Catacomb:
A Database-Backed WebDAV and DASL Repository

Jim Whitehead, Sung Kim

{ejw, hunkim}@cs.ucsc.edu

1 WebDAV Overview

Consider the following Web site authoring and document collaboration scenarios:

- A task force comprised of people from geographically dispersed business units need to develop a document together. (See Figure 1)
- A home computer user wants to develop a Web site, but has no Unix shell or programming experience. Their Internet service provider (ISP) must host their Web site, since they do not have a server computer in their home, and the ISP limits up channel traffic. Furthermore, the ISP would prefer not to give out shell accounts, for security reasons. (See Figure 2)
- A geographically diverse project team needs to be able to maintain their project's Web site, allowing all project members to make Web site updates. (See Figure 3)
- A large corporate Web site needs to accept content contributions from any of the company's geographically separate business units. (See Figure 4)

All of these scenarios have in common the need to *remotely author Web pages and other documents.* The Web Distributed Authoring and Versioning (WebDAV) protocol has been developed as an open, standards-based infrastructure that supports these authoring scenarios. Figures 1-4 below show how the WebDAV protocol makes it possible to do each of these scenarios.

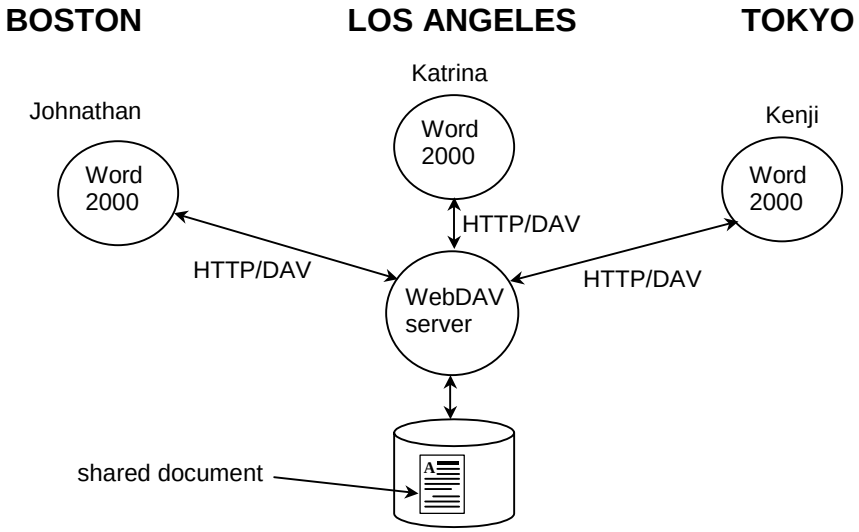

Figure 1 - Three collaborators, located at three different sites, are jointly authoring a document using the WebDAV capabilities of Microsoft Word 2000. Word 2000 uses the WebDAV protocol to interact with the shared document, stored on a WebDAV server located in the Los Angeles office.

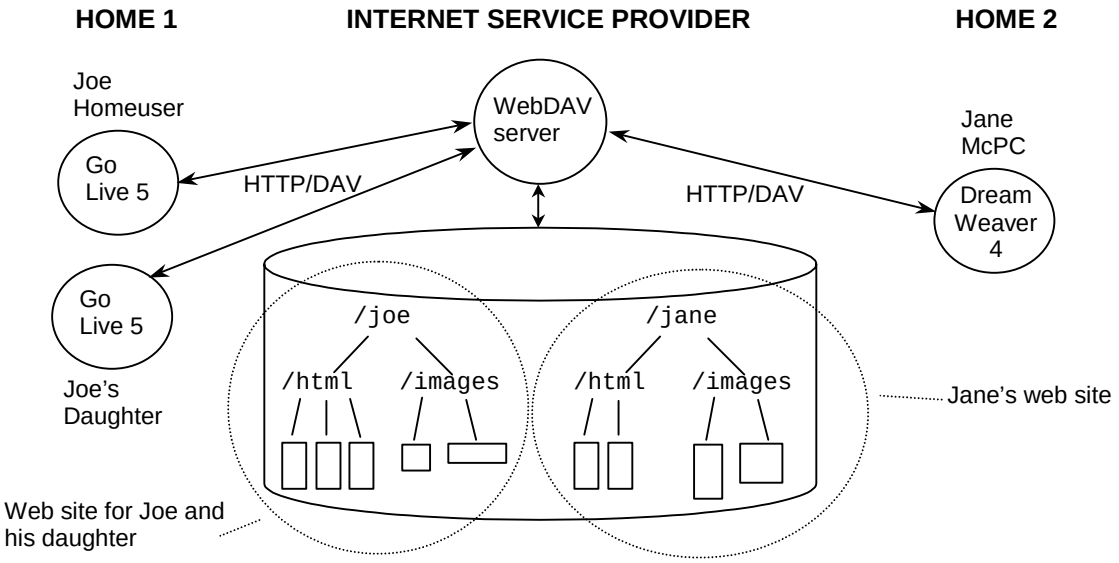

Figure 2 - Two groups of home users are using Web site authoring tools to develop and maintain Web sites hosted by their Internet Service Provider (ISP). In "Home 1", Joe Homeuser and his daughter both use Adobe Go Live 5 to manage their Web site, while in "Home 2" Jane McPC uses Macromedia DreamWeaver 4 to manage her Web site. Go Live and DreamWeaver both use the WebDAV protocol to communicate with the WebDAV server operated by the ISP.

