Active Pages: Intelligent Nodes on the World Wide Web

Henry Houh, Chris Lindblad, and David Wetherall

Telemedia, Networks and Systems Group
MIT Laboratory for Computer Science

31 March 1994

Abstract

Active pages provide a common interface to World Wide Web applications, crossing browser, platform and operating system boundaries. They are hypertext documents that present a front−end to intelligent applications. Typically implemented as interpreted programs with an associated database, they use the *forms* extension for application input.

There are three advantages to the *active pages* approach for application interfaces. Application interfaces are widely *accessible* because they leverage off of the accessibility of the Web by using HTTP to bridge application and interface. Applications are *self−documenting*. The hypertext model of the Web makes it simple for active pages to contain embedded documentation and links to auxiliary material. Finally, applications *integrate seamlessly* with the Web. Active pages may be accessed in the same way and with the same browser as other pages.

In this paper, we present our active page design methodology and demonstrate it with two examples from our server: WEBDNS, a facility for editing Internet Domain Name System master files; and The People Directory, an editable personnel database that includes hypertext links to biographical pages.

Introduction

The World Wide Web can be an interface to shared applications, as well as a mechanism for electronic publishing and collaboration. Applications on the Web can leverage off of the portability of the Web for access. In turn, Web users can leverage off of the flexibility of programs to improve their collaboration.

The pages of a shared application change in appearance and content as viewers progress through them. Input from one viewer can become visible to others. The same information can be presented differently, depending on the viewer and their preferences. From the viewers' perspective, the pages appear *active* and *intelligent*. These *active pages* contrast with other pages, which appear the same from invocation to invocation.

We developed an approach to the design of active pages suited to shared applications. To implement the active pages, a server interface for executing programs and a client interface for accepting input is required. We adopted the *Common Gateway Interface (CGI)* extension to HTTP servers and the *forms* extension to Web browsers, respectively, because they were immediately available and in widespread use.

In this paper, we describe our design and illustrate it with two examples. The first, WEBDNS, facilitates the interactive update of Domain Name System (DNS) master files. The second, The People Directory, presents an editable personnel database.

The Advantages of Active Pages

Active pages provide several advantages to shared applications. They are widely and conveniently *accessible*, especially suited to *collaborative* tasks, encourage *documentation*, and are *seamlessly integrated* with other Web pages.

Accessiblity

The broadest feature of active pages compared to other application interfaces is widespread accessibility. The Web is portable across platform and operating system and network boundaries. Just as for other Web pages, active pages may be accessed interactively, across the network, and from a variety of computer platforms. This makes them appropriate for services that benefit from being freely available to a hetrogeneous community.

Collaboration

Many users may simultaneously access one active page. This, coupled with the fact that the application executes on the server, rather than the client, allows many users to share few special resources. This type of access is especially useful for collaboration within a user community, and can be applied to many database systems. Two large examples of database systems that can be interfaced to the Web with active pages are SABRE, the United Airlines reservations system, and Lotus Notes.

Documentation

From the programmers' perspective, a hypertext interface encourages applications to be self−documenting. Instructions and other forms of documentation are typically embedded in the program to be given as output. Auxiliary material may be provided by links.

Seamless Integration

Active pages are accessed in the same manner as other Web pages. Viewers may use their preferred browser, complete with their customizations. No additional applications are required. This seamless integration makes active pages convenient to use. They are even appropriate for small and frequent tasks, especially those often used in conjunction with Web browsing. For example, a Visitors' Book active page may allow each visitor to a certain server to "sign", appending a comment to a publicly visible log.

Approach

An application can be made accessible to Web users by mapping its functions into a book of active pages. This design task is completed by resolving navigation, concurrency, access restriction, and input validation issues.

Navigation

The interface of the shared application must be mapped into a collection of navigable Web pages. Several different styles of navigation are commonly used and can be combined with active pages. The linked structure of hypertext pages can serve as an application menu. Selected portions of the application database may be revealed through forms, in a manner similar to searchable indexes. Forms also provide a stronger commit mechanism than links, and are more appropriate for application input.

Concurrency

Because many viewers may be using an active page simultaneously, concurrent operations on the application database must be managed. Queries may overlap without affecting each other, but updates may not be able to be safely overlapped. Standard database techniques such as locking are used to address this problem. For non−critical applications, it is sufficient to minimize the update time so that overlap is unlikely, and to fail gracefully if overlap still occurs.

Access Restriction

Different kinds of access to services are appropriate for different viewers. Generic access restriction mechanisms are provided by many servers: NCSA httpd supports domain and password protection. With active pages, the flexibility of program execution can be used for less intrusive and more sophisticated restrictions. Viewing a page from within a company, for example, may show more information than from outside the company.

