Shared Web Annotations as a Platform for Third−Party Value−Added, Information Providers: Architecture, Protocols, and Usage Examples

Technical Report CSDTR/DLTR

Martin Röscheisen
`rmr@cs.stanford.edu`

Christian Mogensen
`mogensen@cs.stanford.edu`

Terry Winograd
`winograd@cs.stanford.edu`

Computer Science Department
Stanford University
Stanford, CA 94305, U.S.A.

Abstract:

In this paper, we present an architecture, called "ComMentor", which provides a platform for third−party providers of lightweight super−structures to material provided by conventional content providers. It enables people to share structured in−place annotations about arbitrary on−line documents. The system is part of a general "virtual document" architecture ("PCD BRIO") in which−−with the help of lightweight distributed meta information−−documents are dynamically synthesized from distributed sources depending on the user context and the meta−information which has been attached to them. The meta−information is managed independently of the documents themselves on separate meta−information servers, both in terms of storage and authority. A wide range of useful scenarios can be readily realized on this platform. We give examples of how a more personalized content presentation can be achieved by leveraging the database storage of the uniform meta−information and generating documents dynamically for a particular user perspective. These include structured discussion about paper drafts, collaborative filtering, seals of approval, tours, shared "hotlists" with section−based visibility control, usage indicators, co−presence, and value−added trails. Our object model and request interface for the prototype implementation are defined in technical detail in the appendix.

Keywords:
Virtual Documents, Meta−Information, World−Wide Web, Group Annotations, SOAPs, Collaborative Filtering, Shared Workspaces, CSCW, Workgroups.

1 Introduction

There are many different reasons why people want to communicate to each other about specific things they find as networked information resources. These include comments and annotations of the kind a workgroup would share about their common area of interest, the ability to have newsgroup−like fora associated with specific items on the "net", value−added trails that link items together that someone considers being connected under a particular view, systematic critique and review information, "Seals of Approval" (SOAPs), or filters in support of enabling people to make sense of whatever information is presented to them, as well as usage indicators (such as how often a particular documents has been looked at by a group of people), "hotlists" generalized to shared structures which look different depending on who (with which access rights) look at them, or filters in support of enabling people to make sense of whatever information is presented to them.

All of these applications have in common two properties that are not associated with the standard mechanisms for hypertext (e.g., HTML):

- The overlay meta−information ("annotations") is not provided or managed by the owners of the annotated documents, and it is also in general not stored as a part of them.
- There are coherent "sets" of annotations that can be treated as a single document in terms of version maintenance, access control, archiving, etc.

For example, consider a set of "consumer reports" annotations provided by a review organization and attached to product catalogs on the web, or the private (within the group) comments made by our local research group as part of their joint reading of the WWW conference proceedings. In each case, the annotations themselves need to be kept separate from the annotated documents and access to them handled in a uniform way for appropriate subscribers.

We have developed a general mechanism for shared annotations, which enables people to annotate arbitrary documents at any position in−place, share comments/pointers with other people (either publicly or privately), and create shared "landmark" reference points in the information space. The framework represents a further step towards giving people a presence on the Web, laying the foundation for on−line communities.

This paper describes a general meta−information architecture (referred to in earlier documents as "PCD BRIO") and an implementation of this general architecture, called "ComMentor", which we have developed for annotating pages on the WorldWideWeb. which realizes such usage scenarios in a particular instantiation; this includes a basic client−server protocol, meta−information description language ("PRDM"), a server system (currently based on an NCSA http 1.3 server with CGI scripts written in PERL), and a remodeled NCSA xMosaic 2.4 browser with interface augmentations to provide access to our extended functionality [CHI95, SLIDES].

The idea of a system enabling a multiplicity of independent individuals to create lightweight value−added "trails" through a document space was envisaged most prominently by Vannevar Bush as early as 1945 [BUSH]. The ComMentor system can be thought of as a tool which such "trail blazers" use to add value to contents, and which other people use to find guidance based on these human−created super−structures. The overall architecture can be seen as a platform where value−added providers can provide their services (as a third player next to content providers and end users).

There are a number of existing systems that incorporate mechanisms that are related to our current architecture. These include the annotations facility in Lotus Notes (which requires making available "hooks" for annotation attachment in a given document), annotations in Acrobat (which are in−place but not shared), and various other in−place facilities which are based on a shared file system to allow multiple people to share comments (these include ForComment(TM), and some versions of MS Word−−both of which are not incremental, that is, only one user can write comments at a given time, and then pass on this right.) In the World−Wide Web arena, there has been ongoing discussion about the appropriate mechanisms for group annotations, but these have generally assumed whole−document attachment (rather than attachment in place) and universal (rather than access−controlled) distribution. They have not been developed into widely used facilities. There are a few experimental systems dealing with annotations of various kinds (mentioned in the references) including Ubique [UBIQUE], which uses a proprietary architecture, geared towards synchronous user−user communication rather than value−added structures.

In the first section of this paper, we describe the overall architecture including a description of the user view and a note on how documents are synthesized. In the second section, we present sample scenarios for the client−server interaction. In the remainder of the paper, we discuss our experience using the facility and describe a variety of usage scenarios that we are currently aware of.

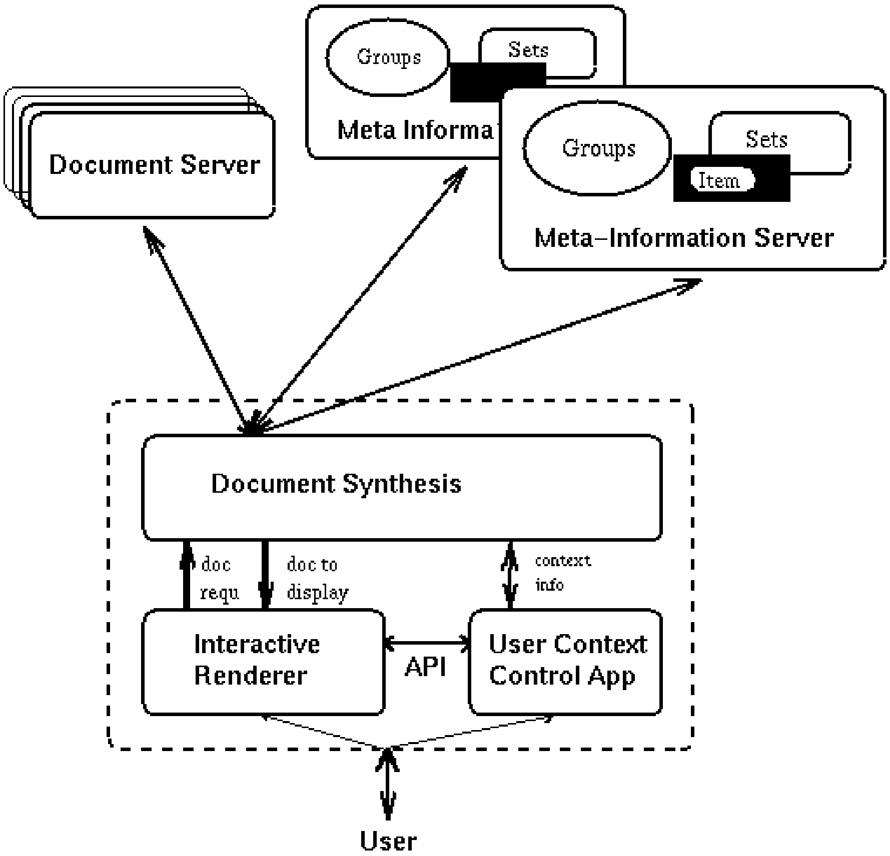
Much of the technical detail is left to the appendix: It contains, next to a glossary of terms, the specification of the client−server protocol as well as the definitions of the meta−information objects relevant for implementing shared comments to distributed documents.

2 System Architecture

In this section, we describe the basic system structure, the meta–information server design, the user view of the annotation system, and the current interface design of the browser. We also include brief technical notes on how documents are dynamically synthesized, and which meta–information objects we use.

2.1 Basic Structure

The basic system architecture is a depicted in Figure 1:



Users interact with a browser to retrieve documents from various document servers. In addition, there are meta–information servers from which relevant information can be retrieved according to the protocol defined in this paper. meta–information "items" like annotations are organized into "sets" to which members of "groups" have access. For example, depending on the context set by the user, the browser can decide to retrieve annotations from certain meta information servers for every document the user looks at, and display a version of the document in which annotations to various segments are shown at their appropriate position in the text.

