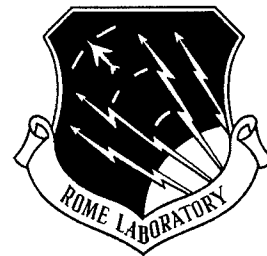


RL-TR-97-145
Final Technical Report
October 1997



WIDE AREA INFORMATION BROWSING ASSISTANCE

The Open Software Foundation Research Institute

Sponsored by
Advanced Research Projects Agency
ARPA Order No. B323

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the U.S. Government.

Rome Laboratory
Air Force Materiel Command
Rome, New York

DTIC QUALITY INSPECTED

19980209 067

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-97-145 has been reviewed and is approved for publication.

APPROVED:



PETER A. JEDRYSIK
Project Engineer

FOR THE DIRECTOR:



JOHN A. GRANIERO, Chief Scientist
Command, Control & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL/C3AB, 525 Brooks Rd, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

WIDE AREA INFORMATION BROWSING ASSISTANCE

Contractor: Open Software Foundation Research Institute
Contract Number: F30602-94-C-0209
Effective Date of Contract: 30 June 1994
Contract Expiration Date: 25 February 1997
Program Code Number: 4D30
Short Title of Work: Wide Area Information Browsing Assistance

Period of Work Covered: Jun 94 – Feb 97

Principal Investigator: Charles L. Brooks
Phone: (617) 621-8758
RL Project Engineer: Peter A. Jedrysik
Phone: (315) 330-2158

Authorized for public release; distribution unlimited.

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by Peter A. Jedrysik, RL/C3AB, 525 Brooks Rd, Rome, NY.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE Oct 97	3. REPORT TYPE AND DATES COVERED Final Jun 94 - Feb 97		
4. TITLE AND SUBTITLE WIDE AREA INFORMATION BROWSING ASSISTANCE			5. FUNDING NUMBERS C - F30602-94-C-0209 PE - 61101E PR - B323 TA - 00 WU- 01	
6. AUTHOR(S) Charles L. Brooks and Misha B. Davidson			8. PERFORMING ORGANIZATION REPORT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) The Open Software Foundation Research Institute 11 Cambridge Center Cambridge, MA 02142			10. SPONSORING/MONITORING AGENCY REPORT NUMBER RL-TR-97-145	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Advanced Research Projects Agency Rome Laboratory/C3AB 3701 North Fairfax Drive 525 Brooks Rd. Arlington, VA 22203-1714 Rome, NY 13441-4505				
11. SUPPLEMENTARY NOTES Rome Laboratory Project Engineer: Peter A. Jedrysik, C3AB, 315-330-2158				
12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for public release; distribution unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) This report contains information from the work performed under the Wide Area Browsing Assistance (WAIBA) contract to investigate the design of novel, intuitive environments for using the web maintaining notions of location - independent information and emphasizing the web's use by collaborative teams. The WAIBA effort is also the basis for graphical annotation work currently under the development as part of the DARPA-Sponsored Advanced Logistics Planning (ALP) program.				
14. SUBJECT TERMS World Wide Web, Browsing Associates			15. NUMBER OF PAGES 100	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

Part 1: Wide Area Information Browsing Assistance	1
1.0 Abstract	1
2.0 Introduction	1
3.0 Rationale	2
3.1 Mediated Access	6
3.2 Group Annotation	6
3.3 Browsing Associates	6
3.4 Notification	7
4.0 Accomplishments	8
4.1 The Strand Toolkit	8
4.1.1 Indexer Strand	9
4.2 Group Annotation	10
4.3 Mediated Access	10
4.4 User Profile Service	11
4.5 Browsing Associates	11
4.5.1 HistoryGraph	11
4.5.2 WhatsNew	12
4.5.3 Linktree	12
4.6 Blackboard Assistance	13
4.7 Summary	14
5.0 Results/Lessons Learned	14
5.1 Mediated Access	15
5.1.1 Proxy-Based Services	15
5.2 Browsing Assistance	15
6.0 Influences	17
6.1 Who influenced us?	17
6.1.1 Who have we influenced?	17
7.0 Publications/Conferences	18
7.1 Conferences	18
7.2 Workshops	18
8.0 Next Steps	18
8.1 Web-based Group Collaboration Services	19
8.1.1 Mediator	19
8.1.2 Group Annotation	19
8.1.3 Notification	20
8.1.4 Blackboard	21
8.2 Desktop services	21
8.3 Influence of Other technologies	21
8.3.1 JAVA (Javascript, LiveConnect)	21
8.3.2 Server plug-in modules	22
8.3.3 World-Wide Web standards	22
Part 2: Logistics Anchor Desk Option	25
9.0 Abstract	25
10.0 Introduction	25

11.0 Initial Plan for Building Graphical Annotation	26
11.1 System Architecture	26
11.1.1 GUI.....	26
11.1.2 Components and topology.....	27
12.0 Master/slave event architecture for Graphical Annotation Applet	28
13.0 Functional Interface for Integra-Integrated Text, Graphical and Audio Annotation System	31
13.1 Introduction	31
13.2 Usage Scenarios.....	31
13.3 Functional interface	32
14.0 Types of annotations	33
14.1 Main Application User Interface	34
14.1.1 Building blocks	34
14.1.2 Summary of Actions.....	38
15.0 First ALP Workshop, May 29-31, 1996	38
15.1 Overview of ALP.....	38
15.2 The RI's role in ALP.....	39
15.3 Collaboration Possibilities.....	39
15.4 Adding Web capabilities to other projects.....	40
15.5 ALP IFD -1.2.....	41
16.0 Second ALP Workshop, Sept. 10-12, 1996	41
16.1 Overview	41
16.2 Meeting Highlights.....	41
16.2.1 Brian Sharkey - overview.....	41
16.2.2 Future Plans.....	42
16.2.3 Conversation with Major Hamm.....	42
17.0 Third ALP Workshop, Dec. 11-13, 1996.....	43
17.1 Overview	43
17.2 Meeting Highlights.....	44
17.2.1 Day One.....	44
17.2.2 Day Two.....	44
17.2.3 Day Three.....	44
Appendix A: <i>Application-Specific Proxy Servers as HTTP Stream Transducers</i>	46 - 55
Appendix B: <i>Dynamic Integration of HTTP Stream Transforming Services</i>	56 - 65
Appendix C: <i>Pan-Browser Support for Annotations and Other Meta-Information on the World Wide Web</i>	66 - 79
Appendix D: <i>Transducers and Associates: Circumventing Limitations of the World Wide Web</i>	80 - 84

Part 1: Wide Area Information Browsing Assistance Final Technical Report

Charles L. Brooks

1.0 Abstract

The Wide Area Information Browsing Assistance (WAIBA) project was funded by DARPA to investigate the design of novel, intuitive environments for using the Web, maintaining notions of location-independent information and emphasizing the Web's use by collaborative teams. The WAIBA effort is also the basis for our graphical annotation work currently under development as part of the DARPA-sponsored Advanced Logistic Planning (ALP) program. Specific systems included within the WAIBA effort are:

- Group Annotation-allows users to create and share annotations to Web pages. Annotations may be retrieved based on selection criteria. The annotations may also be themselves annotated, permitting a threaded discussion facility.
- Mediator-permits teams to co-develop pages within the Web. Teams and teams' document Web spaces may be defined and access to the URLs in the defined Web space is mediated for orderly update.
- HistoryGraph-displays user's browsing history as a tree that can be restructured, pruned, collected into named sets and saved as files.

Various infrastructure libraries and services have also been developed, including the Strand toolkit for application-specific HTTP request/response interception and the User Profile Service for storing and maintaining user information.

2.0 Introduction

This paper represents the final report for the Wide Area Information Browsing Assistance (WAIBA) project. As such, it documents technical work accomplished and information gained during the past two years.

This paper is organized into the following sections:

- **Rationale:** In this section, we trace the project history with an eye towards explaining how we arrived at certain decisions.
- **Accomplishments:** here we list significant project artifacts, specifically those software modules that have been incorporated into larger systems.
- **Results/Lessons Learned:** here we summarize the impact of the project and the lessons learned from the development and fielding of the software
- **Influences:** we have been influenced by a number of people, and we have influenced several other projects and DARPA funded efforts in the course of our work.
- **Publications/Conferences:** a listing of significant papers published, conferences attended, and other technology transfer initiatives.
- **Next Steps:** directions for future work.

We conclude the paper with a list of references and a collection of appendices that contain copies of a selection of our published papers.

3.0 Rationale

We started our investigations from the point of view of using Web-based technologies as a way to provide better support for browsing material stored in information bases accessible via a wide-area networking environment. At the time the project began (mid-1994), the World Wide Web was just beginning to be widely accepted. The primary software available at that time was the NCSA and the CERN HTTP servers and the NCSA Mosaic Web browser. The primary platform for this software was UNIX (various systems).

Our initial proposal read as follows:

The OSF RI proposes to investigate fundamental issues in the design of human-computer interfaces for the emerging national information infrastructure. In particular, we will investigate the design of new hypermedia environments that support wide area information browsing. The challenge is that these interfaces must make the underlying wide area information environment transparent. The user must experience the same intuitive characteristics of point-and-click browsing, whether the information is on his local machine or a continent away. This research project will identify the appropriate meta-knowledge needed for effective wide area information browsing and deliver prototype hypermedia browsers, browsing associates and hypermedia servers that employ this information to preserve the intuitive character of point-and-click browsing. This work will build upon the multimedia Mosaic browser developed by the University of Illinois and the network-capable hypermedia servers developed by CERN

We will extend the Mosaic network browser with a wide area information browsing associate (WAIBA) that can ameliorate the many areas of potential confusion for the user. [OSFRI94]

Our initial definition of the Browsing Associate was a set of cooperating agents employing a blackboard to provide assistance to the user. Our model was that each agent would implement a restricted action of

interest to the user (such as observing time and cost, network availability or changes to previously examined information structures) and would report its results using the blackboard.

Our initial architecture is depicted in **Figure.1**. The server in this picture provides the blackboard function, recording comments from pre-existing "free agents" that are roaming the Web: these agents can query the server to ascertain requests from users or other agents, and can record results.

Users interact with agents through the same blackboard server, looking at the information recorded by the agents. Users can create or control agents by leaving notes on the server using simple HTML-based forms.

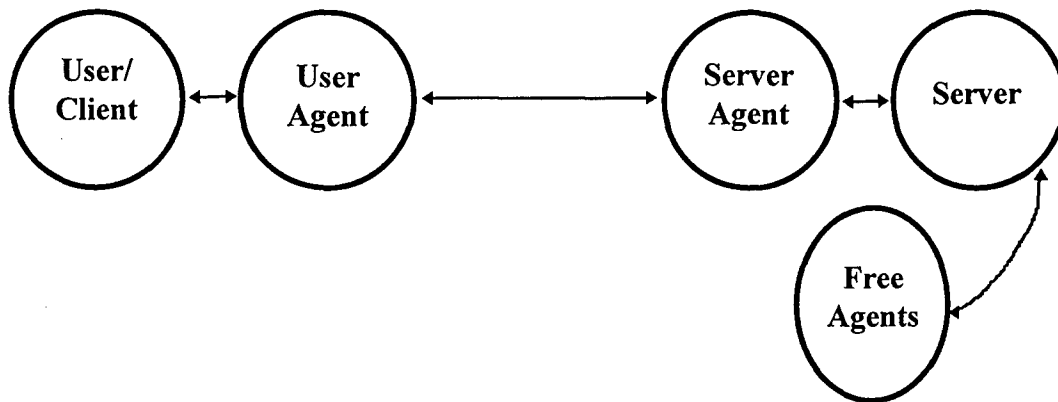


FIGURE 1. Original WAIBA Architecture

The User Agent can see the URLs requested by the user (the agent is a special form of a proxy gateway), as well as the results returned by the server. The agent uses this ability to build a model of the user's interests and maintains a special set of web pages for the user. Similarly, the Server Agent watches the activity of a given server and tracks the kinds of requests it is receiving. It hides the server and can therefore act as a cache for the data in the server, as well as tracking activity.

As part of the original brainstorming sessions, the WAIBA team identified several areas where agents could enhance browsing activities.

- **Network Map (*)** By measuring connectivity, latency, and bandwidth to remote servers, an agent can maintain a database that a user agent consults whenever a document is about to be fetched.
- **Intelligent Gateway (*)** The user agent can try to predict where the user will click next and prefetch these pages. It can also keep a predictive list of sites contacted to direct the network map agent's attention, as well as providing an estimate of the average size of documents transferred from given servers or URL trees.
- **Enhanced Hotlist Services (*)** The agent periodically polls the blackboard and updates a set of WWW pages that replaces the current "hotlist" paradigm. By allowing both agents and users to treat the hotlist as ordinary hypertext data, we enable a wide variety of new automated and interactive services. For example, by pooling the information from a large set of (public and private) agents, a global view of the activity of the net can be tailored to the user's interest.

- **HTML source-to-source transforms (*)** As data is transferred from the server to the client, the user agent can parse the HTML and recognize certain constructs based on user preferences, such as strings that are likely to be place names, ZIP codes, telephone numbers, and e-mail addresses. These strings are then converted to hyperlinks to known value-added services.
- **Web (Re)Structuring tools (*)** By performing depth- or breadth-first walks of subsets of the Web, we can provide:
 - navigation and visual aids
 - different views of the Web, either in part or in whole
 - graphic representations of the shape, size or other attributes of a sub-Web
 - graph structures for sharing and distribution
- Based on data acquired by watching the activity of human, agents, or programs, we can construct:
 - navigation histories (graphical or otherwise)
 - statistical analyses
- **Correlation services** Once a sufficient number of authoritative databases were accessible via the Web, agents could build secondary, derived information sources. These derived sources could combine, sort and correlate the authoritative information in a wide variety of ways. These can be as simple as reconciling data from two different sources or as complex as inducing new relationships between independently discovered data sources.
- **Directed "fishing expeditions"** Given some basic starting information and knowledge about the types of information available on the Web, the user agent "fishes" for related information in particular spots, gathering anything that looks interesting.
- **Interactive Environments** Presently the WWW provides virtual locations where data exists, while Internet Relay Chat (IRC), Multi-User Domains (MUDs) and MOOs (MUDs, Object-Oriented) provide locations where people reside. There is no technological reason why these infrastructures cannot be effectively merged, providing locations where both people and data co-exist.

We identified those items marked with an * as areas that could be implemented either partially or wholly by filtering the HTTP/HTML stream between the browser and destination Web server. The notion of filtering the HTTP protocol and the content returned led us to the notion of extending HTTP proxy servers [Luotonen94]. From this original description, we chose to focus on and expand on the following functions.

Firstly, specific client services could be developed based on the redirection of an HTTP request through a filtering entity that could modify the request in some specified way. Secondly, such services could be

either hosted on the local machine or somewhere on the network. Finally, if run locally, a caching service could be used to support disconnected access.

Our initial experiments in creating filtering-based associates were performed in conjunction with work on a customized browser that we called *Ariadne*. The *Ariadne* browser was built using the Library of Common Code (libwww) from the World Wide Web Consortium, with an HTML display widget developed by Ilog: the user interface portion of the browser was developed using Ilog Views, a C++ based GUI development, and Ilog Talk, a variant of the Lisp programming language. *Ariadne* was notable for the inclusion of two major features: a hierarchical hotlist structure that could be used to directly control the browser, and a machine independent back-channel interface in the form of a TCP/IP connection. The presence of this connection was advertised via an *X-BackChannel* HTTP header: our proxy based services were instrumented to recognize this header and to send information and operations back to the browser via executable LISP statements. This technique was used to build a dynamic browsing history tree that showed the combined browsing history of a group of users.

As our work progressed, our focus changed from software tools that would benefit the individual to tools that would enhance the user's browsing experience by leveraging the activities of their workgroup. The following is from a position paper entitled *Using the World Wide Web to Support Groups*.

We have been building tools to support both individuals and collaborating groups of people through the use of World Wide Web technology. Our approach is to work from scenario to technology to tool. That is, we start by considering a number of realistic scenarios drawn from current work practices. From this we notice places where our technology might be useful, and then construct tools to facilitate the scenario.

Our focus has been on the work environment, and our central interest is in helping groups of people work well together. We take a special interest in situations where the groups (or teams) are formed for specific, short-lived purposes. These groups tend to share several characteristics that make the World Wide Web (WWW or Web) a particularly useful tool:

Groups are transient. That is, they must form quickly, function efficiently, and then disperse. Their membership changes often over their lifetime. "Membership" may not even be a well defined term: some people function as part of the team even after the end of their official connection; others make important contributions without official connection.

Groups are distributed in several senses of the word. They are geographically distributed, with members from all around world. They are temporally distributed because members need not share working hours. That is, each member does team work as part of their regular assignment, but may not do that work at the same time that others are doing team work. Each member folds their team-based work into a larger job or task. And they are organizationally distributed: members are not part of a single unit within an overall organization, but are collected from a wide variety of organizations. For example, a single group within a computer company might contain programmers from an applications organization, hardware designers from an engineering unit, marketing specialists from corporate marketing, customer representatives, graphic artists contracted for a specific project, and top-level management representatives.

Groups are cooperative in nature. This, of course, is not true of all groups, but we have restricted our attention to situations where all group members (eventually) share a set of goals. They organize themselves to divide the work up into units that can be handled by sub-groups (including individual work assignments) and units that require one of several forms of group consensus [Miller95]

We had two major scenarios upon which to draw: our work with BBN, Inc. on the Logistics Anchor Desk effort, and the second with ISX Corporation in the exploration of how Web technologies might aid the effort of DARPA program managers.

These were the beginnings of our Mediated Access technology (also known as the Mediator), which drew on two notions extracted from the above scenarios.

3.1 Mediated Access

First is the notion of collaborative activity as a means to achieve a shared goal and the representation of that goal as an HTML document where individual hyperlinks represent subtasks. The responsibility for completing that subtask could be assigned to different team members; adding content to the object referenced by that hyperlink thus modeled on-going work on that subtask, while the notion of "sealing" a document represented the completion of that subtask. Secondly, the notion of Web as workplace was intrinsic to both scenarios: the Web created by the hyperlinking of artifacts representing both the product and the process of the task. And from this rose the notion of a *Group Server*: an information store that becomes the distributed team's virtual workplace. This group server would be comprised of an HTTP server extended in various ways to provide a set of *group services* that would aid the team in carrying out their day to day tasks.

3.2 Group Annotation

Our next major addition to the notion of group services was a general annotation system that could be used to review and comment on those documents managed by the Mediator. We used the ComMentor system from Stanford [Röscheisen95] as our base, but, in keeping with our philosophy of implementing technologies that would be useful with any Web client or server, we moved certain functions that had been present in a modified version of the NCSA Mosaic browser into a Strand-based service that could be accessed by individual group members. The resulting system provided the ability to annotate any document on the Web, and implemented the user interface to the service as a series of "special" URLs that were interpreted as directives to the service itself. These directives (the results of filling out an HTML form) represented the ability to manipulate and control the value-added server itself using the same mechanism as that used between the clients and the information server (HTML).

3.3 Browsing Associates

Although the above services addressed issues of collaboration within the workgroup, we did not have a complete end-to-end solution: we needed to tie together the notion of group servers and group-oriented services with the user's desktop. Our mechanism for establishing a presence on the user's desktop was via the notion of the *browsing associates*, "relatively small and simple applications that are not coupled to a particular HTTP stream and can asynchronously access the WWW on the user's behalf" [Meeks96]. Note that this definition of the browsing associate varies from our initial definition: here the term associate is used to refer to what had been previously labeled an *agent*.

The intent of the browsing associate was to address the following deficiencies in locating and managing information:

- **Navigating:** becoming lost or disoriented when navigating in a large hypertext space

- **Finding:** Web clients support browsing of a large information space, and one can access Web-based search engines (either for content accessible via the Web, or for more specialized information), but most are not well suited to find a given object that requires an *organized* set of information.
- **Adding:** very little support exists to add value (such as annotations) to existing Web structures.
- **Receiving:** passively receiving timely information is poorly supported

Associates differ from transducers (filtering agents) in that they are not tied to the browsers HTTP connection. That is, failure in an associate will not incapacitate the browser: failure in a transducer has the same impact as failure in a server (activity is suspended until the browser is re-homed).

Browsing associates were a means to add functionality to the browser without requiring a special (and insecure) form of communication with the browser or by actually modifying the browser source code itself. The need for the browser backchannel was lessened given the presence of APIs for communicating with the browser[Cusack95], APIs that can be subsumed under the label *Common Client Interface* (CCI), the name originally provided by the NCSA Mosaic team. These APIs provided several powerful capabilities, including the ability to register an external program as the interpreter of a given MIME type or URL, and to request that an external program be notified whenever the user requested the retrieval of a URL via clicking on a hyperlink.

3.4 Notification

The presence of a number of different associates, either instantiated as browsing associates or as proxy-based services, anticipated the need for a notification infrastructure that would tie the server-based functionality together with the desktop systems, and the desktop systems with each other (we expressly allowed for direct peer-to-peer communication). We chose to implement this notification system using the Zephyr notification system that had been developed as part of MIT Project Athena and an implementation of the Linda programming language that used a TCP/IP-based server to represent the Linda notion of a shared tuple-space [Schoenfeldigner95].

We chose to exploit several capabilities in the Zephyr notification system

- Zephyr's notification classification scheme on class, instance, and recipient was a good match with our group-oriented work. Specifically, notifications could be targeted to a single user, or to a (time-varying) list of recipients (a recipient field code as "*" implies all users currently subscribed to the a specific <class, instance>). In addition, a Zephyr notice consists of a 4-tuple of <class, instance, recipient, body>: this matched well with the underlying notion of a tuple-space as presented via the Linda language primitives, and made it almost trivial to store notifications in Linda tuple-space and retrieve them via the pattern-matching primitives.
- Zephyr's notion of location-independence meant that a user could receive messages regardless of what machine they were using (in the office, working at home, logged in remotely): in fact, Zephyr supports the notion of a user having multiple points-of-presence on the network and will deliver the message to all instances of the recipient.

- The notion of access based on a subscription-list ensured that only those users interested in a sets of messages would receive them; the programmability of the existing UNIX Zephyr implementation meant that the user could also control how a message would be viewed. In addition, the ability to subscribe and unsubscribe at will meant that the certain interactions between a client and the blackboard were trivial: for example, a client could simply ask the blackboard to record messages of class *X*, instance *Y*: the blackboard could easily subscribe to these notices and unsubscribe at the behest of the user.

We chose to implement our blackboard facility on top of the Linda server. As stated previously, the match between the Linda notion of tuple-space (a content-addressable datastore consisting of an arbitrary collection of N-tuples) and the Zephyr definition of a notice as a 4-tuple (class, instance, recipient, message-body) meant that Zephyr notices easily could be stored into the tuple-space by utilizing the existing <class, instance, recipient> fields, adding a time-stamp, and encoding the data field in standardized way (we ultimately chose the URL encoding scheme used by the HTTP post primitives). In addition, the basic Linda primitives supported pattern-matching against the tuple-space: this made it simple to retrieve all instances of a given <class, instance> pair.

In order to achieve the goal of location independence for the blackboard, we developed a front-end associate for the Linda server. This server is responsible for listening for notifications of a particular type: when such a notice is received, the agent responds to the sender with the TCP/IP address of the Linda server. The client then communicates directly with the server via a higher-level blackboard specific API that masks the use of the Linda language primitives.

The associate also provides another service for a client: the client can request the agent to subscribe to and record notifications of different types on behalf of the client. The client can, at a later time, inform the agent to cease collecting these notifications, and then extract these notifications from the blackboard.

It is interesting to note how the final design of the system has adhered fairly closely to the original architecture proposed at the beginning of the project. The idea of associates and transducers have proven to be a useful distinction in locating system functionality. The biggest difference is in a shift from the more aggressive vision of "agents everywhere" to a more constrained notion of an associate-based services that exist both on the clients desktop and on the targeted server that cooperate via a notification system. The next section will describe several instances of associates and group services.

4.0 Accomplishments

We have developed several novel software systems in the course of our project work. The following sections describe our most successful software artifacts generated as part of the WAIBA project. Most of the software described below is available as part of our WebWare software release, available via the Open Software Foundation's home page at: <http://www.opengroup.org/>

4.1 The Strand Toolkit

As a way of learning about implementing proxy based services, we developed a toolkit for creating these filtering services. This toolkit is called the Strand (Stream TRANsDucer), since we borrowed the notion

of stream transforms from [Abelson86]). The toolkit consists of a shell application that is intended to be combined with a user written application that will actually perform the transformation of the HTTP request and/or response stream. The application writer could chose to observe and modify the request and response stream or both, and could choose from three implementation modes

- transform the request and response stream via separate modules; no communication between them.
- transform the request and response stream in a single process such that state could be maintained for each HTTP transaction
- transform the request and response stream such that state could be maintained across connections (this is known as the co-process model)

The Strand toolkit has been the basis for most of our group services. The toolkit itself consists of the Strand "shell" that provides an environment in which the application can run and a low level API library for interfacing with the shell.

Based on our usage of the toolkit, we have extended the basic toolkit in several ways. For example, we have added the capability for the application code to create the downstream connection itself (normally this was handled by the Strand shell) or to actually resolve the URL itself and contact the destination server directly. In order to improve the performance of Strand programs written using various scripting languages, we have extended the Strand shell to support embedded interpreters (in our version 2 release, only Perl is supported). Code is interpreted once, and there is a concomitant saving in execution time, since a new process doesn't have to be created to run the interpreter for each new request. These enhancements also allow a developer to build the application using interactive development tools, and then embed the resulting script into the Strand for increased performance. The toolkit currently runs on the Solaris, HP-UX, and Linux operating systems.

4.1.1 Indexer Strand

We developed our initial Strand-based service to allow a group to utilize a shared browsing history; our second implementation was a full-text indexing application. The Indexer ran as a proxy service: each document requested would be indexed and the results stored in a local database. An HTML query form was provided that permitted the user to query the results of these searches using a simple keyword query interface. As implemented, this service was focused on supplying these services to a workgroup; more recently, this notion has been adopted by various commercial products that work with COTS browsers and the user's document cache, thus providing an individual search capability based on the user's own browsing history.

Although we chose not to follow on with this prototype, the development of this application was significant in that it demonstrated the following:

- A Strand based application could be used by multiple users without undue delay or other performance impact (this is in part because of the variation in response time seen in the normal course of browsing various web sites)

- Generation of document meta-information on-the-fly (the full-text index) and use of a database to store document meta-information (in this case, UNIX DBM files) was feasible when using Strand-based filters.
- Communications between a Web client and a service could consist of a forms-based interface that utilizes URLs to address the proxy service itself. In effect, the proxy service reserves a portion of the namespace for its own use: the difference between this and a standard Web server is that the URL in question could not be used directly by a Web client, since the URL encoding for the proxy service may not be resolvable by a non-proxied client.

4.2 Group Annotation

The original ComMentor architecture combined the merging of documents and the management of a users' group membership and their access to various annotation sets in the browser itself. We initially factored these capabilities into a Strand-based Annotation Service that contacted the Annotation server on behalf of the user, merged the resulting annotations into the returned document, and returned that modified document to the user. In our initial implementation, we continued to use a simplified implementation of the Annotation server as developed at Stanford.

