most popular hardware platform on campus, is also the most familiar to our users. Wherever there exists a correspondence between a Phoenix feature or function and a similar one on the Macintosh, we have defined the Phoenix behavior to mimic that of the Macintosh. For instance, Cut, Copy and Paste in Phoenix have key-bindings and behaviors corresponding to those on the Macintosh. Moreover, we have enabled Macintoshes running remote Phoenix clients through their X-server to share the contents of their clipboard with the Phoenix client both to reinforce this common interface and to facilitate the integration of the editor within the user's local (Macintosh) application environment.

However, this general approach cannot be adopted for Phoenix features that lack a Macintosh counterpart. In these cases we have attempted to maintain the general spirit of the Macintosh interface and suggest what their eventual behaviors should become. For example, clipboard commands treat a link as an object and operate on the HREF markup element and its enclosed anchor as a whole, rather than simply treating it as text. Correspondingly, double-clicking on a link invokes a dialog window through which to edit the properties of the link.

The Web's greatest departure from the standard Macintosh or PC interface lies in its transparent integration of distributed documents and file services. While interface paradigms exist for conventionally distributed file systems (Appleshare, NFS, etc.), they typically represent file systems that support standalone documents lacking a comparable degree of integration. We have therefore chosen to depart from the standard file system interface and instead transparently provide file services to our Phoenix users. While allowing direct access to HTML and HTTP for the cognoscenti, we also provide a user-friendly interface in which URLs, HTTP servers, and comparably intimidating creatures are hidden behind aliases. Thus, all basic file operations can be performed via indirect references to the underlying URLs, user and group names that use existing URL aliases (such as Hotlist, and History items), as well as novel aliases corresponding to users and groups.

Additional HTML+ Support

We intend to support currently implemented provisions of the HTML+ specification and to maintain our Phoenix HTML+ support apace with the evolution of the specification itself (including forms and tables). Toward this end, we have extended tkWWW's support of HTML+ by providing in-line image support (GIF format) in our beta-release Phoenix editor in both browse and edit modes. Phoenix supports the <IMG SRC="URL"> markup both within and outside <A HREF="URL">...</A> markup elements. The Phoenix clipboard supports the IMG markup either as a link to the image file specified in the IMG-URL, or as the image itself. The distinction between these two modes is made by providing two paste commands: Paste, and Paste Image. Cut and Copy remain unique.

DISTRIBUTED HTTP FILE SERVICES

The Web infrastructure we are deploying for the Phoenix Workstation environment will consist of a dozen or so Web servers and hundreds of Web clients distributed around campus. Existing servers provide a variety of information services, including personal 'desktop' file service, course information (class notes, curricula, homework drop-off, quizzes), and dedicated content-based information archives (bio-medical images, Health Information Resources, Medline, etc.). Users enjoy read-write access to these resources according to the authoring/browsing permissions governing them. Integrated organization of these distributed information resources is achieved using the existing features of HTML and HTTP. The corresponding integration of user authentication and authorization, however, requires extending the feature set of current HTML/HTTP implementations.

Specifically, provisions for these services in the leading HTTP servers are designed around locally maintained user name-space and access-control data structures. While these solutions are not unduly cumbersome within the existing Web environment, (predominantly world-read, local-write), they are insufficient to support the wide-area multi-server authoring environment we are deploying for the roughly three thousand potential Phoenix users in the BSD. The model underlying our integrated wide-area authentication and authorization support for NCSA HTTP1.2 service is outlined below.

Authentication

Two authentication models are currently available: host filtering, and user-authentication. In host filtering, access is granted on the basis of the network IP address of the client issuing the service request. While this level of authentication is ideal for distributing documents covered by site-licenses or organization level subscriptions, it is insufficient to provide the granularity and security required of a distributed file-system. User-authentication better satisfies such requirements, but proves unwieldy in its current implementation; access to "protected" documents requires client software (browser/editor) to explicitly prompt the user to provide server-specific authentication information (name, password) for HTTP requests of every server queried in a particular session. The model's reliance on local user name-space precludes transparent HTTP service integration across multiple servers, from the perspectives of both users and system administrators.

We are addressing these shortcomings in our Phoenix environment by implementing the HTTP-specification's provision for Kerberos authentication:

```
Authorization:kerberoskerberosauthenticationparameters
```

This solution relieves the HTTP server of any ancillary authentication responsibility and provides the client software with appropriate hooks to a dedicated Division-wide Kerberos authentication server.