Conceptually, a "browser" can be understood as consisting of

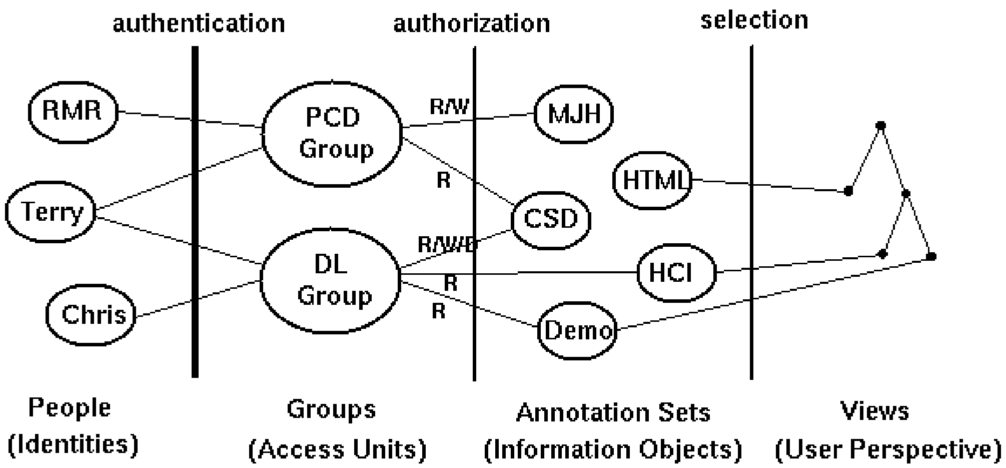
- an interactive renderer: an application like "Mosaic",
- a document synthesis module, which queries appropriate servers for documents and meta–information, and knows how to synthesize them together into a format which can then be displayed by the renderer (e.g. native HTML), as well as
- a context control application: a module which communicates with the renderer and the document

synthesis module, and keeps state information about the extended functionality from the user interaction (such as which of the additional options the user has selected at a given time).

Note that there are different ways to implement such a structure: For better interactivity, the complete dashed box can be realized in one address space (which amounts to "augmenting Mosaic", for instance); alternatively, the document synthesis module server can be factored out into a proxy server, and the renderer controlled by an independent application (which amounts to a "remote controlled Mosaic plus a proxy server"). For example, instead of reloading a possibly large document after adding or deleting an annotation, a browser that incorporates the document synthesis function can provide the same visual feedback by just adding or deleting the annotation internally. The main disadvantage of incorporating the document synthesis function is that people cannot use their unaugmented browsers for this functionality. We have designed the code structure in a way that allows us to experiment with the potential trade−offs involved in this spectrum.

2.2 Server Design

The following depiction shows the internal structure of a meta−information server, with annotation information as an example of meta−information items.



The figure shows the basic entities: *members*, who form *groups*, which access *annotation sets* that contain *annotations* (from left to right).

The basic information unit which can be linked to other documents and change their behavior is called an PRDMitem. This can include text content and other links, but will in general be a lightweight entity. Examples include meta information for annotations to specific locations in arbitrary documents, but also remotely stored user profiles.]]

The architecture is based on "annotation sets". Every annotation belongs to a particular set and annotates a particular page at some specific location. Every set is associated with a particular server ("annotation server") and identity (like a URL). The server holds the index (but not necessarily the contents) of all the annotations in the set, and will in general be distinct from the server which provides the annotated documents (and will not necessarily be under the control or resource allocation of the original writer). Each annotation set can be thought of as a "distributed document," stored in one place but consisting of a set of annotations that are associated with pages anywhere on the Web.

Access control is managed per annotation set. Some obvious examples are

- private sets (like private annotations in NCSA Mosaic), only visible to one person
- group sets (e.g., for commenting on work within a research group)
- public sets (corresponding to the way Usenet newsgroups are used).

Sets are particular information objects to which authenticated people stand in a certain authorization relation; groups are the human−organizational objects. There may well be more than one annotation set associated with a group, each used for different purposes.

The mechanism for authenticating access to meta−information sets (by virtue of being member in a group with appropriate access authorization) is an issue which is kept orthogonal in the design: any authentication mechanism (private keys, public keys, etc.) can be used. This allows us to separate out group membership (with a possibly very secure approval mechanism) and multiple sets of annotations (e.g. on different topics) which anyone in a specific group can easily access. Users can either have read or read/write access to a set by virtue of their group membership. Since there is no net−wide standard for individual identity and access control group membership, we have implemented a simple mechanism for establishing individual identities (with associated passwords) and group membership. We expect in the future that these will be generic network functions, and we will use the standard mechanisms that emerge.

For illustration, consider the equivalent in the UNIX file system: before being able to use a workstation, one needs an account ("group membership"), and once this is established, the user needs to authenticate herself. Then, once "logged in", users can create files and directories ("Sets") in whatever way is authorized by their group memberships for a particular location (e.g. write access for the group 'users' in the home directory). This distinction allows for differently strong authentication methods to be used without inhibiting the ease of creating directories/sets.

All meta−information items, including annotations, can be thought of as objects. In our current implementation they are represented in a textual object description language (PRDM). In future developments we plan to implement them as CORBA objects, whose interface definitions declare the protocol between the user proxy server and the meta server. However, the protocol specification is independent of this layer of representation.

In the current (non−CORBA) implementation, meta−information to the browser is transferred as part of a MIME message of a new MIME content type ("application/x−PRDM") which we are drafting [MIME]. This in conjunction with client accept information can be used to transfer meta−information as part of a MIME multi−part message along with other contents. meta−information can also be embedded into HTML: We have been experimenting with an extension to HTML in our prototype implementation, where the only addition is that of a META tag with 3 attributes LANGUAGE, VERSION, and CONTENT, each of which have opaque strings as values. The first two specify the identity of the meta−language being used, so that extended browsers will be able to parse the content by appropriate callback routines. (Anchors can have meta−information about the nature of the link as well.)

The PRDM meta−information formalism is being developed for a number of other uses (e.g., templates, maps, resource description, etc.). meta−information items can also be stored in a separate database (as done by the annotation server) and retrieved as part of a separate content type−−or even as part of some other (private) protocol (e.g. one which is not layered on top of http such as the ones based on CORBA Object Requests [CORBA93]).

2.3 User view of the annotation system

1. The browser provides a simple mechanism to identify a place on a currently visible page and write a

comment to it within a set you specify (assuming, of course, that you have write privileges to the set). At any time you will have a default set for writing. The browser makes it possible to have comments from any of the open sets show up in place on the page whenever you view a page that has a comment in that set (i.e., you do not have to know in advance what is annotated in which set).

2. The mechanism for specifying locations within a document includes redundant information and update information that makes it possible to determine whether a document was modified. In making an annotation, the user simply selects a region in the document being annotated, and the browser stores redundant information about it. When a page changes, an attempt is made to relocate the attachment point for the comment based on mechanisms such as embedded tags and/or string match. If that cannot be done, it is placed at the end of the document with a notation that it has been displaced, along with the (human−readable) context it was attached to.

3. Users can browse for annotation sets, and put together their personal selection. The selected sets are stored as part of the user's profile for their browser, which is built from the information about the user on the annotation server. Among these sets, users can designate specific ones as "active". If a set is active, then the comments in this set for a document being retrieved are retrieved from the comment server for this set. Information integration is supported both at server−side and at browser−side.

4. Annotations can be indicated in an interface in a number of ways, including marginal markings (as in LaTeX), format−marking of annotated text (as in WWW browsers with underlined anchors), in−line presentation of the annotation text, and in−place annotation indicators. We are experimenting with several of these. The current browser uses in−place markers, with character−size in−lined images marking annotation points, and optional highlighting of the annotated text element. The type of marker image can be user−determined, depending on the perspective a user wants to gain on the ensemble of annotations: if interested in authorship, the images can show each author's face or individual icon; if the identity of the group sharing the annotations is more central, then the images can show a small icon representing the group or the specific annotation set. Although the images are kept small in order not to interfere with document content (often as small as 16x22 pixels), informal experiments have suggested they have sufficient discriminability and identifiability even for faces.

5. The browser interface includes mechanisms which support different styles of reading annotations. The annotation icons themselves are "hot" links: if selected with the left mouse button (which is the Mosaic convention), a full document view of the comment is displayed. This view can then contain other images of the author (in larger size) as well as further links, for instance, to longer elaborations; it may also contain further annotations or follow−up comments (see below).

   Annotations can also be examined with a lightweight viewer which pops up a small window (which looks like a PostIt(TM)) when the icon is selected with the middle button and removes it when the button is released. This tool is a generic meta viewer which can be used in a variety of contexts to get "preview" information faster than it is possible with a full document−view window. It is useful as a general interface augmentation for mosaic browsers in contexts such as examining whether or not to follow a hyperlink, which might lead to an expensive document.