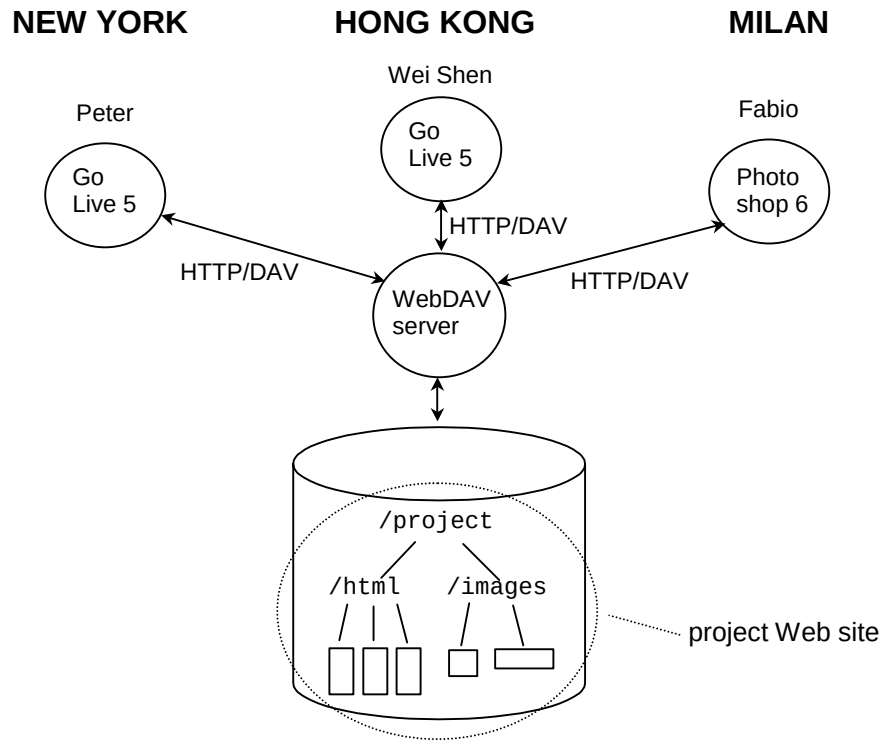

Figure 3 – A project team with members located in New York, Hong Kong, and Milan collaboratively maintain their project's Web site using a combination of Web site authoring, and image authoring tools, in this case Go Live 5, and Photoshop 6, both of which use the WebDAV protocol to interact with the project Web site server.

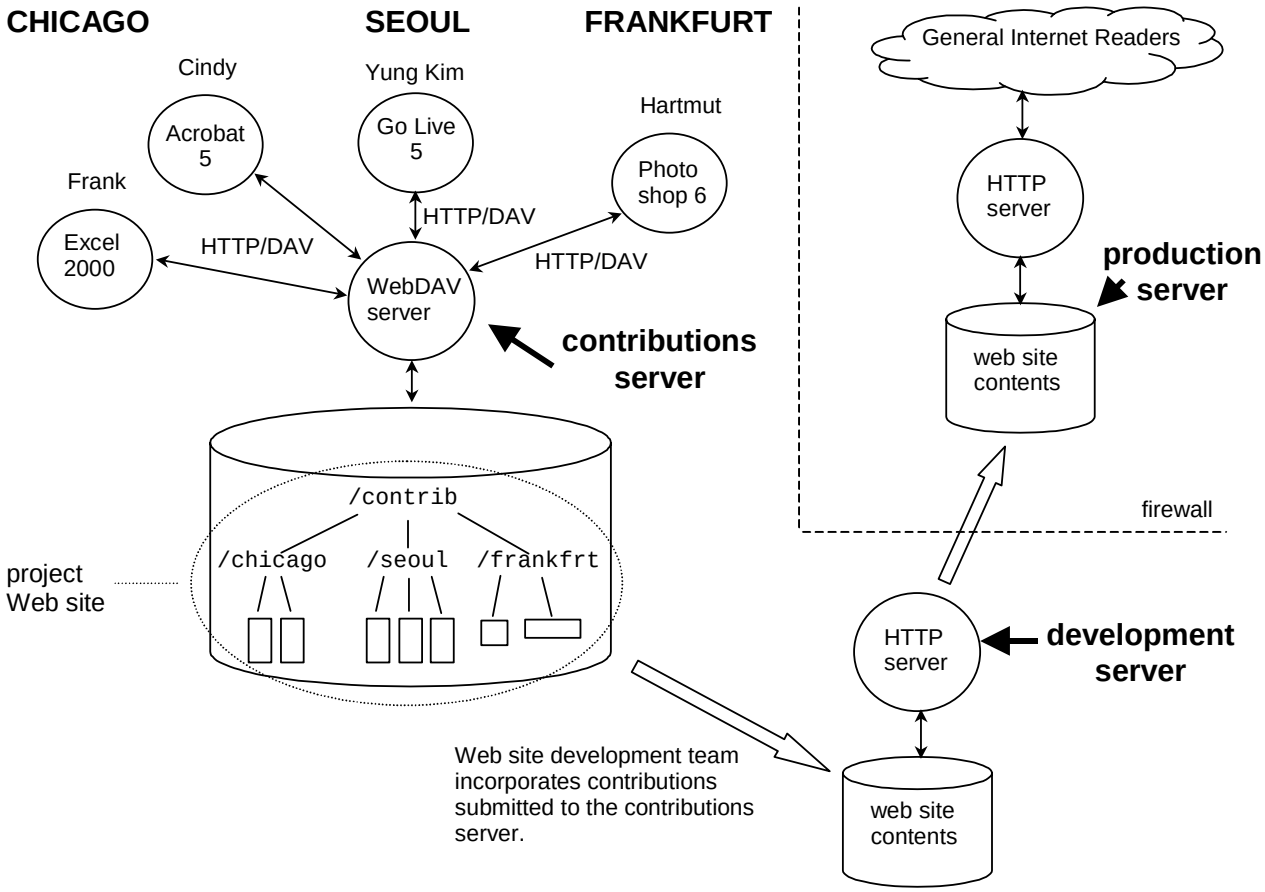

Figure 4 – A large corporate Web site accepts content contributions authored using a wide variety of applications, from its offices located in Chicago, Seoul, and Frankfurt, by hosting a WebDAV contributions server. The Web site development team then incorporates these contributions into their non-publically visible development server. Once the development server has been approved for release, it is mirrored into the actual production server, which is outside the firewall and visible to all Internet users.

WebDAV is a series of extensions to the Hypertext Transfer Protocol (HTTP), the core network protocol that carries Web traffic between a Web server and a Web browser. HTTP protocol operations are called *methods*, and WebDAV adds seven new methods to the set of methods defined by HTTP/1.1 (GET, HEAD, POST, OPTIONS, PUT, DELETE, TRACE). The WebDAV methods provide overwrite protection (LOCK, UNLOCK), metadata management (PROPFIND, PROPPATCH), and namespace management (COPY, MOVE, MKCOL). Just as the user of a Web browser is largely unaware of the HTTP network traffic that request and download Web pages, so too a user of a WebDAV-enabled authoring tool is largely unaware of the use of the WebDAV protocol. The WebDAV protocol is designed to be integrated into existing authoring tools, adding Web-based remote authoring capabilities to the tools users already know how to use. To date, this has been a successful strategy, with WebDAV support in document authoring tools such as Word 2000, PowerPoint 2000, and Excel 2000 (via the "Web Folders" feature) as well as in Acrobat 5 and Photoshop 6, and in Web authoring tools such as Go Live 5, and Dreamweaver 4. You may already own a WebDAV-capable application!

