Fifth International World Wide Web Conference
May 6-10, 1996, Paris, France

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.

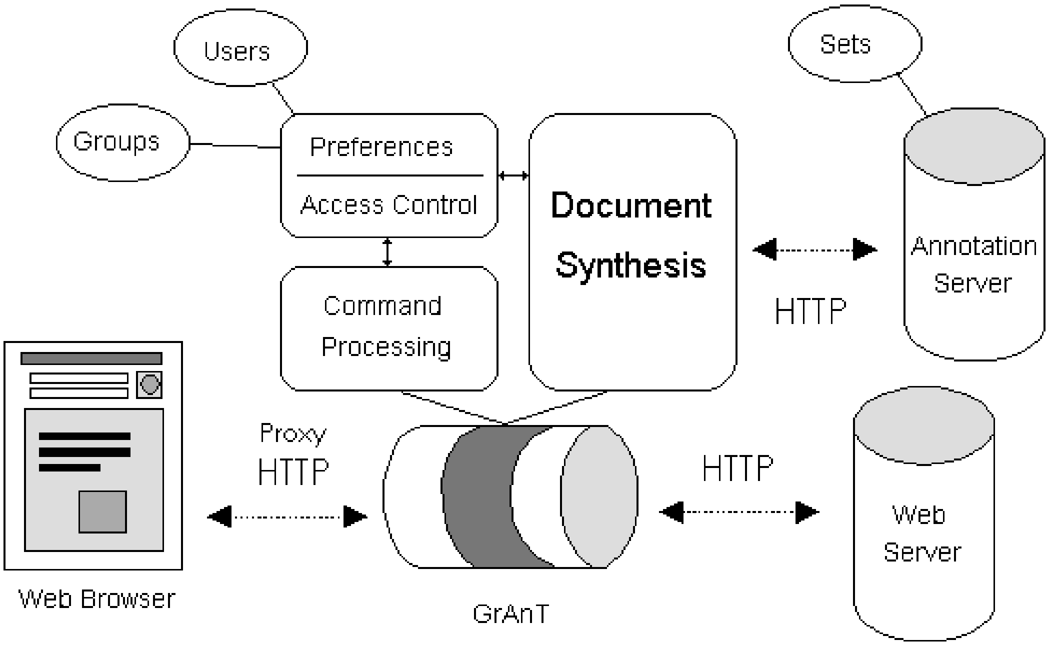

Figure 1: Initial architecture: annotation transducer, annotation server, unaugmented browsers, and unaugmented WWW servers.

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.

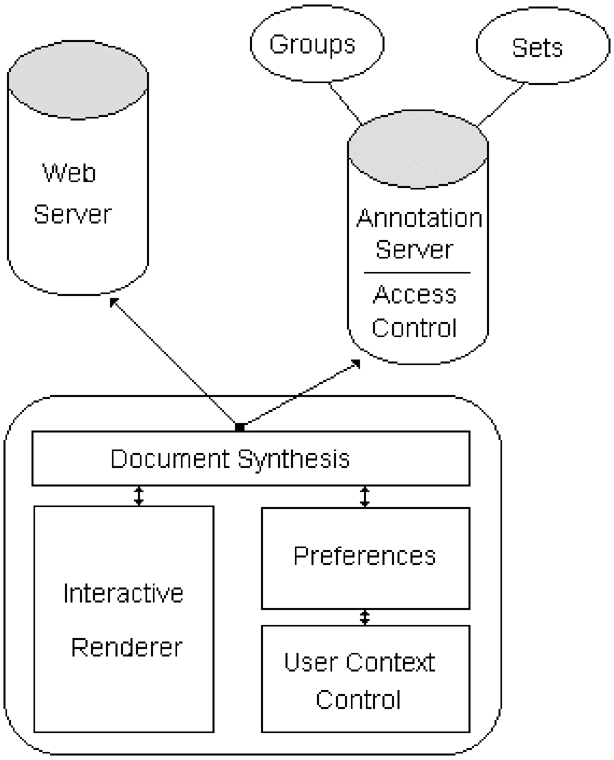

Figure 2: ComMentor architecture: modified Mosaic browser, annotation server, and unaugmented WWW servers.