   Annotations can be filtered according to different selections; a typical usage is to filter review information for category labels to see only documents with a certain rating. Note that queries for particular items from the meta information server are supported by its database backend.

   Since annotations are documents too, they can be recursively annotated.

6. Annotations are considered write−once. They cannot be edited (but of course additional ones can be made). People with appropriate privileges can remove them. Annotation sets can have different policies with regard to deleting.

7. A tour of the annotations in a set that have been created more recently than a given time can be queried from the servers. Such a tour is a list of links, each of which will bring the user to where the comment in the annotated page is located. This greatly extends the value of the annotations, since it enables users to find annotations that are distributed over a number of documents, without needing to check all the pages of those documents for changes. The fact that the list is limited to a specific set makes it more useful than a general "what's new" list.

2.4 Meta−Information Objects

Meta−information is used to convey information about a document in a uniform, machine intelligible way. The meta−information description used in this work is code−named PRDM *(Partial Redundant Descriptive Meta−language)*. It is a typed object language which we are developing for a broader range of purposes in the context of the Stanford Integrated Digital Library Project [DL94]. Some of the general properties of the PRDM meta language are:

1. PRDM is a declarative (not a procedural) object−based language. The basic structure is frame like with the intent to have a uniform way of expressing descriptions of objects:
2. It is self−identifying: name and version can be identified without knowing about the language structure.
3. It is semantically grounded: attribute names are chosen to correspond to concrete semantic notions.
4. Descriptions can be partial, distributed, and redundant: Some PRDMitems at a given site might only describe parts of the complete meta−information known about a given object. Unified together from different distributed sites, some PRDMitems might then express redundant information.
5. It can be easily embedded into HMTL and other text formats.

Since the work on the kind of meta language to use is very much in flux as part of the more general context of a digital library, we present here a simplified version of PRDM which is more specifically geared towards the application of this paper; it represents the form in which it is used in the prototype implementation ComMentor version 1.0, and all disclaimers apply towards the language which ends up being used in the digital library project or in future versions of ComMentor.

Here is an example:

```
(annotationSet
        (name "SampleSet")
        (email "sampleset@pcd.stanford.edu")
        (admin (person (name "Christian")
                       (email "mogensen@cs.stanford.edu"))))
```

This information would indicate to a browser that the page describes an annotation set and some of the properties of this set. meta−information can always be ignored without harm (e.g. by browsers which do not speak the particular meta language)−−but at the cost of decreased functionality.

For the details about the specific object types used in the shared annotation architecture, see Appendix B.

2.5 Dynamically Synthesizing Documents: The Merge Library

The merge library is the set of procedures that actually synthesize the document that is then rendered from the documents and annotations that are relevant in a given context. It contains procedures that take a document and a PRDM list of comment items, and return a document where the comments are in−lined and given an appropriate rendering. The procedures are specific for the content type of the document.

The method of attachment for HTML and plain text is currently based on string position trees (Patricia trees for positions; cf. [KNUTH]) applied to a canonical representation of the document. Each comment object has associated with it the highlighted text string as well as the position identifier string.

Position identifier strings are useful in this context because they are by definition the smallest internal identifying string, and therefore are likely to be robust against modifications of the underlying document. They also allow for changes to be detected.

If the position cannot be recovered, the browser appends the corresponding item marked as "Unassigned" to the end of the document.

For a flexible system with maximum interactivity and generality, it is quite useful to have incremental insertion mechanisms; that is, not all relevant meta–information pertaining to a document in a given context has to be known in the beginning. An example would be that a user activates a new set, or one set is returned by a slower server, or another server has already pre–merged the meta–information from some of the sets while the browser merges the rest of the sets. Then we want to be able to merge in the additional meta–information from the relevant sets incrementally. Note that the requirement of incremental insertions determines how the merge algorithm has to look like. For example, global position identifiers such as simple position counts would not work in a straight–forward sense since they are affected by previous insertions. This raises the need for a canonical representation of a text in a certain format which embodies all the features that are significant for attachment but do still allow text transformations which are invariant with respect to the canonical form. (For example, inserting an HTML comment into an HTML file should preferably not affect the position identifiers.)

Procedures for other content types, especially images (and possibly external viewers), have not yet been examined. If the type already supports a concept of attached annotations (e.g., PDF in the Acrobat viewer), then we could make use of that directly in identifying the position of annotations which are stored on a group server.

3 Client–Server Interaction: Sample Scenarios

In this section, we give two sample scenarios of how browser and server interact:

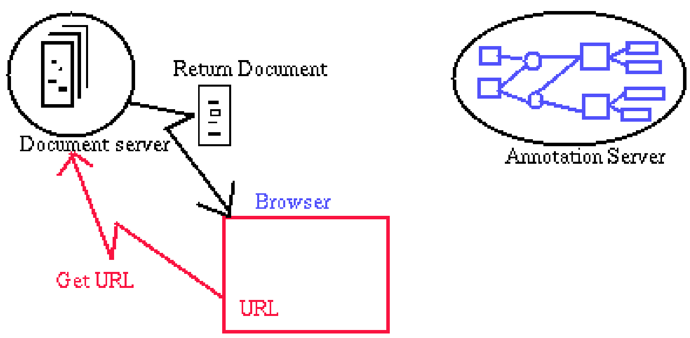
- Retrieving a document and its annotations.
- Joining a group and adding a annotation set.

A technical specification of the client–server protocol and the meta–information objects can be found in the appendix.

3.1 Retrieving Document and Annotations

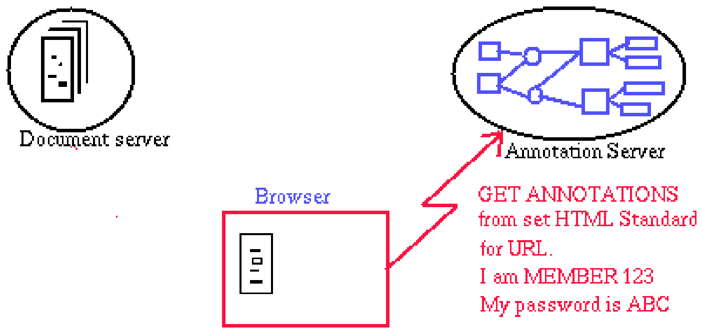
A document and the annotations of the activated annotation sets are retrieved concurrently from document server and annotation servers, respectively.

Fetching a document



Retrieving the base document is the standard document browsing ("GET") interaction.

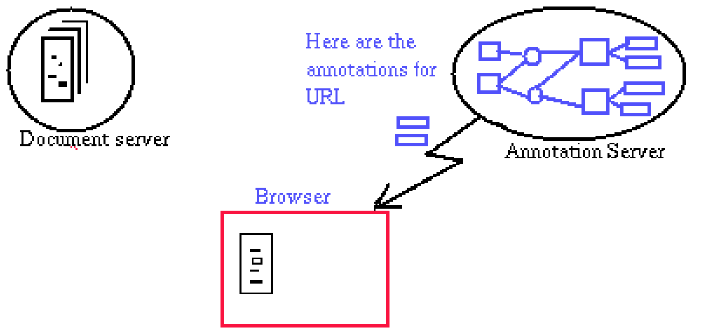
Requesting Annotations



Whenever Joe accesses a new page, the browser sends out a request for annotations related to the URL just loaded to the annotation server; this request includes:

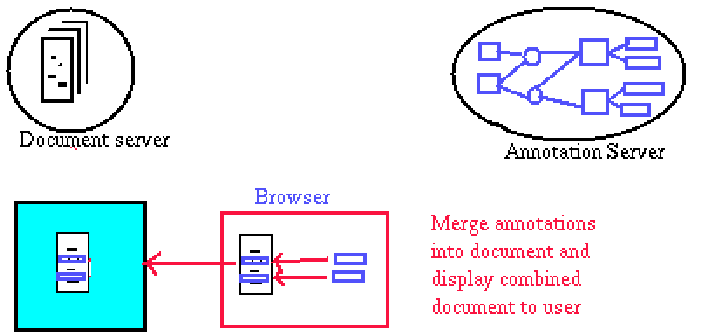
- the URL of the document,
- which annotation sets we want annotations from this particular server: "HTML Standard", and
- Joe's authentication information.

Server Returns Annotations



The server sends back a string of meta–information which the client uses to merge with the original document. The document is then rendered with the annotations inside it.