The core defining document for the WebDAV protocol is Proposed Standard Request for Comments (RFC) 2518, which was developed by the WebDAV Working Group of the Internet Engineering Task Force (IETF). It is a common misconception that the World Wide Web Consortium (W3C) developed this

protocol; while the W3C hosts the working group's mailing list, hosted some early WebDAV meetings, and has contributed to discussions on the protocol, this work was done in an IETF working group, following IETF procedures. This reflects a de-facto breakdown of standardization responsibility for the World Wide Web: the IETF controls the development of the network protocol (HTTP/DAV), as well as naming standards (URI, URN, URL). The W3C typically standardizes content standards, such as HTML, XML, CSS, RDF, etc., and provides proof of concept research on new network protocol standards, such as the HTTP-NG project.

2 Functionality Overview

The WebDAV Distributed Authoring Protocol contains a set of features usable in a wide variety of settings by applications that support collaborative work on remotely authored documents. These features can be partitioned into three groups: overwrite protection, metadata management (properties) and namespace management. A detailed overview of these capabilities is presented in the sections below.

2.1 Overwrite Prevention

Once two or more people start collaborating on the same document, the issue of write control comes to the fore. If everyone can write to the same, unversioned document, then it is possible to lose changes made by one or more contributors as first one collaborator, then another, writes their changes without first merging in previous updates.

There are many techniques that can be used to alleviate this "lost update" problem. Several of the more common ones are:

- *POTS (plain old telephone service) or "over-the-wall" control*. This scheme is a social convention in which collaborators agree to communicate verbally when one author has finished working, and it is safe for another to begin. E-mail can also be used to implement this write control scheme, as can a physical object (e.g., a baton) which is passed from author to author in environments where authors are co-located.
- *Shared locks (also known as advisory locks or reservations)*. In this scheme, an author indicates to the computer controlling access to the document that he or she intends to modify it, and the computer records this author's intent to edit. If another author similarly tries to indicate an intent to edit, the computer will announce that the document is currently being edited. However, if the second author still wishes to edit, it can be done, presumably by contacting the other author to negotiate access or by taking advantage of extra-system knowledge that no conflict will result (e.g., the other author is in a meeting).
- *Exclusive locking*. In this scheme, an author indicates to the computer controlling access to the document that he or she intends to modify it, and the computer responds by locking the document. Once the document is locked, it may not be modified by anyone other than the owner of the lock. Other authors who try to edit the same document are refused, because they do not own the lock.

These schemes vary from least protective and most flexible (POTS) to most protective and least flexible (exclusive locking).

Currently, the WEBDAV approach is to provide facilities for both shared and exclusive locking. Both kinds of lock are requested using the LOCK method, and removed using UNLOCK. This dual lock support provides sufficiently flexible locks to accommodate a wide range of collaborations. While shared locks best support collaborators who have a lot of awareness of each other's activities, exclusive locks provide a more stringent guarantee of conflict avoidance for less aware collaborators or during periods of high contention for a document. Locks may have a scope of a single resource, including all non-live properties on the resource, or a hierarchy of resources (for example, a collection and all of its member resources). A lock discovery mechanism (the "lockdiscovery" WebDAV property) allows authors to find out if any locks exist on a Web resource. Since the Web is designed so that no lock is required to read a Web page, there is no concept of a read lock. An implication of this is the contents of a resource may change without warning if a write lock is not owned on the resource.

To protect against extended network outages, or crashed applications, both of which can cause a lock to remain on a resource long after it is no longer desired, WebDAV locks have an associated timeout value. When the lock times out, the server automatically removes it. Lock durations vary, with some applications taking out long duration locks, on the order of days, while others take out much shorter locks, on the order of minutes, with frequent lock refreshes (this is the case with Office 2000).

WebDAV locks exist independent of the TCP connection that was used to transmit the LOCK request. As a result, it is possible to take out a lock, then work disconnected from the network, and then reconnect to submit editing updates. This type of work pattern is very useful for wireless networking, where network connections can frequently get dropped. Since WebDAV locks are network connection independent, and since there can be multiple clients operated by a single person interacting against the same set of resources (especially multiple instances of the same application running on separate machines), there is a need for a mechanism to identify a specific lock on the WebDAV server. This is satisfied by a *lock token*, a globally unique identifier that uniquely identifies a lock across all WebDAV server. In order to perform WebDAV operations that modify the state of the resource (PUT, PROPPATCH, DELETE, MOVE) on a locked resource, a WebDAV client must submit a lock token with the request, in the “If” header, along with authentication credentials. This demonstrates that the client is (a) operated by the person that took out the lock, and (b) the client is aware of the lock, and has specified a particular lock. If condition (a) isn’t satisfied, then anyone could overwrite a locked resource. If condition (b) isn’t satisfied, it is possible for application A to take out a lock, and then application B (authenticated as the same person) could overwrite the same resource, oblivious of the lock. However, this feature is not universally lauded – some WebDAV developers feel it eliminates one, very rare problem, at the expense of additional complexity in lock token management by clients, all the time.

2.2 Metadata Management

Information on the Web has many pieces of associated information, such as the title, subject, creator, publisher, length and creation date. This information about information (called *properties* within WebDAV, but also known as *metadata*) can be used to search for Web resources, enforce copyrights or provide bibliographic information. Properties are particularly useful in searching for Web resources due to the inadequacies of existing index-based Web search engines that often return a large number of undesired results to any query. By focusing a search on a the value of a particular property (e.g., the author), properties can be used to reduce the number of undesired query results; the DASL effort (described below) is concentrating on providing solid support for queries on properties of resources.

Development of a useful set of properties is extremely important – one schema, or set of metadata, which was developed to assist Web searching is known as the Dublin Core (for more information, see: http://purl.org/metadata/dublin_core/). Since other groups have focused on developing metadata sets, the WebDAV group decided to focus on developing facilities for creating, modifying, deleting and retrieving metadata. These facilities allow for the manipulation of metadata from multiple schemas, allowing the schema itself to vary with domain of use. For example, even though the Dublin Core is appropriate for use in the general Web context, it may not be ideal for use in other settings, such as the legal community. By being metadata schema neutral, the WebDAV approach allows the most appropriate schema to be used in any context. It allows WebDAV to focus on "how," as in how properties are stored and retrieved, rather than on "what," as in what do they mean?