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.

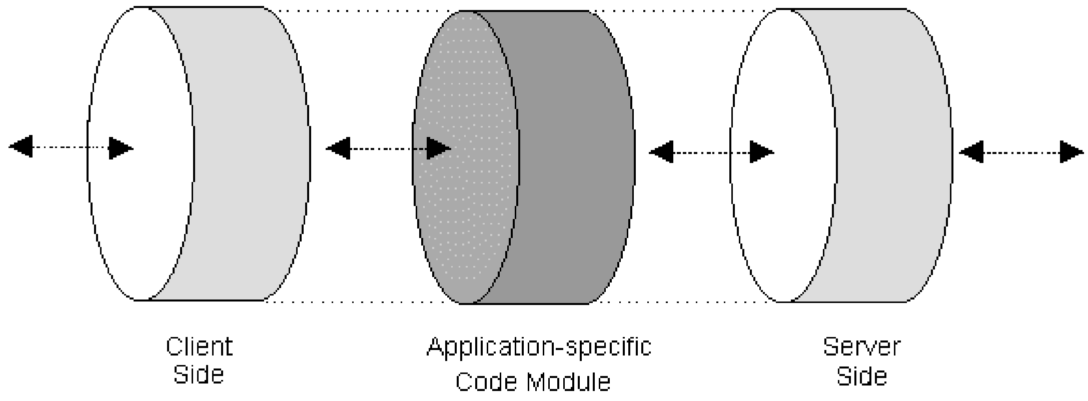

Figure 3: The Strand architecture

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.

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).