Combine Annotations with Document

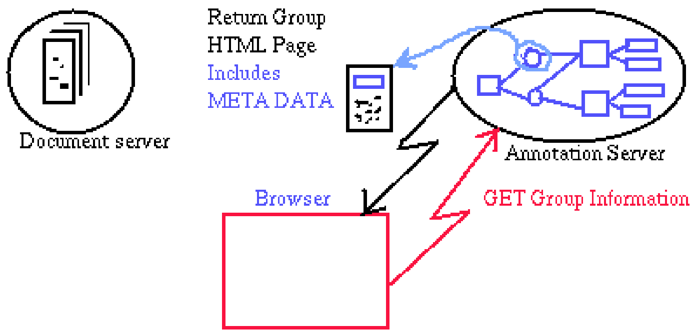


The merging of the document and the annotations is done by a set of procedures based on redundant information and minimal length tree descriptors which uniquely identify the position in a document and which are designed to be maximally robust against change of the underlying document. (See also the section on the "merge library" for synthesizing documents from other documents and accompanying meta information.)

3.2 Joining a Group

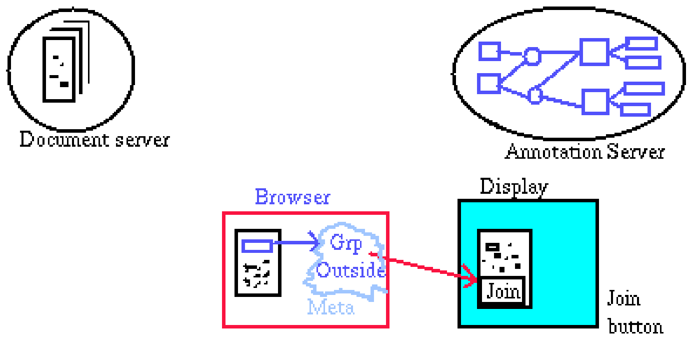
Before we can read comments in a set, we must belong to a group with access to that annotation set.

Fetching group meta–information



Setting up an annotation server requires augmenting a conventional http server with the appropriate scripts. The groups which are available at such a server are described in conventional HTML pages which are augmented by meta–information as to who is the owner, whether the group is public, and other properties. These pages can be browsed in the usual way. We have set up an initial *What's New* type list which contains currently only our servers, but others are likely to come, and people can use such lists to browse for groups they want to join in the same way they also browse for other documents (which also means that such information defines the interface for search "agents" which can be looking for new groups of relevance).

Group: User Interface



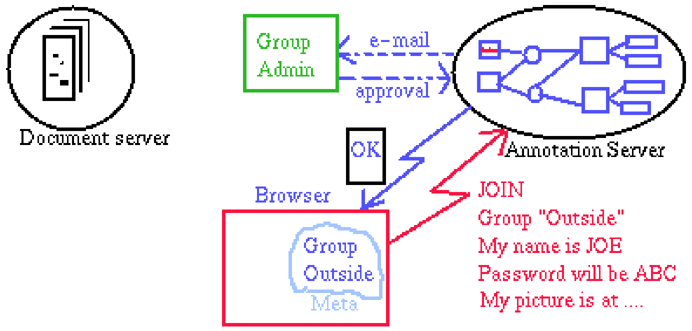
The annotation server sends back a page containing meta–information about the group. The browser extracts this information, displays the page, and indicates in its user interface that the current group is eligible for a request for membership.

Example: The `Join Group` button is made active.

A new user 'Joe' checks out the server and discovers a few interesting annotation sets there. Unfortunately, they are all closed to non–members, so Joe applies to join a group named *Outsiders* which is described as:

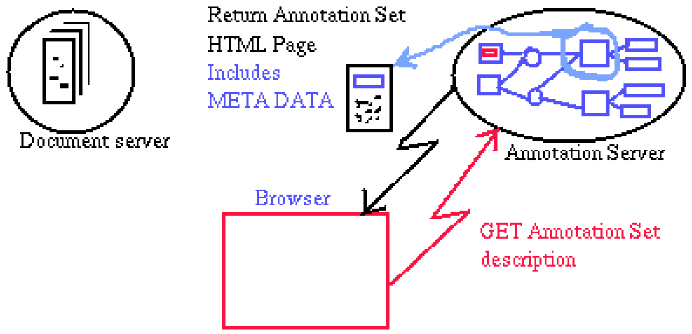
This group is intended for interested parties outside the lab. It will allow you to access the following annotation sets and contribute to the discussion of our research by adding your own annotations. The documents of interest tend to reside on this server. Check each annotation set for details.

Join–Group Request and Approval



The *group administrator* receives e–mail notification of the request and approves the request by sending a reply back to the annotation server's mail gateway. In other words, joining a group is much like joining a mailing list. However, a group is merely an access authorization unit. Groups have nothing to do with the actual creation of annotations; they are dealt with fairly infrequently (like setting up a new UNIX account for someone).

Adding an Annotation Set



After receiving an e–mail notification of the approval, Joe decides to go back to the annotation server and select a few annotation sets. On viewing a page describing an annotation set, his browser detects the embedded meta–information and enables the `"Add Annotation Set"` button in the user interface.

Joe adds the annotation set "HTML Standardization" to his list of subscribed sets. His browser pulls out the embedded meta–information and adds the information to the list of stored annotation sets. The meta–information includes the set's name and location as well as some caching data such as which documents have been annotated.

Joe then *activates* the set. Annotations for all active sets are always requested from annotation servers by the browser––conceptually for every browsed document (but a caching mechanism by which only those document servers are queried which are known to be annotated leads to an efficient implementation).

4 Usage Scenarios

In this section, we present some usage scenarios which follow naturally from the described architecture.

Shared WWW Comments

This is the basic annotation scenario described above with annotations organized into sets accessed by declared groups. It is possible to share comments with undeclared users by declaring the pseudo−person `anyone` to have read access to the annotation set.

A collection of annotations are usually rendered in−line as small icons. Each icon is a link to a dynamically generated HTML page containing the annotation text. Such a page can then be annotated recursively, or a follow−up comment can be made (which is usually rendered linearly)−−providing a threaded stream of comments.

Note that the actual visual presentation of comments is independent of the meta information transmitted; different browser designers can experiment with different renderings without affecting the underlying functionality.

A useful feature when using this sort of set is the ability to query the set for recent additions. If one is interested only in annotations which are less than a day old, or written by other members of a particular group, then the meta information server can perform these selections as queries over the meta information database, and make this information available as links to the relevant locations. This makes it easy to track new annotations to a given document or a particularly interesting discussion.

Personal Annotations

Anyone can create an annotation set which does not give access to anyone else except oneself. This is the case of personal annotations stored on a server. By storing annotations remotely (instead of in the local file system), the same generic searching and cataloguing functions as on any other set are available.

Another use of personal sets is the possibility of creating very informal shared sets that exist outside the formal group structure. In other words, if someone wants to share information with his office mate, it is possible to create a set to which both have read and write access.

Landmarks, Tours, and Trails

In order to navigate a large distributed web of documents, it is useful to have "landmarks", that is, places which people are familiar with and which they choose as reference points. The annotation mechanism can be used to generalize existing concepts used on the WWW such as the notion of "hotlists". Leaving a mark on interesting documents and then querying the annotation server for a list of these annotations, in fact provides a "hotlist" which can be shared. The fact that this is embedded in a generic mechanism (with all other searching and cataloguing functionality available) in conjunction with being more dynamic (for instance, ‘most recent’ queries are possible which give only the recently modified part of a structure) makes it even more useful. Landmarks are particularly useful in combination with a tour mechanism which we have implemented: Taken together, this realizes a form of collaborative filtering.

The tour mechanism is currently implemented simply as a (specifically typed) list of links for each comment thread and a two−pane browsing interface such that the tour context is always maintained in one view and the tour focus is rendered in the other view (e.g. the annotated document with the comment included). Each such tour link will bring the user to where the comment in the annotated page is located. Since it enables users to find annotations that are distributed over a number of documents without needing to check for changes all the pages of those documents, this greatly extends the value of the annotations. We are investigating the advantages of more sophisticated graphical visualizations, such as maps.

Trails can be applied to implement multiple guided tours through the same content without confusing the users: For illustration consider a set of paintings and their descriptions such as they can found in the "Web Louvre". It is now possible to create annotation sets "impressionist painters", "chronological tour", and others, and add trails marks (a special annotation type) to each set which form a linked tour according to the set semantics. Then, if people wish to look at the paintings in a chronological order, they can activate the corresponding set, and follow along a tour by clicking on the trail mark icons. If they wish a different tour, then they can turn on another set. In any case, they will not be confused by multiple signs on a given page, because the annotation architecture allows for underlying contents and connecting super−structures to be separated (in the static representation) and dynamically synthesized together based on a chosen user context.