WebDAV properties are name-value pairs. The name is a Uniform Resource Identifier (URI), such as a URL, and the value is a well-formed sequence of Extensible Markup Language (XML) content. If, for instance, a property name is a URL, it can be given uniqueness without central registration by using URL property names chosen from within a domain whose name is controlled by the party defining the property. So, for example, a company that controls a given domain name, like "widgets.com," can choose a property name from within this domain, like "widgets.com/properties/color."

An example WebDAV property defined by the Distributed Authoring Protocol is the DAV:getcontentlength property, which gives the length, in bytes, of the response generated by a GET on the resource. The property name is a URI, with a URI scheme of "DAV," which is reserved for use by WebDAV. A sample value of this property is:

Name: DAV:getcontentlength

Value: <DAV:getcontentlength> 3422 </DAV:getcontentlength>

In this case, the length is 3422 bytes, which is enclosed within the <DAV:getcontentlength> XML element. By convention, the enclosing XML element for a WebDAV property takes the same name as the property itself.

WebDAV provides two methods for interacting with properties, PROPFIND and PROPPATCH. PROPFIND is used to retrieve the value of one or more properties, from one or more resources, with the values wrapped up in a "multistatus" XML element. The multistatus XML element allows responses from multiple resources to be combined together into a single XML sequence that can be transmitted as the response to a single HTTP method invocation, thus allowing, for instance, the results of a single PROPFIND across an entire collection hierarchy to be returned in a single response. The PROPPATCH method allows one or more properties to be added or removed from a resource in a single operation. The act of modifying a property is submitted as remove/add pair. The operation of PROPPATCH is atomic, that is, either all add/remove operations are performed, or none are performed.

Using XML to encode the value of properties provides three major benefits. First is extensibility. Since all content within XML is encoded between start and end tags, it is easy to add additional elements to a property by inserting new tagged content. Internationalization is the second major benefit. Since XML mandates support for the UTF-8 and UTF-16 encodings of the ISO 10646 character encoding standard, as well as language tagging, properties can express content in the vast majority of human languages. Finally, by using XML, WebDAV properties can support other metadata activities that are also based on XML, such as the Resource Description Framework (RDF).

2.3 Name Space Management

In the current, publish/browse model of the Web, there is scarce need for a user to duplicate or rename Web resources. However, once the Web is used for distributed authoring, the need for these capabilities, plus the ability to get a listing of a directory, becomes extremely important. Being able to discover what resources currently populate a portion of the name space of a Web server and the ability to copy, move and delete these resources, together form the key elements for managing a Web name space.

There are several justifications for adding copy and move capability. A resource may need to be copied due to changing ownership, prior to major modifications, or when making a backup. It is often necessary to move (i.e., change the name of) a resource, for example due to adoption of a new naming convention, or if a typing error was made when originally entering the name.

Copy and move have ramifications with respect to properties: how should properties behave after a copy or a move? It would seem that all properties on the duplicated or moved resource should be identical to the properties on the original. However, there are really two classes of properties: live and static. *Static* (or *dead*) properties have the quality that their value, once set, remains the same until a client explicitly modifies it. *Live* properties, in contrast, have their syntax and semantics enforced by the server and may vary at any time. One example of a live property is the content length of a resource – every time the resource is updated, the value of the property will also be updated. WebDAV also attempts to resolve conflicts between the existing properties of a resource being moved and those that may be enforced by the server or directory in which it is to be located.

Listing the contents of a collection, an operation similar to listing a directory of a file system, is accomplished using the property retrieval method, PROPFIND. Most existing directory listing operations (such as "ls" or "dir") provide the name of a file and an option for retrieving limited sets of properties about the file, such as its size, owner and access permissions. However, since WebDAV has an existing property retrieval method, it made little sense to define another method just for listing a collection. Instead, the existing property retrieval method, PROPFIND, was used. Since PROPFIND allows the hierarchical retrieval of properties on a collection, returning for each resource its name and requested properties, it has enough expressive power to do double-duty as the "list a collection's members" operation as well.

Rounding out the namespace management capabilities is the MKCOL method, which creates a new collection.

2.4 WebDAV Summary

A summary of the HTTP and WebDAV methods, and their area of impact on a resource, can be seen in **Figure 5** below.

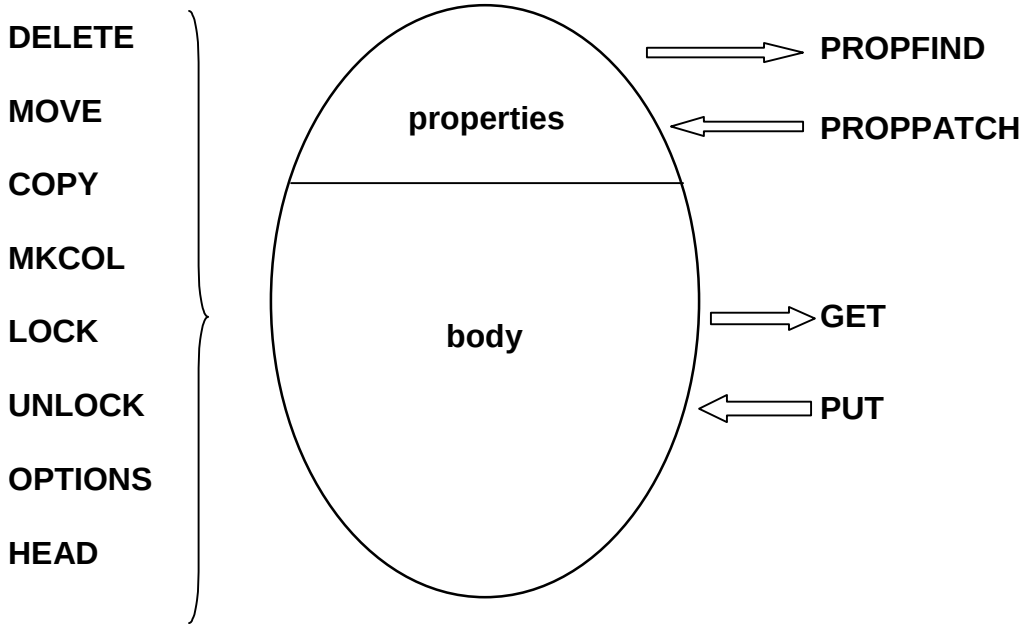

Figure 5 - A summary of the WebDAV methods, and their scope. Methods listed on the left side affect the entire state of the resource, both body and properties. Methods on the right only affect specific parts of the state Specifically, PROPFIND and PROPPATCH only affect property state, while GET and PUT modify the body state (with some side-effects on live property values, such as getcontentlength).

