On Web Annotations: Promises and Pitfalls of Current Web Infrastructure

Venu Vasudevan and Mark Palmer,

Object Services and Consulting Inc.,

venu@objs.com and mpalmer@nh.ultranet.com

Abstract

Annotations are a broadly useful mechanism that can support a number of useful document management applications (third-party commentary, design rationale, information filtering, and semantic labeling of document content to name a few). The ubiquity of web content motivates the need for web annotation systems that are lightweight, efficient, non-intrusive (preferably transparent), platform-independent and scaleable. Building such a system using open and standard web infrastructures (as opposed to proprietary ones) facilitates widespread applicability and deployment. In practice, there are a number of ways to do this, all of which instantiate a common abstract architecture based on ***intermediaries****. The paper describes our experiences with client and proxy-server based implementations of the annotation system architecture. The implementations point to missing elements in the current web infrastructure that make any implementation of annotation systems less than completely satisfactory. This paper discusses these elements of current web infrastructure, and potential changes to the web architecture that might make the implementation of annotation systems more complete.*

1. Introduction

Large organizations increasingly use Intranets and the World-Wide Web as a shared organizational memory for their business processes. While the web has simplified the process of publishing and retrieving documents, the web collaboration model is an asymmetric one with active information publishers and passive information consumers. *Annotations* allow third-parties to interactively and incrementally augment web documents. An annotation system supports the creation and retrieval of annotations, and composes personalized "virtual documents" from the authored document and associated annotations.

By varying the annotation vocabulary and composition semantics, an annotation system is usable in a number of document management applications. For instance, explicitly authored textual annotations are usable in review and rationale capture applications. Systems that annotate documents with relevant information retrieved from search engines, newsgroups and other forums [Elo96], are useful in intelligently contextualizing the document to reader's interests. Effector annotations (which are not visibly presented, but determine the contents of the virtual document) are useful in content labeling and collaborative filtering. Document ontology specifiers [Luke96] provide an example of semantic annotations that label the concepts covered in sections of a web document, thus overlaying a concept map on the document content.

An annotation framework needs to be customizable to support this variety of document management function, and to be non-intrusive to enable easy insertion into enterprise Intranets or the public Internet. This paper describes our efforts to build such an annotation system using open Internet frameworks, and avoiding proprietary extensions to the infrastructure. The web annotation framework described here can be viewed as a specialization of the *intermediary architecture* [Thom98, Barr98]. Intermediaries are expansion joints in the web client-server connection where the web client-server interaction can be customized on a per interaction basis without bringing down web clients or services to do so. The intermediation approach also allows annotation systems to be built using the open API of current web clients, servers and proxies, and without having to invent proprietary web tools.

In this paper, we focus on annotations as one behavior supported by intermediation. First we outline the elements of an intermediary-based annotation architecture. The next two sections discuss our experiences in building proxy-based and client-based annotation systems. It has been our experience that annotation systems are constrained both in capability and efficiency by the limitations of current web infrastructure. We find that the intermediary approach offers a reasonable, uniform structure for extending web client capability - externally to the browser. Yet there is little support to date for extending capabilities within popular browsers in the same principled way, due to security mechanisms and divergent browser designs. The last section discusses changes in web architecture that would make it easier to build

0-7695-0001-3/99 $10.00 (c) 1999 IEEE

annotation systems, and emerging standards that may help in this regard.

2. Architecture: Theme and Variations

Figure 1 shows an abstract annotation system architecture that can be concretely implemented using client or server-side Internet frameworks. The main components of the architecture are *interceptors*, *annotation repository services* (AReS) and *composers*, with the annotation delivery and composition styles being personalized based on a user model. In concrete implementations of the architecture, elements within the dashed rectangle are relocatable to the client, the server or to a mediating proxy.