Shared Hotlists with Section−based Visibility Control

Queries over a range of annotation sets allow people to use the annotation architecture to maintain a "hotlist" which is shared among different people, and where certain parts of it are visible only to people with appropriate access authorization. A hotlist URL would be the query to the document synthesis module which collects the lists of pointers from the different annotation servers, and merges them into a comprehensive list (e.g. a list of sections where each section corresponds to a topic/a set, and each topic/set contains a list of pointers to documents).

To add a document to one of the sets which constitute a hotlist, a user simply marks the page with respect to a particular set (which might correspond to a "section"/topic in the hotlist). Marking a page adds a piece of meta−information to the chosen set. A straightforward extension is the use of multiple hotlists, like one for each topic: different sets. Such hotlists would then be retrieved by asking the server for a list of all documents in the set. The benefit is that the hotlist can be shared by several clients simultaneously, and it is a by−product of the architecture that access control is at the level of granularity of the sections of the hotlist, which thus allow to control the visibility of the different sections.

Seals of Approval (SOAPs)

A seal of approval (SOAP) is an idea which occurred in the Interpedia Project [IP93] and later assimilated into the URC draft [DM94]. A seal of approval is meta−information containing a rating describing another document. In our decentralized model, there may be several SOAP authorities (that is, servers containing SOAPs), and several dimensions along which ratings are made. The semantics of the various dimensions can also be described using meta−information.

As a hypothetical example, consider the case that someone has a high opinion of the quality ratings which a certain academy issues about documents, and wants to be advised by these ratings while browsing for documents. Then, such an academy could create a Guidance SOAP; this is just an annotation set where the access authorizations are chosen such that anyone can read it, but only fellows of the academy can write new comments. Then, interested users can can enable the agency's annotation set, and when they browse documents, they can quickly get an idea of how valuable it will in all likelihood be to spend time on a given document: by glancing at the seals. Note that with the tour mechanism it is moreover possible to look explicitly at those documents for which a very positive rating exists.

SOAPs can be implemented in the given system in a straight−forward way; they are just annotations whose content follows some category system. Browsers can then exploit this systematicity by associating some action model to the various categories; such actions can range from popup hints to not retrieving a particular page.

Usage Indicators

In many current WWW browsers, the browser keeps a global history list and indicates to the user which URLs have already been looked at (e.g. by underlining them in a different color). The annotation mechanism can be used to extend this useful idea to indicators of usage by arbitrary groups of people: the browser would simply write a mark to a certain "usage set" of a group, and then always check with that set whether there is a mark in it for a particular URL. One simple possibility for the interface would be to use a grey−scale to indicate more frequent usage.

Usage indicators are important in combination with attention management. If there are indicators which show that a particular group of people hasn't seen something yet, then one can easily go ahead and notify these people about it (if interesting) without taking up attention span unnecessarily.

5 Future Work

We are currently testing the various usages as part of our internal project communication in the context of the Stanford Integrated Digital Library Project [DL94]. That project is developing a general architecture, called an InfoBus, for integrating "library services" of all kinds for the production, dissemination, maintenance, search and access of information objects. The kinds of services described in the above scenarios are among the services that would form the overall capacities of a digital library. The InfoBus architecture is based on a CORBA object model, and our implementations will be modified to work within that model.

We are gathering experimental data to be able to evaluate the design of specific annotation−based services, in simple pilot applications used by participants in the digital libraries project. These include extensions to materials that are not in HTML (e.g., Adobe Acrobat), and to using the mechanisms as a basis for "virtual place" in on−line communities.

Acknowledgements
Members of the Project on People, Computers, and Design at Stanford Unversity, in particular Michelle Baldonado and Steve Cousins, have provided valuable feedback throughout the design and development of the system.

References

[BUSH]
Vannevar Bush (1945). As we may think. *The Atlantic Monthly,* July. URL: http://www.csi.uottawa.ca/~dduchier/misc/vbush/as−we−may−think.html.
[CHI95]
Martin Röscheisen, Christian Mogensen, and Terry Winograd (1995). *Interaction Design for Shared WWW Comments*. Short Paper, CHI95.
[CORBA93]
DEC, HP, HyperDesk, NCR, ObjectDesign, and SunSoft (1993). *The Common Object Request Broker: Architecture and Specification.* OMG Document Number 93.xx.yy. December 1993.
[DL94]
Hector Garcia−Molina, Yoav Shoam, and Terry Winograd (1994). *Stanford Integrated Digital Library Project*. Computer Science Department, Stanford University. (NSF/ARPA/NASA proposal). URL: http://www−diglib.stanford.edu/diglib.
[DM94]
Ron Daniel, and Michael Mealling (1994). *URC Scenarios and Requirements.* Draft, Internet Engineering Task Force, November 21.
[DT94]
Daniel LaLiberte (1994). HyperNews. URL: http://union.ncsa.uiuc.edu/HyperNews/get/hypernews.html.
[DW94]
David R. Woolley (1994). Conferencing on the Web. URL: http://www.well.com/Community/drwool/webconf.html.
[IP93]
Interpedia Project (1993). URL: news:comp.infosystems.interpedia.
[JD94]
Jim Davis (1994). CONOTE: small group annotation experiment. Jim Davis and Dan Huttenlocher. URL: http://dri.cornell.edu/pub/davis/annotation.html.
[FH94]
Francis Heylighen (1994). The Principia Cybernetica Web. URL: http://pespmc1.vub.ac.be/.
[GRA]

Gramlich (1994). Public annotation systems. URL: http://playground.sun.com:80/~gramlich/1994/annote/.
[KNUTH]
Knuth, D. (1973). *The Art of Computer Programming*. Vol. 3. Addison−Wesley.
[MIME]
Nathaniel Borenstein, and N. Freed (1993). *Multipurpose Internet Mail Extensions*. Draft, Internet Engineering Task Force.
[NCSA]
Mosaic Design Team (Sept 1993). *Group Annotations in NCSA Mosaic*. URL: http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/group−annotations.html.
[OPM]
John Mallery (1995). Openmeet. URL: http://www.npr.gov/OpenMeet/openmeet.html.
[SLIDES]
Martin Röscheisen, Christian Mogensen, and Terry Winograd (1995). Slides for Presentation. Third International WWW Conference, Darmstadt, Germany. URL: http://www−diglib.stanford.edu/rmr/WWW95/talk
[UBIQUE]
Virtual Places Architecture, the Doors Server, and the Sesame Navigator. URL: http://www.ubique.com/

Appendix A: Glossary

PRDM
A meta−information language (*Partial Redundant Distributed Meta−information*) which is describing the objects that are passed forth and back by the clients and servers of the architecture. The version presented in this paper is a simplified cut at what we are developing as part of a larger project.

ComMentor
The name of the World−Wide Web prototype implementation of the system described in this paper, that is in terms of code, an augmented NCSA Mosaic browser and an NCSA HTTP server augmented by a set of server scripts.

PRDMitem
A piece of PRDM meta−information, the top−level structure.

PRDMitem set
A collection of PRDMitems accessibly by one or more groups.

Annotation
A specific PRDMitem which refers to some location in some document; it does not necessarily have to be stored as part of this document.

Annotation set
A collection of annotations accessibly by one or more groups. Sets are like discussion threads on a mailing list.

Group
A collection of members, like a mailing list is the list of subscribers to the mailing list. Groups have names, icons, places and a list of members associated with them. They are defined in PRDM.

Member
A description of a person stored on a server. Members are uniquely identified by the distinguished name and a password. A fuller description of a member is usually stored as a *user−profile*.

Place
The partial URL location of a meta−information server. Needs to be augmented by the type of request.

Icon

A tiny picture intended to be inlined in the text to represent an annotation, a group or a person.

Pict

A larger picture intended to symbolize ownership or membership. Think of it as the headline version of an icon.

Admin

The person responsible for the day−to−day approval and maintenance of a group or set.

Owner

The person responsible for the content of a group or set. May be different from the Admin.
e.g: The ComMentor list is owned by Terry Winograd, but administered by his trusty graduate student, Christian Mogensen.

User

The person interacting with the browser. A user has a unique ID, her Distinguished Name (DN) and a user−profile.

User−Profile

A collection of information about a person. This user−profile may be stored in the browser's local file system, or remotely on a user−profile server (just another meta−information server). By storing the profile remotely on a server, the user gains a place on the web, where other servers may put or read notification information.