As part of our ongoing work, we re-implemented the system to provide a cleaner architectural specification as suggested in [Schickler96], as well as to better support different modes of filtering. The new Annotation service is designed to run either as a client-side (an HTTP proxy service) or as a server-side filter (a proxy service acting as a front-end to the HTTP server). The service accesses user profile information on a specified server: this is usually the same server that runs the annotation service. The Annotation Server that stores and manages the annotations themselves has been re-implemented using the mSQL database system, and management of user access to annotation sets has become part of the administration of the Annotation Server itself.

The Group Annotation service provides a very flexible means to provide commentary on Web documents: client side applications can theoretically annotate any document on the Web, while server-side Annotation services provide commentary only on documents served by a particular server. User's can choose to view annotations only from particular categories (sets), and, for those annotations, chose to filter them in various ways. The current user interface utilizes the frame technology pioneered by Netscape Communications Corporation in their Netscape Navigator browser: annotated documents are displayed in one frame, user control in another, and the annotations themselves are displayed in a third.

4.3 Mediated Access

The Mediator software was developed to provide functionality to support teams in creation, publishing, and maintenance of collaborative webs from anywhere on the Internet. This system provides the following benefits to end users:

- Supports asynchronous distributed authoring
- Provides authorization policy for creating, updating and deleting documents

- Provides centralization of project information
- Notifies selected team members via email of new and updated documents

The Mediator software has undergone significant changes since our initial implementation. Information concerning document classification and other meta-information is now stored in an mSQL database; the document index is now generated dynamically using special header and footer files to add project specific information. Information about project members has been factored into two separate data stores (see the User Profile service description below). Finally, the Mediator not only sends email notification to users who are on the notification list, but also generates a notification message indicating which document has been changed: suitably instrumented applications can receive these notifications can trigger application-specific processing based on the type of event (document creation, modification, "sealing" [completion], or deletion).

4.4 User Profile Service

The User Profile Service was developed to answer a shared need for both the Mediator and the Group Annotation services. Specifically, when recording an annotation or when generating lists of team members, information about individual team members is included as part of the annotation. This information includes the user's e-mail address, project specific home page, the user's actual home page, and the user's picture. This service had previously been duplicated as part of both the Mediator and the Group Annotation services; in the re-design effort undertaken during the second year of the project, this information was factored out of each application and instantiated as a separate service that is used by both services. In particular, this service is extended by the Group Annotation service that uses this service as a means to store the user's annotation set view (which sets the user is interested in seeing) and other annotation filtering information.

4.5 Browsing Associates

4.5.1 HistoryGraph

The HistoryGraph visualizer grew out of the browsing history tree generated as part of the early Ariadne experiment in sharing group browsing histories. Originally developed in ILOG Talk and Views, it was re-written using the Tcl/Tk programming language and a CCI interface such that the HistoryGraph could request notification from the browser whenever the user accessed a URL by clicking on an embedded hyperlink. The HistoryGraph creates a tree-structured representation of the user's browsing history: the tree may be subsequently edited using drag-and-drop. The tree can also be used for navigating: clicking on a node cause the URL associated with that node to be displayed in the user's browser.

Named sets of URLs can be created either by selection or by specifying a regular expression to search either the URLs or the titles of the URLs retrieved: these sets can be saved as HTML files for later perusal. The entire state of the visualizer can also be saved and reloaded at a later time. Named sets of URLs can be forwarded as input to other browsing associates: in this regard, the HistoryGraph provides a convenient means to interface with other applications as well.

4.5.2 *WhatsNew*

WhatsNew represents our efforts to develop a browsing associate to explore how changes in a weblet of interest might be communicated to the user. We factored the notion of change into three separate functions:

- How to specify those *pages* of interest
- How to specify *what changes* were of interest, and
- How to specify how *frequently* to check for changes

In addition, we wanted to explore alternatives for notifying the user when changes of interest had occurred. Our initial implementation implemented the following choices. Pages of interest were denoted by a file containing a set of HTML formatted anchor (<A>) elements; each reference was a candidate for examination. Using HTML formatted documents permitted using existing browser bookmark files or HTML documents. Changes to Web documents were based simply on modification date; one interesting variation was to allow the sense of the test to be inverted, such that all files that had not changed since a given date would be identified. Time could be specified either as an absolute date and time, or as a relative time ("3 days ago"; "1 hour ago"). The frequency of iterations could be specified as an interval (e.g. "5 minutes"): the default was "never", indicating a single iteration.

The user could choose four different means of notification. WhatsNew automatically generates an HTML page containing a list of the URLs that have changed: documents that are unreachable or encounter an error are treated as changed (under the assumption that the user is interested in such information for purposes of modifying their lists of pages of interest). The user can choose to receive no notification at all (in which case they can access the generated HTML file at their leisure), to have the page displayed automatically in their browser, or to be notified via the WebWare notification mechanism (this usually results in a pop-up notification indicating which URLs have changed)

Initially, the WhatsNew associate ran as a completely separate program. During our second project year, we made two significant changes to the WhatsNew program. First, WhatsNew was instrumented to accept a set of URLs to examine from an external program (in this case, the HistoryGraph visualizer). Secondly, WhatsNew was further instrumented to return the set of changed URLs to the HistoryGraph for inclusion in the visualization hierarchy. This changed the WhatsNew application from a (rather trivial) Web-aware application to a more interesting and powerful means of augmenting an awareness strategy for the desktop.

We do not believe that we have exhausted the possibilities of managing change with Weblets. For example, the evaluation of last-modified date could also be augmented by comparison of document content to a set of keywords or to a previous revision of that document. Only if significant changes were found would a document be marked as changed.

4.5.3 *Linktree*

The Linktree associate was designed to supply the functionality of World Wide Web *robot* [Koster94]: namely, a program that, when given a starting URL, would explore the hypertext graph starting from

that node. The search is limited by specifying both the depth of the resulting tree and by an expression that each URL would have to match (for example, indicating that all URLs must be found on the same machine).

The Linktree is hooked into the client's browser such that it is informed of new page being displayed. The search could be invoked automatically, or only when instructed by the user. The Linktree generates an HTML page representing the hierarchy of hyperlinks starting at the given page: if so requested, the Linktree will automatically cause this page to be redisplayed in the browser.

Future plans call for a re-integration of the Linktree with the HistoryGraph visualizer. The user sets operational parameters for the Linktree and then sends the node in question to the Linktree which will generate the tree of URLs stemming from that node and return the resulting data structure to the HistoryGraph for display as a separate subtree. From this point, the user can then use the display to navigate the tree or simply prune the resulting subtree if the information isn't interesting.

4.6 Blackboard Assistance

We have discussed our notification service in a previous section of this paper; notification is the key technology that links our server-side group services with the browsing associates on the user's desktop. The blackboard service presently is the major application to utilize the notification service.

Our goal in building a blackboard service was to provide a common mechanism for applications to share semi-structured information in a networked environment. In particular, we wanted our browsing associates, Strand-based services, and our CGI-based services to utilize the same mechanism to store information that was neither as ephemeral as Zephyr notices nor as persistent as, say, the User Profile information. Our blackboard service was implemented using a shared TCP/IP based implementation of the Linda programming language [Schoenfeldinger95].

The design for a blackboard system based on Zephyr and Linda consists of three parts:

1. a Zephyr-enabled Tcl-based front-end server that connects directly to a companion Linda server and implements the blackboard protocol and library,
2. a protocol that specifies the format of the Zephyr notices exchanged between the server and client to implement the basic blackboard functions,
3. a Tcl library of functions that provide the abstract interface to the blackboard.

In particular, we believe the implementation of the blackboard service provides the following improvements over using Linda alone or some other database-like mechanism.

1. **Location independence:** the service does not have to live at some well-known location that applications need to be configured.
2. **Simple Tcl interface:** the Tcl blackboard library provides a simplified interface to the underlying Linda server, access to a simple application naming service, and (eventually) support for more

powerful abstractions; all these functions can be taken advantage of by any Tcl application that can also use the Tcl Zephyr package.

3. **Name service:** by providing a central registry for generating unique but meaningful application names, the blackboard can provide support for services like the ZSend protocol (a Zephyr-based replacement for the Tcl/Tk *send* primitive) and eventually for a service brokering system.
4. **Application proxy:** the blackboard can provide a "stand-in" for an application by capturing information for that application while it is off-line. For example, HistoryGraph could use the blackboard to save Zephyr notices from the Mediator while it is off-line and then retrieve these notices when it comes back on-line.

We have developed two scenarios utilizing the blackboard and notification service. The first is an implementation of a "guided tour": one individual acts as a "tour leader" to provide a "guided tour" of a particular hypertext space; tour members "follow along" via their Web browsers. The tour facility is provided by an associate that receives messages from the leader and then positions the browser to the correct node. Should another tour member arrive late for the tour, they can "catch up" by accessing the blackboard and retrieving all pages up to and including the current page in the order accessed (thus enables sensible usage of the browser's *Back* button).

Our second scenario integrates the blackboard with the Mediator and the HistoryGraph visualizer. In this scenario, notices from the Mediator are incorporated into the HistoryGraph by modifying the icon representing a given document, thus providing a visual cue of changes in the mediated namespace. When restarted, the HistoryGraph application consults the blackboard for change notifications: additions and deletions are automatically incorporated into the visualization tree.

4.7 Summary

In sum, then, we have developed a suite of powerful, interlinked services that together provide a platform for a powerful workgroup environment aimed at improving the efficiency and communication of a distributed project team. The combination of the Mediator, Group Annotation services, and the Notification service provides a flexible means for team members to create shared documents and commentary on those documents, while being assured that they are notified whenever changes are made to this shared workspace. The use of the HistoryGraph visualizer, in combination with the WhatsNew and Linktree associate, addresses the issues of organizing browsing information as well as integrating the mechanisms of notification and browser control on the user's desktop.

5.0 Results/Lessons Learned

Our efforts of the past two years have resulted in two mature technologies: Mediation and Group Annotation. Both of these technologies are being integrated into the Web Integrated with Security Enhancements (WISE)[OSFRI96b] effort that incorporates groupware technologies with the OSF/RI's DCE-Web [Lewontin96] technology. DCE-Web implements the HTTP protocol via the DCE RPC interface to a DCE Web server: this interaction leverages user authentication and privacy on the client side and leverages the use of DCE authorization services on the server side in the form of DCE ACLs for

individual objects in the namespace. These technologies are expected to be deployed both at Sweden Post and Hewlett-Packard.

5.1 Mediated Access

The notion of "mediated access", where the server is privy to all user interaction with the document store (both reading and writing), is extremely powerful. The canonical Web publishing model assumes that the server just returns (existing) content and does not address how that content is made available or organized on that server. The mediated access model assumes that the server can add value to the content stored on that server: a server that mediates document creation can provide on-the-fly keyword indexing and immediate notification based on stored queries that would match those keywords. Additionally, the server can add relationships via HTML LINK elements and incorporate other document meta-information via HTML META elements. These elements can then be interpreted by suitably instrumented browsers or desktop associates to provide a more highly connected structure not only in terms of navigational aids but also in terms of automated links to other relevant documents. Content can, in short, be tailored to the interests and goals of the individual based on the role that the individual plays on a given team.

By tracking the state of the document, the state change itself can be used to trigger other activities. For example, the sealing of a document (no further modifications are permitted) could trigger the movement of that document into a "read-only" area of the server or could cause access rights to be given to additional individuals.

5.1.1 Proxy-Based Services

The notion of mediation also appears in other proxy-based technologies. In the canonical instance, an HTTP proxy server forwards the request to the actual server and may also act as a cache. Proxy-based services extend this notion by modifying the HTTP protocol itself or the content that is returned by the request.

The notion of proxy-based services that provide value added transforms on information content has become widely successful. Several examples of proxy-based services were reported at the 4th International World Wide Web conference: among them were services in support of vision-impaired users, a community based content-rating and access service, and an implementation of the Millicent payment protocol that used a proxy-filter to add the payment information to communications from an uninstrumented browser [WWW4].

5.2 Browsing Assistance

The state-of-the-art in Web browser implementation has increased at an almost alarming rate over the two years of the project. We are now in a period where various vendors are competing by adding increasing numbers of features to the basic Web browser with the resulting increase in complexity and desktop footprint.

There are two different approaches currently being espoused in the marketplace: the "desktop as browser" metaphor (Microsoft's Internet Explorer and the Nashville release of the Windows/NT

operating system) where browsing functionality is incorporated into traditional desktop tools such as the canonical file browser, and the "browser as desktop" metaphor, embodied by Netscape Navigator. In this instance, the browser becomes the central application on the user's desktop, supporting all user activity directed toward internet information services (this is the monolithic approach). We espouse a third alternative that we label the component model. In this view, the browser is broken into component parts: an HTML display engine (presentation) and an HTTP protocol engine. Thus, the browser exports a set of services to the desktop; other Internet technologies, such as file transfer (FTP) or e-mail (SMTP) can be integrated as a set of tools that may support a protocol engine but may utilize the same display engine for displaying similar content type. An object of mime type *text/plain*, for example, could be displayed by the browser regardless of whether it was obtained by the Hypertext, File, or Simple Mail protocol. A commercial example of this kind of "outside-in" approach is Softquad's Panorama, an SGML viewer and browser. Panorama registers with the user's browser as an external viewer for SGML documents (mime type *application/x-sgml*), but then uses the browser to resolve URLs contained within the document (the content is handed back to Panorama). This is an attractive scenario, and an example of how a Web browser can be usefully integrated into the desktop environment as both HTML display engine and HTTP protocol engine.

The hotlist (bookmark) capability of the *Ariadne* browser has been transformed into the HistoryGraph application. This application demonstrates how alternative interfaces to information resources can utilize a hypermedia interface but maintain an arm's length relationship with existing browsing tools. Our LinkTree and WhatsNew associates are further examples that build not only on the display capabilities of the visualizer, but on the existence of browser APIs as well. Thus, we believe it is important to maintain browser independence. We continue to need general purpose browser APIs to support the component based style of interaction.

The HistoryGraph visualizer had its roots in the hierarchical hotlists developed as part of the *Ariadne* browser work which then was spun off as a separate browsing associate. The added ability to manipulate and save the results of a browsing session, combined with the ability to save and create arbitrary sets of nodes, generates a powerful utility. This leads us to the notion of the WAVE (Weblet Activity Visualization environment), where the visualization function can be extrapolated from the application and developed as a separate component (the WADE - WAVElet Display Engine).

The problem with low-hanging fruit is that others will be quick to harvest it as well. We have ceased focusing on these technologies that have become part of value-added features of other commercial browsers. Hierarchical hotlists are almost universal, as is the ability to determine changes in pages of interest. WhatsNew is useful now as an example of a browsing associate that can be invoked via particular application protocol, but it is not worth pursuing unless there are more interesting applications of detecting and reporting on change within specified weblets of interest.

Mediated technologies offer a wide range of possibilities for providing enhanced services within the context of the World Wide Web. The integration of client and server side services with a location-independent notification service provides a very rich environment for group collaboration.

6.0 Influences

Our research endeavors don't exist in a vacuum: we strongly believe in building on the best efforts of others and of returning our results to the community. In this light, the WAIBA project has benefited since its inception from the work of members of the Perl, Tcl/Tk, Apache, and WWW communities, and we have contributed time and code back to these communities as part of our WAIBA efforts.

6.1 Who influenced us?

Our initial influence regarding proxy-based servers was most assuredly the paper on *World Wide Web Proxies* [Luotonen94]. We also benefited from discussions with Henrik Frystk Nielson at the World-Wide Web Consortium on proxy-based services and on the facilities available in libwww.

The work on the ComMentor system at Stanford [Röscheisen95] was a major influence on our thinking of how one might add commentary and other meta-information to Web pages not owned by a given workgroup.

The notion of Web robots, especially the tkWWW robot [Spetka94], influenced our design and work on the Linktree associate.

Our choice of Linda as an implementation for the blackboard was strongly influenced by Gutfreund's work on WWWinda [Gutfreund94], and by Schoenfeldinger's work on building transaction-based schemes using Linda [Schoenfeldinger95].

6.1.1 Who have we influenced?

Researchers at INRIA [Lang96] have credited us with systematizing the approach of filtering the HTTP streams via a collection of services.

The Jigsaw effort at W3C credits our work on the Strand technology as providing the inspiration for the use of filters in the Jigsaw server [Baird-Smith96].

We have significantly influenced work on the Logistics Anchor Desk effort, both for our Mediation work and our Group Annotation efforts. The ability to annotate textual representations of a logistic plan had lead to efforts to model and build a system that will support annotations of graphical images [Davidson96]. This work will carry on into the Advanced Logistics Planning (ALP) effort.

We have contributed source code and ideas to the W3C consortium for the Windows/NT version of their library of common code (libwww).

We have contributed HTTPd administration modules to the Perl community and to the Apache server development effort, specifically in the areas of embedding Perl interpreters into the Apache server.

We have demonstrated our WebWare software at the fourth and fifth International World Wide Web Conferences, and at the DCE Developers Conference help in August, 1996. We have been pleased to

demonstrate our software to representatives from Hewlett-Packard, Digital, and the U.S. Department of Transportation.

We have commercial contracts for our software from Dascom, Utopia, and Sun Microsystems.

Finally, we have incorporated WebWare functionality into the WISE cell; this development is also planned to be fielded at Sweden Post and at Hewlett-Packard.

7.0 Publications/Conferences

We have been fortunate in being able to present the results of our work at several conferences and workshops. The following lists our most significant contributions during the past year.

7.1 Conferences

- *Pan-Browser Support for Annotations and Other Meta-Information on the World Wide Web*, a paper describing the Open Software Foundation RI's Strand-based approach to supporting group annotations; presented at the Fifth International World Wide Web Conference, May 1996.
- *Transducers and Associates: Circumventing Limitations of the World Wide Web*, a paper on the OSF RI's Strand and Browsing Associate technologies presented at the etaCOM (Emerging Technologies and Applications in Communications) '96 conference, Portland, Oregon, May 1996.
- *Application-Specific Proxy Servers as HTTP Stream Transducers*, a presentation at the Fourth International World Wide Web Conference, December, 1995.

7.2 Workshops

- *Modular, Composable, High-level Stream Transducers*, a position paper by the OSF/RI WAIBA team for the ACM SIGCOMM95 Workshop on Middleware, August, 1995.
- *An Architecture for Supporting Quasi-agent Entities in the WWW*, a paper published in the proceedings of the Intelligent Agents Workshop of the Conference on Information and Knowledge Management, December 1995.

In addition, we also attended the World Wide Web Consortium (W3C) Workshop on WWW and Collaboration, held in September, 1995.

8.0 Next Steps

Based on our successes and our "lessons learned" from the past two years, we believe that we should move forward in the following areas.

8.1 Web-based Group Collaboration Services

Our notion of WWW-based Group Collaboration services encompasses our use of the Mediator, Group Annotation, and Notification technologies, and their use in an enterprise computing environment. Note that these services can and should be integrated with other facilities: e.g., fault-tolerant servers, real-time computing, etc., just as World Wide Web services can benefit from technologies such as distributed file systems, etc. The area of CSCW (Groupware) that we have adopted is still an emerging technology: this is especially true now that Web based technologies are being applied to the problem of collaborative work.

Our focus is thus on the intersection of enterprise webs and group webs, specifically in the area where solutions that are required for enterprise computing (distributed naming, authentication and authorization) are applied to the "friendlier" workgroup environment.

8.1.1 Mediator

In this environment, the notion of Mediated access needs to be extended. Our earlier model of mediated access for writing and HTTP access for reading no longer applies in the enterprise context, since document visibility may need to be managed within the enterprise. We therefore propose that the Mediated Access technology must function more like a DCE "junction server" or an extended CGI interface, where the actual task of name resolution and object identification must be taken on by the mediator itself. The notion of mediated read and write access actually provides a more general environment. In the current environment, the Mediator is responsible for generating the actual URL returned (this is consistent with the HTTP POST method); this makes it difficult for user's to add documents lower down in the hierarchy. By causing all access to be mediated, then Mediator can now perform URL pathname resolution to any depth, returning appropriate content and error messages. The Mediator needs better search capabilities. Whether this should be supplied as part of the HTTP server environment or by the Mediator on a per-project basis is unclear.

The Mediator user interface could be enhanced to more clearly represent project structure and navigational aids to the user. For example, the document index page could show each category as a collapsible list: clicking on the list would show the documents contained in each category. Additional user interface enhancements should include how much information is presented and in what order (e.g. should documents be returned by author, date, etc.).

8.1.2 Group Annotation

Group Annotation needs a more powerful database engine than mSQL: we had to perturb our design based on performance problems with joining multiple tables. At this time, we see no need for an object database; we believe that a relational database is sufficient for our needs.

We need to incorporate results of the Open Software Foundation/RI's related work on visual annotations with textual annotation work. Furthermore, we need to extend the notion of annotation for other application objects using the "wrapping" technique developed for visual annotations. The initial visual annotation technology used a Strand-based service to modify each document. All references to image data contained with the hypertext page were modified to become an *applet* tag that presented that image

as an argument to the visual annotation applet. We propose an extension to this technique as follows: all references to objects of application or image type are modified by a filtering Strand in the following manner: an HTML page is generated that includes the object as a hypertext link, and all related annotations are included on that page (this processing model is similar to the existing textual annotation interface using HTML frames to present the annotations, the control interface elements, and the actual text of the document).

We have recently implemented a prototype Java-based application for textual annotations: we will be evaluating this prototype further to determine if the applet implementation meets on-going needs, or if an independent browsing associate provides a better basis for annotation work.

The current Strand-based application of the Group Annotation service needs to be re-evaluated. The current implementation is single-threaded: we need to investigate the potential performance gains if we enable multi-threading of requests through the service.

Finally, we need to evaluate the above mechanisms in light of on-going efforts in the Web community. The ComMentor authors report that the PICS effort (Platform for Internet Content Support) [Miller96] has incorporated elements of the ComMentor architecture into their efforts at providing content ratings: we must continue to be responsive and aware of on-going efforts within this community.

8.1.3 Notification

The issue of notification when content changes is still on-going within the Web community and is a critical element of any groupware strategy.

There are effectively two different methods for determining which users need to be notified when changes occur in a mediated portion of a web:

- Notify based on changes to individual objects. A user "subscribes" to a given object (which may be a single document, a search query, or a collection of objects). The user is then notified whenever the object changes. This can either be a change to the object itself (in the case of a document) or to the implied internal state of the object (in the case of document collections or of queries).
- Notify based on subscription list. A user chooses to subscribe to a set of known notification types (e.g. *class mediator@opengroup.org*, instance *, recipient *). The user received all notices that match this specification: it is up to the user to determine if they are interested in a *particular* change.

The object-oriented notification has the benefit of only notifying a user when that particular object changes; however, the user can miss changes to other (related) documents. The subscription-list approach puts the onus on the end-user for filtering the potentially large number of notifications: this might well-be a job for a browsing associate.

We are currently thinking about a general-purpose Subscription Manager that would include the functionality currently supported in *zctl* (Zephyr subscription list control application) and *zwgc* (Zephyr windowgram client - responsible for displaying notifications). Users would be able to graphically view and manipulate their subscription lists, and associate actions with each element. Each action could be specified via a simple scripting language based on Tcl or Expect.

8.1.4 Blackboard

The blackboard service serves two useful purposes: as a message repository, and as a rendezvous service for cooperating browsing associates. We have already prepared a demonstration of how an application can invoke the blackboard as an agent to record notifications of interest to the application, and then download these notifications once the application is re-started. Since the blackboard can always be located via its Zephyr interface, cooperating agents (either working together on a single desktop, or cooperating across user's systems) can register their presence as well as the capabilities they provide. In addition, associates can chose to synchronize behavior via the underlying Linda primitives: an `in()` operation will block until a tuple that matches that query is available in the tuple space. Associates thus have both a synchronous and asynchronous method for process coordination.

8.2 Desktop services

We are already contemplating the transition of the HistoryGraph technology into new areas; this new initiative will be re-christened WAVE (Weblet Activity Visualization Environment). This environment will consist of two components: the WADE (WAVE Display Engine) and various Wavelets, small applications that use the services of the WADE to both display and manipulate various weblets and as a means for the user to specify sets of URLs for various operations. The existing WhatsNew and Linktree will be updated to this new interface; and the HistoryGraph application will be re-written under this new model, and re-christened the HistoryViewer.

8.3 Influence of Other technologies

As this project has demonstrated, time does not stand still: the phrase "Web years" frequently is used to describe the time necessary to generate new generations of software - where once this would have taken a year or more, new versions of software now appear every few months.

8.3.1 JAVA (Javascript, LiveConnect)

The Java programming language has captured the imagination of the Internet community, However, one might well ask when the use of this technology is this truly mandated. Our position is that the use of Java applets is warranted when one needs to maintain client-side state when accessing the resources offered by a given site. There are no other good mechanisms: Netscape cookies require explicit cooperation between a client and a server; using hidden fields in forms begs for failure when internal limits are (silently) exceeded. The Java applet definition, however, provides a means for an applet to connect with a secondary service on the Internet host where the applet was downloaded, and enter into a stateful dialog with that server as long as the Web page on which that applet was loaded is still in view.

Alternatives to Java are browser specific plug-in modules or the Active-X technologies from Microsoft. The disadvantage of these two technologies are that they are compiled for specific platforms: this platform must be taken into account when versions are downloaded, and that they offer no built-in security guarantees. This will undoubtedly change in the near future; at present, Java has the slight edge in portability, although evidence is beginning to surface that the Abstract Windowing Toolkit (AWT) is not well-specified enough to guarantee consistent behavior across different platforms.

8.3.2 Server plug-in modules

Many commercial servers are now offering the ability to incorporate server-side processing into the server itself: this can be done either when the server is initially compiled, when the server is loaded into memory for execution, or when the functionality is invoked explicitly. Servers that currently offer this functionality are the Netscape family of servers, the Apache server, Microsoft Information Server, and the OSF/RI Wand server. This functionality takes two forms: either that of user specified processing that could have been implemented via the Common Gateway Interface (CGI), or by adding functionality to the server, which emulates the ability of some servers to process specific directives called "server side includes".

These technologies offer improved performance characteristics beyond that possible with either the CGI interface, or by interposing a filtering Strand before the actual HTTP server. Both the Mediator and the Group Annotation service are candidates for this enhancement: an additional advantage to this arrangement is that issues of authorization and authentication can be carried out by the server code (as is now the case for CGI), but not for server-side filtering services.

8.3.3 World-Wide Web standards

As this paper is being written, the HTTP 1.1 specification is being prepared for submission to the IETF as a proposed standard. This specification contains definitions for the PUT and DELETE methods for the HTTP protocol which will imply a more general publishing model for the Web than is currently extant. In addition, the HTTP 1.1 protocol contains other directives that apply specifically to proxy-servers: our technology must be kept up to date with evolving standards. We are currently participating in a working group on Distributed Authoring and Versioning: changes proposed by this effort will undoubtedly influence our directions and our results.

9.0 References

This section includes references mentioned throughout the text, as well as seminal writings that have influenced our views of collaborative efforts via the WWW.

[Abelson86] H. Abelson. et al., *Structure and Interpretation of Computer Programs*, MIT Press, Cambridge, MA 1986.