3 DASL Overview

The DAV Searching and Locating (DASL) protocol defines a lightweight SEARCH method for executing server--side searches of resource contents and metadata properties. The full DASL protocol consists of the SEARCH method which transports the query from the client to the server, the DASL response header which indicates the server support of the SEARCH method and the query grammars recognized, the DAV:searchrequest XML element which contains the actual query, the DAV:queryschema property which allows the client to discover what properties can be referred to in the query, the DAV:basicsearch query grammar which implementations must support, and the DAV:basicsearchschema which defines how properties may be used in the query (selectable, searchable, or sortable), and which of the special DAV operators are supported.

The DASL protocol specification defines the basic lifecycle of a SEARCH request as follows. First the client constructs a XML query using the desired search grammar. After constructing the query, the client invokes the SEARCH method on a search arbiter, a resource that performs the search on the client's behalf, and includes the query in the request body. The arbiter then executes the query and sends the results back to the client in the body of the response. The text/xml MIME type is used for both the request and response bodies. The response body must conform to the PROPFIND response body, as specified in RFC 2518.

4 Design of the Catacomb Repository

4.1 Database Schema

We use MySQL so that we can make use of the strong conditional capabilities of SQL queries to search metadata and resources. In Catacomb, all resources and metadata are stored in the database. The text and the binary contents of resources are stored in different fields. Unlike the hierarchical structure in a file system, the resources in Catacomb are organized in a flat structure in the database table. The hierarchical containment relations of resources and collections can be realized by the URI of each resource. There is a 'resourcetype' field in the table to indicate whether the resource is a regular file or a collection. With a hierarchical structure, it is difficult to efficiently implement conditional searching because the search process must recusively process the subdirectories. In contrast to this, one can easily implement search function in a database due to its flat organization of records in tables and the relations between the tables.

There are two tables in the database. The resource table contains resources and live properties, while the property table contains user defined dead properties. We have to use a separate table for the dead properties because, unlike live properties, dead properties are not known ahead of time. Any resource can have its own, unique dead properties and new ones can be created at any time. The relationship between the resource and property tables is one-to-many, related by serialno.

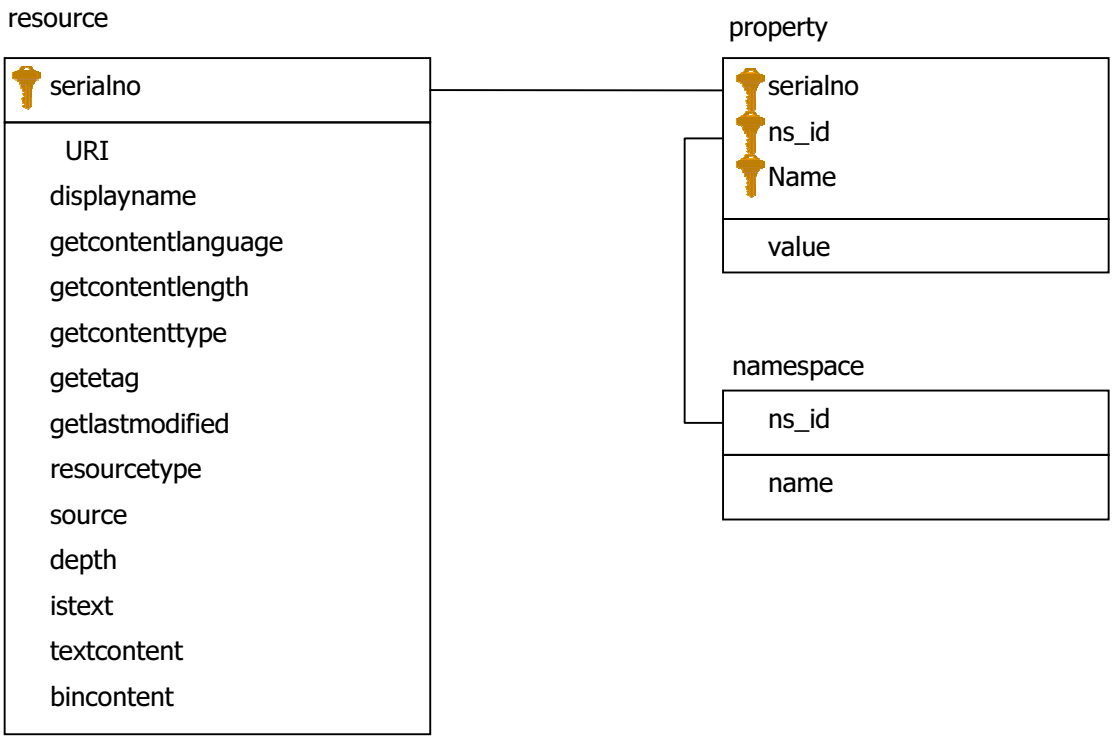

Figure 6 – Resource, property and namespace schemas

The resource table contains both the resources and their live properties. The fields are shown in Figure6. The catacomb table contains the dead properties associated with a resource. The fields in are shown in Figure6. To facilitate efficient searching, there is also an extra index placed on (name, value(245)) for the property table.

4.1.1 Primary key

Which is a better choice for the primary key for the resource table, URI or serialno? Though URI is a unique field in the resource table and can serve as the primary key, we introduce a serialno field as the primary key for efficiency. serialno is a long integer field, while URI is a text field which could be at most 65535 bytes in length. If URI were used as the primary key, it will take much longer when we perform joins on the resource and property tables. In addition to the above, serialno has another function of keeping track of the order the resources were inserted into the table. The cost for using serialno as the primary key is that we add one more field to the resource table and MySQL should maintain its serial property. In practice, this cost is not considered high.