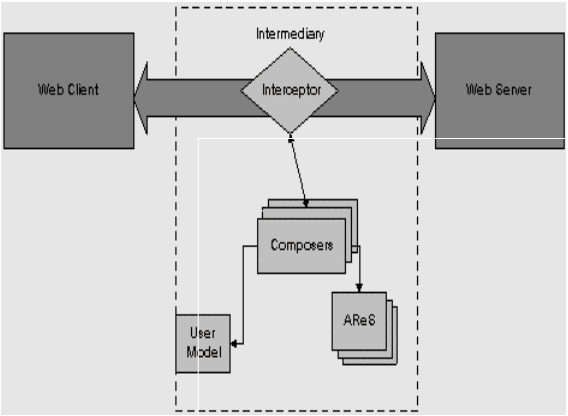

Figure 1: An Abstract Annotation System Architecture

Interceptors tap into a web client-server interaction and trigger the annotation process. To manufacture annotated content, they invoke composers that understand the process of rendering the document content and annotations into a composite that is personalized to the end user. Composers communicate with one or more AReS'es to retrieve annotation sets appropriate to the document, user and context. A reason for communicating with multiple AReS'es is that a user may belong to multiple groups, and the composed document may therefore require the merging of private, group and public annotations. AReS'es provide the repository function for the annotation system.

To be able to efficiently compose annotation sets with document content, composers operate on a document abstraction known as the document object model (or DOM). The DOM provides a high-level API for composers to directly access locations in the document where annotations are to be inserted, and facilitates efficient composition. While it is not depicted as a separate module in the abstract architecture, the efficiency and level of abstraction of DOM support is a critical component of concrete annotation system architectures. The DOM may be provided by an application module, or inherently by the annotation infrastructure. The interceptor-composer-AReS architecture outlined above can be customized to handle a wide variety of annotation-related document management functions depending on the levels of capabilities in the interceptor, the composition framework and the AReS. The kinds of functionality supported by these components and its effect on the overall annotation capability are described below.

2.1. Interceptors

Interceptors can modify the semantics of document retrieval by either trapping and modifying the outgoing request, the returned content, or by triggering other actions as side-effects of the act of document retrieval. Accordingly we classify them as *request*, *page* or *event* interceptors. *Request interceptors* intercept an outgoing document URL request, and redirect the request to a URL that returns the document augmented with annotations. *Page interceptors* trap the contents of the web document being retrieved and pass it along to the annotation system to be augmented. *Event interceptors* detect some event related to document retrieval by subscribing to an event channel. The document event triggers the annotation mechanism. An example of event interception that is addressed subsequently in greater detail is one of detecting that a document is being loaded into the web browser, and using this information to present the appropriate annotations.

2.2. Annotation Repository Services (AReS)

The *basic* AReS functionality is the ability to create annotation objects with attributes specifying the author, timestamp, URL of the annotated document and anchor information about the placement of annotation sets within the document URL. Additionally, a basic capability of AReS'es is an API to create/edit annotations, and to filter and retrieve *annotation sets* based on the above fields. Annotation set based filtering is a broadly useful capability and can be used in a number of ways. Author or group based filtering is used to personalize the annotated document to the end user. Timestamp-based filtering is useful to incrementally retrieve new annotations for an annotated document. Annotation sets may be a unit of access control, in that they determine who can create or retrieve annotations corresponding to a particular document group [Rosc96]. In a public and physically distributed AReS, annotation sets may be a unit of storage and distribution, with the set being stored on a server that is close to the author or group that created it.

It is common for AReS'es to export their API via HTTP or other Internet protocols. Such AReS wrappers are referred to as *annotation set servers*, and the protocol for querying them as the *annotation protocol*. The Hypernews annotation set server [Brav] for example, supports an annotation protocol that provides the basic AReS capability but without anchor support. Annotation servers facilitate *indirect* annotations in that the an annotation (or annotation set) can be included as a hyperquery (i.e. a hyperlink that is a really a query to the annotation server) rather than necessarily being embedded by value. This is useful for richly annotated documents or for thin clients, where delivering all the annotations by value to the client might overwhelm the system or the user. In the more semantic annotation applications, annotations need to be delivered in a structured, parseable form as the consumer of annotations is more likely a program, than a human. HTML, while adequate for visually rendering annotation sets, is cumbersome to parse and limited in its expressiveness. It is useful for annotation servers to deliver annotation sets in metadata formats such as SOIF [Hard96] (and now XML), which are easily parseable as they are *self-describing* (i.e. they describe both the structure and the content of the annotation set).