[Baird-Smith96] *Jigsaw: An Object-Oriented Server*,
<http://www.w3.org/pub/WWW/Jigsaw/User/Introduction/wp.html>

[Cusack95] Software Development Interface,
<http://www.spyglass.com:4040/newtechnology/integration/iapi.htm>

[Davidson96] M. Davidson, *Technical Plan for Graphical Annotation*,
<http://www.opengroup.org/RI/www/waiba/tasks/lad/ALP/plan-2.0/>

[Gutfreund94] Y. Gutfreund, et. al., *WWWinda: An Orchestration Service for WWW Browsers and Accessories*, The Second International WWW Conference '94: Mosaic and the Web, Advance Proceedings, October, 1994.

[Koster94] M. Koster, *Robots in the Web: threat or treat?*,
<http://info.webcrawler.com/mak/projects/robots/threat-or-treat.html>

[Lang96] B. Lang and F. Rouaix, *The V6 Engine*,
<http://pauillac.inria.fr/~lang/Papers/v6/>

[Lingnau95] A. Lingnau, et. al., *An HTTP-Based Infrastructure for Mobile Agents*, World Wide Web Journal, Issue 1: Conference Proceedings, Fourth International World Wide Web Conference, O'Reilly and Associates, December 1995

[Lewontin95] S. Lewontin, *Securing the Enterprise Web*,
<http://www.opengroup.org/www/enhance/swwhitep.htm>

[Luotonen94] A. Luotonen and K. Altis, *World Wide Web Proxies*,
<http://www.w3.org/hypertext/WWW/Proxies/>.

[Miller95] J. Miller, *Using the World Wide Web to Support Groups*, February, 1996,
<http://www.opengroup.org/www/waiba/papers/2-17-95.htm>

[Miller96], J. Miller, P. Resnick, and D. Singer, *Rating Services and Rating Systems (and Their Machine Readable Descriptions)*,
<http://www.w3.org/pub/WWW/PICS/services-960323.html>

[OSFRI94] *Hypermedia Browsing Technology for Wide Area Information Environments*,
<http://www.opengroup.org/ri/contracts/2.InnovativeClaims.frame.html>

[OSFRI96a] *Web Integrated with Security Enhancements*,
<http://www.opengroup.org/RI/PubProjPgs/wise.html>

[Röscheisen95a] M. Röscheisen and C. Mogensen, *ComMentor: Scalable Architecture for Shared WWW Annotations as a Platform for Value-Added Providers*,
<http://www-pcd.stanford.edu/COMMENTOR/>

[Röscheisen95b] M. Röscheisen, C. Mogensen, and T. Winograd, *Beyond browsing: shared comments, SOAPs, trails, and on-line communities*,
http://Walrus.Stanford.EDU/diglib/pub/reports/brio_www95.html

[Schickler96] M. Schickler, M. Mazer, C. Brooks, *Pan-Browser Support for Annotations and Other Meta-Information on the World Wide Web*,
<http://www.opengroup.org/www/waiba/papers/grpannot/grpannot.html>

[Schoenfeldinger95] W. Schoenfeldinger, *WWW Meets Linda: Linda for Global WWW-Based Transactions*, World Wide Web Journal, Issue 1: Conference Proceedings, Fourth International World Wide Web Conference, O'Reilly and Associates, December 1995

[Spetka94], S. Spetka, *The TkWWW Robot: Beyond Browsing*, The Second International WWW Conference '94: Mosaic and the Web, Advance Proceedings, October, 1994.

[WWW4] World Wide Web Journal, Issue 1: *Conference Proceedings, Fourth International World Wide Web Conference*, O'Reilly and Associates, December 1995.

Part 2: Logistics Anchor Desk Option

Misha B. Davidson

9.0 Abstract

This report covers the work performed by the Open Software Foundation Research Institute under the Logistics Anchor Desk (LAD) option of the WAIBA contract in 1996. The highlights of the work include development of Graphical Annotation technology and design of mixed-media Web-based annotation system, participation in a Technology Integration Exercise (TIE) and in an Integrated Field Demonstration (IFD), and attending three Advanced Logistics Planning (ALP) workshops. Note that during the course of the project, DARPA asked that we move our research from the LAD program to the needs of the ALP program.

10.0 Introduction

This report is a summary of the work we did under the LAD option of the WAIBA contract in 1996. It presents key elements of the effort along with the results that were produced.

Our role in LAD was to bring advanced Web technology into the realm of LAD Technology Integration exercises. We fulfilled this role by participating in a TIE in March and in an IFD in July of 1996.

During the year, the LAD program was transformed into Advanced Logistics Planning (ALP) program. Our role in it has changed accordingly. We participated in a number of ALP meetings aimed at determining the system goals and architecture. Reports from these meetings are included. In addition we wrote a number of research proposals aimed at solving the long-term technical problems relevant to the ALP program.

In the process of preparing for the TIEs and IFDs, we designed and developed a number of Web-based collaboration technologies. For the initial TIE in March, we adapted a Web-based annotation system to work with text documents generated by the GLAD software. In preparation for the July IFD, we developed the Graphical Annotation technology that enables logisticians to collaborate on logistics maps using their Web browsers.

As a part of our participation in ALP, we also developed and designed Integra, an Integrated Text, Graphics and Audio Web-based annotation system. Integra will enable logistics planners to collaborate on full logistics documents consisting of text and graphics. It will make collaboration easy, allowing users to comment via text, graphics or speech, or any combination of the above. Finally, it will provide sophisticated notification and group membership facilities enabling users to obtain accurate information as soon as it is available.

The following sections present key information describing these activities. The first section describes the design of the Graphical Annotation system that was presented at the July IFD. The second section addresses the functional specification of Integra. The final sections provide information about the ALP Workshops we attended throughout the year.

11.0 Initial Plan for Building Graphical Annotation

11.1 System Architecture

The Graphical Annotation provides users with the ability to create and read graphical annotations attached to the images on a server where the system resides. The system will modify the documents residing on that server by replacing images with applets which display those images and allow users to annotate those images.

11.1.1 GUI

The GUI supporting that functionality is presented in Figure 1:

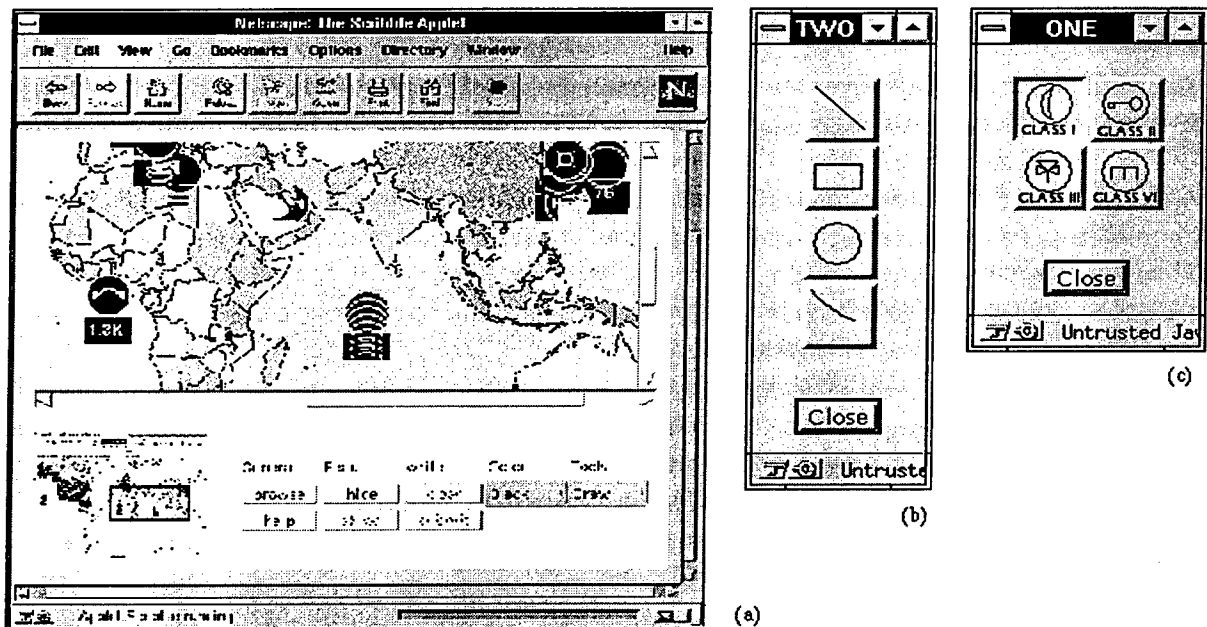


Figure 1. Graphical Annotation GUI. User interface consists of the main applet window imbedded in the document (a), and of pop-up windows (b) and (c), containing the pallets of symbols and actions supported by the system. The main applet window contains three major components. They are the "drawing window" on the top where a user can create and read annotations, a "positioning window" that allows users to find and move the drawing window on the map, and the control panel.

Users will create annotations by selecting one of the tools from the pop-up pallets and applying this tool to the main drawing window, (much like in Frame or xfig). Scrolling the main application window will enable users to read annotations in various parts of the image.

Each annotation consists of a number of individual symbols or shapes, plus an annotation anchor. An annotation can be presented in one of two ways: expanded, when all components of the annotation are showing, or collapsed, when only the annotation anchor is visible to the user. Clicking on the anchor toggles the state of annotation, allowing users to access the view they prefer, whether it is the unobscured view of the underlying map or certain graphical annotations on that map.

11.1.2 Components and topology

The current version of Graphical Annotation is **server-side**. This means that it will only allow users to annotate images on a single server where the system is installed. We made this decision because this is the simplest configuration that allows us to explore the functional requirements and the key complexities of the system. Should the need for supporting annotations to arbitrary documents arise in the future, it can be accommodated based on our past experience with text annotations.

The key system components and their interactions are shown in **Figure 2**:

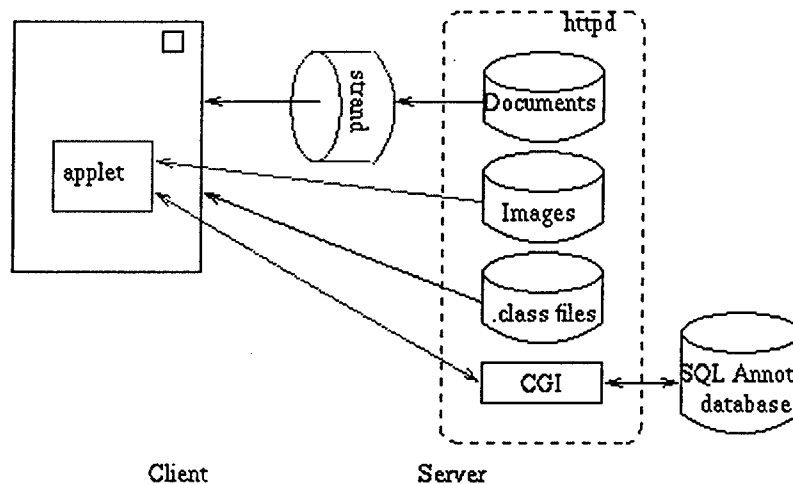


Figure 2. Graphical Annotation design. The HTTP server contains most of the system components except for the Strand.

Graphical Annotation is a three-tier system. The applet running in the browser is the client, the http server is a mediator of all interactions, and the SQL database provides the annotation information. The design uses the HTTP server to bring the whole system together. It serves the documents, images and applets, and it also provides the CGI interface that enables the applets to access annotation information (applets can talk to the server they came from without any restrictions).

The Strand sitting in front of the HTTP server bootstraps the Graphical Annotation process by replacing the image anchors like:

```
<IMG SRC="image.gif">
```

found in a document returned in response to the request for document `/~mbd/index.html` with the applet that displays that very same image and retrieves and overlays the annotations over it. The replacement HTML will look something like this:

```

<APPLET code="ButApp.class" width=500 height=300>
<PARAM name="IMAG" value="image.gif">
<PARAM name="DOCT" value="/~mbd/index.html">
<PARAM name="HOST" value="defiant.osf.org">
<PARAM name="PORT" value="80">
<PARAM name="PATH" value="/cgi-bin/aserver-demo/tester.pl">
<PARAM name="AUTH" value="bWJkOmlZA==">
</APPLET>

```

When the browser loads the applet, the applet will use its parameters to connect to the annotation server via the CGI on the HTTP server in order to retrieve and store annotations to/from the database.

The most complex part of the system, however, is the Annotation applet itself. The next section of this document deals with the applet design.

12.0 Master/slave event architecture for Graphical Annotation Applet

This section describes the key design principles which guide the construction of the graphical annotation system.

There are many interactions between various components of the system. We introduce a uniform interface structure that allows system components to communicate with the rest of the system (see **Figure 3**). By allowing only "vertical" communication, we make the system uniform and reduce the number of dependencies among its components.

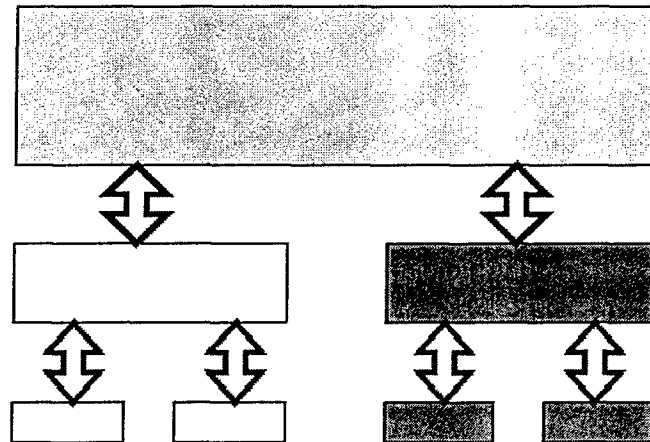


Figure 3. Module communication. In order to minimize the complexity of the system, "sibling" modules are not allowed to communicate directly. Instead modules communicate with their "containing" module and with the modules they "contain".

In order to insure the end-to-end processing of all events, unify the architecture, and simplify interfaces, the event handling system is implemented as a number of slave components and a single Event Master component. Each slave component accepts user events, notifies the master about these events and knows how to carry out certain actions. The event master contains a number of event-specific handlers. When a

slave component notifies the master about an external event, the master evokes a handler which dispatches appropriate actions to the relevant slave components, as shown in **Figure 4**.

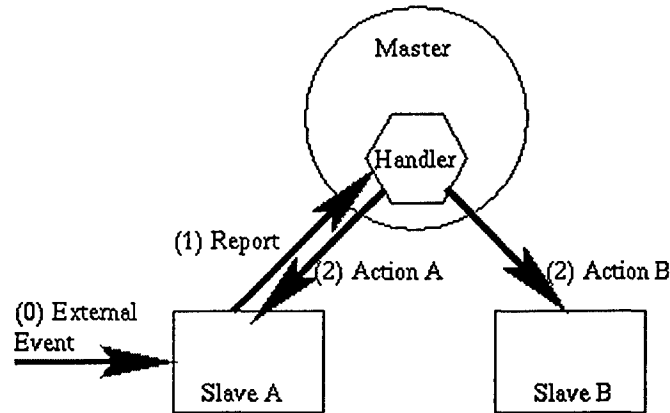


Figure 4. Master/Slave architecture for event handling. This architecture insures an end-to-end flow of events. When an *external event* (0) is accepted by a *slave component* (A) it does not handle it directly, but passes the event on (1) to the *Master* which invokes an appropriate *Handler*. The Handler then determines the correct action and dispatches it (2) to the relevant slave components. As a result, more than one component may react to an external event.

All top level components in the system conform to this structure. **Figure 5** presents the system components grouped by their function.

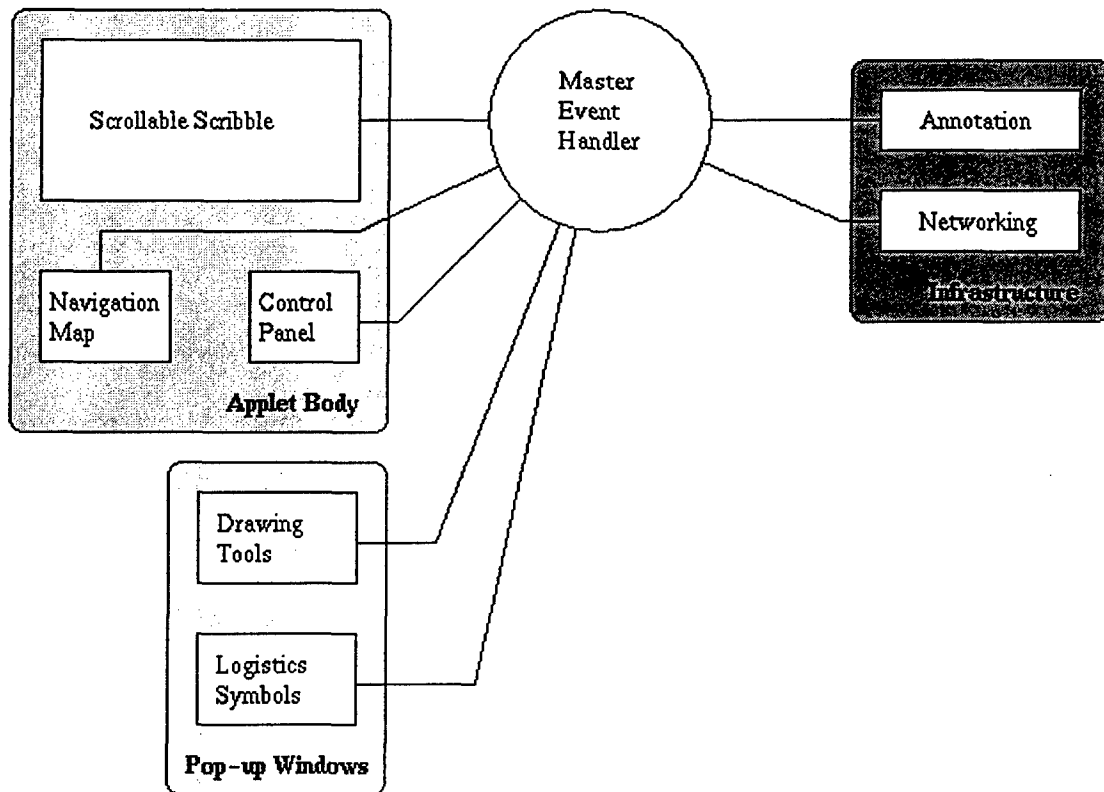


Figure 5. Top level components of the Graphical Annotation Applet. All major components, no matter where they are located in the system and whether they are exposed to the user or not, obey the master/slave architecture outlined above and handle all incoming events and communicate with each other via the Event Master.

Another key design principle of the system is to keep only one copy of state, preferably in the Master Event handler. However, when the state is tightly associated with one of the components, it is acceptable to keep the state within that component as long as interfaces for retrieving this data are provided.

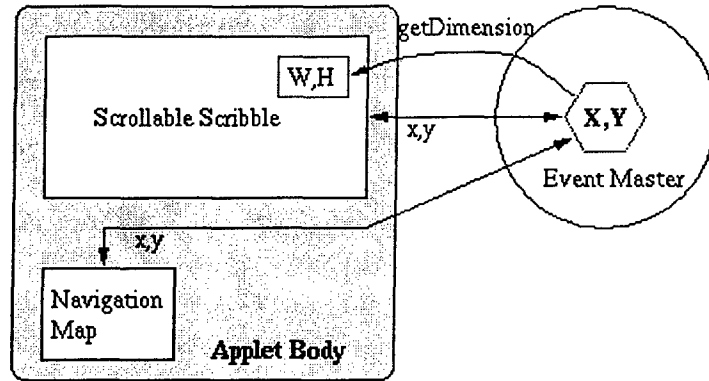


Figure 6. Uniqueness of state. In order to preserve the end-to-end nature of the system, we try to avoid replicating data throughout the system. Scrollable Scribble and Navigation map share their current coordinates through the variables X and Y stored in the Master Event Handler. Not all data, however, needs to be stored centrally. The dimensions of Scrollable Scribble may change as the applet window is resized. Thus, the dimension data belongs to it, and whenever this data is needed it is read by the event master

Drawable objects constitute a drawing mechanism used for representing user annotations. Each drawable object has a symbolic representation and can either be drawn in full (as in drawable shapes and symbols) or act as a tool (e.g. delete or move tools). The classification of drawable objects is presented in Figure 7.

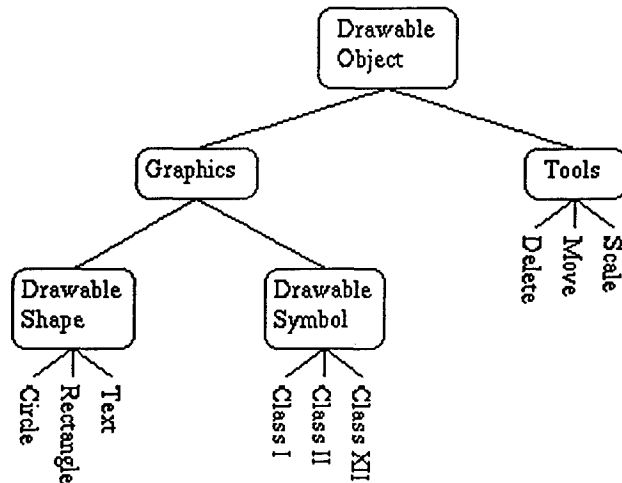


Figure 7. Classes of Drawable Objects. Drawable objects are objects that have a drawable symbolic representation and can possibly be drawn on a map in full representation

Each subclass defines its own way of handling user events, which should make it easy to implement drawable objects like "delete" and "move." This hierarchy can be easily extended to accommodate new types of objects or symbols specific to a different application domain (e.g. architecture).

13.0 Functional Interface for Integra-Integrated Text, Graphical and Audio Annotation System

13.1 Introduction

Integra is an integrated text, graphics and audio annotation system which will enable users to annotate HTML documents and images with text, graphics and audio. Integra will make the annotation process as easy and seamless as possible *without actually rewriting* a browser. This document describes the key functionality of Integra and outlines the design choices that make this functionality possible.

Design goals for Integra are: *integration, unification, consistency* and *responsiveness*.

- *Integration*: all modalities of Web documents can be annotated with all modalities of annotations
- *Unification*: the system does not simply allow users to create text or graphical annotations, but rather, each annotation may consist of a variety of modalities, containing, for example, text, graphics and audio. Moreover, creation of such a compound annotation should be an easy and seamless process.
- *Consistency*: even though there are several active annotation modalities, the interfaces to those annotation modalities are coherent. For example, writing a text annotation is in some way analogous to writing a graphical or audio annotation
- *Responsiveness* is key to the system success. Functionality is the our most important goal. Performance, however, is the key to the success of the system. Even though we will not implement all the performance features initially, we will put in the hooks for future enhancements.

We are designing a full-featured system, but first we will be building only a baseline version for the following platforms: Netscape on UNIX and PC and Internet Explorer on a PC.

13.2 Usage Scenarios

Integra will provide a powerful collaboration support mechanism for technical professionals and other users who jointly develop and evolve documents using an electronic media. Thus, the function of the system must reflect the kinds of work and patterns of communication followed by such users.

A typical scenario would be, for example, that of a design team working on a technical plan for their project. After a series of meetings, the members of the team have more or less agreed on the high-level system design. At that point, the team leader creates a high-level design document and places it on the Web for review by the members of the team. Using the annotation system, he/she notifies the rest of the team that the document is ready for review. The team members then comment on the document using Integra, and the team leader is notified when their input becomes available.

There are three types of first class objects in our system:

- Users
- Groups (or Tasks)
- Documents

And of course there are annotations. What does it mean to be a first class object? It means that a user will be able to see a list of such object's system-wide (e.g., a list of all users) and add new objects of this type to the system independently of any other conditions. This is in contrast to annotations that are attached to existing documents.

Users are organized into groups. Groups are short-lived and are created to fulfill a specific task (such as creation of a technical design document). Once the document is created, the group that has worked on it is no longer needed, although a different group using this document as an implementation guideline may be created.

Users should be able to create new groups and users and add users to existing groups on line via the standard Integra interface. The creator of the group should be able to send a notification to the member of the group announcing the addition of the new document to the scope of the discussion.

13.3 Functional interface

The Integra annotation system UI consists of two major components—the browser and the associate/proxy itself. The goal is to create a clear functional distinction between the text and graphical annotations. The browser displays the annotations or annotation anchors of both text and graphical annotations inline, but does so statically, without allowing the user to add new or edit existing annotations. The Integra associate/proxy is used to view the content of annotations represented only by anchors in the browser. Most importantly, it is used to add new annotations (both textual and graphical) to the documents displayed in the browser.

On a very high level, the browser and the Integra associate/proxy interoperate in the following ways:

- When the browser requests a document, the proxy function of Integra merges existing annotations and the activators that enable users to create annotations into the document requested. The browser displays the document with the annotations and activators merged into it.
- When a user clicks on one of the annotation anchors in the document, the associate part of Integra displays the information about the corresponding annotation.
- When a user clicks on one of the annotation creation activators, the associate presents the user with an annotation creation interface (more detail later) that enables the user to write a new annotation.
- When the annotation is submitted, the associate merges it into its internal representation of the document, forcing the browser to reload the document with the new annotation via the proxy.
- Finally, when a user selects one of the annotations within the associate, the associate displays the information about the annotation and then forces the browser to scroll the document to the relevant annotation anchor.

The implementation of these interactions may differ slightly for different annotation media; however, all types of annotations in the system should support these interactions.

14.0 Types of annotations

Consider the relationship between the document and the annotation media. The Web consists of HTML documents, some incorporating **graphics** and **audio**. We will provide **text**, **graphic** and **audio** annotations. How do these two categories mesh together? In the most general system, it would be possible to annotate any kind of Web media with any other kind of annotation media. To simplify the task, we propose some constraints on this relationship. Later, when the skeletal system is in place, the types of media that are allowed in the system can be extended.

Consider the use of audio, first in HTML and then in annotations. In HTML documents, audio is just an embedded anchor; it is no different from any other URL. So, as far as the annotation system is concerned, the audio fragments in HTML documents should be treated just like the rest of HTML. Now consider the use of audio in annotations. We could have a separate type of an annotation - an audio annotation - or we could integrate audio within the text and graphical annotation. The latter solution appears more useful. For example, integrating text and audio in an annotation referring to a related Web site will make it easy for a user to type in the URL of that site and at the same time record an audio annotation. In some cases, the integrated text+audio annotation will have only a typed title, and the rest of the annotation will be audio. Alternatively, a combined annotation without audio will simply be a text only annotation.

The above argument for integrating text and audio annotations equally applies to graphical annotations, which could also benefit from the addition of audio comments. And of course, graphical annotations should include text. In fact, there are three different kinds of text. One is title, another is free floating text over the image generated with a text tool, and the third is the description text that is entered separately in a dialog box (the last two kinds are optional).

Taking all of the above considerations into an account, the following table shows the relationships among the Web/Annotation media:

HTML elements	Annotation Media	
	<i>Text + Audio</i>	<i>Graphics + Audio + Text</i>
HTML with embedded audio	Yes	No, but design applet with this use in mind.
Graphics	No, but is implemented within graphics	Yes

We could possibly implement text annotations to the whole image by placing a paragraph marker right after each image. This may, however, confuse the appearance of some documents. More importantly, having the text field as part of the graphical annotation fulfills the need for commenting on the image.

Annotating plain HTML with graphics is no doubt useful, but will require a separate inlining mechanism for putting new annotated images into the text. The Graphical Annotation applet is designed so that it can be used both with existing images and with the new visual annotations.