4.1.2 Datetime format

In the resource table, we use 'bigint' datatype to represent the datetime fields, such as creationdate and getlastmodified. The values in those fields indicate the seconds since Jan. 1st, 1970, 00:00:00. Besides this decision, there are two other choices to present the datetime.

The first choice is to use a string type. For example, "20020128080601" indicates 08:06:01 on Jan. 28th, 2002. We can only use the "YYYYMMDDHHMMSS" format to present the datetime with a string because we need to perform comparison operations on the field. There are two disadvantages to this approach. First, the format to present the datetime in this approach is not flexible since we have to transform it when we want to display it in different forms. Second, arithmetic operations cannot be easily applied. For example, the arithmetic operation that calculates the difference of two datetimes would require inefficient conversion routines.

The other approach considered was to use the datetime data type supported by MySQL. The advantage of this approach is that we can make use of the datetime functions in MySQL. The drawback is that it is vendor specific since we have to depend our code that deals with the datetime on MySQL. If we decide to use another database in the future, this approach dictates changes to the source code. On the other hand, it is not flexible to process the datetime in our program, if our code depends on the presentation format and process functions of datetime in MySQL.

4.1.3 Text and Binary Fields

We use a longtext field to store text files and a longblob field to store binary files (e.g. image, pdf, mp3, video, et cetera). There are three reasons that we store the text file and binary file in different fields. First, the content of the text file can join the conditional search. For example, we can easily use this field in a search condition to find resources that have certain content, while it makes no sense to apply this on binary files. Second, we can retrieve the text content from a document with a special format and store it in the 'textcontent' field. For example, .pdf and .doc files have a special document format. We can save these files in the bincontent field and, at the same time, extract their text content and save it in the textcontent field. It is then possible to not only make use of the search abilities of SQL on their content, but also keep the original documents in the repository. There is an istext field maintained that indicates whether the resource is a text file or a binary file. Third, the database can apply full-text searching on text fields. Full-text indexes can be created on varchar and text fields to allow a query to search for resources with specific contents.

There is not much space wasted when we create both textcontent and bincontent fields in the resource table. The size of a record in the table depends only on the size of the actual contents of the record. For example, a text file only has content in the textcontent field and its bincontent field is empty. Thus, the bincontent field contributes zero bytes to the overall size of the record. The only extra space required is for the record that has contents in both textcontent and bincontent.

4.1.4 Live and Dead Properties

We put live properties together with the resource due to the considerations of the search efficiency. Since the search condition may be over the content and live properties at the same time, we can avoid table joins by putting them in the same table. It was necessary to put live properties and dead properties in different tables because dead properties are different from live properties. Dead properties are not predetermined so

users can define their own properties of their resources. Therefore, dead properties have a different representation than live properties. Live properties are organized in columns, while dead properties are organized in rows. We will see later that this representation of dead properties in this way causes some difficulties when parsing the SEARCH request to generate the SQL query.

The organization of the live properties in the resource table ensures the searching efficiency when using live properties as the searching conditions. But there is an extensibility problem with the design of the resource table. If someone defines a new property, both the resource table and the source code should change. As we can see, it is a tradeoff between efficiency and extensibility.

5 Implementation of Catacomb

We implemented an Apache 2 server module that interfaced the WebDAV protocol with a DBMS layer over a MySQL database, as shown in Figure 7.

All resource contents and metadata properties are stored in MySQL. The decision to use MySQL was made because it is lightweight and supports a simple C API. Within this architecture, we implemented the DASL protocol SEARCH method. For testing out the server, we added search capabilities to Cadaver, a command line WebDAV client based on the Neon library.

5.1 DBMS

The DBMS module uses the MySQL C API to implement the database operations. This module is at the lowest layer in Catacomb, and conceptually sits on top of the DB. The functions in this module provide database operations for the other modules. The functions in this module can be divided into the following groups: database connection, property processing functions, file content processing, and searching.

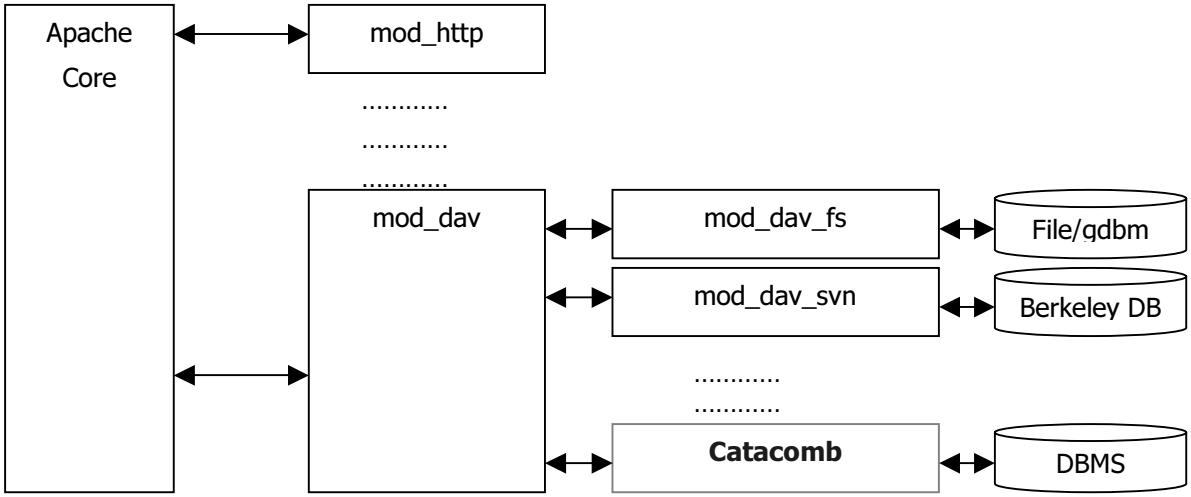

Figure 7 – Catacomb architecture overview

5.2 Data Structures

The main data structure used in the DBMS module is dav_repos_resource. It serves as a container that returns query results to the search module. dav_repos_resource is a linked list, every node in which is a resource structure containing major information for a resource. A dav_repos_resource node also contains a list, which holds the list of dead properties belonging to the resource. This structure can also serve as the container for the individual resource record and the container for resource records in a collection. dav_repos_property is a data structure that stores dead property records. It is also a link element of dav_repos_resource.

5.3 Functions

The major parts in the functions of DBMS module are the SQL statements in them. In this section, we explain a few of the more interesting functions.