A category of document management applications require annotations to be first-class objects that can themselves be recursively annotated. In dialogue management and document review applications [Sumn96], annotations represent assertions or comments by an author, and recursive annotations represents responses or clarifications of the original annotations. In design rational applications, annotations are used to explain or justify a decision, and recursive annotations may provide references to authoritative texts that justify the explanation. Support for these applications requires AReS'es that are *annotation graph servers.* Annotation graph servers support links between annotations, and a query API for graph-based queries of the annotation repository.

An orthogonal dimension to annotation set servers and annotation graph servers is that of extensibility. *Extensible annotation servers* export a schema API that allows new classes of annotation objects and link types to be dynamically added to the server. This is useful where the same AReS is supporting a variety of annotation applications with differing annotation semantics.

2.3. Composers

Composers determine what it means to merge the document with one or more annotation sets. We categorize the composers (and composition) as either *stylistic*, *versioned* or *semantic* depending on the complexity of the composition algorithm. In stylistic composition, annotations are data objects with presentation semantics only, and composition is the process of combining document data with annotation data in a particular presentation style. Decisions for stylistic composers include locating and anchoring the annotation sets, and choosing a customized presentation scheme for the annotation sets to visually distinguish them from document content (unless visual distinctions are undesirable). Stylistic composers may support explicit (e.g. at a named HTML element) or implicit (e.g. before or after a phrase) anchoring. In the case of both anchoring schemes, composers may vary in how they deal with anchor degradation, which is the partial or total deletion of annotation anchors in documents that are editable. Composers that support annotation graphs may either use a flattened HTML rendering of an annotation graph, or use specialized graph display applets as embedded viewers to permit navigation and edits to an annotation graph.

Versioned composition includes stylistic composition, but takes the versioning semantics of both the document and the annotation sets into account. Composers that support versioned composition can reason about how annotations interact with document versioning. For instance, annotations may apply only to a certain version of the document, or may tunnel through to subsequent versions. To do lists that might be used by collaborating authors of an online document, are annotations that migrate to future versions of the document until they are flagged as done. Composers that deal with versioning semantics allow the specification of version related annotation policies.

Semantic composition applies to structured annotations, which may or may not be visibly presented along with the document. Semantic annotations may be operations to be applied to the document that are authored as annotations, or annotations that are associated with the document by a knowledge-based processing (as opposed to explicit authoring). Composers of semantic annotations may know how to interpret the microlanguages in which annotations are specified, or be experts in intelligently retrieving annotation sets that are relevant to the document. Filter annotations that conditionally elide parts of a document, and systems like SHOE [Luke96] that overlay the ontology of a page's contents as annotations on the page are examples of annotations that have internal structure. PLUM [Elo96], a system that annotates news articles with the related information personalized to the reader, is an example where the annotations are textual, but are the result of a knowledge-based search.

3. Implementations

3.1. InterNote - A Proxy-Based Annotation System Implementation

InterNote transparently annotates web content using a request interception architecture. A proxy server intercepts requests from browsers for web documents. The proxy server then redirects the request to the appropriate composer, depending on the kind of stylistic, versioned or semantic composition dictated by the user model and document type. The composer retrieves annotations from one or more AReS'es and returns the composed content to the web browser. Other than some initial configuration, the user of the web browser is unaware of mediation by a proxy server, and the fact that (s)he is receiving a personalized and virtual document. The implementation uses existing proxy server mechanisms [Luot] not for the typical firewall proxy function, but to inject application logic into a web transaction. We call the proxy server an *application proxy server*, to distinguish its role from security and firewall proxy servers.

The InterNote application proxy server is implemented using Jigsaw, a Java web server distributed by the World-Wide Web consortium. Since the server is implemented in Java (and therefore object-oriented), more of the server's internal architecture is exposed as objects than is typical of web servers implemented in other languages. This makes it easier to customize the application proxy behavior. Requests in Jigsaw are exposed as objects that can be modified by pre and post methods. A request interceptor can therefore be built using pre-methods that modify the request before the server processes it. A page interceptor can similarly be implemented as a post-method to the request object. URLs in the server are not documents, but instances of (document handler) object types. The fact that all URLs are programs, not data allows InterNote to define composers as URLs, and for the request interceptor to simply use the standard HTTP protocol to communicate with composers. New composer classes can be defined in Jigsaw by the usual object-oriented mechanisms to support various kinds of stylistic, versioned and semantic composition. The next two paragraphs describe some concrete details of the InterNote implementation.