In summary, we can define a generic annotation as:

Generic Annotation = Title [+Text] [+Graphics] [+Audio]

with the two specific cases we described above defined as:

Annotation to Text = Title [+Text] [+Audio]

Annotation to Graphics = Title + Graphics [+Text] [+Audio]

The rest of this document will refer to the above Annotation to Text as *Text Annotation*, and to Annotation to Graphics as *Graphic Annotation*.

In the future, however, we could add a simple graphical capability to the Text annotation, which would allow a user to include a simple drawing along with the text of the annotation. Other users would be able to view this drawing, but they would not be able to draw on it as they would in the case of normal Graphical Annotations.


14.1 Main Application User Interface


This section introduces some constraints to the system behavior. Having such constraints makes it easier to describe the functional behavior of the system. Please consider the *principles* rather than *specifics* in this section.

The main design goal for this system is to provide an integrated annotation environment. This tight integration of the system is exposed to the user via the GUI. What follows is one possible way of structuring that GUI, presented here in conjunction with the states of the system.

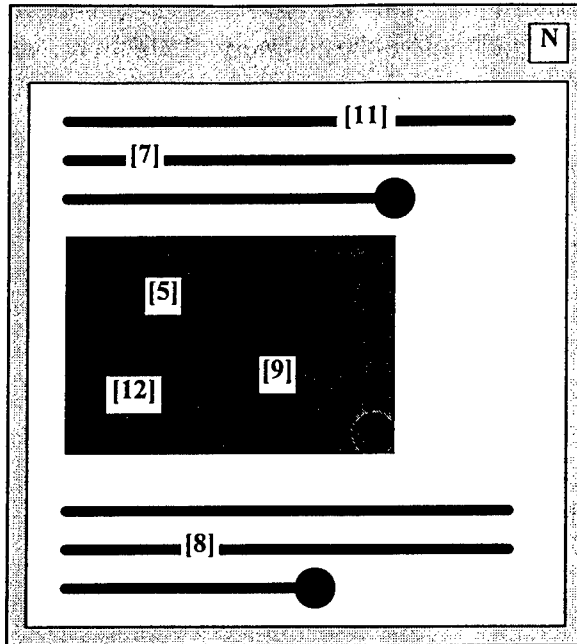
14.1.1 Building blocks

From the perspective of the annotation system, any document consists of a number of *chunks*. An example of a chunk is a paragraph, an unnumbered list or an image (notice the parallel: an image to a graphical annotation is what a paragraph is to textual annotation). Within every chunk, there may be a number of annotations. Of course, a graphical annotation to an image is the same as a textual annotation to a paragraph. To emphasize this parallelism, we are going to use the same symbols for chunk and annotation activators for both text and graphics.

 **Chunk activator**
for both text and
image chunks

 **Annotation activator**
for both textual and
graphical annotations

Using these activators, Integra will present a user with a **consistent** view of the annotations in the context of a document. The consistent use of activators will also reinforce the uniformity of the annotation interface provided by the Integra associate.

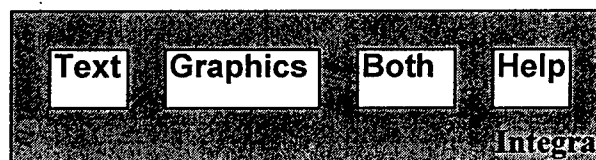


The Integra associate has a small number of states defining the functionality and the behavior of the system at any moment. These states are:

- **Initial** - annotation is disabled, no processing obvious to the user is done on the documents viewed via the browser
- **Reading** - either Text or Graphic annotation or both are enabled, annotation activators are appropriately merged into the document. User can read annotations by either selecting annotations from the annotation list presented by the associate, or by clicking on the annotation activators in a browser.
- **Writing** - a user is presented with the tools appropriate for annotating the chunk of the document he/she selected (e.g. drawing tools and dialogs for graphics and insertable paragraph text for text).
- **Search** - provides a user with a search interface that allows to search for all annotations that match the given criteria across the whole document space.

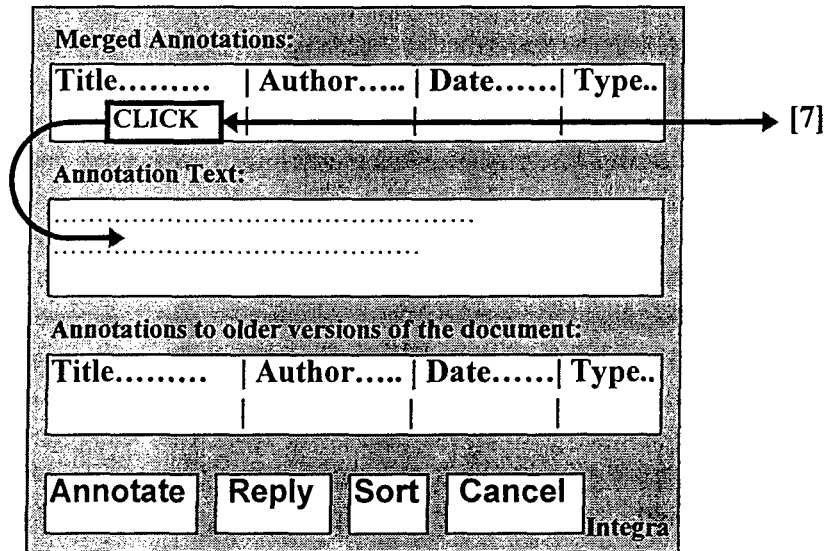
As was mentioned above, audio controls are included in all annotation interfaces.

Below is a sketch of the GUIs corresponding to these states. When the system is in the **Initial** state, it presents a user with the controls for activating the services.



Once the user activates one of the available services, the system goes into the **Reading** mode. In this mode the user can see a list of all annotations in a document (along with their author and type, which is either text or graphics). By double clicking on an annotation from that list he/she can read the text of the

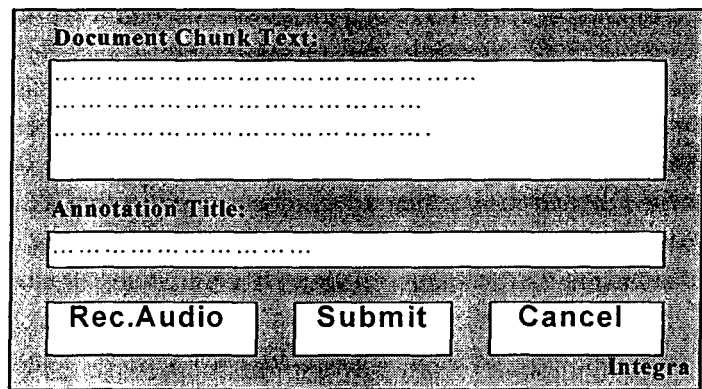
annotation (remember, graphical annotations may have text associated with them). In addition, if there is an audio clip associated with the annotation, the system will play it as well. Finally, Integra will tell the browser to scroll the document so that the chunk to which the selected annotation belongs is in the top of the window.



All annotations that could not be merged into the document will be displayed in a separate clickable area. Later if a **Document Change Tracking** system were developed, double clicking on an annotation appearing in that window will force the browser to load the version of the document to which this annotation has been written originally.

Clicking on Annotate or Reply buttons will send the system to the **Writing** state. If the last annotation that the user has looked at was an annotation to a text chunk of the document, then the user will be presented with a Text Annotation Interface with that particular chunk pre-loaded. If the last annotation was attached to a graphics chunk, that particular image will be pre-loaded into a Graphics Annotation interface. A user can also get to this screen by clicking on the paragraph activator in the browser.

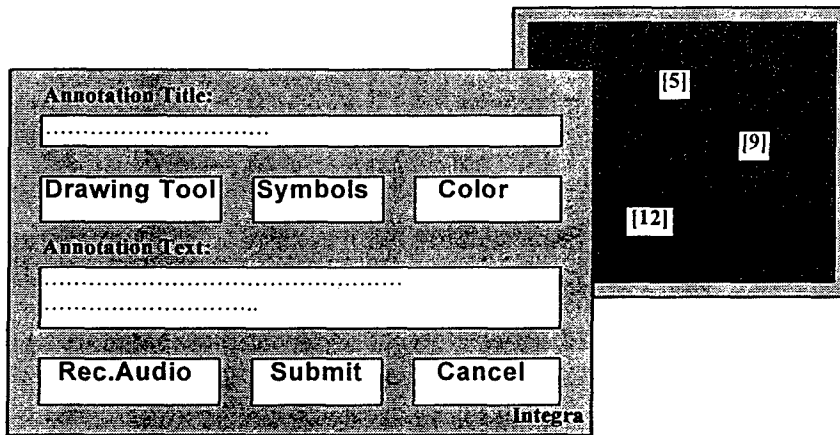
First consider the Text Annotation interface. There is a text area where the system places the text of the paragraph that is being annotated. To annotate any part of this text, a user simply clicks in the text and then starts typing. When he/she is done, the user clicks the Submit button, the annotation is sent off to the annotation server, and the document is reloaded with a new annotation marker merged in.



The interface for Text Annotation Reply is similar, but is more restrictive. While the user can see the paragraph to which the original annotation is attached and he/she can see the text of that annotation, neither is editable. Instead a user is given a text area to type in the reply.

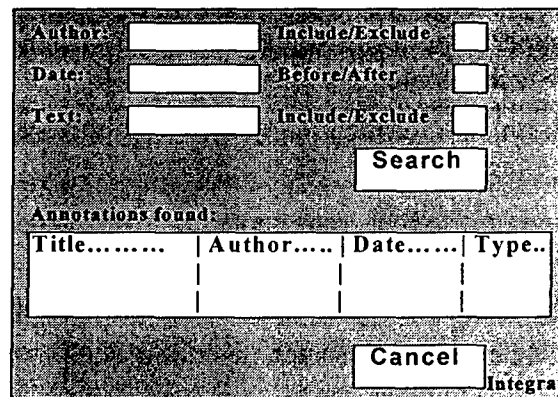
Second, consider the Graphic Annotation interface. A user can get to this interface from the **Reading** mode when he/she clicks on the Annotate button while viewing a Graphical Annotation. Alternatively, a user may simply click on the chunk activator associated with an image in a browser. In order to preserve consistency with the case of Text Annotations, Graphical Annotations are created within the Integra Associate, not within a graphical applet running within the browser.

The Graphic Annotation Interface consists of two windows. One contains the image that is being annotated (it serves as a drawing canvas and also displays the previously written annotations). Another contains the drawing tools and text boxes.



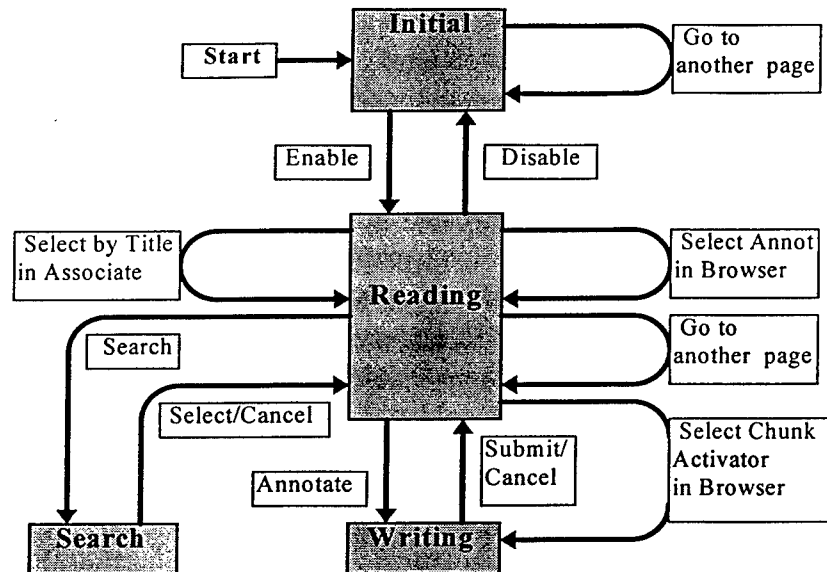
Some of the drawing tools could also open additional windows, dialog boxes and pallets. The Graphical Annotation Reply Interface is very similar to the interface described above. The only difference is that the graphical representation of the original annotation is highlighted and its text is presented to the user.

Last, but not least, the Search interface that users to examine the whole annotation space and find the annotations that match the given criteria. The interface consists of a search form and an output list containing the annotations found. Clicking on one of the annotation titles in the list forces the browser to load the document to which that annotation is written.



14.1.2 Summary of Actions

The following diagram presents a high-level summary of the interactions of the various states of the system described above:



Note that the above diagram omits any issues dealing with system administration, such as adding new users or creating new groups. These actions are out of the scope of this project.

15.0 First ALP Workshop, May 29-31, 1996

15.1 Overview of ALP

The Advanced Logistics Planning (ALP) is jointly run by DARPA and Defense Logistics Agency (DLA). The goal of ALP is to improve the efficiency of and to speed the logistics planning process by bringing in modern computing technology. The ALP conference we attended initiated such technology infusion by:

- Describing the overall goals of ALP to the participants
- Demonstrating the scope of the current technological efforts within the logistics community
- Bringing together domain experts and technologists in order to determine the match between the existing needs and available technologies
- Attempting to determine a base plan for integrating the available technologies in each of the separate technical areas.

There were over forty companies and universities, represented by over 120 people, at this meeting.

The main directions for ALP were outlined by the three PMs (Brian Sharkey, Doyle Weishar and Bob Neches) during the first day of the conference. They emphasized the concepts of rapid planning tied with execution, continuous replanning, and end-to-end system operation.

Sharkey concentrated on the idea of rapid replanning using sentinels that watch the execution activity, compare it to the original plan, and change the plan accordingly to compensate for any deviations. He

also emphasized the use of HCI technology to provide operators with a suitable visualization and collaboration environment.

Weishar talked about the core technical issues of the program, emphasizing the use of mediators that facilitate an efficient and timely integration of heterogeneous databases. It is interesting that he mentioned the use of Blackboards as one of the core technologies to support collaboration.

Neches outlined the division of labor within the ALP process. TI and BBN will be the principal contractors in charge of integration of technologies provided by subcontractors such as the RI. As the project progresses, their participation in the project will increase from being observers and facilitators to leading the development effort. At the end of the meeting, he was very clear about requiring developers to be responsive to the needs of end-users from the very beginning, instead of building throw-away technology.

Overall the emphasis was on building technology that is useful to the logistics community and that can be integrated to form an end-to-end system that makes logistics analysis faster and more precise.

15.2 The RI's role in ALP

The participants in the conference were divided into groups doing related work. The RI was assigned to the *Group Decision Support* area. The goal of the participants of that group is to provide tools that allow logistics planners to make better decisions based on the information available to them and to better coordinate their activities. There were about a dozen organizations present in the group, divided approximately equally between collaboration technologies and information gathering agents. Two of the collaboration support systems were Web-based and most of the participants planned to have Web-based interfaces to their systems available soon. For more details, see the section on *Collaboration Possibilities*.

A template for a 15 minute presentation was provided to participants. Misha Davidson gave a presentation on graphical annotation work. The proposed work was incorporated into the final plan for Group decision support.

Jeff Berliner compiled all the suggestions from all participants into a large table with the RI's row reading:

Now	1 Year	Later
<ul style="list-style-type: none">• Text Annotations	<ul style="list-style-type: none">• Graphical Annotation• Group Membership, Authentication	<ul style="list-style-type: none">• Integrated Annotation Service• Document Change Tracking

Additional information on the RI's role can be found in the section on *Future Work*.

15.3 Collaboration Possibilities

Based on the presentations we heard and the people we met during the conference, the following several projects are possible candidates for collaboration.

There were two Web-based systems presented at the session on Group Decision Support. One, developed by the ISX corporation and called "Tapestry," is a collection of specialized Web servers that interface to simple databases, each carrying out one function. For instance, Tapestry has a:

- Calendar server
- Mediated Access server
- Rolodex server
- Bug report server

While the idea of providing a variety of application specific servers is interesting and potentially useful, the ISX team did not build any infrastructure supporting communication among these components.

Another Web system is the Telecon from Volpe Transportation Labs. Telecon has functionality similar to that of the Mediator and the HotPage developed at the RI. A user can check in a document and ask other users to comment on it. The comments are shown in a separate frame, as in HotPage, but unlike Group Annotation. The system also supports search and the functionality of the Personal Profile Server.

The system is simply a Web server with a Python interpreter compiled into it; all the databases are monolithically implemented directly in Python. The system, however, has a sleek user interface with frames and graphics.

15.4 Adding Web capabilities to other projects

An interesting non-Web project was presented in our group by Roger Barga from the Oregon Graduate Institute. His system, DIOM, supports continuous queries on databases. Such queries are applicable to "what's new" and "what's changed" systems. Every query is a tuple consisting of (DB_Query, Reevaluation_Condition, Termination_Condition) The query is reevaluated every time when Reevaluation_Condition becomes true until Termination_Condition is met.

The system is currently using database log scraping to determine when reevaluation conditions become true. In future plans, Roger talked about putting the databases on the Web and possible integration with Harvest, also developed at OGI. It was suggested that the RI's Strand technology may be helpful in an end-to-end replacement of log scraping when databases are put on the Web, particularly in a scenario with a large number of distributed databases. We agreed to exchange more information on our projects.

Another interesting system presented in our group was Visage from Maya Design Group, originally developed as Sage at CMU. Visage provides a portable tool-set for large database visualization. It operates on live database data, supporting such visual operations as drill-down/roll-up and painting, where all instances of a particular class are painted a user-defined color. The system is highly configurable and scriptable.

Misha Davidson also spoke to Lynn Schmoll (lscmoll@dlsc.dla.mil) from the Defense Logistics Services Center (DLSC). Schmoll said that DLSC is in the business of publishing electronic catalogues for the Army procurement, and was interested in using the Group Annotation system to support the procurement process and asked for more information.

Finally, Misha Davidson spoke to Lou Mason, ALP/LAD domain expert. Mason has expressed interest in extending the Graphical Annotation system to support synchronous collaboration, such that several

people could co-author a graphic annotation together. We agreed that when he visits Boston, he could come to the RI for more discussions.

15.5 ALP IFD -1.2

Conversations with Doyle Weishar and Bob Neches indicated that ALP funding would start no sooner than August 1996. Meanwhile we continued to work under the LAD contract, participating in the ALP IFD on July 30 - 31. As the demonstration was classified, our Graphical Annotation was installed on a BBN machine for this demo. BBN then copied the setup onto a classified machine in order to give the demonstration.

16.0 Second ALP Workshop, Sept. 10-12, 1996

16.1 Overview

The three major accomplishments of the second ALP conference were:

- communicating our technical vision to the ALP team
- better understanding of the human-to-human communications in the logistics process
- demonstrating our software and establishing contacts with other participants for future cooperation/early adoption

During the conference Misha Davidson successfully presented the RI's vision for an integrated annotation system (Integra) at the breakout session on Course of Action (COA) and Sustainment support technologies. This presentation resulted in inclusion of Integra into the ALP technical plan as a key communication/collaboration component for COA/Sustainment. He also presented the proposed role of Integra to the meeting of all ALP participants. Finally, Jeff Berliner suggested that Integra be used as a collaboration tool during the process of ALP development. Such use could give the RI broad exposure among over forty companies and government agencies.

During the meetings and especially at the demonstrations of our software, a number of people came forward who are interested in our work either as collaborators or early adopters. Maintaining contacts and working with these people may help us to build better software *and* discover new ideas.

16.2 Meeting Highlights

16.2.1 Brian Sharkey - overview

For project information see the following URLs

- <http://darpa.mil>
- <http://darpa.sra.com/alp>
- <http://hokie.bs1.prc.com>

Anticipatory contracts were stated to be the temporary solution, sufficing until the end of September, 1996.

Key points made by Sharkey:

- Revolutionary vision is more important than revolutionary technology.
- Products, not technologies
- take a "systems view"
- reduce labor intensiveness
- automation, networking, laptops
- seamless and continuous modeling and simulation

Brian Sharkey wrote a paper explaining his vision. It is available at the SRA site.

- J3 NEO planning support - plan developed and REVIEWED by j3, j4
- Ship stow plan-schedules are distributed via the Internet

In another presentation, Sharkey outlined the key requirements for the logistics technology

- Build tools to operate on data
- Maintain reliable data
- World-wide high-bandwidth communications

16.2.2 Future Plans

Jeff Berliner proposed Integra as a major component of the following tools:

- Collaborative Plan Executive
- Allocation Module w.r.t. Apportionment guidelines

As a minor component for the remaining COA/Sustainment tools:

- Logistics Analysis Manager
- Plan Execution Monitor
- Requirements Estimation Manager
- Dynamic Critical Item List Generator

16.2.3 Conversation with Major Hamm

Major Hamm is a logistics planner stationed in St. Louis. He has been with the Air Force for 21 years. He outlined the communications typical of creating a logistics plan.

Major Hamm said that logisticians participate in two types of planning:

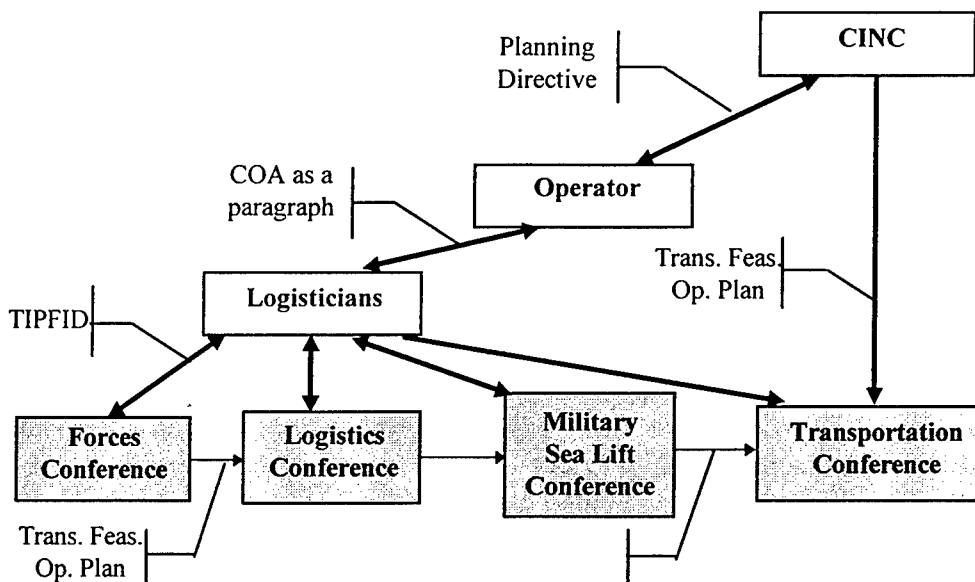
- Deliberate Planning - 1 year cycle
- Crisis Action Planning - days or weeks

The Major's stated that he should be able to use the same tools for both types of work. It seems that there are many data integration problems. The Major said that there are dozens and dozens of databases

maintaining a tapestry of information that is manually integrated every time anyone needs to access a single piece from point A to point B.

Major Hamm wants a system that would allow him to say to say: "I need 25 fuel filters for F-16s delivered to Alpha Base two weeks from now." The system would respond, "Sending filters from Depot X on Transport Y arriving at Alpha Base by Date Z." Ideally, the system should know that F-16s require new filters every 1500 hours of flight time and that they fly 300 hours per month on average. Therefore, filters would be shipped automatically to Alpha Base every 5 months.

In terms of the work that we could do, the Major outlined the communication pattern that occurs during the Deliberate action planning.



There is a lot of communication happening, which is currently handled by arranging conferences and sequentially passing the plan among participants. There is much room for improvement, for example, by providing parallel access to the evolving plan document, having the working groups, etc.

17.0 Third ALP Workshop, Dec. 11-13, 1996

17.1 Overview

The goal of the third ALP workshop was to define specific functional goals and a timeline for achieving them, giving the program participants a clear understanding of their tasks and of how they fit into the bigger picture. In the words of Brian Sharkey, the workshop did not succeed in achieving these goals.

To insure better information exchange among the ALP participants in the future, SRA has set up a Website that will contain the key ALP documents and information. The URL of the site is: <http://darpa.sra.com/alp>. Access to this site is restricted.

Finally, during the workshop Misha Davidson met with a number of technology providers, some of whom may be interested in collaborating on this program or providing the technology suitable for use in other RI projects.

17.2 Meeting Highlights

17.2.1 Day One

During the first day of the workshop there were a number of general speeches covering various aspects of current logistics practices and the desired results of the ALP program.

17.2.2 Day Two

During the second day the participants held separate meetings, organized by the Integrated Product Teams (IPTs). Misha Davidson spent most of the time in the Logistics Planning IPT, headed by Jeff Berliner, and about an hour in the Electronic Commerce IPT. (Other IPTs included Rapid Supply, Transportation, etc.)

In the Log Planning IPT, Brian Sharkey was very helpful in steering the discussion away from the particular revolutionary capabilities that ALP is going to provide to discussion of the Logistics planning process and the best ways technologically to support it. This was the most useful ALP meeting attended, because it clarified the information flow in the Logistics "information loop." Most of the emphasis of the discussion was focused on specific technologies and algorithms, leaving the human component completely out of the picture.

17.2.3 Day Three

During the last day of the workshop participants met in technology groups. Misha Davidson participated in the User Interface Services (UIS) group. There everyone noted that as long as the core technology of the system is not defined and the processes that are going to be supported are not well understood, there is no way to determine which common services to provide. An attempt to concentrate on a specific application and determine what sort of services could be provided was largely unsuccessful.

The most potentially useful result of this session came from Bill Crowder, a Research Fellow at the Logistics Management Institute (LMI). He suggested organizing an overview course for the participants of the UIS group to demonstrate the kind of problems and interactions that a logistics planner faces in his/her everyday work.

The overall conclusion of the UIS meeting was that everyone should continue working on their respective technologies with the eventual goal of integrating them to make them useful to the ALP community as a whole.

The concluding "plenary" did not offer much new information. Sharkey pointed out the disappointing outcome of the workshop, and notified everyone that the next workshop is scheduled for the second weekend in April. The next preliminary TIE will be in July and the main IFD is planned for September.

Appendix A: Application-Specific Proxy Servers as HTTP Stream Transducers

Application-Specific Proxy Servers as HTTP Stream Transducers

Charles Brooks,
Murray S. Mazer,
Scott Meeks, and
Jim Miller

Abstract

If one wishes to execute specialized processing on the HTTP requests and responses that flow between WWW clients and servers, one can add the processing in the clients, in the servers, or between them. We describe a novel approach to the latter: we generalize the notion of proxy servers to construct application-specific proxies that act as transducers on the HTTP stream. We have built a sample set of transducers, to demonstrate the idea, and an initial toolkit, to ease the task of constructing these transducers and attaching them to the HTTP stream.

Keywords

proxy servers, stream transducers, HTTP

Introduction

Typically, a WWW client (usually a browser) sends an HTTP request directly to the target WWW server, and that server returns the response directly to the client. WWW proxy servers [7] were designed to provide gateway access to the Web for people on closed subnets who could only access the Internet through a firewall machine. In the proxy technique, the WWW client sends the full URL (including protocol and server portions) to the designated proxy, which then connects to the desired server via the desired protocol, issues the request, and forwards the result back to the initial client. The key observation is that the client uses HTTP to communicate with the proxy, regardless of the protocol specified in the URL. An addition to proxy servers gave them the ability to cache retrieved documents to improve access latencies [7].