Person

A Distinguished Name − usually as seen as part of a member record or a user−profile. Basically people and their DN's are at least one−to−one. Usually defines name, email address and the DN, as seen in
`(person (name "some name")(email foo@bar)(dn "blatz.in"))`

Distinguished Name

A unique identifier that refers to a particular person. The term Distinguished Name comes from the X.400 catalog.

Anyone

A special member which anyone can be. This member exists on all servers. The member `Anyone` with no password is allowed limited access to the annotation server. It is up to the creator of a set to allow read access to the member `Anyone`, in effect making the group publicly available. Giving write access to `Anyone` effectively creates a newsgroup−like forum. Note that since 'Anyone' is just another member, this case fits naturally into the access control mechanism without much special code: Access to a server is authenticated with name and password/key of the user at a particular server; if the user has an "account" at this server, the appropriate name and password is taken; if he doesn't, then name is simply 'anyone' and password is the empty string.

Appendix B: Object Specifications for Annotation System

This section has primarily two sections: In the first, we quickly explain how we specify PRDM objects in this paper. In the second, we use this notation to describe the major PRDM objects for the annotation system application. In a subsequent section, we point out some implementation details about how ComMentor 1.0 transfers meta objects (for developers).

Conventions: Specification Language

We use in this paper the following conventions for purposes of exposition in order to specifiy particular examples of objects in PRDM for the application of the architecture in this paper,

We use templates of the following syntax to define the frame structure of PRDM expressions:

```
ObjectType {
        attribute_1 = ObjectType_1
        ...
        attribute_n = ObjectType_n
}

ObjectType {
        ObjectType_1
}

ObjectType {
        primitive
}
```

Such a template would be instantiated by PRDM expressions of the corresponding structure (self−identifying objects)−−the textual representation of specific meta−information.

The notational convention is that type names start with upper−case letters, and selector names (and atoms) with lower−case letters.

For example, the hypothetical type `Document` could be specified by a template like

```
Document {
        author = HumanName {
                    firstName = String
                    lastName = String
                 }
        content = String
}
```

which then could be instantiated by an expression

```
(Document
        (author (HumanName
                    (firstName (String "James"))
                    (lastName (String "Jones"))))
        (content (String "Hello World")))
```

Note that for the sake of brevity, the general definition of a subtype (here: `HumanName`) can be included in the definition of the template which contains the type.

A suffix `−List` for an object type in a template indicates that the following structure is a set of such objects. Alternatives are indicated in templates with '|' between types. Optionality of attributes can be explicitly indicated by allowing `null` as one of the alternatives. The notation `Type.attribute` is used to refer to a particular substructure in the usual way.

Annotation System Objects: Specification

The following subsections describe the major PRDM objects used in the shared annotations system ComMentor 1.0. We first define some Common Base Objects, then give the specification for Access Groups, PRDMitems, PRDMitem Sets, and User Context Profiles as well as other descriptors sich as those for Errors and Caching Information.

Common Base Objects

Frequently used base structures include the following:

- String: a sequence of characters; quotes indicate the presence of a string (even in absence of the type specifier). Example: "This is a string"
- URL: as specified in the corresponding RFC. Example: http://pcd.stanford.edu/
- Email: as specified in the corresponding RFC. Example: foo.bar@twiddle.twaddle.com
- DN { String }: distinguished name, i.e. a unique identifier for a person. Example: "foo.bar@twiddle.twaddle.com"
- Name { String }: some name. Example: "ComMentor"
- HumanName { Name }: name of a person. Example: "Christian Mogensen"
- Bool { 0 | 1 }: Boolean value (0 = false, 1 = true).
- Integer: the integers (range as per 32 bit ints)

Note: The Distinguished Name (DN) is currently just a fully qualified e−mail address (not an X500 string). In the future, it may contain spaces, so DN's are String.

The notion of a `place` is used to talk about where entities like groups, sets, or scripts are located. A place has currently two access methods (depending on which gateway is used).

```
Place {
        url = URL
        email = Email
}
```

Persons are described by the following template:

```
Person {
        email = Email
        name = HumanName
        dn = DN
        pict = URL
        icon = URL
        home = URL
}
```

A person describes a member or a very partial user profile. A person construct is usually found as an element defining a larger role. Hence in the `PRDMitemSet` definition we find `(admin = Person)`. This would expanded in an instantiation to

```
(admin (Person (email mogensen@cs.stanford.edu)
               (name "Christian Mogensen")
               (dn "LN=Mogensen CN=Christian MI=L
                    OU=CSD O=Stanford")))
```

The above gives us a tag telling us about a role *admin* and information about the role, namely who performs it.

Access Group

The following template defines the frame structure which is used to describe access control groups.

```
Group {
        name = HumanName
        home = URL
        icon = URL
        pict = URL
        description = String
        mailList = Bool
        openGroup = Bool
        admin = Person
        owner = Person
```

```
        place = Place
}
```

Group information is important for knowing what a group is about, which restrictions it has, and how one can join it. The `mailList` attribute designates whether it shall be possible to send e−mail to all members of the group. `description` is a human−readable description of the purpose and intent of the group; it might be displayed for people which are browsing for groups. `pict` and `icon` are a picture and an icon standing for the group; `home` is its home page, if any. `openGroup` clarifies whether administrator approval is necessary for people to become members of the group (much like many mailing list servers have approval procedures).

Here is an example instantiation:

```
(group
        (name "Comments on ComMentor")
        (home (url http://foo.com/foo.html))
        (icon (url http://foo.com/foo.tiny.gif))
        (pict (url http://foo.com/foo.gif))
        (description "For comments on ComMentor")
        (mailList 1)
        (openGroup 0)
        (admin (Person (email mogensen@cs.stanford.edu)
                       (name "Christian Mogensen")
                       (dn "LN=Mogensen CN=Christian MI=L
                            OU=CSD O=Stanford")))
        (owner (Person (email mogensen@cs.stanford.edu)
                       (name "Christian Mogensen")
                       (dn "LN=Mogensen CN=Christian MI=L
                            OU=CSD O=Stanford")))
        (place  (url   http://pcd.stanford.edu/cgi-bin/postit/)
                (email www-server@pcd.stanford.edu)))
```

ComMentor 1.0 Implementation: The group's `place URL` is the partial URL to the root of the PRDMitem server. It indicates to the browser where where to send join requests, and PRDMitem requests; `email` is a mail gateway which is used among others for approving new membership requests. In other words, `(place (url http://foo.com/bar/) (email baz@foo.com))` implies that

1. the PRDMitem server is at `http://foo.com/bar/annotation/server`
2. the group server is at `http://foo.com/bar/group/server`
3. the group e−mail can be sent to `baz@foo.com` with `[GroupName]` in the subject line.

PRDMitem Set

Each PRDMitem set can be accessed by many groups: Hence there is a structure (namely, `groupAccess`) which describes which groups are authorized do what to a set.

There is no mailing list associated with a set−−mail can only be sent to groups.

```
PRDMitemSet {
        name = Name
        groupAcess = GroupAccess-List
        home = URL
        icon = URL
        pict = URL
        admin = Person
        owner = Person
        annotatedServers = FQDN-List | all | none
        active = Bool  |  null
        description = String
        place = Place
}
```

where

```
GroupAccess {
        name = Name
        setAccess = SetAccess {
                       read | write | delete
        }
}

FQDN {
        String
}
```

`FQDN` is a fully−qualified domain name such as "www−pcd.stanford.edu". It is part of a list which is associated to a set in order to indicate which ones are the servers to which items in the set are referring in some form (caching information).

PRDMitems

This section looks in detail at the structure of the meta−information items, the PRDMitems themselves. Such information is returned, for example. as part of the PRDMitem_get request. Examples include annotation items, user profiles, and more.

```
PRDMitem {
        PRDMannItem  |  PRDMprofileItem
}

PRDMannItem {
        author = Person
        content = DocItem {
                     title = String
                     url = URL
                     text = String
                  }
        annid = PRDMitemID {
                     date = Date
                     number = Integer
                     url = URL
                     set = PRDMitemSet
                }
        posid = PosID {
                     type = ContentType
                     info = PosID_HTML  |  PosID_PS  |  ...
                }
}

PosID_HTML {
        url = URL
        string = String
        relpos = 0 | 1 | 2
        id_len = Integer
        sel_len = Integer
}
```

The `PosID` structure is a position identifier descriptor where `string` is the position identifier string (cf. [KNUTH]), `id_len` is the length of it (allowing storing redundant information in `string`), `sel_len` is the length of the region in the document which the `PRDMitem` is referring to in the document at the specified URL. The `annid` attribute is an identifier containing the set to which the item belongs, a unique number in this set's items, the date of creation as well as a URL by which the item can be requested separately.