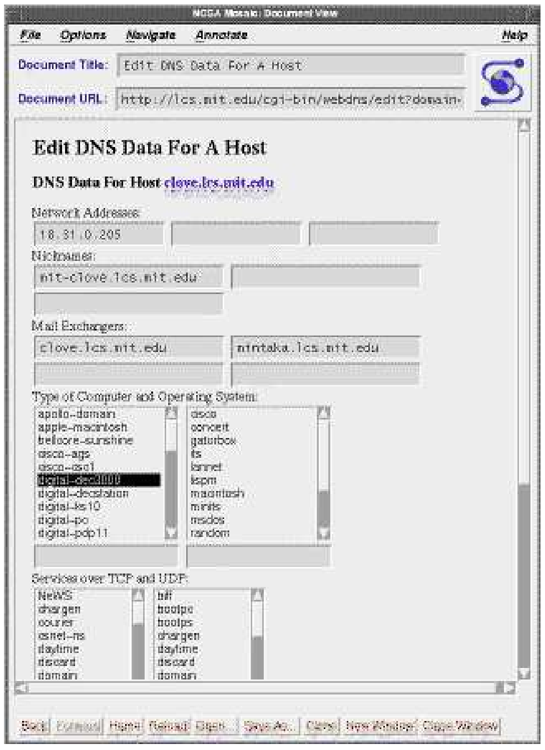
Input Validation

All user input needs to be validated before it is acted upon. Dynamically manufactured forms may be useful to restrict input and provide preliminary screening of data. Even with this approach, viewer input must be checked to ensure security. The wide accessibility of pages and the statelessness of the HTTP protocol make validation more important than it would be otherwise.

Examples

Our example active page applications use the CGI interface to the HTTP server, and the forms extension for input. The first demonstrates how a database application residing outside the Web may be manipulated from within the Web. The second manipulates a database maintained solely for Web use. Both are scripts written in interpreted languages that operate on text–based UNIX databases. As a convention, our active pages also provide a source listing of themselves when requested.

WEBDNS



Document Title: Edit DNS Data For A Host

Document URL: http://lcs.mit.edu/cgi-bin/webdns/edit?domain-

Edit DNS Data For A Host

DNS Data For Host clove.lcs.mit.edu

Network Addresses: 18.31.0.205

Nicknames: mit-clove.lcs.mit.edu

Mail Exchangers: clove.lcs.mit.edu | mintaka.lcs.mit.edu

Type of Computer and Operating System:

apollo--domain	cisco
apple--macintosh	concert
bellcore--sunshine	gatorbox
cisco--ags	its
cisco--csc1	lanmet
digital--dec1000	lispm
digital--decstation	macintosh
digital--ks10	minits
digital--pc	msdos
digital--pdp11	random

Services over TCP and UDP:

NeWS	biff
chargen	bootpc
courier	bootps
csnet--ns	chargen
daytime	daytime
discard	discard
domain	domain

WEBDNS is a Web facility for editing Internet Domain Name System (DNS) master files. These files are used to translate hostnames to Internet addresses, a basic service depended on by most network applications. The files are shared amongst a community, any of whom may need to alter them.

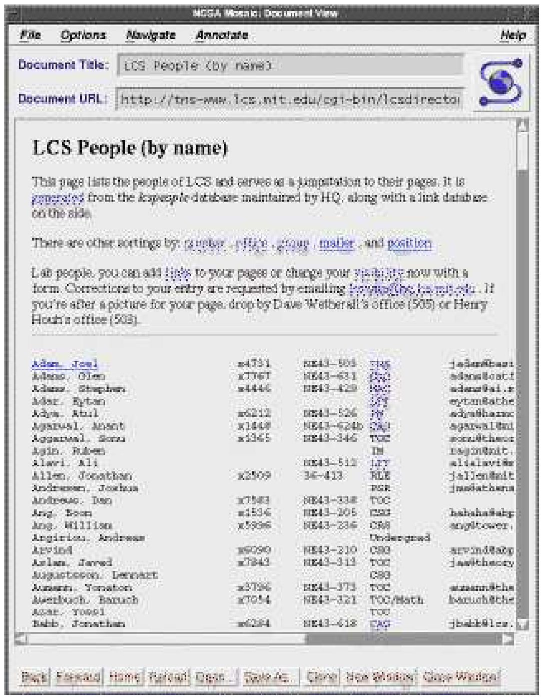
At its top level, WEBDNS provides the viewer with a small menu of links, allowing them to choose whether to edit, copy or delete DNS records. Each of these operations requires the viewer to specify the host they wish to change. Two types of editing are supported. A formatted facility constrains viewer input, providing a straightforward means of affecting anticipated changes. Editing of the raw DNS records provides a fallback for more experienced administrators to enter broader changes.