The dbms_read_content function reads the contents of a resource from the DB and saves the content to a file. MySQL provides the ability to retrieve the contents of text or blob fields directly into a file. Thus, we don't need to write the code to handle the file operation. Conversely, the dbms_write_content function uses UPDATE statement to write the content of a file into the database. Examples of the SQL statements used are below.

```
SELECT bincontent
INTO DUMPFILE <filename>
FROM resource
WHERE serialNO= aSerialno ;
```

```
UPDATE resource
SET textcontent = LOAD_FILE("aTextFileName")
WHERE serialNO = aSerialNo ;
```

By simply changing the URI field of a resource, the dbms_move_resource function moves the resource. Just like moving files or directories in a file system, we only change the path (URI) of the resource(s) instead of copying and deleting files. We make use of the string operations provided by MySQL to construct the SQL statement that can change the URI(s) of the resource(s) in only one statement. In doing so, we avoid recursing through the DB and ensure the atomicity of the move operation on a collection. For example, if we want to move the resource /project to /project_bak, the following statement will change /project to /project_bak, and change all the resources in /project, /project/*, to /project_bak/* if it is a collection.

```
UPDATE resource
SET uri = concat( '/project_bak' +
        SUBSTRING( uri , 8 ))
WHERE uri like '/project%' ;
```

The dbms_copy_resource function copies a resource (and it's properties) to another URI. New records are created in the DB to contain the copy of the resource. If the resource being copied is a collection, all the files contained therein are copied as well. The procedure to copy resource(s) in the database is described below.

First, create a temporary table and save source resource(s) with changed URI(s) into the temporary table. The SQL statements used are:

```
CREATE TEMPORARY TABLE tmp
SELECT ...
FROM resource
WHERE ...
```

Second, copy the records in the temporary table to resource table. The SQL statement is:

```
INSERT INTO resource
SELECT *
FROM tmp
```

5.4 Catacomb module

The Apache 2 server configuration directives are used to configure the database options used by catacomb, thus each instance of the server can have only one DB host, DB name, DB user id, etc. specified. The advantage of this approach is that we can connect to the DB when the Apache thread starts and disconnect from the DB when apache thread exits. This connection life cycle improves the database performance which is responsible for the overall performance of Catacomb. The limitation of this approach, of course, is that we can only use one database per server instance.

Since the interface of mod_dav is designed for a file system, there are open, read, seek style interfaces on the back end. MySQL doesn't support these sorts of file handling mechanisms, so we needed to use temporary files as a medium to deliver content between the server and the DB. This process makes Catacomb slower than it would be otherwise.

5.5 SEARCH method

The search module handles the processing of the SEARCH method. The general approach is as follows. First the XML body of the request is parsed, and the corresponding SQL query constructed. The query is then executed on the underlying database, with the results returned as a linked list of resources and their properties. The results of the search are packaged into an XML body that is sent with the HTTP response. Due to the DB schema we used in, there are a few difficulties that were faced in implementing the SEARCH method.

One of the difficulties faced was that dead properties are stored as rows of the property table. This necessitates the aliasing of the property table in the SELECT portion of the query and then using self join operations to get the desired results. Aliasing is an operation that allows one to refer to a table by two names, allowing the property table to be effectively used as multiple tables. Self-joins refer to the fact that we are really joining result sets from the same underlying table. To see why this is necessary, recall that the property table has three fields, serialno, name and value. In order to find the resources that are written by 'Patrick' with the keyword 'Germany', we specify a query such as:

```
SELECT ...
FROM property
WHERE name='author' and value ='Patrick' and
       name='keyword' and value='Germany';
```

This query will return an empty set, as fields cannot have more than one value (as with name='author' and name='keyword' above).

```
SELECT ...
FROM property t1, property t2
WHERE t1.name='author' and t1.value ='Patrick' and
       t2.name='keyword' and t2.value='Germany';
```

This query must instead be written as where the property table is aliased twice to t1 and t2. This example also illustrates how the dead properties that occur in the WHERE element of the XML request affect the FROM portion of the SQL query. In reality, the dead properties in the WHERE element will also affect the SELECT portion of the SQL query as well. To make matters worse, dead properties in the SELECT element will also affect both the SELECT and WHERE portions of the SQL query. These dependencies dictate that we must process the XML request out of order, collecting the dead properties first from the different elements of the request, and then building the SQL query from string fragments generated during the parsing. More work is needed to determine if there is a better DB schema that will result in a more direct parsing of the SEARCH request without negatively affecting the performance. If MySQL supported 'EXISTS' operation, sub queries or stored procedures this self-join approach would not be necessary.

6 Client Writing Using the Neon Library

Neon, *http://www.webdav.org/neon* is an HTTP and WebDAV client library, with a C language interface. As described on the Neon web site, its features include:

- High-level interface to HTTP and WebDAV methods (PUT, GET, HEAD etc)
- Low-level interface to HTTP request handling, allowing implementation of new methods
- Persistent connections
- RFC2617 basic and digest authentication (including auth-int and md5-sess)
- Proxy support (including basic/digest authentication)
- SSL/TLS support using OpenSSL (including client certificate support)
- Generic WebDAV 207 XML response handling mechanism
- XML parsing using the expat or libxml parsers
- Easy generation of error messages from 207 error responses
- WebDAV resource manipulation: MOVE, COPY, DELETE, MKCOL

- WebDAV metadata support: set and remove properties, query any set of properties (PROPPATCH/PROPFIND)

- autoconf macros for embedding neon directly inside an application source tree

We took advantage of Neon's support for defining new methods to develop a DASL client. For any new HTTP/DAV/DASL request, Neon requires the programmer to go through several steps. This section briefly explains how to issue DASL protocol requests using Neon. More detailed information on programming Neon can be found on its web page.

6.1 Create session

Neon has special data structure called a *session* that represents the network interface, storing connection-related state such as the schema, hostname and port number or the URL that identifies the HTTP/DAV/DASL server. Using this information, Neon can reconnect the server, if necessary. We create a session by calling 'ne_session_create'.

```
ne_session *sess = ne_session_create(scheme, host, port);
```

All API calls to create HTTP/DAV/DASL requests require a session.