Upon launch, the Phoenix client (X-windows browser/editor) prompts the user for a login and password and obtains a ticket from the Kerberos server. This ticket is used by the client to request server-specific kerberos tickets from the kerberos ticket granting server. These Kerberos service tickets are then passed by the client to "Phoenicized" HTTP servers in subsequent service requests.

Our HTTP servers are correspondingly "Kerberos aware," maintaining a kerberos key with which to decrypt Kerberos tickets issued by Phoenix clients. Successful decryption of incoming Kerberos tickets, passed in the HTTP service requests, authenticates the identity of the requesting user. The HTTP request is then passed, along with the decrypted username, to the server's authorization service to determine whether the particular request can be satisfied. Otherwise, an error message is returned to the client. While this scheme provides integration of HTTP service user name-space and transparent authentication, it preserves the "flavor" of the WWW by supporting the integration of a single user name-name space distributed across multiple Kerberos servers.

Authorization

Current HTTP authorization models are based upon a directory based organization of served documents. According to this scheme, access is granted to the content of directories rather than to individual HTML documents *per se*. Files in a given directory are thus subject to identical access control, specified either in the HTTP server's main configuration file or in a subsidiary control file located in the directory itself.

This approach suffers from three principal limitations. First, it imposes an awkward document/directory structure upon the HTTP server's file-system, whereby documents are organized on the server according to access-permission rather than upon a more "natural" basis such as content. Second, it provides no effective means of modifying the access permissions of individual documents without moving them to a new location in the file-system, thereby breaking any existing hyper-links for which the given document is a target. Finally, it makes no provision for client-side document access-control management.

The CTRL Markup element

Our answer to these limitations is to implement an experimental HTML+ markup extension and associated support services. We define the HEAD HTML markup element, <CTRL SRC="URL">, to specify the location of the access-control file governing the access to the document:

```
<HEAD>
<TITLE>Demonstration of The CRT markup
element</TITLE>
<CTRL SRC="protocol://hostname/path/access-
controlfile ">
...
</HEAD>
<BODY>
...
</BODY>
```

The access control files are served both in browser--->server and in server--->server requests.

This authorization scheme satisfies our requirements for document level permission granularity and unhindered file-system organization of served HTML documents. Modification of document access permissions is accomplished by simply updating the URL of the CTRL markup element to point to the new access-control specification file.

Document READ-WRITE

In our authorization model, our HTTP servers are supported by a dedicated authorization service; it is to this service that all authorization requests are referred. Authorization requests, issued by the HTTP server, comprise the following elements of the original HTTP request: the METHOD, the CTRL-URL, and the authenticated user-name. Authorization of the request is performed by retrieving the access control file (locally or via HTTP) and comparing it to the user-name and METHOD provided in the service request. A pass or fail message is then returned to the HTTP server for further processing of the user's original HTTP request.

Client-side document permission modification

Our model provides two client-side methods for editing document-permissions: modifying the CTRL-URL mark-up element in the underlying document, and editing of the CTRL-URL document itself. These correspond, respectively, to changing the access to the current document, and to altering permissions of the entire set of documents currently governed by a given CTRL-URL.

The first is performed with the Save as... dialog in the Phoenix client. The set permissions option, available in this dialog, retrieves the CTRL-URL (GET-CTRL) and parses it into a user and a group list. The parsed entries are displayed to the user in the document permissions dialog, along with those maintained for the user by the client in the user's local .users and .groups file. Following the user's selection of appropriate entries from the user and group lists, the client issues a POST-CTRL service request for the revised list to the HTTP server specified in the CTRL-URL. The HTTP server authenticates and authorizes the request, and checks for an existing permission specification matching that defined in the request. If one is found, its URL is returned to the requesting client; if not, a new access-control document is generated and its URL returned. In each case, the CTRL-URL is returned to the client in the same POST-CTRL transaction.

The second method, performed independently of a particular document-editing session, is accomplished through the Web Permissions dialog. The client retrieves a specified CTRL-URL from its server and updates it as above, then issues a POST-CTRL service request. The server authenticates and authorizes the request, and overwrites the existing control file with the newly submitted one.