WWW clients and servers normally expect that the network will transport messages between them with the content unchanged; this is true whether a proxy is in the stream or not. That is, the correspondents expect the client's request to arrive at the server with exactly the content transmitted by the client, and they expect the server's response to reach the client intact. The underlying assumption is that the network is simply a mechanism for transporting messages between the communicating entities but is not a mechanism for applying application-specific processing to the message contents. A caching proxy, which may elect to serve the response itself instead of forwarding the request to the designated server, starts to challenge that assumption. Similarly, a load-balancing proxy, which might select a mirror site for a given request instead of the designated server, in order to reflect current network behavior, would further challenge that assumption. In both of these cases, the request may not reach the designated server, but the response likely reaches the client intact.

We challenge this assumption even further: we suggest that, for some classes of client/server applications and their network transactions, substantial value may arise from inserting, into the communication stream, application-specific transducers that may view and potentially alter the message contents. We are testing this hypothesis in the context of the World Wide Web, by building a sample set of proxy-based transducers that are bound to the HTTP request/response stream. This approach extends the "standard" WWW architecture in a way we believe is both novel and useful.

The rest of this paper proceeds as follows. First, we describe the motivation for introducing transducers into the HTTP stream. Then we review our approach to building and using this enhanced architectural component. We describe some examples, followed by a discussion of our publicly available toolkit [8], its implementation, and its performance. Finally, we consider some future work.

Motivation

We are developing this technology in the context of a larger research program, into approaches to building computerized *browsing assistants* [12] to aid the human user in accessing relevant information on the Web. We hypothesize that some of the browsing assistance is best achieved by monitoring and processing the HTTP stream between the user's browser and the Web. As an independent example, the idea of using an application-specific proxy was recently advocated for non-intrusive community content control[2]. Further, group-related assistance may arise from processing the stream between a workgroup's browsers and the Web; we are investigating a

number of issues in this area.

We view the transducers as having four classes of functionality:

- *Filtering* individual HTTP requests and responses.
- *Characterizing* sets of messages.
- *Transforming* message contents.
- *Additional processing* indicated by the messages.

Here are some illustrative examples of each class in the WWW context:

- *Filtering*: checking the validity of request headers (eliding improper ones or returning an error response to the sender); redirecting requests according to dynamic models of current network behavior; and eliding images from responses being sent to bandwidth-limited clients.
- *Characterizing*: building a dynamic model of current network behavior based on request/response latency; and constructing a full-text index of all responses received by a workgroup (so that individuals may leverage the browsing behavior of the group).
- *Transforming*: converting response formats into formats better suited for the client (e.g., downgrading video quality for a portable client); adding value to the content based on additional data sources (e.g., recognizing zip codes inside responses and converting them into anchors that link to census bureau data on the appropriate regions); and interpreting location-specific references inside requests to determine the appropriate destination server.
- *Additional processing*: prefetching links embedded in each response and applying appropriate transforms, such as constructing a tree showing all document titles lying within two links of the current document.

These transducers are essentially specialized processing modules, whose inputs include at least the HTTP stream. They may also take input from other sources, may communicate with other processing modules, and may create output of any kind appropriate to their function. For example, a full-text indexer could produce an index searchable via a Web form.

These transducers may serve at least four classes of users:

- *Individuals*: individuals may support specific interests and preferences through selection of transducers.
- *Groups*: group transducers may effect group policy or provide benefit to individuals based on the actions of their peers.
- *Enterprises*: as with groups, but on a larger scale.
- *Public*: innovators, information providers, and entrepreneurs may offer transducers to the public.

Approach

Our approach clearly derives from the proxy server model: as illustrated below in Figure 1 for WWW browsers and servers, one can use the mechanism to insert other kinds of processing entities into the stream.

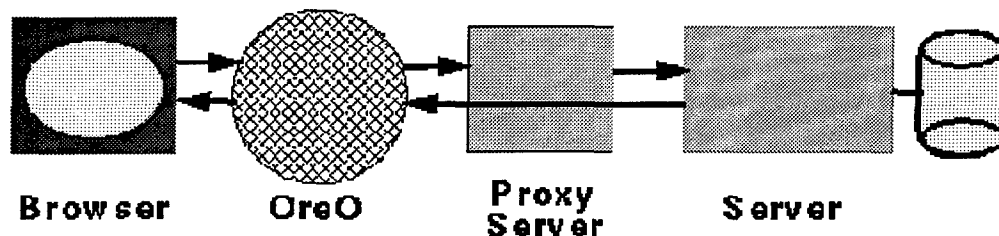


Figure 1. An OreO as part of the HTTP stream

We refer to our HTTP transducers as *OreOs* (with appropriate apologies to the cookie makers), because the transducer is structured with one 'wafer' to handle browser-side communication, another 'wafer' to handle server-side communication, and a functional 'filling' in the middle. As illustrated below in Figure 2, the OreOs take advantage of the HTTP proxy mechanism, essentially appearing as a server to the client side and as a client to the server side. Because the full URL is delivered intact to the OreO, the filling can use the scheme, server,

server-relative URL, or request data in its processing.

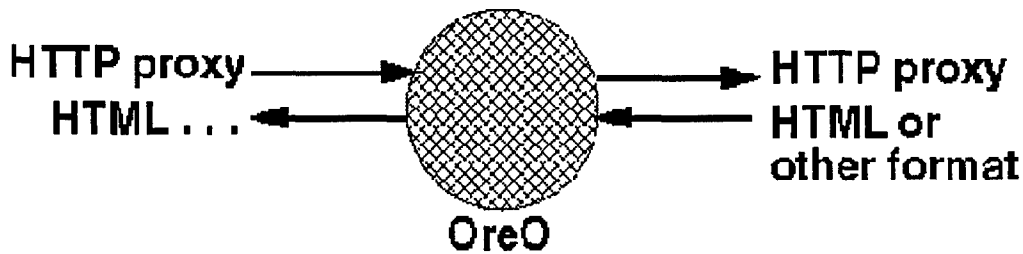


Figure 2. OreOs can use the HTTP proxy mechanism

In some cases, the OreO will forward the request to another proxy (using HTTP proxy, as illustrated above); in other cases, the OreO may forward the request directly to the designated WWW server.

Figure 3 shows various possible compositions of OreOs to serve the needs of individuals, groups, enterprises, and the public.

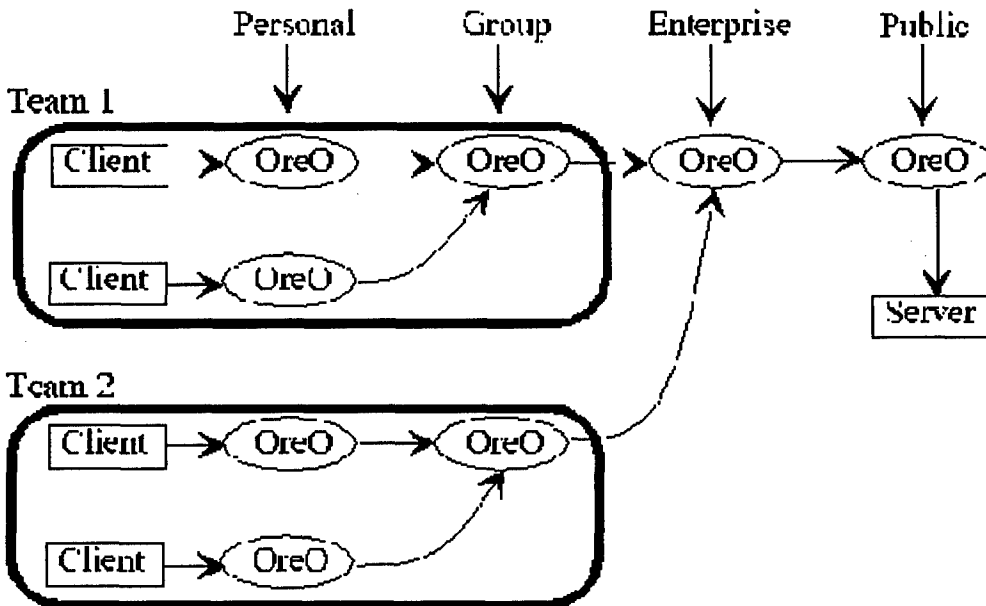


Figure 3. Personal, group, enterprise, and public OreO servers

We advocate constructing highly specialized transducers that can be composed to produce more sophisticated aggregate behavior; this contrasts with building monolithic components that are hard to reuse. We propose that these composed functional chains may be configured dynamically as well as statically, on a granularity as fine as per-request or as coarse as per-session or across sessions. Currently, we have experience only with static configurations.

Our approach reflects the "stream model" of signal processing (later adopted and extended by the functional programming community [1]). Viewing the connection between the client and the server as a stream of data, one can build a standard box which accepts one or more such streams as input and produces one or more such streams as output. Each box can execute specialized processing on the stream(s). All of the boxes have the same basic interface (stream-in/stream-out), so they can be connected together, producing sophisticated processing from the composition of simple processing elements.

We expect that the selection of which transducers to use in which situations will in some cases be preconfigured, in others be under administrative control, in others be under user control, and in still others, be under OreO control. For applications which offer the user some direct control over the selection and invocation of network transactions, users will be able to select the functional modules to be used.

Examples

We have built several example HTTP transducers; these are meant to be evocative, rather than definitive. The first was constructed by deconstructing a CERN httpd and inserting the desired processing (written in C). The remainder were constructed in various languages using the toolkit discussed below. In addition, some of the functionality is enhanced by use of our internally-developed experimental browser, Ariadne[9]. Here are some examples; others are under development:

- The first OreO performed several different functions:
 - URL validation, to identify malformed URLs and report errors directly back to the browser, short-circuiting the request. In a sense, the OreO is "serving" part of the "errorful URL space."
 - Measuring data transfer rates from the various servers, and reporting these to the user via a TCP *backchannel* (a separate communication channel) to the Ariadne browser.
 - Creating a *group history*, based on the HTTP requests of the set of browsers passing requests through the OreO, and reporting the group history in one of two ways:
 - Through a dynamic HTML document, created and updated by the OreO, and accessed at a well-known URL.
 - Through the backchannel to Ariadne, and presented by Ariadne as a multi-colored tree in a special window.
- A full-text indexing OreO, which creates a full-text index of all of the non-trivial terms appearing in the responses flowing through it. The OreO stores the index in a file system accessible by a CGI script sitting on a WWW server, and users may query the index via the usual forms interface. This OreO could serve an individual as well as a workgroup; the goal is to ease the problem of recalling the documents in which certain terms have appeared.
- A "protocol monitoring" OreO that is used to display the request/response traffic and to diagnose malfunctioning client/server interactions.
- A specialized "wafer" that acts as a gateway between unmodified browsers and WWW servers protected by DCE security mechanisms [OSF95c][10]. Communication between the wafer and the browser is via proxy HTTP, and communication between the wafer and the server is via DCE RPC [OSF95d][11].
- A "group annotation" OreO, that supports the Stanford annotation approach [Ros95][14] but allows that functionality to be available to users of a variety of browsers, not just the Stanford one.
- A "rewriting" OreO that encapsulates each anchor inside the Netscape *Blink* extension, making anchors easier to spot on monochrome displays.

As becomes clear through these examples, an OreO will often create some sort of information space (e.g., full-text index or group history) that may be made available to the user through a variety of mechanisms, such as

- Query forms (e.g., the OreO writes data to a file that a CGI script can use to answer queries on the data)
- A well-known URL (e.g., the OreO writes an HTML page to a location that a server can access to respond to requests for the information)
- Interprocess communication (e.g., the OreO can send information updates or commands to a browser via a control channel to get the browser to display the relevant information to the user)
- URLs that the OreO serves itself (e.g., the OreO traps requests to a particular part of URL space and returns the corresponding HTML document, which it might create dynamically)

Toolkit

To ease the task of building these transducers, we have produced a toolkit [OSF95a][8] which allows the filling developer to focus on the application-specific aspects of the transducer. In the initial version of the toolkit, we provide a "shell" that implements the "wafers" between which the filling is placed. The developer may use any program development system to create the filling, which is simply executed by the shell after it performs appropriate setup functions. The shell ensures that the filling is connected into the request/response stream, so that the developer can simply operate on the contents and ignore the network-specific issues.

Processing Modes

The shell supports four processing approaches:

- At most one request/response pair extant at any time, each pair flowing through a newly instantiated filling.
- At most one request/response pair extant at any time, flowing through a single, persistent filling.
- Multiple outstanding request/response pairs flowing through a single, persistent filling.
- Multiple outstanding request/response pairs, each one flowing through its own filling instantiation.

Further, the filling developer may configure the shell so that the filling handles HTTP requests only, HTTP responses only, or both requests and responses; the shell handles the requests or responses (as a pass-through) if the filling does not.

Implementation

The OreO shell is implemented as a single Unix process that runs as a Unix daemon; this is due largely to the original example OreO's origin as the `main()` for the CERN (now W3C) HTTP daemon process (*httpd*). The shell accepts a series of command-line arguments that indicate the TCP/IP port on which it should accept connections, the program that it should execute to process the HTTP request and/or response, and whether the program should be executed in parallel or sequentially.

For each new connection, the shell can either open a connection to the downstream process (which can be specified either as an environment variable `OREO_PROXY` or via a command line argument), or simply allow the filling code to determine the destination for the request. The latter mode allows filling code to support new protocol implementations (such as an RPC based protocols).

A Single, Persistent Filling

If a single, persistent filling has been specified, the shell `fork()`s and `exec()`s a copy of the filling: this filling is expected to meet the rules for the design of a *co-process*: the process must retrieve a pair of Unix file descriptors corresponding to the client and server sockets from a private IPC channel created between the shell and the co-process. Once the process has retrieved these sockets, it must send an acknowledgement back to the shell (the above is encapsulated in an API provided as part of the toolkit). At that point, the co-process is responsible for the connections: it must read and write to the appropriate sockets, and close them as necessary.

The issue of multiple versus singular request/response pairs is thus the responsibility of the co-process. Normally, the processing will be singular, since the co-process will effectively execute the following loop:

```
do {
    get a connection ;
    process the connection ;
} while(1);
```

Newly instantiated fillings

If a single persistent filling has not been requested, then the shell determines what kind of processing has been specified. The options are as follows:

- process the HTTP request, the HTTP response, or both (but provide a different filling for the request and response streams).
- process both the request and response streams, but do so in a single filling.

In the first case, if the filling chooses not to handle the request or response, the shell simply reads and writes data from one socket to the other. Each filling is set up appropriately: a request filling has its standard input set to the client socket and its output set to the server socket; the process is reversed for a response filling. This model is useful if there is no need to maintain state in the filling between the HTTP request and response streams, and is most similar to the notion of Unix filters (one way transforms).

In the second case outlined above, the shell sets the filling's standard input to the client socket, and the standard output to the server socket; a supplied API hides this implementation detail from the filling writer. This mode allows the filling writer to maintain internal state across the HTTP requests and responses.

Future versions of the toolkit should add new levels of abstraction for the developer; for example, we have specified a higher-level API that can present the requests and responses as pre-parsed HTML entities, so that the developer does not have to replicate that effort. Further, a toolkit might provide appropriate interfaces to the underlying WWW protocol support.

Performance

To gauge the effects of OreOs on HTTP request/response latency, we measured the additional delay introduced to a series of HTTP transactions by a PERL-based pass-through OreO (one that merely forwards requests and

responses without change). Our results are encouraging: the delay experienced was between 3% and 6%, depending on the mode of the OreO. We also performed simple tests to determine whether humans could perceive the additional delay. Four identical browsers were configured to point through various configurations of OreOs and proxies (the "worst" being a chain of four pass-throughs), and test subjects were asked to identify each of the four configurations; the subjects could only identify the browser proxied through four pass-throughs. This suggests that users are already conditioned to the variability experienced in network transactions, and the addition of a small extra delay does not stand out. The OreO shell appears to add no perceptible delay--the filling, however, can be arbitrarily complex and therefore add arbitrary delay.

Futures

We have already begun thinking about extensions to our basic model. These extensions fall into three general categories: implementing the OreO toolkit on multiple platforms, extending the notion of filtering agents themselves, and increased browser/OreO interactions.

Other Platforms

At present, the OreO shell has been compiled and tested under HP-UX and under OSF/1 running on Intel platforms. Time and resources allowing, we plan to re-implement the OreO shell under Microsoft Windows/NT. This will require minor modifications to the networking code (vanilla BSD Unix to WinSock compliant code), as well as modification to the process creation and process execution model. The latter should not prove difficult, as the details of this mechanism have been encapsulated in a single routine. Other possibilities including re-implementing the OreO shell to run as a Windows/NT service.

Extensions

We have several ideas on how to extend the notion of filtering agents, as illustrated by the OreO. These ideas break down into three categories:

- Improvements to the OreO shell,
- Improvements to the OreO/Browser interaction model, and
- Improvements to the actual processing of content.

Improvements to the OreO shell

We believe that the notion of the OreO shell is a good one: a layer of code that isolates the actual filling from the details of obtaining and processing network connections. The inspiration for this model is the standard input and output model for the Unix operating system [KER84]: programs can read from the standard input unit and write to the standard output unit without regard for whether these are files, terminals, network connections, etc.

Our initial thought was that the filling developer would encapsulate the invocation of the OreO shell and the appropriate filling code inside a shell script. While this has been true, we have imagined other models that would provide enhanced functionality in packaging and exporting transducing services to a user community. At present, if one wished to support several long-running OreOs, one would have to arrange explicitly for each script to be run when the machine is initially started. The following suggestions offer improvements to this situation.

OreO shell as *inetd*

In this model, the OreO shell is implemented similarly to the standard Unix *inetd*. Transducer fillings are configured similarly to the specification in the *inetd.conf* (perhaps specifying service name, protocol, port, program to run, etc.) As in the current *inetd*, the OreO shell would accept connections on the various TCP/IP ports listed in its configuration file. New connections would result in a new process being created to run the filling code. Stdin, stdout, and stderr for the filling would be connected to the upstream (client) program by the shell; the filling would be responsible for connecting to the downstream (proxy) server by either reading a command line argument or an environment variable.

OreO shell as *portmapper*

In this model (inspired by the RPC daemons of both ONC and DCE [OSF95d][11]), the OreO shell functions as a registrar for the above transducers. Transducers register their services with the shell, at which time the shell listens on the port specified by the transducer. Clients connect to that port, and the shell redirects the connection to the appropriate IPC channel established between the transducer and the shell. Such a technique would permit dynamic registration of transducer code: the interface for clients would remain the same (proxy HTTP), whereas the

interface between the shell and the transducer code would become more complex.

OreO shell as request broker

In this model, the OreO shell functions as a location-independent agent registrar, incorporating aspects of both the DCE *rpcd* and the DCE CDS (Cell Directory Services) servers. In addition to the functionality represented by the *portmapper* approach above, the shell would also be responsible for updating and maintaining a distributed database of agent functionality, such that an individual shell would be able to redirect a request for service to an appropriate location.

Generalized Agent Factory

In this model, the OreO shell becomes a "generalized agent factory" similar to the Softbot[13] or Sodabot [3] environments. Individual transforms are coded as functions that transform the HTTP stream as it is passed from transform to transform. The user may choose to implement these transforms in a programming language of some kind that provides its own specific GUI and other functionality.

OreO/Browser Interaction

We also intend to explore transducer/browser interaction. At present, our *Ariadne* browser sends an additional HTTP header (*X-BackChannel:*) that indicates a host, protocol, and port number on which the browser is willing to accept connections. This header is recognized by our OreO agents; servers simply ignore them. One can imagine extensions to this mechanism that are similar to the current *Accept:* headers, except that these headers indicate languages that the browser is willing to process: *Safe-TCL*, *Python*, *Java*, etc.

Finally, provided that browser's network point-of-presence is known (host, protocol, port), one can imagine using the NCSA Common Client Interface (CCI), the Spyglass Software Development Interface (SDI), or the NetScape API to communicate between a browser and an OreO. At this time, however, the mechanisms for establishing that point-of-presence are loosely specified, so it is not clear how well the above interfaces will work when the transducer would be running on a separate machine.

Improvements to processing information content

At present, the OreO shell presents a byte-stream interface to its client "fillings." While this is a very general and flexible model, it is too low-level to provide the kind of productivity improvements that we had initially hoped for when designing the OreO shell. Achieving the next level of productivity will require a higher level of abstraction: we have been evaluating the W3C's Library of Common Code (*libWWW*)[5] as a basis for that higher level of abstraction. We believe that, at a minimum, the next level will provide an abstraction of an HTTP request/response object: a series of HTTP headers optionally followed by an opaque content body. Parsing of the input byte stream, then, becomes part of the process of constructing this object; destructing this object might include converting it to a byte-stream and directing it to the downstream sink. This functionality would be provided by the API and not directly by the filling code itself.

Once the request/response object has been constructed, we then need to determine how to transfer this object between various OreO transducers. At present, this process is quite inefficient, in that each transducer must construct the request/response object, transform it, and then re-convert it to a byte-stream representation in order to pass it on to the (potentially) next transducer. Transducers may desire to see only the HTTP headers or the content body, or both. Modifications to content may require modifications to the headers (e.g., conversion of GIF images to JPEG would require modification to the Content-Type header). The above model then leads to a view of transducers as functions in a programming language, or as "functors" (functions as instantiable objects) [4]: the HTTP protocol object is passed from one such functor to another based upon some user-specified sequence.

Finally, we may wish to expand on the notion of the transducer as HTTP server. As suggested in the introduction, the CERN HTTP proxy server with caching [7] enabled is one kind of transducer. Another kind of transducer is an OreO that manages its own virtual Web-based namespace that has no mapping to an underlying file store. In this model, the OreO simply caches various information in memory: browsers send proxy requests to "service.machine.org" which fulfills these requests from memory. The CGI interface to this OreO would be to invoke internal functions with the appropriate arguments: each function returns an object of type HTML.

We are beginning to work on how OreOs might communicate with each other for control purposes, and to share information. At this time, we are pursuing the notion of implementing a network blackboard via HTTP (the blackboard server would be implemented as a stand-alone OreO) using the techniques described above (OreO manages its own internal namespace, etc.)

Conclusions

We have presented a novel approach to introducing specialized processing on the HTTP requests and responses that flow between WWW clients and servers. We generalized the notion of WWW proxy servers to that of application-specific proxies that act as transducers on the HTTP stream. Our prototype OreOs demonstrate the utility of the concept, and our toolkit aims to ease the task of building such OreOs, without adding undue performance penalties. We encourage experimentation to determine the kinds of application settings to which such an architectural component is especially suited and ways in which to extend the WWW architecture further.

Availability of Software

FTPable source and binaries for HP-UX (and possibly other platforms) are available by anonymous ftp from riftp.osf.org in `/pub/web/OreO`. Please read the copyright notice.

References

1. Abelson, H. et al., *Structure and Interpretation of Computer Programs*, MIT Press, Cambridge, MA 1986.
2. Behlendorf, B., *A Proposal for Non-Intrusive Community Content Control Using Proxy Servers*, <http://www.organic.com/Staff/brian/community-filters.html>
3. Cohen, M., *The SodaBot Home Page*, <http://www.ai.mit.edu/people/sodabot/sodabot.html>.
4. Coplien, J., *Advanced C++: Programming Styles and Idioms*, Addison-Wesley, Reading, MA, 1992.
5. H. Frystyk, *Library of Common Code*, <http://www.w3.org/hypertext/WWW/Library/>.
6. Kernighan, B., Pike, M., *The Unix Programming Environment*, Prentice-Hall, Englewood Cliffs, NJ, 1984, "Filters", pp. 101-132.
7. Luotonen, A., and Altis, K., *World-Wide Web Proxies*, <http://www.w3.org/hypertext/WWW/Proxies/>.
8. OSF RI World-Wide Web Agent Toolkit (OreO), http://www.osf.org/ri/announcements/OreO_Datasheet.html.
9. Ariadne, http://www.osf.org/ri/announcements/Ariadne_Datasheet.html.
10. , *DCE-Web Home Page*, <http://www.osf.org:8001/www/dceweb/DCE-Web-Home-Page.html>
11. , *OSF Distributed Computing Environment*, <http://www.osf.org:8001/dce/index.html>
12. *Wide-Area Browsing Assistance for the World Wide Web*, <http://www.osf.org/www/waiba/>.
13. Perkowitz, M., *Internet Softbot*, <http://www.cs.washington.edu/research/projects/softbots/www/softbots.html>.
14. Roscheisen, M. and Mogensen, C., *ComMentor: Scalable Architecture for Shared WWW Annotations as a Platform for Value-Added Providers*, <http://www-pcd.stanford.edu/COMMENTOR>.

About the Authors

Charles Brooks
OSF Research Institute
11 Cambridge Center
Cambridge, MA 02142
Murray S. Mazer
OSF Research Institute
11 Cambridge Center
Cambridge, MA 02142
Scott Meeks
OSF Research Institute
11 Cambridge Center
Cambridge, MA 02142

*World Wide Web Consortium
Massachusetts Institute of Technology
Laboratory for Computer Science
545 Technology Square Cambridge, MA 02142
jmiller@mit.edu*

Appendix B: *Dynamic Integration of HTTP Stream Transforming Services*

Dynamic Integration of HTTP Stream Transforming Services

Misha Davidson, Charles Brooks, Murray S. Mazer

Abstract

One way to add value to HTTP request/response transaction is to filter it through application specific HTTP stream transforming services. Given a set of such services a user needs a mechanism for dynamically selecting and applying a useful subset. This paper describes an experiment to integrate such multiple stream transforming services on the Web. We describe an architecture and implementation that allows users to select dynamically a subset of available services; we evaluate its performance and propose a number of ways that will make this scheme technically feasible for a widespread deployment in the Web community.

1.0 Introduction

Web sites vary greatly in the kinds of information they serve, the way this information is presented, and access policies protecting that information. However, all these sites can be considered *services*. Some of these services provide access to static data, some serve as gateways to frequently updated databases, some are used as navigational aids on the Web, and there are many others. Regardless of the kinds of data and underlying mechanisms, these services all share the same client-server model of operation a browser makes an HTTP request to a service and the service sends back some data in response.

There is a possibility, however, of providing a different type of service. Such a service does not actually serve the data but rather *transforms* a user request and/or the server response. The most familiar example of such a service is a caching proxy server that monitors user requests and serves documents from its own cache. A more general mechanism for building such services was outlined by Brooks et. al. [4]. Using this mechanism, we have built a number of services including Group Annotation, Document Correlation and Full Text Indexer [10]. Other groups have also built transforming services, for example WebFilter [3], Signet [1] and W3-Access for Blind People[11], or used transforming services to implement payment protocols (e.g. Millicent [6]).

These services are useful for increasing personal and group productivity and communication. Here are some examples. WebFilter decreases document transmission times and reader irritation by removing advertising images from the responses returned by some Web sites. A Full Text Indexer builds an inverse index of all the words in all the documents requested by a group of users connected to it. It provides a forms interface that allows users to share their explorations of the Web by querying this index for documents that contain a given set of words. The Document Correlation service appends a forms interface to every page, allowing users to search a local database for documents similar to the currently viewed document. The Group Annotation service allows users to create and share comments about any document on the web. The text of annotations associated with the documents is stored in a separate database, and when a document is retrieved, the Annotation service merges the text of annotations into the text of the document.