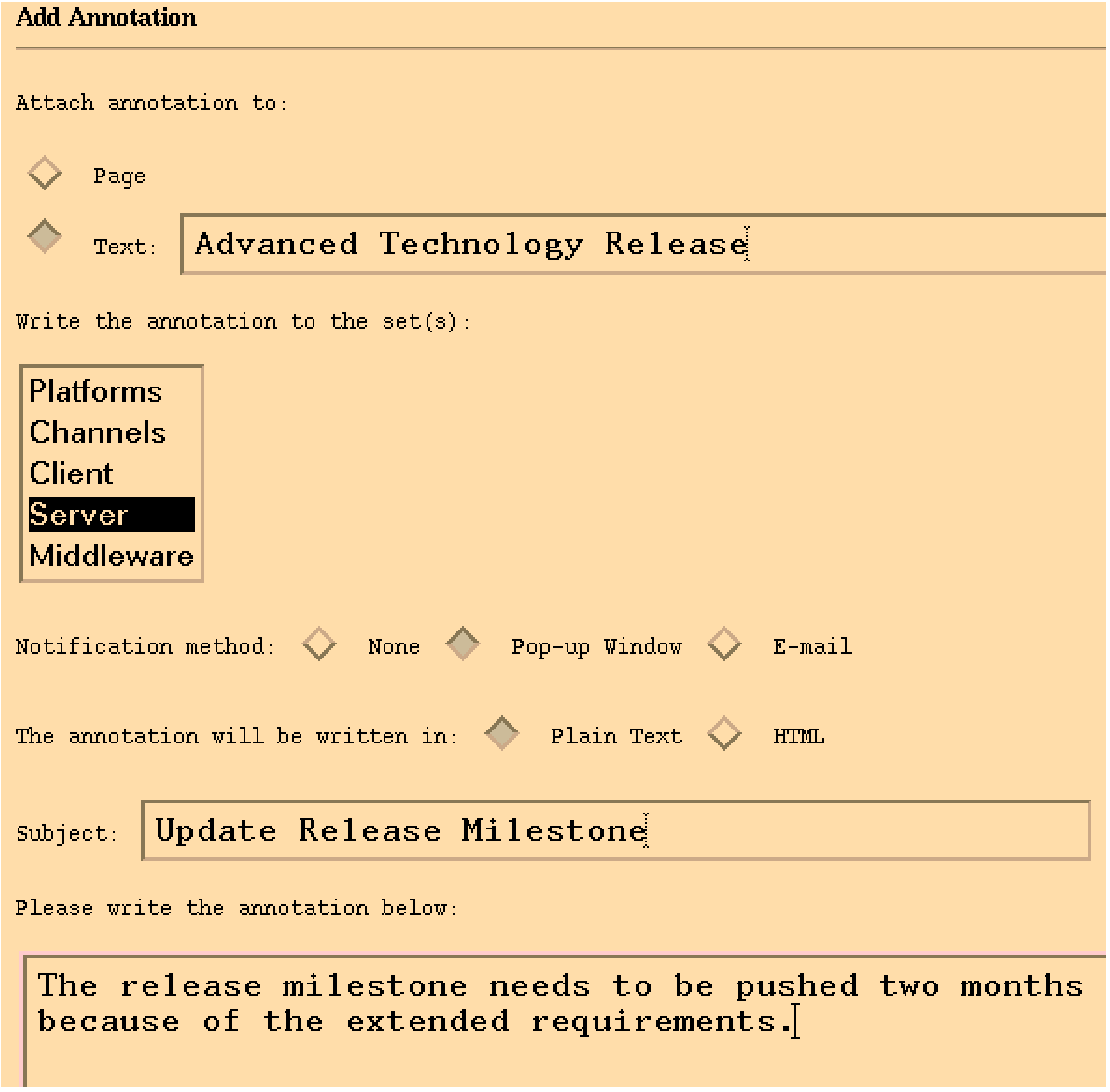

Figure 5: A form for creating annotations

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.

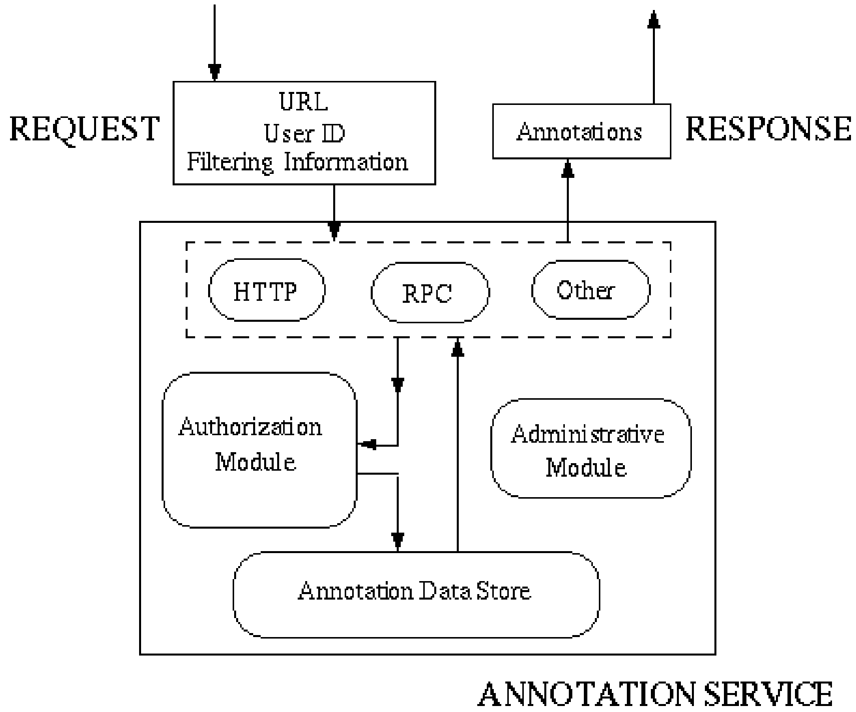

Figure 6: New annotation service architecture

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/annote

[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/.