APPLICATION OF PHOENIX IN THE BIOLOGY CURRICULUM

It has long been a goal of instructional technology efforts to develop a "virtual classroom" that provides a forward looking, learner-directed exploratorium in which students and teachers alike can explore, discover, communicate, collaborate, and learn. With the advent of the World Wide Web and sufficiently powerful tools such as Phoenix, building such an environment is now within reach. This section describes our initial applications of Phoenix technology to the Biological Sciences Division curriculum.

In our efforts, we have sought:

•to develop utilities that dynamically generate personalized "home pages" on logging into the system. These home pages contain links to custom built and existing information targets that range from the individualized personal information source (a student's independent research project, for example), to shared group targets (a class home page), to local and remote information servers (a library catalogue or remote database);

•to provide default frameworks for presenting common information constructs, such as a class home page;

•to build these default frameworks in a way that permits easy enhancement and modification by individual faculty and students;

•to scale this framework to support the Division's entire class offerings -- over 100 classes serving over 1700 students per term; and

•to provide a series of shared information utilities (an Image Archive, for example) that users can contribute to and draw from in support of the teaching and learning process.

From these objectives, we have implemented a first generation *Class Information Architecture* that includes script-generated Class Home Pages and a set of supporting information retrieval utilities that operate within this framework.

At the beginning of each academic term, a series of scripts uses a data feed from the University's Office of the Registrar to build default Home Pages for each class offered in the BSD during that term. These Home Pages contain appropriate graphical banners that are built up from a series of elemental graphic constructs (subject headings, course numbers, etc.). In addition, these banners are also links to the corresponding target's home page (a high level graphics for the BSD Office of Academic Computing for example, points to the BSDAC home page).

The principal content of the Class Home Page, however, is a set of standard headings and links representing the following areas:

•Course Instructor
•Course Syllabus
•Class Announcements
•Class Notes

With the exception of the Course Instructor heading (which points to the appropriate faculty biographical HTML target page), the targets for each of these headings/anchors is a blank page. It is up to the course instructor(s) to create and add structure and content to this sub-web. Of course, with the aid of the WYSIWYG editing functions in Phoenix, this is now an simple routine.

For example, an instructor may wish to use the (initially blank) target page of the Class Notes anchor to build a simple notes index based on lecture date. Alternatively, an instructor could choose to organize his/her class notes using a subject-based framework. The ability for the instructor to decide how to organize and present this information represents a deliberate effort on our part to provide some level of consistent structure (allowing automated generation and maintenance of a framework) while at the same time giving the users the ability to "personalize" the structure.

An additional feature of the Class Home Page construct is the inclusion of a searchable WAIS-index of the class' sub-directories. With this, a user can search for any text string within that Class' Notes, Announcements, Syllabus, etc. After a search is specified, Phoenix returns the results by appending the hits, an anchor to the target destination, and the content of the Head element, onto the Class Home page. In this way, searching and navigating through what can become a large amount of course information becomes easy and intuitive.

From a user's perspective, Phoenix uses its security features along with daily data feeds from the Registrar's office to generate (using scripts) users' home pages. These pages, which we call a user's "Nest," contain a series of links that range from the personal (a User's Personal home page) to the shared (the list of classes in which the user is either a student or instructor), to the general (the BSDAC and University's Home page, general Internet sources, etc.).

With this functionality, Phoenix provides a personalized distributed computing environment to the user. Because Phoenix "knows" the classes in which a student is enrolled and provides links to those classes' home pages, the system becomes a powerful platform to support a secure, distributed hypermedia authoring environment.

Currently, three pilot courses (an undergraduate molecular biocomputing course, a upper level genetics course, and a medical school immunology course) either have used or are using Phoenix as a Class Information Architecture. Anecdotal reports from these users have been very positive. As additional Phoenix features are added (WYSIWYG image editing, for example), we will make them available for testing by additional pilot courses. Our plan is to have by Fall the Phoenix Class Information Architecture configured and ready for adoption by the roughly 100 classes offered during a given term.

With these efforts, we are working to make Phoenix a true "virtual classroom," able to support a collaborative learning environment where students can acquire and practice the skills needed to access and manage information, formulate effective questions, test hypotheses, solve problems, make judgments, and express themselves logically and lucidly.

ACKNOWLEDGMENTS

The authors wish to thank the BSD Academic Computing staff and student assistants who have contributed, and continue to contribute, to the development and implementation of the Phoenix Project. Special appreciation is extended to Lee Newberg and to Phillip Stylianos, without whose contribution Phoenix would be but still a dream.