User Context Profile

The following context information is kept about a user (in the browser or on the user profile server). Such information can be stored on the meta–information server with a protocol equivalent to the one used for annotation structures.

```
PRDMprofileItem {
        userID = String
        person = Person
        lastUpdate = Date
        nonce-SetUserProfile = Nonce
        configServerScript = URL
        configServerName = Name
        options = Options {
                     defaultPRDMitemIcon = URL
                     fancy-selection = Bool
                     delay-img-load = Bool
                     font = Font
                     ...
                  }
        hotlist = Hotlist {
                     title = String
                     section = Section {
                                  item = Item {
                                            url = URL
                                            title = String
                                            comment = String
                                            date-added = Date
                                         }  |  Item-List
                               }  |  Section-List
                  }
        grouplist = Group-List
        setlist = PRDMitemSet-List
        serverlist = Server {
                        url = URL
                        password = String
                     }  |  Server-List
}

Nonce {
        Integer
}
```

The `options` include an extensible list of browser interface settings. `grouplist`, and `setlist` describe which groups a user is a member of, and which sets she has chosen. Nonces are used to guard against intercepting and against spurious meta–information, that is meta–information which is associated in the proper format with documents without that it was the users's intention to see the effects of this meta–information.

Errors

The following object is sent as a response when something goes wrong, such as not having sufficient access authorization by not being member in a specific group. The string is a message which the client can display to the user.

```
ServerError {
        code = Number
        error = String
}
```

Example:

```
(ServerError (code 1)
             (error "The server cannot access this document."))
```

Caching Information

The caching information is returned whenever PRDMitem or PRDMitemSet information is returned. This information helps the client to determine where specific PRDMitems are, and whether it might be unnecessary to look for them at certain places. This leads to a substantial performance improvement since in general only few documents are referred to by an PRDMitem.

The browser queries the PRDMitem server in sufficient frequency to ensure the currency of its cache. Consider the scenario with one annotated server. If the user rarely visits the annotated site, then PRDMitems for that set will rarely be retrieved, hence the caching information for that set will be seldomly updated. Therefore the browser needs to refresh its cache every now and then.

```
CachingInfo {
        distribInfo = PRDMitemDistrib {
                          PRDMitemSet-List
                      }
}
```

The special case of `PRDMitemSet.annotatedServers==none` means that no servers have been annotated, so periodic checks on the cache status are all that is needed. Conversely, `PRDMitemSet.annotatedServers==all` implies that too many servers have PRDMitems referring to documents on them that it is not viable any more to keep a cache about this list.

ComMentor 1.0 Implementation Specifics

This section briefly describes some of the details necessary to transfer meta–information objects: transport and embedding, escaping conventions, and parsing.

Transport and Embedding

Descriptions in PRDM can be passed to clients (that is, browsers or proxy servers) as the content of a special MIME type message which knowledgable clients can then exploit by performing appropriate actions. Alternatively, as in the ComMentor 1.0 HTTP–based implementation, they can be conveyed as part of a message of text/html content type by using the HTML tag META in the form added by us. Here, the only extension to HTML is the addition of a META tag with the three attributes LANGUAGE, VERSION, and CONTENT, each of which has an opaque string as value, and an added tag to anchors. The first two specify the identity of the meta–language being used, so that clients can parse the content by appropriate callback routines and act on them.

PRDM meta–information can be embedded into HTML documents in the following way:

```
<META   LANGUAGE="PRDM"
        VERSION="1.0"
        CONTENT="opaque string understood only by browsers
                 with appropriate extension">
```

HTML anchors can be augmented with meta–information about the link:

```
<A      HREF="url"
        META_LANGUAGE="PRDM"
        META_VERSION="1.0"
        META_CONTENT="opaque string understood only by browsers
                      with appropriate extension">link</A>
```

HTML anchors are used to represent some of the meta items (inlined annotations, highlight markers, etc.). Specifically, in the absence of a separate tag for annotations in the HTML standard, we use the anchor tag with the following additional attributes:

```
<A      HREF="url by which the full annotation can be retrieved separately"
        NAME="identifier of an annotation (set name plus unique number)"
```

```
TYPE="annotation (or link)"
INFO="whatever is to be displayed by the lightweight viewer"
REFTO="which text to highlight"
```

The INFO attribute is an orthogonal extension related to the lightweight viewer [CHI95]; it can be used independent of the intentions of this paper for a number of different purposes, including hard−coded help information, pricing information, and more.

If the type is annotation, then REFTO specifies how much of the preceeding text to highlight.

The TYPE attribute is another attribute which is not yet in the standard, but which we feel has sufficient generality to be worth assuming.

Escaping Conventions

This section describes some technical detail about how data is escaped when it is inserted into HTML.

All special characters are escaped to %hh where hh is the hex value of the char, *Server:*

- *Sending*: Percent−escape the whole PRDM list expression to be sent. This avoids problems with embedded quotes.
- *Parsing*: In a received string, treat everything inside =" and "& as an opaque string.
  Strings not enclosed in quotes are %unescaped and + converted to spaces. (for compatibility with POST interfaces).

Browser:

- *Sending*: All values in the command language which might contain ampersands are surrounded by quotes.
  e.g. post_data = `key=value&key="value&value"&`
- *Parsing Meta*: Percent unescape the whole SList expression first.

  The escaped characters are:

  ```
  LF== 0A hex
  CR== 0D hex
  " == 22 hex
  % == 25 hex
  & == 26 hex
  + == 2B hex
  > == 3E hex
  ? == 3F hex
  ```

  The use of escaped strings facilitates PRDM parsing and stops embedded quotes from confusing parsers (like the one in the installed base of Mosaic).

Parsing PRDM

Parsing recursive s−expressions is a well−established technique which can be found in the basic textbooks. The recursive nature of PRDM lends itself to simple tokenizers, simple recursive−decent parsers, easily extensible objects, and flexible yet redundant encodings.

Tokenizer: There are five tokens: "(", ")", alphanumeric strings not containing a spaces or parentheses, quotes strings (like (type (atom "This string is an atom"))), and EOF.

Recursive Parser: See the standard literature on recursive−decent parsing. The first element of a list is always an atom such that an "assoc" procedure can be used conveniently.

Appendix C: Object Request Interface Specification

This section specifies the individual requests which the client can send to the server in order to perform some action. For each request, arguments and return type are given. Here is an overview for what is specified in the remainder of this section:

PRDMitem and Group Request Interface

Access to the PRDMitem server requires valid authentication.

- PRDMitem_new : create a new PRDMitem to a URL
- PRDMitem_get : retrieve PRDMitems for a given URL
- PRDMitem_rem : remove an PRDMitem
- PRDMitemList_get : get a tour to all PRDMitems which have a certain property.
- PRDMitemSet_rem : remove an PRDMitem set (removes all PRDMitems in that set)
- PRDMitemSet_info : retrieve the set's attributes
- PRDMitemSet_list : list the sets on this server

- grp_create : create a new group on the server
- grp_change : change a group's attributes
- grp_okmember : ok's a new member (after approval by group administrator)
- grp_remmember : removes another member from a group (administrator)
- grp_editmember : edits a member's info
- grp_quit : removes a member from the group's memberlist
- grp_info : retrieve a group's attributes
- grp_list : list groups on a server
- grp_join : apply to join a group

- member_update : update the member info on a server

Browsing Request Interface

This interface has unrestricted access (Requests are also available through the protected interface).

- PRDMitemSet_info : retrieve the set's attributes
- PRDMitemSet_list : list the sets on this server
- grp_info : *identical with above*
- grp_list : *identical with above*
- grp_join : *identical with above*

PRDMitem and Group Request Interface

The PRDMitem server is the principal secured part of the server: all requests to this part of the server must be authenticated by a valid *member name* and *member password*.

ComMentor 1.0 Implementation: Access restricted to `htmembers` with the http access control mechanism.

The following describes the individual requests and their arguments.

PRDMitem_new

Create a new PRDMitem in a set for a URL.

```
Authentication: Member distinguished name, member password on this server
Arguments:
PRDMitemSet      (name) PRDMitem set the PRDMitem is being added to
url              (url)  The URL of the document about which the PRDMitem is.
title            (text) Title of PRDMitem
pos              (prdm) Position within document "(relpos position)"
type    (type)  Type of the item.
Optional:
data             (text) Short text PRDMitem (if needed)
annurl           (url)  pointer to a separate page (if needed)
name             (text) name of author (for Anyone)
email            (email) email address of author (for Anyone)
iconurl          (url) url of author’s icon (for Anyone)
picturl          (url) url of author’s mug shot (for Anyone)
homeurl          (url) url of author’s homepage (for Anyone)
```

Add an PRDMitem record to the PRDMitem database for the appropriate URL.