6.2 Create Request

The first step in sending a request is to create a request object for a specific session. A session can have several requests. When creating a request object, the HTTP/DAV/DASL method is specified. The following code creates a SEARCH request:

```
ne_request *req = ne_request_create(sess, "SEARCH", uri);
```

6.3 Request preparation

Now, we have a session and request object. Before the request object is transported to the server, we need to add request-specific headers and the request body. The API call, ne_add_request_header adds HTTP header to the message for you. This example shows how to set the Content-Type header, used to convey the MIME type:

```
ne_add_request_header(req, "Content-Type", NE_XML_MEDIA_TYPE);
```

Since we're building a SEARCH method, the content type is text/xml, reflecting that the request body will contain the XML-encoded DASL basicsearch query. The Content-Length is automatically set by Neon based on the size of the request body. To set the body, you need to create a character buffer containing the contents of the request body. Let suppose we have request body data in a buffer pointed to by char *body_data, we can then set the request body by calling ne_set_request_body_buffer:

```
ne_set_request_body_buffer(req, body_data,  strlen(body_data));
```

6.4 XML Response Body Handling

If you aren't interested in the response body sent by the server (such as for successful MOVE or COPY method invocations), we can just send a request and check the status. To handle response bodies sent from

server, such as the response to a SEARCH method, you need to set a call back function. This is especially true if the response body is XML and needs to be parsed, since this requires setting an XML callback function.

```
ne_add_response_body_reader(req, search_accepter,
                                    ne_xml_parse_v, search_parser);
```

The response body for the SEARCH method is XML, containing search results. The first step in creating an XML parser is calling the function ne_xml_create(), which generates an XML parser.

```
search_parser = ne_xml_create();
```

Two callback functions then need to be set, to handle parsing at the beginning and end of an XML element. These two callback functions are called “start_element” and “end_element”.

For example, consider this XML:

```
<apachecon>
  <catacomb>DBMS Backed repository</catacomb>
</apachecon>
```

When XML parser reaches the <apachecon> start tag, the “start_elemet” function will be called with the element name, “apachecon”. When the XML parser reaches “</apachecon>, the “end_element” function will be called. To keep parsing status, we can define our own context handler and set the handler data when we set the callback function. The data can be used in each callback function.

If an element has data (such as the catacomb element in the example above), the data can be passed to the end_element function so it can manipulate the data. The following example shows simple way to push call back functions and handler data.

```
ne_xml_push_handler(search_parser, search_elements,
                                    validate_search_elements,
                                   start_element, end_element, sctx);
```

6.5 Sending request

When we are ready to translate the in-memory request object into an on-the-wire request, use the ne_request_dispatch function:

```
ret = ne_request_dispatch(req);
```

The return value indicates whether the request was successful. After receiving the response body, the previously set XML parser and callback functions will be called to process the response data.

6.6 Destroying the Request

After finishing the request, we should destroy the request object. Since each request has several dynamic memory allocations, we need to free all memory by calling ne_request_destroy.

6.7 Destroying the Session

If you want to connect send another request to the same server, you should not destroy the session, since a session can be reused. A session can have more than one request, and will automatically reconnect to the server, if the TCP/IP connection was closed. If the TCP/IP connection is still alive, Neon will reuse the connection for high performance.

If you want to finish your client program or change a server, it is good idea to release all memory by calling ne_destroy_session().

7 Contributions and Future Work

Several contributions of this project can be identified, including the reference implementations of the DASL protocol on both the client and server side and the initial work towards developing a generic, yet reasonably efficient database schema for combined resource and metadata management. Catacomb is an open source DAV/DASL web server with a database backend, and is an important first step towards the development of a personal document management/digital library system.

There are a number of ways in which this project could be extended and developed further. Research into developing a high performance, flexible database schema that can store arbitrary metadata is high on the list of priorities, as is finishing the work necessary to be fully compliant with the DASL protocol as it evolves towards RFC status. Adding auditing facilities, a more flexible security model than the one provided by Apache, versioning of resources and collections via the Delta-V protocol, or developing a richer containment model that allowed multiple containments could also functionally improve the project. The integration with event notification services by adding the appropriate hooks into Catacomb is another promising direction to embark upon. This body of work will likely require the long-term attention of several, perhaps many, developers. To address this need, this work will continue as an open source project in the hope that other people will recognize the long-term benefits to the community and join our effort. In the future, we hope to be able to distribute Catacomb with the Apache server in a standard fashion, thus providing a scalable, database backed, open source web server that supports the entire DAV protocol suite.

Further Reading

WebDAV Distributed Authoring Protocol

- WebDAV Resources
  <http://www.webdav.org/>
  A web site containing a central collection of pages and links to all things WebDAV.
- WebDAV Working Group
  <http://www.ics.uci.edu/pub/ietf/webdav/>
  Contains links to active documents, and a complete list of WebDAV-supporting applications.
- Y. Y. Goland, E. J. Whitehead, Jr., A. Faizi, S. R. Carter, D. C. Jensen, "HTTP Extensions for Distributed Authoring – WebDAV", Microsoft, U.C. Irvine, Netscape, Novell. RFC 2518, February, 1999.
  <http://www.ics.uci.edu/pub/ietf/webdav/protocol/rfc2518.pdf>
  The WebDAV Distributed Authoring Protocol specification.
- E. J. Whitehead, Jr., Y. Y. Goland, "WebDAV: A network protocol for remote collaborative authoring on the Web", *Proc. of the Sixth European Conference on Computer-Supported Cooperative Work*, Sept. 12-16, 1999, Copenhagen, Denmark, pp. 291-310.
  <http://www.ics.uci.edu/~ejw/papers/dav-ecscw.pdf>
  An academic paper giving an overview of the WebDAV Distributed Authoring Protocol.
- WebDAV Book of Why, Yaron Y. Goland
  <http://www.webdav.org/papers/#misc>
  A collection of perspectives on WebDAV design rationale
- WebDAV Frequently Asked Questions (FAQ)
  <http://www.webdav.org/other/faq.html>
  A slightly dated list of DAV Q&A

DAV Searching and Locating (DASL) Protocol

- WebDAV Search
  <http://greenbytes.de/tech/webdav/draft-reschke-webdav-search-latest.html>
  Latest revision of the DASL protocol specification

Catacomb Web Server

- Catacomb Project Web Site
  <http://www.webdav.org/catacomb/>
  Provides access to source code, installation tarballs, and documentation.

Neon Library

- Neon Project Web Site
  <http://www.webdav.org/neon/>
  Provides access to source code, installation tarballs for multiple platforms, links to projects using Neon, and links to wrappers in other languages.