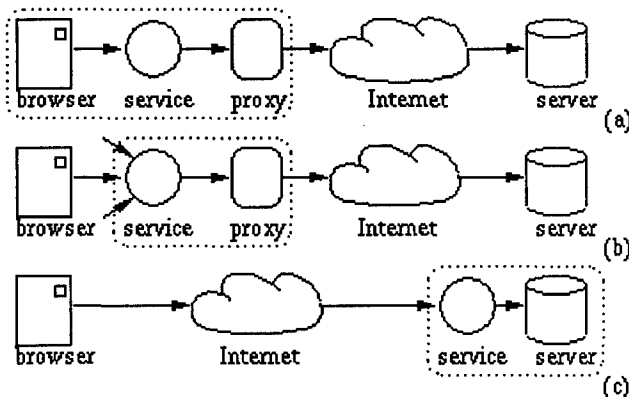


FIGURE 1. Transforming services may function as (a) Personal proxies, (b) Group proxies running on the 'client-side' and (c) 'server-side' processors.

Because transforming services can "transparently" fit into the HTTP stream, these services may be used in multiple ways in the Web architecture (as illustrated in Figure 1). When browsers proxy to these services they can be used as personal or group proxies. When running on the "server-side" directly in front of the server, they can provide services to the community accessing that server.

1.1 Problem Definition

Even though transforming services are potentially very useful and powerful tools, several costs arise in using such services. Using a transforming service involves an extra TCP connection per request, plus the processing time within the service. In addition to latency concerns, there may be concerns about privacy when using an untrusted service, or about monetary cost if the service provider charges for its use. To make services appealing to users, the benefits of using a service should be greater than the costs.

In the case of traditional Web services that provide data in response to a user's request, there is never a question of combining multiple services in one request. If the user wishes to find a weather forecast and to check stock quotes, she first directs the browser to issue a request to a weather server and then another request to a stock quote server. The situation is different with *transforming* services. One may wish to read annotations while searching for similar documents and logging the document content in the Full Text Indexer. Applying multiple transformations to a single document requires a way of combining multiple transforming services.

Moreover, it is not enough simply to combine a fixed set of services. As user needs change in various contexts, users should be able to select the services appropriate to their needs. For example, consider two scenarios: first, of someone using a search engine to find a set of interesting documents on the Web; and second, of the same person examining in depth the pages he found. When a person is using a commercial web index to find an interesting document, his preference is for getting pages as quickly as possible, and for being able to go back to some of the documents that were examined during the search. This situation calls for the use of two services WebFilter to speed up the retrieval of search results, and a Full Text Indexer to keep track of all the documents that were examined.

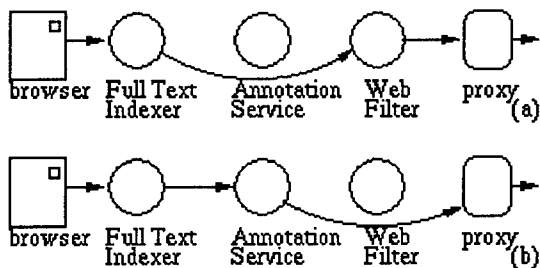


FIGURE 2. Dynamic selection of Transforming services. (a) Enabling WebFilter and a Full Text Indexer when using a Web index server. (b) Enabling Annotation and Full Text Indexer when examining a site of interest.

When the person finally finds the right pages, he may want to examine them in detail. He may want to see if anyone else has annotated these pages, or perhaps he may want to make a comment. This situation warrants the use of an Annotation service and justifies an increased latency of page retrieval by the benefit of getting and sharing additional information. The Full Text Indexer is useful here as well, as it will make it easier for other members of the group to find these pages and annotations later. However now that the search is over, use of WebFilter would only slow things down, and it should be disabled.

The above scenario and Figure 2 illustrate the problem that this paper addresses. Users may need to combine multiple transforming services. As users change the context of their work on the web, their notion of costs and benefits changes as well. In order for the benefits of transforming services to outweigh their costs, users need a mechanism that allows them dynamically to select the appropriate services.

The rest of the paper is structured as follows. We discuss our solution to the problem of dynamically selecting transforming services, examine its performance and critique it. Then we propose two design alternatives that deal with the drawbacks of our initial solution, and evaluate their benefits. We conclude by proposing a promising direction for future work.

2.0 Proposed Solution -- Switching Service

2.0 Proposed Solution -- Switching Service

This section describes a design of a *Switching Service* that allows users dynamically to select multiple transforming services. The Switching Service operates under the following model. There is a session-state associated with every browser. A browser's Session-state is a list of services that are applied to the requests sent from and/or responses sent to the browser. Session-state does not change between requests, except when the user chooses to change it.

The problem of applying multiple services to a request/response transaction can be subdivided into two separate tasks, as illustrated in Figure 3: Service Selection and Service Routing. Service selection allows a user to change the session state. Service Routing applies the selected services to the requests issued by the browser and to the document returned.

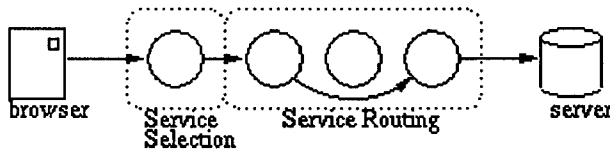


FIGURE 3. Switching Service is composed of two parts Service Selection and Service Routing. Service Selection handles the interactions with a browser and chooses the services; Routing applies the selected services to the request and the response.

Such division of functionality dictates the structure of the Switching Service. It is composed of two subsystems. The Service Selection mechanism handles the browser interactions and defines the services that should be applied to a request. The Service Routing subsystem performs the actual routing of requests and responses among services. This section first examines the Routing subsystem, then describes the Service Selection mechanism, and concludes with an end-to-end example.

2.1 Service Routing

Assume that the Service Selection mechanism has determined the session state of a request; now the Routing Service has to apply the selected services to it. In the case of transforming services, this means channeling the request and response through the services selected. Applying a single transforming service is similar to proxying through it in HTTP. However, HTTP allows only a single level of proxying. Therefore, if multiple services are selected, a special mechanism supporting multiple levels of proxying is required. Moreover, as it was shown in Section 1.1, simply chaining a number of HTTP stream transducers together is insufficient, because service selection may change at any time. Thus, it is necessary to provide a mechanism that allows stream transducers to connect to each other dynamically.

To accommodate this need for dynamic, multi-level proxying, we have developed a simple extension to the HTTP protocol [2] that supports dynamic chaining of transducers. In order to record the sequence of services that should be applied to the request, we introduced an *X-Routing* header. The value field of the header contains a name of the service that should be applied to the request. Any request can have zero or more *X-Routing* headers. If there are no *X-Routing* headers in the message, no services should be applied to it. If there is one or more, the services should be applied in the order in which they appear in the headers.

To implement the described functionality of multi-level proxying, we have added a layer of interpretation to our stream-based services. Normally a stream transducer has its proxy specified at start-up time. Every time a transducer receives a request, it processes the request and then forwards it to that pre-defined proxy. The interpretive mechanism we have introduced alters this behavior. After processing the request the interpretive layer executes the following algorithm:

1. Read headers of a request
2. If there are no *X-Routing* headers, pass the request through to the designated proxy
3. Else, take the following actions:
 - Remove the first *X-Routing* header from the request
 - Map the name of the service from the removed header onto a network address
 - Forward the request to that address

Execution of this algorithm in every transducer running within the Switching Service results in request being routed through all the selected services. By the time the request leaves the Switching Service there are no *X-Routing* headers left in it. Section 2.3 illustrates this protocol in detail through an example.

X-Routing headers left in it. Section 2.3 illustrates this protocol in detail through an example.

2.2 Service Selection

To control the services that are applied to a request and a response going through the Switching Service, it is necessary to associate each request with the session-state of the browser that issued it. Such association in the HTTP context requires every request to carry authentication information, thus forcing every user to authenticate in the beginning of every session. In some cases this may not be a problem as the server which provides information to which services are applied may require user authentication anyway. However, in case a server is unprotected, or the Switching Service is used as a proxy to access information coming from a number of different servers, this approach does not work.

Therefore, instead of making the Switching Service map authorization information onto session-state, we have chosen to send the session-state along with every request using Netscape cookies [9]. Cookies allow browsers to accept information from servers and proxies, store this information, and later send it along with HTTP requests. This way, the browser is responsible for maintaining session-state, and the Switching Service is stateless. A useful implication of this design is that if the Switching Service crashes and is later restarted, the browsers will receive the same services that were enabled before the crash. Unfortunately, at present cookies are only supported by the Netscape browser.

The encoding used to represent the service state of a browser is straight forward. Every service is represented as a cookie with a name *ServiceN* and a value that is either *on* or *off*. When a Switching Service receives a request with a *Cookie* header field containing *ServiceN=on* (see Figure 4(c)), it routes the request through *ServiceN* by inserting an appropriate X-Routing header (see Section 2.1). Alternatively, the Switching Service does not route the request through services whose cookie values in the request are *off*.

When a browser makes a first request that goes through a Switching Service, it has no cookies associated with the Switching Service. Therefore, no cookies are included in the request header (see Figure 4(a)). Upon receiving a request without a *Cookie* header in it, the Switching Service determines that this request is a beginning of a new session. The Switching Service then passes the request through without applying any services. When the response comes back to it, the Switching Service inserts in it the following header

```
Set-cookie: ServiceN=off;
```

for every service that it controls. Thus, after the first connection the browser is configured to request no services.

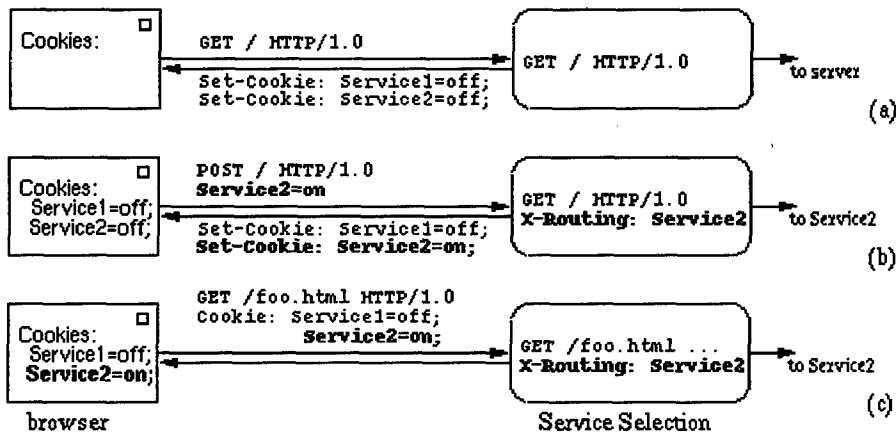


FIGURE 4. Interactions between browser and the Switching Service. (a) Initial Request: a browser first connects to the Switching Service, no services are enabled, and the browser gets the initial set of cookies plus a service selection form; (b) Service update: a user selects Service2 and submits a service selection form, POST request is rewritten as GET, Service2 is enabled and cookies are changed; (c) Normal Operation: a user requests a document the cookies are interpreted and corresponding services are applied.

The Switching Service also implements a special protocol that allows browsers to change a set of services that are applied to the requests (see Figure 4(b)). The Switching Service appends a form, with a check-box for every service controlled by it, to the bottom of every page returned to the browser. The check-boxes in that form are selected according to the session-state read from the cookies in the document request.

selected according to the session-state read from the cookies in the document request.

A user may change the set of services applied to the requests issued by the browser by selecting the desired services on the appended form and submitting it back to the Switching Service. When the Switching Service receives the resulting POST message, it interprets the body of the message to determine the new session-state of the browser, then it reloads the current document with the newly selected set of services applied to it and inserts a Set-cookie header that reflects the new set of services into the response. The detailed sequence of events proceeds as follows:

- The browser sends a POST request to the location of the current document, with the request body consisting of a series of ServiceN=on.
- The Switching Service traps the request, reads its body and determines which services constitute the new session-state and should be applied to it.
- The Switching Service changes the request method from POST to GET, removes its body and then routes it through the services selected by the user by setting the appropriate X-Routing headers.
- When the response comes back through the Switching Service, it sets Set-cookie header values to on for services specified in the POST request body, and to off for services that were not.

Consequently, the browser effectively reloads the current document with newly selected services applied to it, and it also gets a set of cookies that reflect the users new choice of session-state. On the subsequent request to the Switching Service, the browser will send the newly received cookies, thus enabling the Switching Service to apply the set of services selected by the user last time.

2.3 An End-To-End example

Suppose that a Service Selection component receives a GET request that contains the following line in its header (as illustrated in Figure 5):

```
Cookie: Service1=on; Service2=off; Service3=on
```

Consequently, the Service Selection mechanism determines that services Service1 and Service3 should be applied to the request. Thus it inserts the following lines into the header of the message:

```
X-Routing: Service1  
X-Routing: Service3
```

Then it executes the routing header interpretation algorithm described above in Section 2.1. That results in the Service Selection Mechanism removing the first X-Routing header and forwarding the request with one remaining X-Routing header to Service1.

Service1 in turn performs its transformations on the request and then executes the routing algorithm, removing the one remaining X-Routing header and forwarding the request to Service3 with no X-Routing headers in it.

Service3 performs its transformations on the request and attempts to execute the above routing algorithm. However, because there are no X-Routing headers in this request, it simply forwards the request to its pre-defined proxy outside the Switching Service.

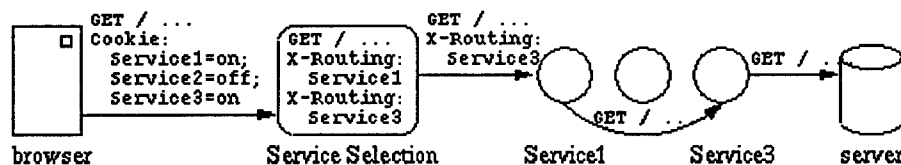


FIGURE 5. Processing of a request within a Switching Service. The Service Selection mechanism translates cookies into X-Routing headers, and services redirect the request accordingly

Meanwhile, the stack of connections from browser to Service Selection to Service1 to Service3 remain open. When Service3 receives the response from its proxy, it performs its transformations on it, and then sends it back to Service1. Similarly, Service1 performs its transformations on the response and sends it back to Service Selection mechanism. Service Selection adds the service control form to the bottom of the document and returns it to the browser.

2.4 Service Level Protocols

The above design of the Switching Service allows any transforming service that obeys the HTTP specification [2] to be used within the Switching Service. However, in order to insure that all components can work together within the Switching Service, we have introduced a small number of additional requirements for service behaviors.

2.4.1 Prefixing Service Input

Normally a service within the Switching Service simply accepts a request, processes it, and passes it to the service's proxy. When the response comes back from the proxy, the reverse action is taken. Sometimes services need to receive user input that is specific to that particular service, and there is no need to pass it to the proxy. An example of such behavior is an Annotation service, that accepts a user annotation along with information about the document and location within that document where the annotation should be placed.

This situation requires the use of the POST method. However, in Section 2.2 we described how all POST requests that come to the Switching Service get rewritten as GET requests. In order to allow services within the Switching Service to receive POST requests, the Switching Service does not convert requests POSTed to a location which begins with a predefined string (for example `/post-bin/`). In addition, in order to allow integration with other software that makes use of the CGI capabilities of traditional HTTP servers, POSTs to locations beginning with `/cgi-bin/` are unaltered as well.

However, servers may map their CGI directories onto URL paths rather than ones starting with `/cgi-bin/`. A more general solution would be to introduce a special `Service_Update` field in the body of service selection request. Using this field the Switching Service will be able to distinguish service update requests from the POST requests directed to other services, regardless of the URL path.

2.4.2 Limiting Service Interaction

Sometimes it may be desirable to limit interactions among the services. For instance, it is logically incorrect to annotate error messages or input request forms. Therefore we have introduced a special `no-modify` pragma.

This simple protocol insures that a transducing service will not modify HTML generated in response to special conditions, such as errors, warnings or user input requests. Any service may insert the `no-modify` pragma into the header of a response in case the document that the service is returning should not be altered by other services. When any service sees a response whose header contains the following line:

```
Pragma: no-modify
```

it should pass the message through unaltered. Server errors are handled in the similar manner by examining the return status code. In general only documents with a return status code of 200 and content-type of `text/html` are processed by our services.

3.0 Evaluation

We have built a system that implements the Switching Service described above. It allows users dynamically to select a subset of available HTTP transforming services. Currently it uses a simple routing model to resolve issues of multiple service integration. This model could evolve to become much richer (e.g., classes of composed services), requiring more sophisticated techniques (e.g., end-point-oriented multi-hop routing from networking [5], message classes and dynamically evaluated, pre-specified routing from office systems [8]).

We have tested the Switching Service system with Group Annotation, Document Correlation and Keyword Highlighter services. The Switching Service ran on the server side, connected directly to an HTTP server. All components of this configuration ran on the same machine HP-710 running HP-UX. All HTTP stream transducers were implemented using the OSF Research Institute's Strand technology [4]. Most of the service code was written in Perl, with the remainder written in C.

Our tests show that the design described above functions correctly. In addition, the service interaction model outlined above provides a sensible solution for handling special cases of service behavior when combining multiple services in one stream. However, we found two problems that need to be resolved to make such multi-service systems truly useful. One such difficulty is in introducing new services into a pre-existing set of services that are running under the Switching Service. Another is high latency of processing the requests within a Switching Service. In the rest of this section we examine these two problems and consider the options available for fixing these problems.

fixing these problems.

3.1 Configuration control

The difficulty in adding to or deleting from the set of available services derives from the fact that every component within a Switching Service must have an identical routing table. This table is used by services to perform request routing as described the algorithm from Section 2.1. Routing tables are initiated at the start of the system. Adding a new component to the system requires changing all tables. Presently this is done by shutting down and restarting all the components with the new routing tables. There are three ways we can introduce services in dynamically:

- Extend HTTP to deliver update information to all services.
- Use addresses of services instead of their names.
- Keep the routing table in only one place.

The first option of extending HTTP to carry configuration information is undesirable, at least because it means making another change to the protocol and an introduction of a rather complex mechanism to insure that all the services see the right routing information at the same time.

The second option (using service addresses instead of translating a service name into its address) is a simple workable solution. However, associating a service with a particular address makes it difficult to replicate services. This particularly is a concern with computationally expensive services that may require replication due to high CPU time requirements.

The last option (having a single copy of the routing table) makes the task of updating it much easier, thus resolving the problem of changing the set of services available. One of our prototype implementations used a one-table approach by placing all routing capability into a single Routing Interpreter service (as illustrated in Figure 6) and having all other services communicate with it every time they processed a request. This approach, however led to doubling the number of connections and was prohibitively slow.

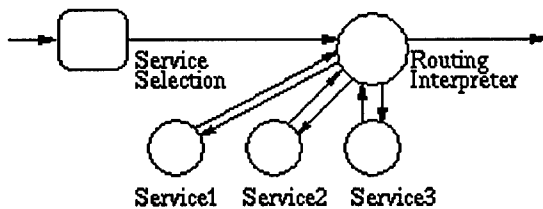


FIGURE 6. Putting a Routing Table into a single process doubles the number of connections required to route a request.

We believe that the correct way of implementing a single routing table solution is to use a Routing Name Server (RNS) that provides the mapping from a service name to its (possibly multiple) locations. In order to minimize communication overhead, services should use caching, so that the only times they have to communicate with RNS is when they see a new service in the routing header and need to determine its address.

Using a Routing Name Server approach has another advantage. It allows anyone to put a transforming service on the net, and for anyone to access it. After creating a service, its author can register the service with a number of Service Selection mechanisms, so that the service appears in the selection forms appended to the bottom of user's pages, or perhaps presented in a "service selection" frame or application. If the user selects the service, RNS will insure that other services can route a request through it.

3.2 Performance

Another problem with the Switching Service prototype is its high latency. Even though going through each service takes only a few seconds, the overall slowdown was noticeable, particularly with several services enabled. This slow down is caused by a number of factors, such as processing delay, inter-process communication and sharing a limited amount of computational resources. Here we examine these factors.

A simple improvement is to raise the efficiency of inter-process communication. Our prototype presently uses TCP sockets to communicate among services running on the same machine. Replacing these with UNIX domain sockets [13] can deliver a significant performance improvements. Better yet, if various services could run as threads within a single process, communication costs as well as process start up overhead would be even lower.

A more complicated, but potentially more effective way, of increasing performance is by replicating services in order to reduce competition for CPU time. In general, some services may be useful but computationally expensive (e.g. Annotation) and many people may want to use it at the same time. In order to provide a reasonable latency, there should be multiple copies of the service running; in the best case, one per user, running on user's machine.

In order to make client-side execution of services usable, we need to provide a way of managing services running there; in particular, bringing in and securely executing new services. These administrative actions should be easy to perform. Trying a new service should be as easy as getting a document from the Web.

All of the above considerations (multi-threading, client-side execution, network code and security) are addressed in the Java programming language [12]. Java may very well be the tool of choice for implementing the next generation of transforming services. In fact, the Krakatoa Chronicle project [5] has already implemented a filtering Java applet. However, this implementation only worked in the HotJava [14] browser. At this point it appears that no other browser supports the capabilities necessary to run transforming services. Therefore, to implement transforming services in Java we will have to build a proxy server that can run Java applets. One way of implementing such a server is to write it in Java, so that we can take advantage of the Java run-time system [16] to execute the service applets, much like HotJava does.

A Java-based Switching Service might operate as follows. Services would be written as filtering applets that transform the requests that go out of the browser on the net and the responses that come back, much like the services described above do. The individual service modules will be controlled by the special Switching Service applet. A user will upload the service modules of his choice into the proxy server, and the Switching Service applet will present her with an interface allowing to selectively enable or disable the use of each service, much like in the current prototype.

3.3 Future Work

Given a choice between the Routing Name Server and client-side Java-based approaches to extending the current work, we believe the latter is more promising because it addresses both configuration and performance problems and scales well. In addition, because it automates the introduction of new services, it is likely to be used by a larger community. Therefore, our next step is to study how the filtering functionality of Strand-based services can be implemented and combined in a Java environment.

4.0 Conclusion

We presented an architecture and implementation which allows users to exercise control over which transforming services are applied to the documents they request. The system functioned correctly with three services connected to it Annotation, Document Correlation and Keyword Highlighter. Even though the system was fully functional it was slow and hard to maintain. To make such multi-component transforming systems feasible for wide-spread use on the Web, we need to be able to run multiple services within a single process, to off-load most of the processing onto the client side, and to make addition of new services easy. A Java client-side proxy implementation is a likely candidate for fulfilling this task.

5.0 Acknowledgments

We thank Doug MacEachern for writing much of the code that allowed us to construct the experimental system, and Matt Schickler for creating an Annotation service, based on the work done at Stanford [12].

This research was supported in part by the Advanced Research Projects Agency (ARPA) under the contract number F30602-94-C-020. The Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Advanced Research Agency of the U.S. Government.

6.0 Bibliography

[1] B.S. Baker, E. Grosse, *Local Control over Filtered WWW Access*, Fourth International WWW Conference Proceedings, pp. 423-432, Boston, MA December 11-14, 1995.

[2] T. Berners-Lee, R. Fielding, H. Frystyk, *Hypertext Transfer Protocol HTTP/1.0*, <draft-ietf-http-v10-spec-04.html>, <http://www.w3.org/pub/WWW/Protocols/HTTP1.0/draft-ietf-http-spec.html>

- [3] A. Boldt, *Filtering the Web using WebFilter*, <http://emile.math.ucsb.edu/~boldt/noshit/>
- [4] C. Brooks, M.S. Mazer, S. Meeks, J. Miller, *Application-Specific Proxy Servers as HTTP Stream Transducers*, Fourth International WWW Conference Proceedings, pp. 539-548, Boston, MA December 11-14, 1995. <http://www.w3.org/pub/Conferences/WWW4/Papers/56/>.
- [5] D. Comer, *Internetworking with TCP/IP: Principles, Protocols, and Architecture*, Prentice-Hall, Inc., Englewood Cliffs NJ, 1988.
- [6] S. Glassman, M. Manasse, M. Abadi, P. Gauthier, P. Sobalvaro, *The Millicent Protocol for Inexpensive Electronic Commerce*, Fourth International WWW Conference Proceedings, pp. 423-432, Boston, MA December 11-14, 1995 <http://www.w3.org/pub/Conferences/WWW4/Papers/246/>.
- [7] T.Kamba, K. Bharat, M. Albers, *The Krakatoa Chronicle, An Interactive Personalized Newspaper on the Web*, Fourth International WWW Conference Proceedings, pp. 159-170, Boston, MA December 11-14, 1995.
- [8] M.S. Mazer and F.H. Lochovsky, *Logical Routing Specification in Office Information Systems*, *ACM Transactions on Office Information Systems*, Vol. 2, No. 4., October 1984, pages 303-330.
- [9] Netscape Communications Corporation, *Netscape: Client Side State HTTP Cookies, Preliminary Specification*, http://www.netscape.com/newsref/std/cookie_spec.html
- [10] Open Software Foundation, *WebWare Release 1.0*, <http://www.osf.org/mall/web/webware.htm>
- [11] L. Perrochon, A. Kennel, *W3-Access for Blind People*, Fourth International WWW Conference Poster Proceedings, pp. 92-93, Boston, MA December 11-14, 1995.
- [12] M. Röscheisen, C. Mogensen, T. Winograd, *A Platform for Third-Party Value-Added Information Providers: Architecture, Protocols, and Usage Examples*, <http://www-diglib.stanford.edu/rmr/TR/TR.html>
- [13] W.R. Stevens, *UNIX Network Programming*, Prentice Hall, Englewood Cliffs NJ, 1990.
- [14] Sun Microsystems, *The HotJava(tm) Browser, A White Paper*, <http://www.javasoft.com/1.0alpha3/doc/overview/hotjava/index.html>
- [15] Sun Microsystems, *Java Language Specification*, <http://www.javasoft.com/JDK-beta2/psfiles/javaspec.ps>
- [16] Sun Microsystems, *The Java(tm) Virtual Machine Specification*, <http://java.sun.com/doc/vmspec/html/vmspec-1.html>

Appendix C: *Pan-Browser Support for Annotations and Other Meta-Information on the World Wide Web*

Pan-Browser Support for Annotations and Other Meta-Information on the World Wide Web

Matthew A. Schickler, Murray S. Mazer, and Charles Brooks
OSF Research Institute
11 Cambridge Center
Cambridge, MA 02142 USA

Abstract

This paper describes an innovative approach for groups to create and share commentary about the content of documents accessible via the World Wide Web. In particular, the system described supports the creation, presentation, and control of user-created meta-information, which is displayed with the corresponding documents but stored separately from them. The typical use for this mechanism is to support annotations about documents accessed through browsing clients of the Web. In contrast to other approaches, the system described herein does not depend on changes in, or specializations to, Web browsers or servers. Rather, the system takes advantage of the notion of *application-specific stream transducers*; most of the functionality of the system is provided by a specialized proxy. We describe design considerations, the system architecture, usage scenarios, our initial implementation, and future work.

Keywords

Annotation; Meta-information; Virtual documents; Groups; Information merging; Proxy servers; Stream transducers; HTTP; World Wide Web.

Introduction

The World Wide Web is increasingly important as an infrastructure for group-oriented activities. As the Web becomes more prominent, users push for greater functionality to support their individual and team needs. Further, the users and their organizations seek the provision of that functionality without dependence on the features of any specific browser (or the whim of any particular browser provider). To support groups effectively, a system must enable communication among members about issues and documents of interest. Our interest is in supporting asynchronous communication. Electronic mail and newsgroups are well-known support tools for asynchronous communication, but they do not provide a natural mechanism for group discussions regarding shared document spaces, such as documents accessible from World Wide Web servers.