Returns a document containing a text message, and a PRDM and an HTML representation of the PRDMitem.

The icon, picture or home URLs are taken from the member’s record unless overridden by the fields in this command. The exception is the member `Anyone` who is an anonymous entity. When accessing or posting to a set anonymously the browser should provide the name, email etc. of the user, since they are not on file at the public PRDMitem site. If the user is a member on that server then the details are stored as part of the member info, and they will be automatically retrieved from the member’s record.

PRDMitem_get

Retrieve PRDMitems in given sets for a given URL.

```
Authentication: Member dname, member password on this server
Arguments:
PRDMitemSet      (name) set(s) we want PRDMitems for (multiple allowed)
url                  (url) URL we want PRDMitems for
Optional:
PRDMitemID       (integer) unique PRDMitem identifier
Select       (string) selection criterion for the database processor
```

Returns PRDMitems for the document at `url` as a PRDM list, with the appropriate attributes filled in (names, icons, content, ...).

If `PRDMitemID` is specified, then no PRDMitem is returned and a HTML list is returned instead. If there is no `PRDMitemID`, then we return only a HEAD with PRDM (no body content).

PRDMitem_rem

Remove an PRDMitem.

```
Authentication: Member dname, member password on this server
Arguments:
PRDMitemSet      PRDMitem set we want PRDMitems to disappear from.
url                     URL we want to remove the PRDMitem from.
PRDMitemID       The specific PRDMitem we want to have removed.
```

Returns OK status message if PRDMitem is successfully removed.

Only authors or owners/administrator are authorized to remove PRDMitems (depending on a set’s chosen policy).

PRDMitemList_get

Get a tour to all PRDMitems which have a certain property. Currently, this property is a time of how recent an PRDMitem was created.

```
Authentication: Member dname, member password on this server
Arguments:
PRDMitemSet     (name) set(s) we want PRDMitems for (multiple allowed)
days_old        (time) time in the format MMDDhhmm about how old the PRDMitem can be.
Select  (string) selection criterion for the database processor.
```

Returns a tour/list that contains for each PRDMitem set all of the hyperlinks to PRDMitems with the required property. The hyperlinks have a relative target anchor to the name of the PRDMitem.

PRDMitemSet_new

Create a new PRDMitem set.

```
Authentication: Member dname, member password on this server
Arguments:
PRDMitemSet     PRDMitem set we want to create
access          An access pair for this set.
Optional:
access          More access pairs
readmembers     Individual members with read access to the set.
writemembers    Individual members with write access to the set.
homeurl         A page describing the set in detail, point to FAQs
iconurl         Pointer to default inline icon for this set
picturl         Pointer to default larger headline icon for this set
admin           DN of administrator of set (if different from admin)
owner           DN of owner of set (defaults to the creator)
description     Text of a brief description of the topic of the PRDMitemSet.

<access pair> ::= <group name>=<set-access>
<set-access>  ::= read | write | delete
Note the use of the = sign as a separator.  The = may be escaped if
need be - the server handles both cases.
Multiple <access pair>s are permitted.
e.g: access=ComMentor=read&access=PCD=write
```

Any member is allowed to create an PRDMitem set. This may be restricted in the future to members of a special `Set_Creators` group.

The owner is set to the member dn of the creator (unless overridden). The administrator is set to the owner (unless overridden).

`readmembers` and `writemembers` are used to authorize access to specific individuals (which might not be in a group otherwise).

PRDMitemSet_list

List the PRDMitem sets on this server.

```
Authentication: Member dname, member password on this server
```

Any member is authorized to access of list of the PRDMitem sets. The PRDMitem sets each have a link off to the `PRDMitemSet_info` for each set. *(see PRDMitemSet_info)*.

PRDMitemSet_info

Display the information about an PRDMitem set.

```
Authentication: Member dname, member password on this server
Arguments:
PRDMitemSet     The PRDMitem set we want info on.
```

The returned page contains meta data for the browser and human−readable HTML data for the user.

The page will display the short description, the icons and pictures for the set, and a link to the set's home page if there is one. The browser should recognize the PRDM meta−information to allow the user to add the set to their personal PRDMitem set list.

grp_create

Create a new group on the server

```
Authentication: member dname, member password on this server
Arguments:
group           (name) name of the group
admin           (dn of member) who manages the group
owner           (dn of member) who is responsible for the group
homeurl         (url)  pointer to home page
picturl         (url)  pointer to picture
iconurl         (url)  pointer to PRDMitem icon
description     (text) A short textual description of the group
mailList        (Bool) Shall we enable mailing list for group?
```

The member creating a group must be a member of the group `Group_Creators`. Otherwise an `(error "string")` is returned.

`admin` is the member responsible for administration of the group, such as approving new members and maintaining the mailing list addresses, etc. The technical details in other words.

`owner` is the member responsible for the content of a group's sets−−usually the owner will be the person who creates the initial batch of sets and seeds the discussion. Maintains the social function of the group.

`place` is generated by the server; it is the email address to which group mail is sent for redistribution and the URL of the script for accessing the group.

The mailing list option determines whether a mail may be sent to all members of a group. It is up to individual members to decide if they want notification of PRDMitems or not.

grp_change

Change a group's attributes

```
Authentication: memberdname, member password on this server
Arguments:
group           (name) name of the group to change
admin           (dn of member) who manages the group
owner           (dn of member) who is responsible for the group
homeurl         (url)  pointer to home page
picturl         (url)  pointer to picture
iconurl         (url)  pointer to PRDMitem icon
description     (text) A short textual description of the group
mailList        (Bool) Shall we enable mailing list for group?
```

Changes a group's attributes, most usefully the admin member and the description. Only changed attributes need be included.

grp_okmember

Approves a member to join the group’s list of participants.

```
Authentication: memberdn, member password on this server
Arguments:
group           (name) group to add new member to.
approvedn       (dn) unique name of member being approved.
```

This creates a member of a group, i.e. not a full user but someone who is already a member of one or more other groups on this server. Only admin or owner of a group may send a `grp_okmember` message.

This command will usually be invoked through the mail interface as the group administrator replies to server generated mail in response to user join requests. There is also a forms interface.

ComMentor 1.0 Implementation: This is also supported by a HTML "form" interface: Send grp=okmember and group=groupname to get a form of pending applications to join.

grp_remmember

Removes a member from the group’s memberlist.

```
Authentication: memberdname, member password on this server
Arguments:
group           name of group to remove member from.
```

Returns a confirming page and a PRDM command to delete the group to the list of subscribed groups. (`grp_quit` can be sent to either the group or the PRDMitem server.)

member_update

Updates the member record used for generating PRDMitems.

```
Authentication: memberdname, member password on this server
Arguments: Optional
name            (name)  New name
email           (email) New e-mail address
iconurl         (url)   New icon url to inline in text
picturl         (url)   A better picture of the user
homeurl         (url)   A newer home page url
```

Effect: PRDMitems a user made earlier now will use the new picture and icon URLs.

Note that some of the fields may be left out, and that this *cannot* be used to change the user’s DN or password on the PRDMitem server. The client should send update commands to all the servers in its server list whenever the user’s profile is changed.

Browsing Request Interface

This interface has unrestricted access; anyone can issue these requests unverified. (Requests are also available through the protected interface.) Most of the group administrator functions are protected through the PRDMitem interface.

The following describes the individual requests and their arguments.

grp_list

List all the groups on this server.

```
Arguments:
None.
```

Returns a list of groups on this server as an HTML page. The list includes links to the home pages of each group.

grp_info

Retrieve a group's attributes.

```
Arguments:
group           (name) Group to retrieve information about
```

Returns a page with details about a group's public info (currently all of it), and containing a PRDM meta−information with the same info, so that the browser can use this information to issue a `grp_join` later on.

grp_join

Apply to join a group as a member.

```
Arguments:
group           (name) Group to retrieve information about
dn              (dn) Distinguished name of user applying
passwd          (passwd) random password for this server if
                joining for the first time, otherwise use the
                same password as previous times.
email           (email) address to send comments to
homeurl         (url) pointer to home page
picturl         (url) pointer to picture
iconurl         (url) pointer to PRDMitem icon
```

The browser user will not be able to retrieve PRDMitems successfully until the user's state has been changed by the administrator (via approving a request, for example, with the e−mail interface).