The InterNote implementation provides a *composition library* with several useful stylistic composers, and utility classes that support the development of other kinds of composer classes. Support is provided for both explicit and implicit anchoring. Annotations can be embedded at HTML anchors by value, as textual hyperqueries, or hyperqueries whose results are rendered by specialized viewers. In the third category, InterNote provides a *treeviewer* applet that display and allows the navigation of annotation graphs. The treeviewer allows the user to view and navigate dialogues that are structured as annotation trees.

InterNote uses the plug-in capability provided by web browsers to handle multimedia annotations. This allows audio or image objects to be attached to anchor points as annotations. Multimedia annotation data is authored using standard multimedia tools and published to a web URL. A multimedia annotation object in the AReS is authored as a hyperlink to the multimedia content URL, with associated author, timestamp and anchor information.

The AReS implemented using Object Design's PSE persistent store provides the annotation set server function and an API for the creation and querying of annotation graphs. . API is exported as CGI scripts, therefore allowing for indirect annotations to be embedded in the document as hyperqueries. The AReS adds a request serialization layer that allows for multi-user access of the AReS. The AReS provides support for link objects, and built-in capabilities for several link types. At this point, schema creation API has not been exported in the annotation server, and adding new link types requires programming the persistent store to extend its schema. The AReS supports the SOIF metadata format, in that annotation sets can be returned as machine parseable SOIF. This API is used by the annotation tree viewer to retrieve all or part of an annotation tree, but is also useful for other programs such as search engines to query the annotation repository for search metadata.

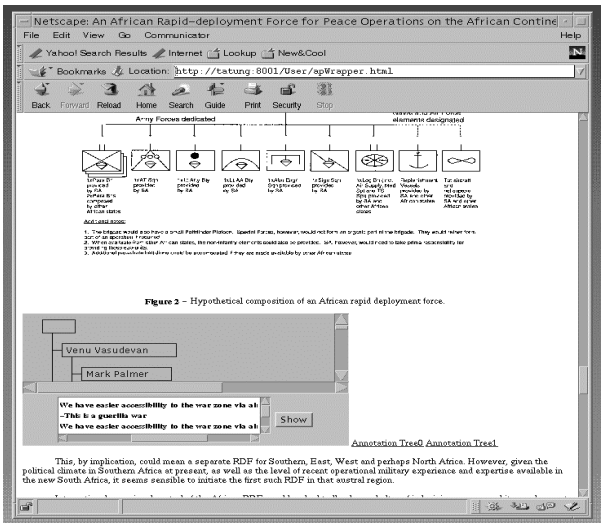

Figure 2: Annotation sets displayed with embedded viewers

Figure 2 presents an example of stylistic composition using embedded viewers. A treeviewer applet is embedded at each HTML anchor point with associated annotations. The document shown in Figure 2

discusses military strategy in an African war, and the treeviewer applet within the browser window presents annotations that elaborate on a schematic that details the armed forces strategy. Two separate dialogues are associated within this schematic, each represented as a hyperlink (labeled “Annotation Tree0” and “Annotation Tree1”). Clicking on these hyperlinks causes an embedded JavaScript program to query the remote annotation server, retrieve the selected annotation dialogue graph as a parseable SOIF data structure, and to transfer this set to the treeviewer for rendering. Clicking on a treeviewer node presents the text of the annotation and other associated metadata in the panel within the applet. Changing the user model for a particular user can change the presentation style of the annotated document.

3.2. JotBot - Client-centric annotations

The JotBot prototype moves the intermediary function within the client browser, so that annotations can be accessed by common web browsers without needing to configure and run an intermediary proxy on the client. The composer is a Java Applet that retrieves and presents annotations. Within browsers, the page and request interceptor functions are not universally available. Event interception was implemented by making calls to the native-code browser interface (DDE on Windows). This requires the user to download native code and makes the applet require privilege to run.