This paper describes an innovative approach to enabling group members independently to create and share commentary about the content of documents accessible via the Web. In particular, the system described supports the creation, presentation, viewing, and control of user-created meta-information, which is displayed with the corresponding documents but stored separately from them. The typical use for this mechanism is to support annotations about documents accessed through browsing clients of the Web. For example, members of a corporate functional team may wish to create a shared, evolving commentary about their strategy documents stored on a private Web server, and they may also wish to comment about the public information offered at a competitor's Web site. As another example, our research group may, in the course of a collaboration with another institution, wish to join an ongoing discussion about joint design documents hosted at the other institution. As suggested by Röscheisen et al. [RoMW], user-created meta-information may be used in ways other than annotations, such as trails, voting, and seals-of-approval, which support notions of editorial control in an often unvetted communication medium.

The following usage scenario illustrates aspects of our target usage model. The user, while viewing a document anywhere on the Web via her favorite browser, notices that one of her department colleagues has added a comment regarding a sentence in the document. When the user asks to view the comment, she is presented with both the comment and the opportunity to respond to it. Instead, she decides to comment about the whole document. To do so, she invokes the annotation submission mechanism. After creating the text of the annotation, providing a subject header, and citing a relevant URL, she submits the annotation. The user then notices that her annotation is now available for viewing by authorized members of her group. Furthermore, the annotation system notifies those group members that an annotation of interest has been posted.

In contrast to other approaches, the system described herein does not depend on changes in, or specializations to, Web browsers or servers. Rather, the system takes advantage of the notion of *application-specific stream*

transducers [BMMM]; most of the functionality of the system is provided by a specialized proxy [LA]. Several systems, such as CoNote [Da], HyperNews [La], and NCSA Mosaic Group Annotations [NCSAb], use a CGI script running on an HTTP server to provide annotation support. This design choice forces the user to direct all requests explicitly to that HTTP server; this is a reasonable choice *if all of the documents to be annotated reside on, and are controlled by, that server*. We believe this to be too restrictive---one should be able to create commentary about documents accessible from any server. Further, annotations in these systems may be attached either at the end of the document or in author-defined locations; again, this may be too constraining for productive group interaction. The Public Annotation System [Gr] prototype uses both CGI support and a modified HTTP server, making the support hard to use with other servers. The ComMentor system [RoMo] separates the functionality of an annotation system into a meta-information server (hereafter, "metaserver," for storing the annotations separately from the target documents) and a client-side component (for querying both the metaserver and the target document server and for synthesizing the two responses). This approach provides more autonomy and potential for scalability, but it unfortunately ties the user interface and query/synthesis functionality to a particular browser implementation; this strategy either inordinately restricts the user community or forces implementers to become tightly integrated with the browser implementations.

To address these potential drawbacks, we hypothesized that a specialized HTTP stream transducer could provide flexible, browser-independent, annotation support. As stated in Brooks et al. [BMMM],

[w]e suggest that, for some classes of client/server applications and their network transactions, substantial value may arise from inserting, into the communication stream, application-specific transducers that may view and potentially alter the message contents. We are testing this hypothesis in the context of the World Wide Web, by building a sample set of proxy-based transducers that are bound to the HTTP request/response stream. This approach extends the "standard" WWW architecture in a way we believe is both novel and useful.

Our initial prototype of a Group Annotation Transducer ("GrAnT") served as a proof of concept to support this hypothesis.

To facilitate construction of our initial prototype, we borrowed elements of the ComMentor prototype system from Stanford University [RoMo, RoMW], which relies on a modified Mosaic browser [NCSAa] with embedded annotation support. While it was not strictly necessary for us to cannibalize the Stanford prototype to demonstrate the value of our approach, this was a reasonable step, as their system design led naturally to a proxy-based approach (as acknowledged in [RoMW]). Our subsequent architecture retains the lessons, but not the detail, of the first prototype.

The rest of this paper proceeds as follows. First, we describe the key design considerations. Then we discuss our system architecture and review the concept of application-specific HTTP stream transducers. The following section describes the implementation and lessons learned. We then discuss the implications for new architectures and consider some outstanding issues, related work, and future directions.

Design Requirements

The base set of functionality we wished to implement in our prototype is as follows:

- For a requested HTML document, query an annotation database and merge the resulting annotations with the response into a coherent HTML document.
- Write an annotation for an HTML document. The target document need not be "controlled by" the user in any way; nor must the document author prepare the document to be annotated.
- Save the annotation to an annotation database.
- Permit multiple users to annotate the same document, sequentially or in parallel.
- Write a single thread of replies to an annotation.
- Support attachment of annotations to specific pieces of a viewed HTML document.
- Support a simple model of users and annotations that permits organizational grouping, user-filtered retrieval, and access control.
- Maintain user profile information which is stamped onto annotations.
- Provide a simple user interface for creating and selecting annotations.

Initial Architecture

Figure 1 illustrates our initial architecture, consisting of Web browsers, Web servers, an annotation-specific stream transducer, and an annotation server.

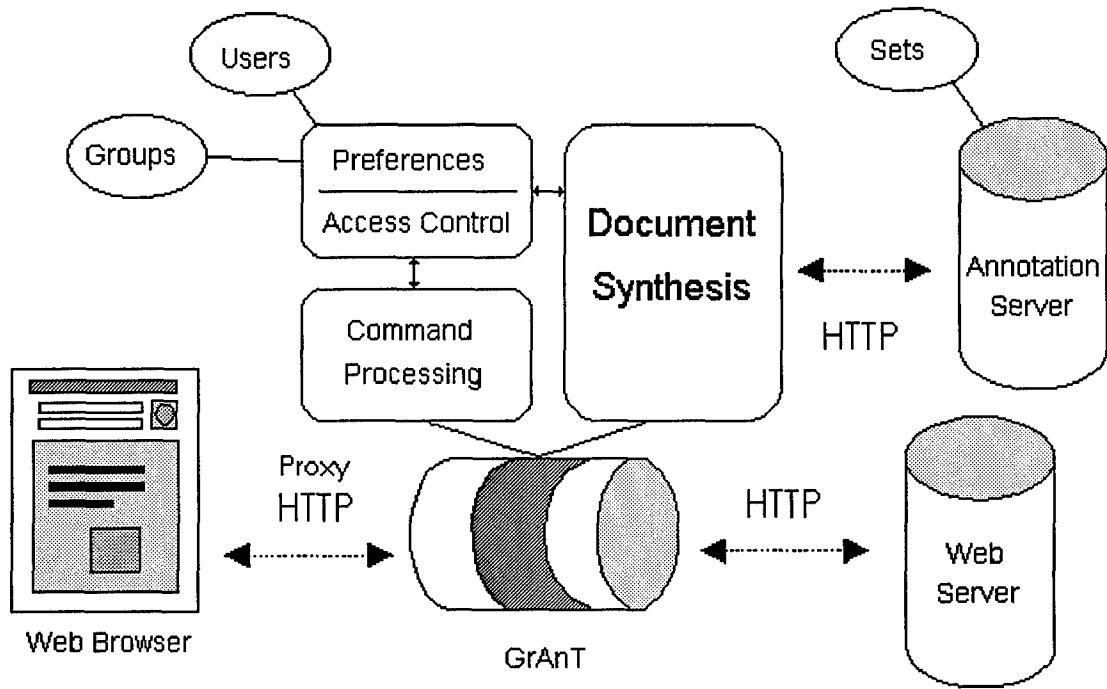


Figure 1: Initial architecture: annotation transducer, annotation server, unaugmented browsers, and unaugmented WWW servers.

The original prototype was able to leverage elements of the ComMentor system; in particular, we borrowed the *merge library* (to create a synthesized document with in-place annotations), the metaserver (to store and serve annotations), and the initial model of users and annotations. For comparison, Figure 2 illustrates the high-level ComMentor design.

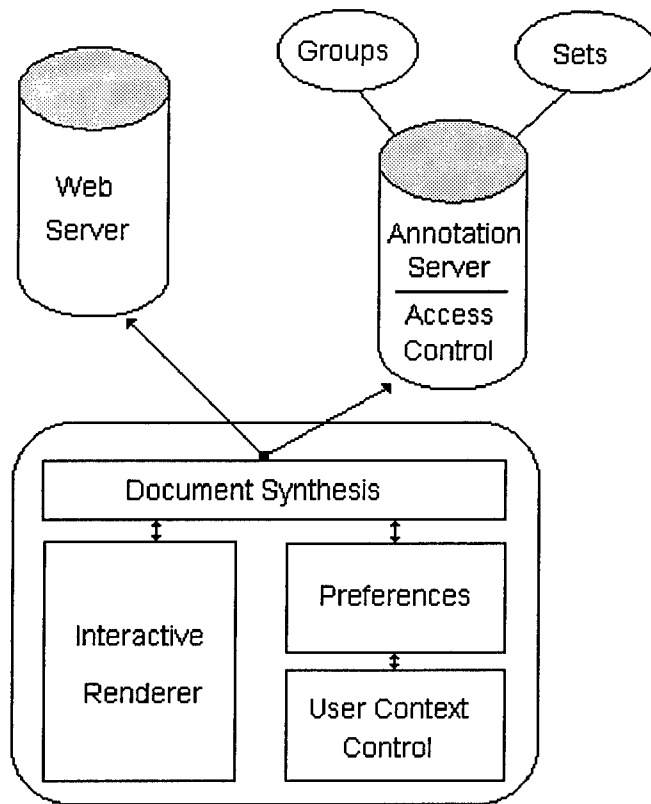


Figure 2: ComMentor architecture: modified Mosaic browser, annotation server, and unaugmented WWW servers.

The model of users and annotations is quite simple; this suited our initial purpose, but our approach can support a more complex model as well. In the prototype model, an individual user may be a member of one or more administratively-defined groups. Each annotation is a member of an administratively-defined *annotation set*, and each set resides on an annotation server. Conceptually, each set may represent an individual topic area, with the expectation that annotations written to a set are related to the topic of interest. For example, consider a workgroup in the purchasing department of a large corporation, browsing an on-line catalog of a competitor's parts; the workgroup may want to separate comments about component cost from their running discussion of product quality. This situation calls for two separate sets of annotations. Annotation sets also provide a user with a primitive way to filter annotations to suit the user's purpose. For example, a member of the purchasing group concerned at this moment with the quality of a particular part may elect to "turn off" all annotations that have to do with the cost.

Each group is permitted access to one or more annotation sets. Access groups are either public or private. Public access groups can be joined by any user of the service. A private access group can only be joined after administrative validation. Our goal was to ensure that the annotation service could support at least this model of grouping and access control; this goal was met.

For the purposes of our prototype, it was not important how or where the annotations were to be stored, only that the GrAnT could easily store and retrieve them via a well-defined mechanism. In the current implementation of the prototype, the mechanism is the HTTP protocol and the storage medium is a slightly adapted version of the Stanford annotation server.

Approach

We first describe a toolkit used to build the GrAnT. Then we discuss the user interface technique, after which we describe the structure of the transducer itself.

Strand Toolkit

We built the GrAnT by using a toolkit [OSFa] developed at the OSF Research Institute to allow a transducer developer to focus on the application-specific aspects of the transducer, instances of which we generically refer to as *Strands* (for "Stream TRANsducing Daemons"). As illustrated in Figure 3, the toolkit provides an 'outer shell' that manages the connections toward the client and toward the server, between which the application-specific code module is placed.

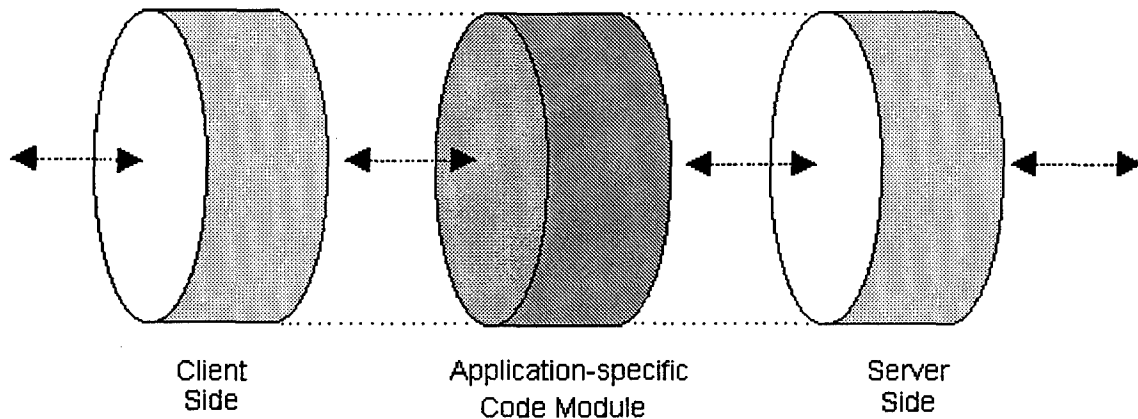


Figure 3: The Strand architecture

The developer may use any program development system to create the Strand code module, which is simply executed by the Strand after it performs appropriate setup functions. The Strand ensures that the module is connected into the request/response stream, so that the developer can simply operate on the contents and ignore the network-specific issues. A previous description of the Strand approach [BMMM] reports that the Strand outer shell adds a minor but user-imperceptible delay to the processing of HTTP request/response streams. The annotation service, then, is a Strand module written in C and executed within a Strand outer shell.

User Interface

The interface for interacting with the annotation service is implemented via HTML-based forms and controls. This choice follows from our design principle of not modifying browsers or servers. Further, this type of interface is relatively straightforward to implement within a Strand module, since all modules deal with HTTP at the byte stream level. Interface components support two types of functions:

- submitting new annotations*
- changing one's profile, thereby changing the annotations retrieved on one's behalf.*

The GrAnT appends control buttons to the bottom of each valid HTML response before forwarding it to the client. Buttons include "Submit Annotation," "View Groups," and "View Sets." In the ComMentor system, these functions are provided via pulldown menus within the Mosaic application itself. In our system, when the user presses one of the control buttons, the GrAnT captures the resulting HTTP request and guides the user with a series of HTML-based interactions.

Implementation

In this section we describe the flow of control within the GrAnT when the user views existing annotations, submits new ones, and uses the control functions to modify preferences.

Viewing Annotations: Flow of Control

When an HTTP request reaches the GrAnT, the module performs several steps:

1. creates a thread of control to handle this request/response transaction; this control thread performs the remaining steps.
2. forwards the request to the server specified in the URL.
3. authenticates the user and determines the user's profile (to determine which annotations to request in the next step).
4. authenticates itself to the annotation server. (Note that the authentication relationship is between the GrAnT and the annotation server---the user has already authenticated to the GrAnT. This is in contrast to the ComMentor model, in which the user authenticates directly to the metaserver.)
5. queries the annotation server for any annotations associated with the requested URL and the requesting user (steps 3, 4, and 5 proceed in parallel with step 2).
6. synthesizes the returned document with the associated annotations and returns the synthetic document to the client.

To synthesize annotations with the document, the merge component does the following actions:

- for in-place annotations, inserts in-place markers (such as [1]) to indicate the presence of annotations. The marker, placed directly after the target text, is an intra-document hyperlink to an element in a list appended to the document.
- appends a list of hyperlinks to the document. Each link corresponds to an annotation; when the user selects a hyperlink from the list, the associated annotation appears with any follow-ups (as an HTML document) in the browser view, with a "follow-up" control button. See Figure 4.

Annotation for <http://cf42.osf.org/ri/presentations/Boutique9502/DCEWebSlides.html>

Posted-by: "Matthew Schickler" (mas@osf.org)
In-reference-to: "Secure remote administration of internet services."
Annotation-set: "Server"
See-also: <http://cf42.osf.org/www/dceweb/ato/interface-spec.html>
Created-on: 08/23/95 13:41
Subject: "DCE RPC management interface"

I've drafted a simple specification of a DCE RPC management interface to the multi-protocol server. The hyper-reference gives the details.

Posted-by: "W. Scott Meeks" (meeks@osf.org)
In-reference-to: Original Annotation (follow-up)
Annotation-set: "Server"
Created-on: 09/14/95 11:30
Subject: "Reply to 'DCE RPC management interface' by Matthew Schickler"

This looks pretty good. Why don't you polish it up and then add a link in the main document?

[Write Follow-Up](#) | [View Groups](#) | [View Sets](#) | [Edit User Profile](#)

Figure 4: A simple annotation and follow-up.

User Interface: Flow of Control

The user interface implementation uses a *short-circuiting* technique that takes advantage of a Strand's ability to trap requests to certain URLs and serve them itself. That is, a Strand module may send to the client a custom response which is not a response returned by a downstream server. A module may use this technique to send Strand-generated error messages to a client based on some failed condition, such as a stream processing error. For the user interface, the GrAnT traps requests that correspond to command directives stimulated by the user's selection of a command button (which was previously inserted by the module itself). There are many ways to

indicate that a request corresponds to a command directive. The current implementation uses a reserved local host name that is effectively overridden by the GrAnT. Whenever a requested URL indicates the reserved host name, the module processes any query data sent via the GET or POST method as a command directive.

If the command directive is to change the state of the GrAnT (with respect to its general operation or with respect to the user's state), the module makes the change and returns a dynamically generated document to indicate the outcome of the operation. For example, when the user presses the "View Groups" button, a *view groups* directive is sent to the GrAnT. The module processes the directive by sending an HTML form back to the client. This form includes a list of groups, the user's current status in each group, and a set of radio buttons associated with each group. The user's status in one or more of the groups may be changed by selecting the appropriate radio buttons. Pressing "Submit" causes another directive to be sent to the GrAnT. This directive is similar to the first but is accompanied by a set of arguments that indicate any changes that were made to the user's group status. After the changes have been processed, the module returns the results of the changes as a new HTML page. The user may then continue browsing as usual.

The case of submitting a new annotation is discussed in the next section. The implications of this user interface choice are discussed further in the "Lessons Learned" section.

Submitting Annotations: Flow of Control

The flow of control for submitting annotations is very similar to that of editing group membership. Directives, with and without arguments, are passed to the GrAnT for processing. The key difference in this case is that the HTML form returned to the client is specific to writing an annotation. This page includes a number of input fields and radio buttons that can be used for entering the annotation text and selecting options. As illustrated in Figure 5, one of the options allows the user to attach an annotation to a specific piece of text in the document by entering this text in an input box (otherwise, the annotation is attached to the document as a whole).

Add Annotation

Attach annotation to:

Page

Text:

Write the annotation to the set(s):

Platforms

Channels

Client

Server

Middleware

Notification method: None Pop-up Window E-mail

The annotation will be written in: Plain Text HTML

Subject:

Please write the annotation below:

The release milestone needs to be pushed two months because of the extended requirements.

Figure 5: A form for creating annotations

Another option can be used to select whether the annotation will be written and displayed as plain text or as HTML. The command directive that is sent after the form is submitted contains a number of arguments that are used when posting the submission to an annotation server. On completion, the module returns the results of the submission and the user may continue browsing as usual.

Lessons Learned From First Prototype

The first prototype supported the hypothesis very successfully, in that users are able to read and write annotations, configure preferences, and so on—all of the design requirements were met. The group annotation transducer has been used successfully on the "client side" (with browsers proxied explicitly to it), on the "server side" (running at a Web server's advertised address, so that requests to that server run through it), and as a user-selectable service [DaMB].

The following problems were identified in building and testing the prototype:

- User Interface
 - Interface elements take up too much space
 - Forms-based interface is not powerful enough
 - Selection of annotation attachment points is inelegant
- The Annotation Server

- Multiple TCP connections and slow CGI scripts create significant overhead
- There is limited support for user-directed filtering of annotations
- There is no support for multiple reply threads that originate at a single annotation
- Scalable Architecture
- Authentication

We discuss each of these in more detail below.

User Interface

The current graphical user interface (GUI) exploits the capabilities of HTML forms. This class of markup proved to be an excellent tool for prototyping a user interface, because it is simple, somewhat powerful, and standard on all browsers. The disadvantages of using such an interface for a production quality system became apparent after building the prototype. The toolbar and annotations that were appended to a document took up too much space on the page. If a document spanned multiple screens, these items would not be visible at all times while reading the document. Editing user data such as group membership requires a three step process: go to the group page; submit the group page; and return to the original HTML document. Browser-augmented approaches such as ComMentor can offer more sophisticated control (e.g., menus) and display (e.g., popups and new windows). It is clear that the next version of the proxy-based annotation software will require a smoother interface. However, such an interface should remain a standard feature of all browsers on the market if we are to continue our goal of developing pan-browser annotation support.

Another problem introduced by the proxy-based approach was the selection of attachment points for annotations within a document. A natural method of selection involves indicating the attachment point with the mouse, by selecting a piece of text that is being displayed by the browser. This is straightforward to implement when using an approach that augments an existing browser. The developer has direct access to important information such as mouse events and internal buffers. Because our approach is browser-independent, in the most general sense, access to this type of information is restricted. Workarounds such as querying a window system manager (e.g., an XServer) for the current text selection are possible but are platform- and/or browser-specific. In order to maintain our pan-browser design goal, the prototype implements a simple HTML forms interface which adds an extra and sometimes frustrating step to the process of selecting a point of attachment. Instead of a direct selection, the user must cut-and-paste the target text into an input box provided by the form. Then, after the annotation is submitted, the GrAnT must perform extra processing in order to discover redundant information that characterizes the selected string of text. More problems arise when the selected string of text is not unique in the document. This can be handled inelegantly by rejecting the annotation outright (i.e., forcing the user to use a new text string) or making the user select the correct occurrence out of a list of ambiguous phrases. In either case, processing power and the user's time are wasted.

Several potential mechanisms exist for providing cross-platform, pan-browser, application-specific interface support, including Tcl/Tk [Ous] and the Java language, class libraries, and runtime [Su]. The ability to run an annotation applet on the client could provide an elegant solution to the problems identified in our prototype. Such an applet could contain toolbar buttons and a list of annotations for the current document. It would no longer be necessary to merge this information directly into the document. The applet could persist from page to page and could function, with the help of the GrAnT, as an extension to the browser. Since it is expected that the Java run-time system will be an integral part of all future web clients, this remains a pan-browser solution. This also permits very innovative presentation and interaction techniques, such as virtual layering of annotation, version, and other meta-information "on top of" the base document. One example of this is implementing inlined annotation markers as embedded Java applets. Clicking on the marker would pop up a window displaying the annotated text and the first few lines of the annotation itself.

Some of the problems inherent in selecting attachment points for inlined annotations may also be solved by doing away with the HTML forms interface and providing all the functionality of the annotation system in the annotation applet. For example, while a user is writing an annotation, the applet could display a stripped-down, textual version of the HTML. A section of the text could be selected as an attachment point for an inlined annotation. Ambiguities no longer exist because the applet is working with an exact point of insertion instead of a single, isolated text string that it must later match in the document.

The Annotation Server

Our prototype uses the ComMentor metaserver, which uses the Common Gateway Interface (CGI) [NCSAc] for storing and retrieving annotations. This approach introduces unnecessary overhead in a number of ways. It requires that a TCP connection be opened for each query that is made to the annotation server. In addition, the metaserver scripts provide slow service, because they must be executed each time the server receives a request. Performance of the annotation server is not the only issue. The prototype annotation server is limited in its

capabilities, offering only the most primitive database functions. An effective annotation service requires that users be able to filter and search for annotations based on a flexible set of criteria, such as author's name, date of creation, annotation type, and keywords in the subject header and content body. With the introduction of a database system for storing annotations, both user-directed filtering and a fully threaded system of replies for an annotation becomes easier to implement. Such a system will be implemented in the next release of the software.

Scalable Architecture

In the current prototype, each GrAnT assumes the existence of a single workgroup that uses a common, local annotation database. There are no provisions yet for permitting sharing among workgroups. In our development of the next generation of the system, we will incorporate an interoperability protocol, such as those under development for the Digital Library Initiative [DLI], to permit an annotation server to accept requests from network clients and process these requests using the local database. This permits the GrAnT to query other annotation databases, local annotation databases to share annotations and indices, and other kinds of clients to access the information.

Authentication

The current HTTP specification does not support proxy authentication. This type of authentication is needed by the GrAnT, because the services it provides are customized to the accessing user. In order to circumvent this problem temporarily, we developed a personal ID Strand Module, which adds an experimental header field to the HTTP request. This field, X-Proxy-Authenticate, uses the Basic Authentication method (encoded username and password) to identify the source of the request. A problem with this method is that it requires a Strand process to be running on each client machine (and introduces an extra TCP connection per request/response transaction). It is expected that the next revision of the protocol will contain support for proxy authentication or multiple intermediate authentication points. Otherwise, an annotation applet could include user profile information that could be sent directly when required. This is an adaptation of the approach taken within the augmented Mosaic implementation of ComMentor. Another alternative is to use security features of DCE [OSFc], as compellingly demonstrated in the DCE-Web system [Le,OSFb].

Futures

New Architecture

Figure 6 illustrates our new architecture for supporting group annotations, based on our experience with the GrAnT.

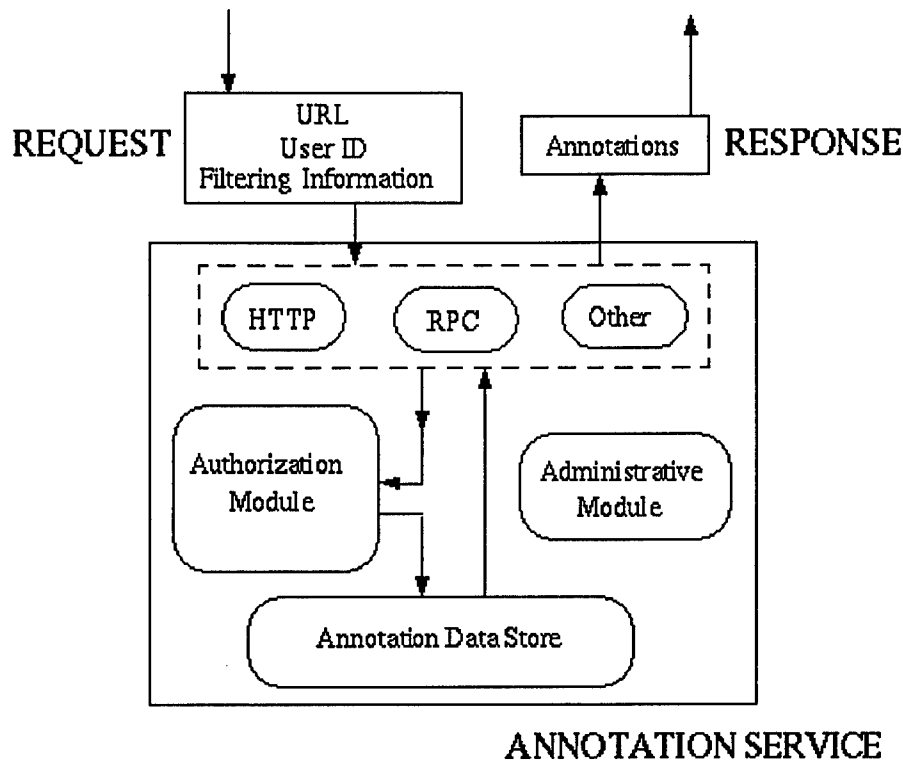


Figure 6: New annotation service architecture

The architecture derives from an analysis of the functionality provided by the GrAnT, roughly:

- request a document referenced by a URL, and request the associated annotations,
- merge the document and annotations,
- provide a user interface,
- manage user state,
- manage annotation and user grouping information, and
- support user authentication and authorization of requested access to annotations.