WEBDNS is implemented as a Perl script. Every time it is invoked, it uses the named.boot file on the machine on which it is running to find all the DNS master files for which the machine is a primary server. It then effectively reads in the master files.

The action WEBDNS performs is determined from the path and query strings of the URL. If it is a request to view data or build a form, WEBDNS extracts and presents the relevant data. If the request is a post, a change has been submitted. WEBDNS checks the contents of the post for consistency, updates its internal data structures, and then dumps out new versions of all the master files and restarts the name server process. To reduce the risk of concurrent updates, WEBDNS applies advisory locks to the master files during the update.

Performance and concurrency issues complicated the design of WEBDNS. Because it may take a long time to read and write the master files, WEBDNS maintains a cache of the data in random–access dbm files. It reads from this cache if the master files have not been updated. Similarly, it may be instructed to write to this cache only, allowing the postponement of updates to the master file until the cache holds the complete group of updates. Caching makes database access much faster.

The People Directory



Document Title: LCS People (by name)

Document URL: http://tns-www.lcs.mit.edu/cgi-bin/lcsdirecto

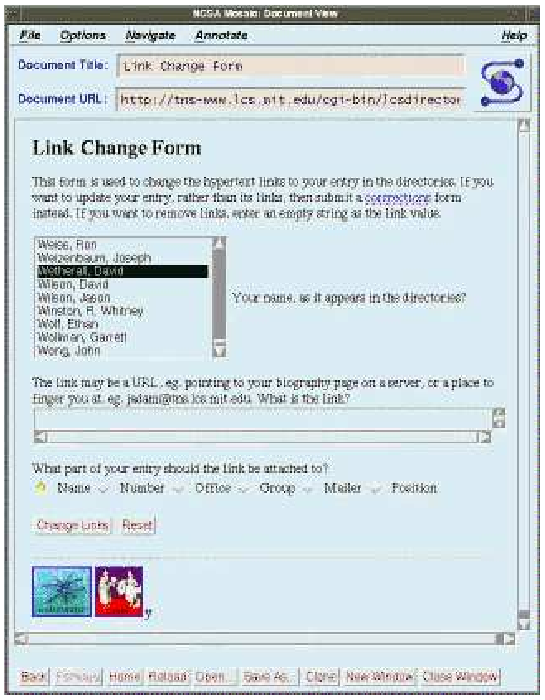
LCS People (by name)

This page lists the people of LCS and serves as a jumpstation to their pages. It is generated from the *lcspeople* database maintained by HQ, along with a link database on the side.

There are other sortings by: number, office, group, mailer, and position.

Lab people, you can add links to your pages or change your visibility now with a form. Corrections to your entry are requested by emailing [illegible]. If you're after a picture for your page, drop by Dave Wetherall's office (505) or Henry Houh's office (503).

```
Adam, Joel              x4731   NE43-505   TNS          jadam@basi
Adams, Olen             x7767   NE43-631   CSG          adams@catt
Adams, Stephen          x4446   NE43-429   MAC          adams@ai.m
Adar, Eytan                                SPT          eytan@athe
Adya, Atul              x6212   NE43-526   PM           adya@harmo
Agarwal, Anant          x1448   NE43-624b  CAG          agarwal@mi
Aggarwal, Sonu          x1365   NE43-346   TOC          sonu@theor
Agin, Ruben                                IM           ragin@mit.
Alavi, Ali                      NE43-512   LPT          alialavi@m
Allen, Jonathan         x2509   36-413     RLE          jallen@mit
Andresen, Joshua                           PSR          jma@athena
Andrews, Dan            x7583   NE43-338   TOC
Ang, Boon               x1536   NE43-205   CSG          hahaha@abp
Ang, William            x5996   NE43-236   CRS          ang@tower.
Argirion, Andreas                          Undergrad
Arvind                  x6090   NE43-210   CSG          arvind@abp
Aslam, Javed            x7843   NE43-313   TOC          jaa@theory
Augustsson, Lennart                        CSG
Aumann, Yonatan         x3796   NE43-373   TOC          aumann@the
Awerbuch, Baruch        x7054   NE43-321   TOC/Math     baruch@the
Azar, Yossi                                TOC
Babb, Jonathan          x6284   NE43-618   CAG          jbabb@lcs.
```



Document Title: Link Change Form

Document URL: http://tns-www.lcs.mit.edu/cgi-bin/lcsdirector

Link Change Form

This form is used to change the hypertext links to your entry in the directories. If you want to update your entry, rather than its links, then submit a corrections form instead. If you want to remove links, enter an empty string as the link value.

Weiss, Ron
Weizenbaum, Joseph
Wetherall, David
Wilson, David
Wilson, Jason
Winston, R. Whitney
Wolf, Ethan
Wolman, Garrett
Wong, John

Your name, as it appears in the directories?