An alternate design for client event interception is to place the browsing window and the composer window in sibling frames. The composer can then poll via JavaScript to read the location (URL) property of its sibling frame. Security policy allows this only if both frames have contents originating from the same file server or domain name. However, a server intermediary can prefix all URL references in each page with the local domain name before sending the page to the client, and remove the localized prefix before fetching a referenced page. The event interceptor also removes the false local prefix to read the actual location. This design has the disadvantages of using frames, though, in that visited pages cannot be bookmarked and the forward/back functions do not work.

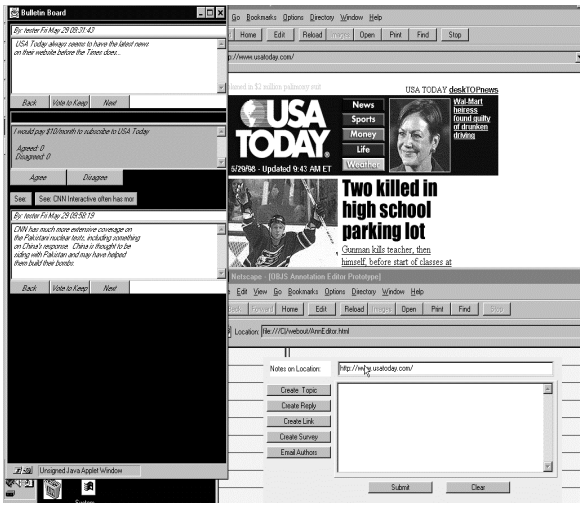

Figure 3: A JotBot Annotation Screen

With either of the above configurations, the browser reports only events at a page or loadable object granularity, so that annotations can only be associated with a page and not placed within it. This limits the ability to be specific with commentary, especially on long pages. The composer applet occupies a second browser instance and displays annotations for each page in a frame that floats beside the browsing window.

Each annotation names a Java class that the composer loads and uses to present the annotation for viewing, so new presentation styles and behaviors are easy to incorporate. By using the Java library, presentation classes can display multimedia types. Examples of presentation styles implemented to date are: audio note, image, text note, link, and survey. Annotations are deleted from the repository after their expiration dates. Text annotations have a rating feature whereby users can extend the life of useful information by pressing a "Vote to Keep" button. This differs from Collaborative Filtering ratings (e.g. as in GroupLens) in that the ratings are not used to make predictions or recommendations. Survey annotations allow the user to pose a question answerable by "agree/disagree" and automatically track the responses of other users. This feature allows an opinion poll to be affixed to a relevant page, for quick consensus formation. An asynchronous messaging facility allows users to discover and send email to the list of authors who created annotations on a given page, via SMTP.

Figure 3 shows three kinds of annotations associated with the *USA Today* document on the right hand side of the figure. The windows on the left of the figure are annotation windows, and illustrate text, survey, and link annotations (reading from top to bottom).

4. Discussion

4.1. Current Web Infrastructures: Missing Elements, potential fixes

One area where the web infrastructure is still maturing is in its uniform support for transparent interception of client-server transactions. Support for interception and customization of a web interaction is far more readily available in web proxies than in web servers (that are not proxies) or web clients. The limitations of client-side interception are particularly unfortunate, because client-side solutions are more readily accepted by the user community, and tend to have different scalability properties than server or proxy based solutions. While it is possible to build event interceptors on the client-side, the open APIs of current web browsers do not support either page interception or request interception. Page interception appears to be within closer reach, as elements of a page interception architecture are already present in web client architectures. In current web browsers, the contents of an HTML page that is in the process of being loaded are reflected as objects in the *scripting language* (e.g. JavaScript or VBScript) supported by the browser. Code written in such a scripting language can directly access and manipulate the document as an HTML tag hierarchy. Potentially, one could then write page interceptors in a scripting language, that would then invoke composers (also written in the scripting language) that would exploit the DOM view of the document to compose annotations into the page being loaded. However, restrictions in current browser architectures make the DOM available only to scripts written in the scripting language *that are embedded* in the page being loaded. This support for client-side DOM, also known as level 0 DOM support [DOM98], does not help in building an annotation system, as annotation systems require agents *outside* the document to manipulate the DOM. Ongoing work at W3C's DOM working group [DOM98] aims to extend the DOM, and make it accessible to programs external to the browser. This would facilitate a full-function, client-side annotation systems based on page interceptors in which composers external to the page.