We believe that the latter two capabilities should rest with the annotation service, rather than with the annotation intermediary. Consequently, in the new architecture, the annotation server performs user authentication, request authorization, and storage and retrieval of annotations. (Perhaps not surprisingly, this division of responsibility is similar to that prototyped in the ComMentor system.) This functionality is accessible to clients via a variety of interface modules, including ones accepting HTTP and DCE-RPC connections. The clients might be a next-generation annotation Strand or other applications that need access to annotations, such as a web editor or a versioning mechanism. Each interface module performs user authentication and passes the incoming request (including the user's identity, the URL for which to retrieve annotations, if known, and a user-specific filter) to the core of the annotation server. The core component uses an authorization module to perform an authorization check, ensuring that the user is permitted to retrieve the specified annotations. Then it retrieves those annotations from the underlying data store, which is accessed via a generic annotation-oriented interface that hides the implementation of the data store (e.g., one of a variety of database systems, or a special purpose annotation storage engine).

A positive property of this architecture is that clients need not use a specific common access mechanism for querying and creating annotations. For example, one annotation server could serve some clients that use the strong security guarantees of DCE-RPC and other clients that use HTTP. Similarly, an interoperability protocol, such as that being developed within the Digital Library Initiative, can offer a diverse set of clients access to annotations maintained by a variety of annotation servers, even when the clients and annotation servers run within different administrative domains. In addition, this modular approach permits us to add functionality for availability and scalability, such as replication and index sharing.

Other Functionality

Although our prototype focused on attaching annotations to HTML documents, the service is really a more general system for attaching different types of meta-information to documents in the World Wide Web. For example, the notion of backlinks, or links to the pages that point to the current page, could be implemented by allowing the GrAnT to annotate documents automatically. When a user visits a page, the annotation service could automatically post an annotation that contains the URL of the referring HTML document. A list of all such "backlink annotations" for a document could be displayed at the user's request. As another example, after reading a page, members of a group could cast votes on the document's content, appropriateness, or style [RoMW]. This information could be stored as a series of annotations. When a document is fetched that contains a link to the document that was voted on, the votes could be tallied by the GrAnT and displayed next to the link. The system could also be used to create special annotations that act as landmarks. Each landmark would contain a comment and as many as two pointers, one that points forward to a new page and one that points back. Chains of landmarks, called trails [RoMW], could be formed in this manner. Trails could be created to allow people to discover information in new or more logical ways. The GrAnT could also be instructed to follow a trail to its completion, returning a page containing all of the links on the trail.

Another important way to extend an annotation service is to tie it in with other, related services, such as:

- a versioning mechanism, which permits annotations to be created and retrieved in correspondence with the appropriate versions of documents as they change.
- a document search/correlation mechanism, so that annotations may be searched for relevance, along with other documents (such as electronic mail and newsgroups).
- a notification mechanism, so that users are notified of annotations of interest.

We will be addressing these related services in future systems.

Conclusions

We have presented a novel approach to supporting the creation, presentation, viewing, and control of user-created meta-information about documents in the World Wide Web. In contrast to several previous approaches, our mechanism may be used without specialization of either browser or server. This implies certain tradeoffs, impacts of which we discussed. Our initial goal was to test the applicability of the Strand-based approach to this problem; this goal has been met quite successfully. Our next goal is to build another generation of annotation support, to address some of the user interface, overhead, and scalability issues and to extend the functionality in new directions.

Availability of Software

FTPable source and binaries for HP-UX (and possibly other platforms) are available; see <http://www.osf.org/mall/web/webware.htm>.

Acknowledgement

This research was supported in part by the Advanced Research Projects Agency (ARPA) under the contract number F19628-95-C-0042. The views and conclusion contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the U.S. Government.

References

- [BMMM] C. Brooks, M.S. Mazer, S. Meeks, and J. Miller, *Application-Specific Proxy Servers as HTTP Stream Transducers*, Proc. Fourth International World Wide Web Conference, 11-14 December 1995, Boston, MA, USA: <http://www.w3.org/pub/Conferences/WWW4/Papers/56/>.
- [DaMB] M.B. Davidson, M.S. Mazer, and C. Brooks, *Dynamic Integration of HTTP Stream Transforming Services*, OSF Research Institute technical report, Cambridge, MA, USA, February 1995: <http://www.osf.org/www/waiba/papers/integrating.html>.
- [Da] J. Davis and D. Huttenlocher, *CoNote Homepage*: <http://dri.cornell.edu/pub/davis/annotation.html>.
- [DLI] Digital Library Initiative: <http://alexandria.sdc.ucsb.edu/digital-libraries/>.
- [Gr] W. Gramlich, *Public Annotation System*, <http://playground.sun.com:80/~gramlich/1994/annotate>.

- [La] D. LaLiberte, *HyperNews*, <http://union.ncsa.uiuc.edu/HyperNews/get/hypernews.html>.
- [LA] A. Luotonen and K. Altis, *World-Wide Web Proxies*, <http://www.w3.org/hypertext/WWW/Proxies/>.
- [Le] S. Lewontin, *The DCE Web Toolkit: enhancing WWW protocols with lower-layer services*, Proc. Third International World Wide Web Conference, 10-14 April 1995, Darmstadt, Germany, <http://www.igd.fhg.de/www/www95/proceedings/papers/67/DCEWebKit.html>.
- [NCSAa] NCSA, *NCSA Mosaic*, University of Illinois (Urbana-Champaign), National Center for Supercomputing Applications: <http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/>.
- [NCSAb] NCSA, *NCSA Mosaic Group Annotations*: <http://www.ncsa.uiuc.edu/SDG/Software/XMosaic/Annotations/overview.html>.
- [NCSAc] NCSA, *The Common Gateway Interface*, University of Illinois (Urbana-Champaign), National Center for Supercomputing Applications: <http://hoohoo.ncsa.uiuc.edu/cgi/>.
- [OSFa] OSF Research Institute, *OSF RI World Wide Web Agent Toolkit (OreO)*: http://www.osf.org/ri/announcements/OreO_Datasheet.html.
- [OSFb] OSF Research Institute, *DCE-Web Home Page*: <http://www.osf.org:8001/www/dceweb/DCE-Web-Home-Page.html>.
- [OSFc] Open Software Foundation, *OSF Distributed Computing Environment*: <http://www.osf.org:8001/dce/index.html>
- [Ous] J.K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley Publishing Company, Reading, MA, USA: 1994.
- [RoMo] M. Röscheisen and C. Mogensen, *ComMentor: Scalable Architecture for Shared WWW Annotations as a Platform for Value-Added Providers*, Stanford University Technical Report, Palo Alto, CA, USA: <http://www-pcd.stanford.edu/COMMENTOR>.
- [RoMW] M. Röscheisen, C. Mogensen, and T. Winograd, *Beyond browsing: shared comments, SOAPs, trails, and on-line communities*, Proc. Third International World Wide Web Conference, 10-14 April 1995, Darmstadt, Germany: <http://www.igd.fhg.de/www/www95/proceedings/papers/88/TR/WWW95.html>.
- [Su] Sun Microsystems, *Java(tm): Programming for the Internet*, <http://www.javasoft.com/>.

Appendix D: *Transducers and Associates: Circumventing Limitations of the World Wide Web*

Transducers and Associates: Circumventing Limitations of the World Wide Web

W. Scott Meeks, Charles Brooks, and Murray S. Mazer
Open Software Foundation Research Institute
11 Cambridge Center, Cambridge, MA 02142, USA
Telephone: (617) 621-{7229, 8758, 7322}
{meeks, cbrooks, mazer}@osf.org

Abstract

There are limitations in the usability of the World Wide Web. One approach to circumventing these limitations involves inserting high-level transducers in the HTTP request/response stream. A second approach involves building more independent programs, called browsing associates. We have built example services and toolkits for both.

1. Introduction

This paper describes two approaches for circumventing important limitations in the usability of the World Wide Web (WWW). The first approach involves inserting high-level transducers, in the HTTP request/response stream. The second approach involves building more independent programs, called browsing associates, which are not tied to a particular HTTP stream, and can monitor the user's web activity indirectly, access the WWW independently, and communicate useful information to the user with a range of methods. Below, we discuss our motivation and hypotheses, the two approaches, and the example services and toolkits we have built.

2. World Wide Web Deficiencies

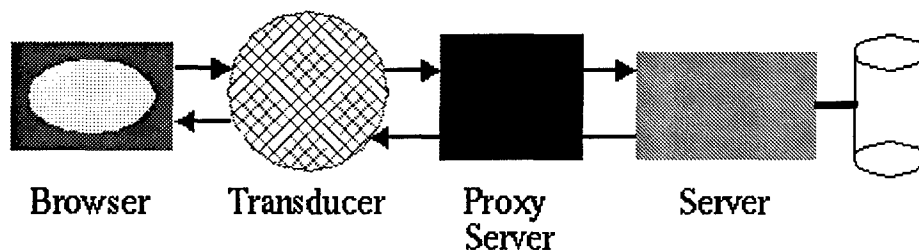
HTTP transducers and browsing associates are tools to increase the usability of the WWW for locating, managing and using information in a timely and efficient manner by addressing the following deficiencies.

- **Navigating**
Becoming lost or disoriented when navigating in a large hypertext space is easy. Browsers provide some limited support in the form of hotlists and history lists.
- **Finding**
Web clients are excellent at supporting *browsing*, and tools for *searching* for very specific information are becoming quite sophisticated, but *finding* objects requires organized or correlated sets of information that may not always exist.
- **Adding**
There is little support for allowing WWW users to easily add value, such as annotations, to existing web structures.
- **Receiving**
Passively *receiving* information in a timely manner is poorly supported.

We are exploring the *just-in-time* paradigm; that is, the user should get just the needed information, as soon as possible, without having to aggressively search for it or filter through an overload of irrelevant information. We want to determine how information can be delivered in a just-in-time fashion within the context of the WWW and produce services and toolkits reflecting our insights.

3. HTTP Stream Transducer Approach

Tapping into and possibly modifying the stream of HTTP requests and responses between a browser and a server in the WWW could help mitigate some of the WWW deficiencies by inserting high-level, application-oriented transducers. A natural way to insert the transducers in the HTTP stream is by using WWW proxy servers. [Luo94] Proxies are often used in network firewalls to enforce communication policies, but as illustrated below for WWW browsers and servers, one can use proxies to insert other kinds of processing entities into the stream.



We liken our HTTP transducers to Oreo cookies, because the transducer is structured with one 'cookie' to handle browser-side communication, another 'cookie' to handle server-side communication, and a functional 'filling' in the middle.

The transducers built to date are meant to be evocative, rather than definitive. They were constructed in various languages, most using the toolkit discussed below. Here are some examples; others are under development:

- Our first transducer, which performs URL validation, measures data transfer rates, and creates a group history.
- A full-text indexing transducer, which creates a full-text index of all of the non-trivial terms appearing in the HTTP responses.
- A "group annotation" transducer, that expands on the Stanford annotation approach.[Ros95]
- A "document correlation" transducer which uses the SMART technology from Cornell[Sal89] to find documents related to the current document.
- A "what's changed" transducer which monitors local URLs referenced by the current page and marks up the page to indicate any that have changed since last visited.

A number of these transducers directly address some of the WWW deficiencies. The group history and text indexer help with both navigating and finding. Group annotation helps with adding and document correlation with receiving. "What's changed" lets the user receive timely information.

We have produced a toolkit which allows the filling developer to focus on the application-specific aspects of the transducer. The initial version of the toolkit provides a 'shell' that implements the 'cookies' between which the 'filling' is placed. The developer may use any program development system to create the filling, which is simply executed by the shell after it performs appropriate setup functions. The shell ensures that the filling is connected into the request/response stream, so that the developer can operate on the contents and ignore the network-specific issues.

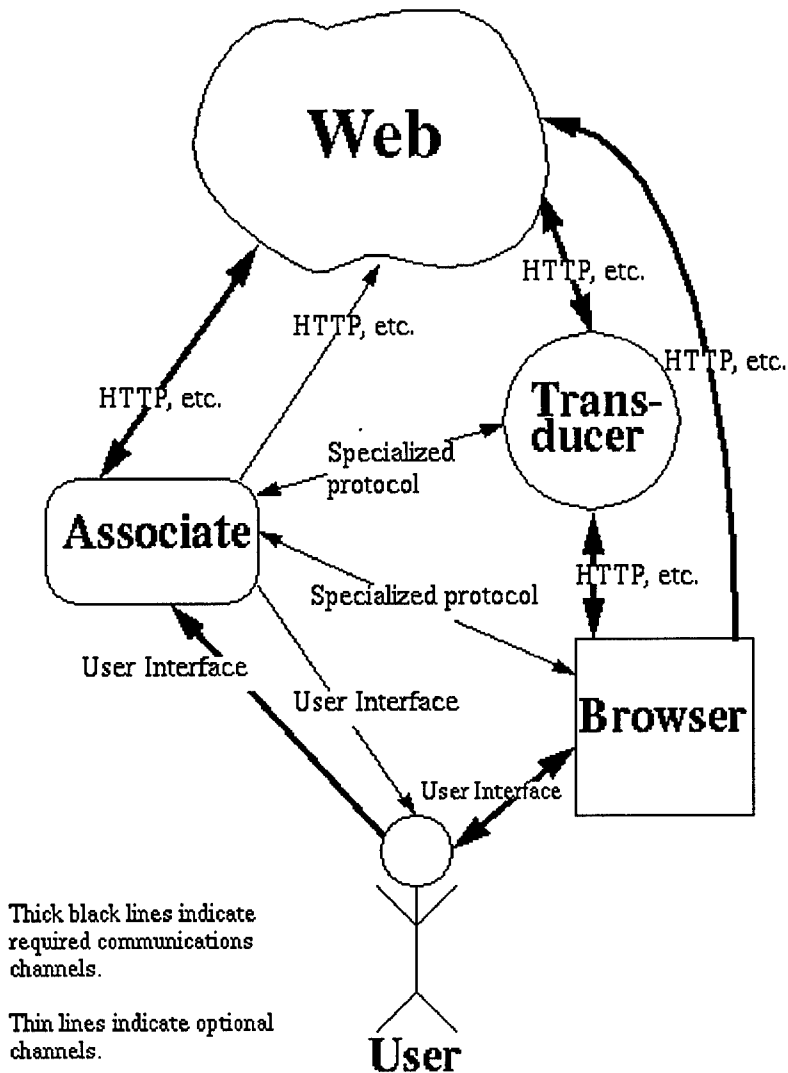
Future work will include enhancing the existing services and toolkit, and building new services. We plan on having several services in common use within the group within the next year. Further development of chaining transducers as a sequence of filters is expected to require a way of specifying a sequence of transforms and the ability to allow a pre-parsed information object to be passed among the transforms.

4. Browsing Associate Approach

A drawback of the transducer technology is that it is tightly coupled with a particular HTTP stream. Browsing associates are relatively small and simple applications which are not coupled to a particular HTTP stream and can independently and asynchronously access the WWW on the user's behalf.

A key requirement for most browsing associates is that they access the Web efficiently. Efficiency is important because many associates will actively seek information on behalf of the user rather than passively observing and thus may be limited in available resources. Associates must also communicate with the user. At a minimum, a user would specify parameters to the associate, and the associate would display results to the terminal. At a more sophisticated level, the user might interact with the associate via a graphical user interface (GUI), by generating a set of HTML pages, or by issuing asynchronous commands to the user's browser.

Associates, unlike transducers, are expected to be autonomous. Once an associate is started, it should go off and do its work, perhaps periodically being interruptible and checking in with its user, but acting independently. In addition, an associate may spawn sub-associates, or instantiate multiple threads of control to provide further asynchrony.



Browsing associates test the hypothesis that fairly simple programs that can access the WWW and communicate with the user can provide timely and useful information without requiring excessive additional work on the part of the user.

To date we have built three prototype Browsing Associates. The first is the "What's New?" associate which is responsible for notifying the user as to whether a particular set of documents has changed over a given interval. Any document containing a series of embedded anchors may be checked, allowing the use of specialized files or other HTML documents such as hotlists. The user can also specify the cutoff time for considering a URL to have changed, whether to note URLs that have or have not changed, and a frequency for automatically repeating the check. The user can also request to be notified either by a popup window or by having a list of the URLs automatically displayed in the browser. A few members of our group are starting to use this service on a regular basis and it is being evaluated for deployment elsewhere.

The second associate is the LinkTree which produces an interactive hierarchical listing of all the links recursively embedded in a particular document down to a particular depth. A link tree allows the user to see just the anchors in a tree of documents rooted at a particular URL and to quickly locate interesting links without having to look at the contents of the intervening documents. The associate uses the NCSA Mosaic Client Command Interface (CCI)[Mos95] to track the current URL in the user's browser, and provides a GUI for setting the depth of the search and other options affecting the search and display of the results.

Our latest example of a Browsing Associate is a graphical history grapher. The GUI consists of a tree of nodes and links where the nodes correspond to URLs and the links point up to the previously visited URL. Using CCI, the associate tracks the current URL in the user's browser. Whenever the URL changes, the history grapher checks to see if it already has a node for the new URL, and if not it will add a new node with a link to the previously visited

URL. If the URL already exists in the graph, then the associate marks the corresponding node as being the current node. If the user then navigates to a new URL, a branch will be created in the graph. The associate acts as a peer with the browser: clicking on a node in the graph requests the user's browser to display the corresponding URL.

We are starting to build up the components of a rapid prototyping toolkit for building associates based on Tcl/Tk[Ous94] and related technologies. We have some useful scripts including one that implements a generic mechanism for communicating with browsers. We are developing useful, reusable object classes such as a class representing WWW anchors.

Our work on browsing associates is really just beginning, but we have a few insights from our examples.

- The degree of coupling between the user's activities on the WWW and the associate will generally be looser than with transducers and will vary greatly from associate to associate.
- Tcl/Tk and related technologies show promise for implementing associates.
- Browsing associates and transducers can benefit from communicating with each other: transducers providing information about the HTTP stream to associates, and associates providing external information to the transducers.

We are enhancing existing associates and the development environment, investigating other tools such as perl and Tcl/Tk based GUI builders for use in associates, and building new associates. We also plan to explore communication between transducers and associates by building some hybrid tools.

5. Conclusion

We have presented two approaches to mitigating some of the deficiencies of the WWW. First, we generalized the notion of WWW proxy servers to that of application-specific proxies that act as transducers on the HTTP stream. Our prototype transducers demonstrate the utility of the concept, and our toolkit aims to ease the task of building such transducers. Second, we developed the notion of a relatively small and simple application that can independently and asynchronously access the Web on a user's behalf. Our prototype associates demonstrate some of the range of possibilities.

FTPable source and binaries for selected portions of our technology are available by anonymous ftp from riftp.osf.org in `/pub/web/Strand`.

6. Acknowledgement

This research was supported in part by the Advanced Research Projects Agency (ARPA) under the contract number F19628-95-C-0042. The views and conclusion contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the U.S. Government.

7. References

[Luo94] Luotonen, A., and Altis, K. (May 1994), World-Wide Web Proxies, <http://www.w3.org/hypertext/WWW/Proxies/>.

[NCSA95] National Center for Supercomputing Applications (March 1995), NCSA Mosaic Common Client Interface, <http://www.ncsa.uiuc.edu/SDG/Software/XMosaic/CCI/cci-spec.html>.

[OSF95] Intelligent Browsing Assistance for the World Wide Web (1995), <http://www.osf.org/www/waiba/>.

[Ous94] Ousterhout, J. (1994), Tcl and the Tk Toolkit, Addison-Wesley, Reading, MA.

[Ros95] Roscheisen, M. and Mogensen, C. (1995), ComMentor: Scalable Architecture for Shared WWW Annotations as a Platform for Value-Added Providers, <http://www-pcd.stanford.edu/COMMENTOR>

[Sal89] Salton, G. (1989), Automatic Text Processing-The Transformation, Analysis and Retrieval of Information by Computer. Addison-Wesley, Reading, MA.

DISTRIBUTION LIST

addresses	number of copies
PETER A. JEDRYSIK ROME LABORATORY/C3AB 525 BROOKS RD. ROME, NY 13441-4505	10
THE OPEN SOFTWARE FOUNDATION RESEARCH INSTITUTE 11 CAMBRIDGE CENTER CAMBRIDGE, MA 02142	1
ROME LABORATORY/SUL TECHNICAL LIBRARY 26 ELECTRONIC PKY ROME NY 13441-4514	1
ATTENTION: DTIC-OCC DEFENSE TECHNICAL INFO CENTER 8725 JOHN J. KINGMAN ROAD, STE 0944 FT. BELVOIR, VA 22060-6218	2
ADVANCED RESEARCH PROJECTS AGENCY 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714	1
ROME LABORATORY/C3AB 525 BROOKS RD ROME NY 13441-4505	1
ATTN: RAYMOND TADROS GIDEP P.O. BOX 8000 CORONA CA 91718-8000	1

AFIT ACADEMIC LIBRARY/LDEE 1
2950 P STREET
AREA B, BLDG 642
WRIGHT-PATTERSON AFB OH 45433-7765

ATTN: R.L. DENISON 1
WRIGHT LABORATORY/MLPO, BLDG. 651
3005 P STREET, STE 6
WRIGHT-PATTERSON AFB OH 45433-7707

ATTN: GILBERT G. KUPERMAN 1
AL/CFHI, BLDG. 248
2255 H STREET
WRIGHT-PATTERSON AFB OH 45433-7022

DL AL HSC/HRG, BLDG. 190 1
2698 G STREET
WRIGHT-PATTERSON AFB OH 45433-7604

AUL/LSAD 1
600 CHENNAULT CIRCLE, BLDG. 1405
MAXWELL AFB AL 36112-6424

US ARMY STRATEGIC DEFENSE COMMAND 1
CSSD-IM-PA
P.O. BOX 1500
HUNTSVILLE AL 35807-3801

COMMANDING OFFICER 1
NCCOSC RDT&E DIVISION
ATTN: TECHNICAL LIBRARY, CODE 0274
53560 HULL STREET
SAN DIEGO CA 92152-5001

COMMANDER, TECHNICAL LIBRARY 1
474700D/C0223
NAVAIRWARCENWPNDIV
1 ADMINISTRATION CIRCLE
CHINA LAKE CA 93555-6001

SPACE & NAVAL WARFARE SYSTEMS 1
COMMAND, EXECUTIVE DIRECTOR (PD13A)
ATTN: MR. CARL ANDRIANI
2451 CRYSTAL DRIVE
ARLINGTON VA 22245-5200

COMMANDER, SPACE & NAVAL WARFARE 1
SYSTEMS COMMAND (CODE 32)
2451 CRYSTAL DRIVE
ARLINGTON VA 22245-5200

CDR, US ARMY MISSILE COMMAND 2
RSIC, BLDG. 4484
AMSMI-RD-CS-R, DOCS
REDSTONE ARSENAL AL 35898-5241

ADVISORY GROUP ON ELECTRON DEVICES 1
SUITE 500
1745 JEFFERSON DAVIS HIGHWAY
ARLINGTON VA 22202

REPORT COLLECTION, CIC-14 1
MS P364
LOS ALAMOS NATIONAL LABORATORY
LOS ALAMOS NM 87545

AEDC LIBRARY 1
TECHNICAL REPORTS FILE
100 KINDEL DRIVE, SUITE C211
ARNOLD AFB TN 37389-3211

COMMANDER 1
USAISC
ASHC-IMD-L, BLDG 61801
FT HUACHUCA AZ 85613-5000

AFIWC/MSO 1
102 HALL BLVD, STE 315
SAN ANTONIO TX 78243-7016

NSA/CSS 1
K1
FT MEADE MD 20755-6000

PHILLIPS LABORATORY 1
PL/TL (LIBRARY)
5 WRIGHT STREET
HANSCOM AFB MA 01731-3004

THE MITRE CORPORATION 1
ATTN: E. LAURE
D460
202 BURLINGTON RD
BEDFORD MA 01732

OUSD(P)/DTSA/DUTD 2
ATTN: PATRICK G. SULLIVAN, JR.
400 ARMY NAVY DRIVE
SUITE 300
ARLINGTON VA 22202

SDIO/S-PL-BM 1
ATTN: CAPT JOHNSON
THE PENTAGON
WASH DC 20301-7000

SDI TECHNICAL INFORMATION CENTER 1
1755 JEFFERSON DAVIS HIGHWAY #708
ARLINGTON VA 22202

NAVAL AIR DEVELOPMENT CTR 1
ATTN: DR. MORT METERSKY
CODE 30D
WARMINSTER PA 189974

ESD/XTS 1
ATTN: LT COL JOSEPH TOOLE
HANSCOM AFB MA 01731

USA-SDC CSSD-H-SBE 1
ATTN: MR. DOYLE THOMAS
HUNTSVILLE AL 35807

NTB JPO 1
ATTN: MR. NAT SOJOUNER
FALCON AFB CO 80912

NASA LANGLEY RESEARCH CENTER 1
ATTN: WAYNE BRYANT
MS578
HAMPTON VA 23665-5225

AL/LRG 1
ATTN: MR. M. YOUNG
WRIGHT-PATTERSON AFB OH 45433-6503

AL/CFHV 1
ATTN: DR. B. TSDU
WRIGHT-PATTERSON AFB OH 45433-6503

WL/KTD 1
ATTN: DR. D. HOPPER
WRIGHT-PATTERSON AFB OH 45433-6503

AFIT/ENG 1
ATTN: LT COL M. STYTZ
WRIGHT-PATTERSON AFB OH 45433-6503

USA ETL 1
ATTN: MR. R. JOY
CEETL-CA-D
FT BELVOIR VA 22060

NUSC 1
ATTN: MR. L. CABRAL
NEWPORT RI 02841-5047

NOSC (CODE 414) 1
ATTN: MR. P. SOLTAN
271 CATALINA BLVD
SAN DIEGO CA 92151-5000

NTSC (CODE 251) 1
ATTN: MR. D. BREGLIA
12350 RESEARCH PARKWAY
ORLANDO FL 32826-3224

NASA
ATTN: DR. J. ROBERTSON
LANGLEY RESEARCH CENTER
MAIL CODE 152E
HAMPTON VA 23665-5225

1

NAVAL OCEAN SYSTEMS CENTER
ATTN: GLEN OSGA, PHD
CODE 444
SAN DIEGO CA 92152

1

ARI FORT LEAVENWORTH
ATTN: MAJ ROB REYENGA
P. O. BOX 3407
FT LEAVENWORTH KS 66027-0347

1

ARI FIELD UNIT
ATTN: JON J. FALLESEN
P. O. BOX 3407
FORT LEAVENWORTH KS 66027-0347

1

RL/ESOP
ATTN: MR. JOSEPH HORNER
HANSCOM AFB MA 01731

1

**MISSION
OF
ROME LABORATORY**

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

- a. Conducts vigorous research, development and test programs in all applicable technologies;
- b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;
- c. Provides a full range of technical support to Air Force Material Command product centers and other Air Force organizations;
- d. Promotes transfer of technology to the private sector;
- e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.