The link may be a URL, eg. pointing to your biography page on a server, or a place to finger you at, eg. jadam@tns.lcs.mit.edu. What is the link?

What part of your entry should the link be attached to?
[✓ Name] Number Office Group Mailer Position

Change Links | Reset

The People Directory is a collection of pages that list the several hundred members of the Laboratory in a White Pages style. It is convenient for such tasks as: finding email addresses, determining the students of a research group, and finding who shares an office.

Several alternate listings can be requested via links, each suited to these different tasks. The default style lists all members alphabetically by surname. Other listings categorize people by research group or position, or order them by office number, phone number, or email address. For each member, any of these contact details may be linked to biographical pages. The listing presented to a viewer also depends on their location. A visibility attribute allows lab members to control who sees their entries. They may elect to be listed for all Web clients, only those around MIT, only those in the Laboratory itself, or for no clients.

Lab members may update their links to biographical pages and their visibility by submitting a form. Once submitted, new information is reflected immediately.

The People Directory is implemented as a pair of Tcl scripts that manage a database. One script presents the listings. It uses the query portion of the URL as presentation parameters. It loads the database and lists it, according to both the presentation parameters and the location of the viewer. A cache of previous reports is checked before generating a new one.

The other script updates the listings. It presents either a Link form or a Visibility form, depending on the URL used to reach it. When the viewer posts the form it validates the new information and updates the database. Changes are reflected the next time a listing is requested. Concurrency is addressed by minimising the overlap time and discarding all but the last update.

Findings

In our implementations we were constrained by our adoption of the CGI interface and the forms extension. For example, CGI programs run to termination to produce a single hypertext page. They cannot be used to maintain state information while the viewer interacts with the page.

More powerful interfaces from the server to programs, and from the client to user interface, would more easily permit intelligent Web interfaces. They would allow pages to be more easily customized to the viewer as well as be more efficient. Coupled with a viewer input model that calls back to the executing program, rather than causing a new program to be run, the browsing model of the Web can more closely resemble a distributed application than can plain hypertext. One example of this is the Common Lisp Hypermedia Server [Mallery].

We have found that active pages may be readily manufactured with interpreted languages, and may add intelligence compared to other pages.

Conclusions

We expect that the proportion of intelligent nodes on the Web will increase as it matures. As the information accessible via the Web multiplies, intelligent navigation, presentation, and analysis will become increasingly useful.

Active pages are Web pages that present a front–end to intelligent applications. They provide three advantages as application interfaces. They are widely *accessible*: they leverage off of the accessibility of the Web by using HTTP to bridge application and interface. They are *self–documenting*: the hypertext model of the Web makes it simple for active pages to contain embedded documentation and links to auxiliary material. They *integrate seamlessly* with the Web: active pages may be accessed in the same way and with the same browser as other pages. These advantages make active pages well suited to collaborative applications, where a community accesses and updates a shared database.

We presented an approach for designing active pages by addressing the issues of *navigation*, *concurrency*, *access restriction*, and *input validation*. The interface of the shared application must be mapped into a book of navigable Web pages. Standard database techniques such as locking are used to address the management of concurrent operations on the application database. Different kinds of access to services will be appropriate for different viewers and may be implemented by the application. And to ensure database consistency and security, all user input needs to be validated before it is acted upon.

We have illustrated this design methodology with WEBDNS and The People Directory, two examples from our server. WEBDNS demonstrates how a database application residing outside the Web may be manipulated from within the Web. The People Directory is an example of an application whose main use is browsing, but also permits its pages to be updated.

We were constrained in our implementations by our adoption of the CGI interface and the forms extensions. More powerful interfaces from the server to programs and from the client to user interface would more easily permit intelligent Web interfaces. They would allow the browsing model of the Web to more closely resemble a distributed application than does plain hypertext.

Acknowledgements

This research was supported by the Advanced Research Projects Agency of the Department of Defense, monitored by the United States Air Force (AFSC, Rome Laboratory) under contract No. F30602−92−C−0019, and by a grant from Nynex. We wish to thank Professor David Tennenhouse for helpful discussions and his support, and Jonathan Soo for his initial work.

References

Rob McCool The Common Gateway Interface, National Center for Supercomputer Applications.

National Center for Supercomputer Applications Mosaic for X version 2.0 Fill−Out Form Support, National Center for Supercomputer Applications.

Berners−Lee, T. HyperText Transfer Protocol Requirements, European Laboratory for Particle Physics (CERN).

Mockapetris, P.V., Domain names − implementation and specification, Internet RFC 1035, November 1987.

Mockapetris, P.V., Domain names − concepts and facilities, Internet RFC 1034, November 1987.

Mallery, J. C, T. A Common Lisp Hypermedia Server , Proceedings of the First International Conference on the World−Wide−Web.