Proxy-based interception is fairly easy and flexible when compared to client-side interception. Proxies, especially those built on top of Java web servers allow programmable access both to the outgoing request object and the incoming document content. The former is used to implement request interception, and the latter can be used to implement page interception. One limitation of proxy-based interceptors, which is really a limitation of the HTTP protocol is that the proxy cannot distinguish between request for a document, and requests for sub-objects (e.g. embedded images) of the document. Unless care is taken in the interceptor implementation, the retrieval of an article with nine embedded images can look to the request interceptor like ten document requests. This is inefficient, because the interceptor and the associated annotation machinery will be triggered far more often than is needed. Another efficiency issue in proxy servers is that they do not provide any built-in DOM support. The composer or other application modules are expected to build a DOM abstraction of the retrieved document to facilitate efficient composition. As more document augmentation functions are supported by web proxy servers, it would make sense that the proxy server also provide built-in support for a DOM abstraction, preferably the same abstraction as is provided via scripting languages on the web clients.

Limitations of web browsers and of HTML as a layout language make digital annotations somewhat more limited than paper annotations. HTML as a layout language is somewhat poorer than its proprietary counterparts (e.g. RTF and MIF). For instance, no HTML syntax for sidebars or change bars has been standardized, although some draft proposals on the subject have been issued. Similarly, there is no easy way in HTML to render annotations on the sidelines of a web page, an annotation style that is quite natural to annotators of paper books. While some of these problems are likely to be addressed in the near future, work in [Mars97] points out the significant gap that needs to be narrowed between digital and "human" HCI's before web content can be annotated as naturally as digital content.

4.2. Related Work

This section summarizes past and ongoing projects that attempt to support the annotation of web documents. No attempt is made to be exhaustive, as the goal is to compare and contrast these with our efforts. The approaches that these projects take to building a web annotation system can be categorized into *customized content*, *customized protocol*, *customized infrastructure* and the *COTS and open standards*. Systems in the first three categories require changes that are apparent either to the information producer or the information consumer (or both). The fourth category, which is the philosophy we adopt, aims to support annotation systems natively and unobtrusively.

CoNotes, a group annotation system from Cornell is an example of a customized content annotation system. It expects the document authors to anticipate places where readers might wish to add commentary or questions by inserting "annotation points" into the original document. These anchors are used by the CoNotes to compose annotations as inline links to discussion threads. Dependence on specialized anchors means that CoNotes (and other customized annotation systems) can only annotate documents under its server's control. This is reasonable in small group

annotation scenarios such as the computer-aided instruction scenario that CoNotes has been used in [Davis95]. However, this approach does not scale to annotating arbitrary web content.

In customized protocol systems like HyperNews [LaLib] and CritLink[Yee98], the request for annotated content is syntactically different from a request for the unannotated document. HyperNews system is primarily a discussion group tool, but has supported annotation in conjunction with the use of modified browsers (Mosaic and HotJava) in the past. The HyperNews user types accesses CGI scripts to create annotations and to retrieve annotated documents. CritLink, developed at the Foresight institute, uses a specialized server called the mediator. Users first visit the CritLink home page (served by the Mediator) and enter the URL of a page they wish to visit and read or write annotations for. The Mediator retrieves that page, prefixes all references in it with its Internet domain, finds any relevant annotations, and returns the page to the user. Thus all further navigation from links within the returned pages are forced to be processed by the Mediator. This approach works with all browsers, but requires users to navigate from within the CritLink-processed page instead of by using the browser's bookmarks or URL entry field. Support for Java applets which expect to access resources on their home server is unspecified. CritLink composes annotations by placing links to singular text comments (stored on the CritLink server) at the end of each page.

Both NCSA's Mosaic project [NCSA] and ComMentor [Rosc96] modify web browsers to augment them with annotation capabilities, and are therefore examples of a customized infrastructure approach to building annotation systems. ComMentor supports embedded annotations, and added browser menu functions for creating annotations and a popup viewer for examining annotation text. A merge library was developed for finding annotation anchors, placement, and handling anchor degradation. Both basic AReS capability and some annotation graph server capability are included. The NCSA Mosaic browser supports private annotations and group shared annotations in several releases. The annotations in NCSA Mosiac are composed as links placed at the end of the page. Modified browsers allow complete control over annotation function, but are unlikely to be widely used unless the modifications are included in widely distributed releases of popular browsers.

Strand/GrAnT [Schi96] and Net Notions [Netn98] provide two examples of systems that introduce web annotation function without modifying web content, browsers or servers. They are similar in philosophy to systems that we have built. Strand uses a client intermediary for page interception. The Strand (Stream Transducer Daemon) intermediary was conceived as a general-purpose service, extendible by adding application-specific modules such as GrAnT (Group Annotation Transducer). By using an intermediary, GrAnT was able to support all browsers. GrAnT inserted annotation function buttons at the end of a page and processed inputs from them directly instead of forwarding directives. GrAnT used some of the ComMentor components - a merge library for placing annotations in HTML pages, and its metaserver. The authors noted some problems with changing page layout when inserting annotation indicators and functions, and envisioned that much of the GrAnT functionality might be moved into a Java applet, an approach that JotBot explores. Net Notions is a commercial product from Sideware Systems, Inc. that allows users to affix text annotations "over" arbitrary web pages. Net Notions runs a client process that apparently performs event interception, interfacing to the browser to record current URL and position information when annotations are created. The client presents annotations as text squares positioned at the coordinates where the annotation was created; the text squares iconify to push-pin symbols. Annotations are stored on a workgroup server and can be grouped hierarchically by topic. Keyword search for annotations is provided, as well as an online messaging facility for communication, but only to those users who are currently connected to the server. Net Notions supports the most popular browsers, which implement support for event interception on the PC platform.

4.3. Future Work

The architecture outlined in the beginning of this paper implicitly assumes that interceptors, composers and AReS'es know about each other. While a single group might have a well known AReS, it is likely that groups around the internet would independently create thousands of such repositories that differ in the annotations they contain and the access capabilities they provide. A desirable way to scale an annotation system, is to dynamically discover and bind to AReS'es that contain annotations pertinent to the document being retrieved so that groups can leverage annotations created by other groups without having to access a common giant public annotation repository. Supporting such dynamic federation of AReS'es requires a resource discovery infrastructure that lets services find other suitable services on the Internet. Work on traders in the distributed object and AI communities [Venu98a] provides the beginnings of a resource discovery framework, but has not yet been applied to internet services. One line of work we are currently pursuing is to build Internet traders [Venu98b] that can support service discovery on the internet, which can then be used for AReS discovery. If composers are downloadable components such as applets, then the same ideas can also be used for the discovery and

downloading of remote composers and viewers that support a particular composition algorithm.

The move from HTML to XML is likely to significantly affect the annotatability of documents, and the annotation capabilities one can unobtrusively introduce into the web. Our near term focus on this front is to use XML as a repository format for AReS'es. An XML repository provides a lighter weight alternative to either persistent stores or full-function databases. Annotate! [Gins98] for instance, stores the annotation set for each document as an XML file. A number of other XML concepts such as semantic tagging and the ability to assert hyperlinks between documents outside either document, are likely to significantly change the architecture of browsers, and therefore the architecture of annotation systems. As WEBDAV [WEBDAV98] and other standards efforts settle on approaches to modeling and storing document metadata (including annotations), it will become possible for annotation systems to exploit these as standard AReS mechanisms.

5. Conclusions

The ability to annotate web documents provides a mechanism that can be the basis of a number of useful document management applications. We have presented an architecture to non-intrusively annotate web content, and two concrete implementations of this architecture. Our experience shows that useful annotation capabilities can be non-intrusively introduced into today's Intranets and Internets without resorting to specialized infrastructure. However, current web infrastructure restricts the flexibility and the implementation mechanisms by which annotation systems can be built. Future work will build useful vertical applications on top the core annotation function, address scalability issues associated with large scale annotation systems, and evolve the capabilities of the systems we have built to exploit evolving web infrastructure.

6. References

[Barr98] Barrett,R., and Maglio, P., "Intermediaries: New Places for Producing and Manipulating Web Content", *in Proceedings of the Seventh International World Wide Web Conference, 1998, http://www7.conf.au*

[Brav] Braverman. et al., "Annotations Protocol", http://www.ncsa.uiuc.edu/SDG/Software/XMosaic/Annotations/protocol.html

[Davi] Davis,J. and Huttenlocher,D., "CoNote - Shared Annotations for Cooperative Learning", *http://www-cscl95.indiana.edu/cscl95/davis.html*

[DOM98] "W3C Document Object Model Specification", *http://www.w3.org/TR/WD-DOM*

[Elo95] Elo,S., "Augmenting Text: Good News on Disasters", *DAGS95: Electronic Publishing and the Information Superhighway, 1995. URL: http://www.cs.dartmouth.edu/~samr/DAGS95/Papers/elo.html#haa95*

[Gins98] Ginsburg, M., "Annotate! A Tool for Collaborative Information Retrieval", *in Proceedings of WETICE'98, also at http://raven.stern.nyu.edu/papers/.*

[Hard96] Hardy,D., "W3C Distributed Indexing Workshop: RDM/SOIF", *Position paper in the Distributed Indexing/Searching Workshop, 1996.*

[LaLib] LaLiberte,D. and Braverman,A., "A Protocol for Scalable Group and Public Annotations", *http://www.hypernews.org/~liberte/www/scalable-annotations.html*

[Luke96] Luke,S. et al., "Ontology-Based Knowledge Discovery on the World-Wide Web", *Proceedings of the AAAI-96 workshop on Internet-Based Information Systems*

[Luot] Luotonen A. and Altiz,K., "World-Wide Web Proxies", *http://www.w3.org/hypertext/WWW/Proxies/*

[Mars97] Marshall,P., "Annotations: From Paper Books to the Digital Library", *in Proceedings of the ACM Digital Libraries '97 Conference, Philadelphia, PA (July 23-26, 1997)*

[NCSA] "NCSA Mosaic Group Annotations", *http://ncsa.uiuc.edu/SDG/Software/Xmosaic/Annotations/overview.html*

[Netn98] "NetNotions Product Details", http://www.sideware.com

[Resn96] Resnick,P and Miller,J., "PICS: Internet Access Controls Without Censorship", *Communications of the ACM, 1996, vol. 39(10), pp. 87-93*

[Rosc96] Roscheisen,M. et al., "Content Ratings and Other Third-Party Value-Added Information Defining an Enabling Platform", *D-Lib Magazine, August 1995, also at http://www.cnri.reston.va.us/home/dlib/august95/stanford/08roscheisen.html*

[Schi96] Schickler,M. et al., "Pan-Browser Support for Annotations and Other Meta-Information on the World Wide Web", *in Proceedings of the Fifth International World Wide Web Conference, 1996*

[Sumn96] Sumner, T. et al., "Open Peer Review & Argumentation: Loosening the Paper Chains on Journals", *Ariadne, Issue 5, Sept., 1996. URL: http://www.ukoln.ac.uk/ariadne/issue5/jime/ or http://kmi.open.ac.uk/~simonb/csca/jime-arg*

[Thom98] Thompson,C. et al., "Intermediary Architecture: Interposing middleware services and ilities between web client and server", *OMG-DARPA Workshop on Compositional Archiectures, 1998, URL: http://www.objs.com/workshops/ws9801/papers/paper103.html*

[Venu98a] Vasudevan,V., "A Reference Model for Trader-Based Distributed Systems Architectures", *OBJS Technical Report, URL: http://www.objs.com/staging/trader-rm.html*

[Venu98b] Vasudevan,V., "Trading-Based Composition for Component-Based Systems", *OMG-DARPA Workshop on Compositional Architectures, 1998, URL: http://www.objs.com/workshops/ws9801/papers/paper041.html*

[WEBDAV98] "Extensions for Distributed Authoring and Versioning on the World Wide Web -- WEBDAV", *WEBDAV Working Group IETF Draft Proposal, http://www.ics.uci.edu/~ejw/authoring/protocol/draft-ietf-webdav-protocol-03.html*

[Yee98] Yee,K.P., "The CritLink Mediator", *http://www.crit.org/critlink.html*