

Custom WebDAV Applications

When WebDAV is used for an application that includes but goes beyond simple distributed authoring, it is a custom WebDAV application. The application can be as simple as a few business rules acting on custom property values or as complex as Exchange 2000. This chapter discusses the advantages of using WebDAV for custom applications, the issues involved, and some examples.

Before the ubiquity of the Web, custom applications typically involved custom client code. That meant that client software had to be installed on users' computers. Users don't like installing software, but a client application is sometimes the only way (even today) to present a usable interface with required functionality. If the custom application ever requires added functionality after initial deployment, then the difficulty and cost of upgrading the deployed base of client applications is huge.

With the Web, many more custom applications can be designed to use server-side code alone. Many user interfaces can be done entirely in HTML, with HTML form submissions when the user needs to perform an action. Web-based custom applications are now very common, featuring many useful tools to cut down on development and deployment costs. However, many Web-based custom applications are also slow, suffering from poor user responsiveness, poor scalability, or both.

The development of client-side scripting helped Web applications designers cut costs and improve the usability of a custom Web-based application. Client-side scripting improves performance when it reduces the number of requests that need to be made. Although the client must download the script in addition to the Web page, if the script reduces roundtrips at all, performance is improved because that's where the real costs are incurred. If you've designed Web

applications to use client-side scripting, then you're aware that scripting requires careful design, making decisions about what gets done where.

In many of the same ways, WebDAV can be another valuable tool to cut Web application costs. As a bonus, WebDAV is far more standardized than client-side scripting!

This chapter discusses first how to use WebDAV to augment an existing application without writing any client-side code (not even a browser-run script) and very little server-side code, in the context of an online photo album or other picture storage repository. Second, a more complex example shows how WebDAV was integrated into Exchange 2000. Among other things, this example shows how browser-executed script can use WebDAV to host an application that is faster than customer Web-hosted applications where the browser downloads server-generated pages without client script. Because so many client applications already support WebDAV, neither of these custom applications requires any software to be installed by the user.

The third example in the chapter is instant messaging. That example includes some suggestions where *not* to use WebDAV as well as some ideas how WebDAV can happily integrate with other protocols. Finally, a problem-solving environment for research organizations shows how much more flexible a WebDAV property schema is than traditional database-backed custom application schemas.

14.1 Online Photo Album

Now let's examine how we would implement a hypothetical Web service using WebDAV. As digital cameras have become more popular, a number of Web sites have emerged to allow people to share, manage, and print their digital images. These services are generally called **online photo albums**.

14.1.1 Application Overview

An online Web-based photo album is a Web service that hosts collections of photos for many users. The service provider can offer the service for free, support it via advertisements, or charge for this service. The site could contain mostly people's travel and family pictures, professional photographers' portfolios, or images collected by a publisher for use in books. We'll assume that the users are regular computer users with no special software-related expertise, so the service has to be easy to use.

Images can be created in many applications. Digital photos can be created directly on the user's computer from a connected camera, downloaded from a portable camera, or scanned in. Applications like Adobe Photoshop are used to create enhanced photos. Most likely, the image is stored at some point on the user's computer (see Figure 14–1).

Uploading files is a special-case function on the Web. Browsers must protect users' privacy. In the upload case, this means ensuring that unauthorized Web sites cannot select and upload users' files from their hard drives. Consequently, Web browsers have security blocks built

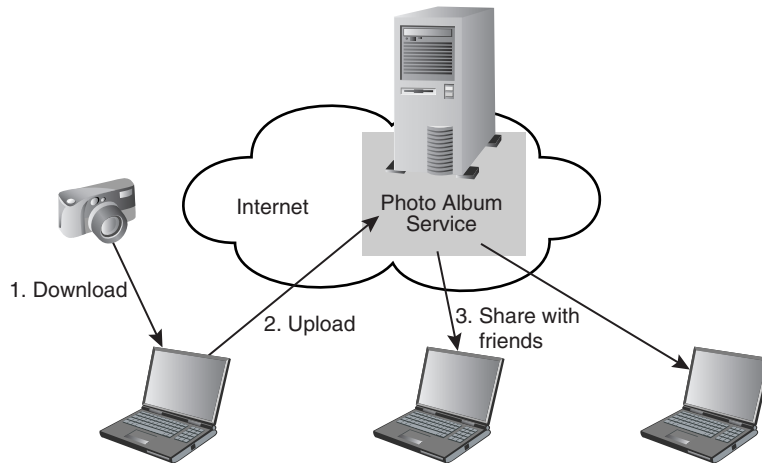


Figure 14–1 Photo-sharing life cycle.

in specifically so that when the server suggests a file upload (through an HTML form), the user must be involved to select a single file and agree to upload it.

Frequently users have several images to upload at the same time. It is possible to use an HTML form to upload multiple files, but it's not exactly easy to use. Only one file can be selected for each file upload input slot. To upload five files, the user must select each file independently in its own input slot and then submit the HTML form (or worse, submit the HTML form five times). It's not possible with ordinary HTTP Web browsers to upload five files by selecting them all together.

To solve the multiple-file upload problem, Shutterfly (a public Web photo album service) implemented a special plug-in for Web browsers. This plug-in is very nice, but not only did Shutterfly have to develop it and maintain it, but each user had to agree to terms of use and download and install this component.

Once images are uploaded to the server, they may be arranged according to how or when the pictures were taken, the subject matter, or some other scheme. In any case, users must be able to organize images according to their own plan. It would be nice if it were possible to reorganize images too, to be able to move them from one collection to another.

Once the images are uploaded, other users can view the images. In most public Web photo services, anybody can view the images, so security is rather minimal. For example, a photo collection may have a password, which every viewer needs to know. Another possibility (as in Shutterfly) is that the album may only be viewable by users who have an account with the Web service. Note that the user must be able to set permissions at the granularity of an album at least, if not on individual pictures.

When sharing photos, users like to give their images names and comments. Some systems might allow more detailed information, such as who took the picture and when it was taken. In addition, the photo albums or collections of photos need to be named.

14.1.2 Existing Architecture

An online photo album service can be implemented using only a Web server with an extensibility model (e.g., Apache with CGI scripts). However, the result would be rather hard to implement, be difficult to use, and perform poorly. We can reconstruct how a theoretical Web-based photo album service would be implemented, and consider the challenges:

- The implementation would take a lot of time because every function has to be implemented from scratch. For example, Web servers don't have native ways to allow the client to move a Web resource (like an image) from one place to another, so the Web service has to be programmed to do that.
- The Web user interface would be difficult to use, particularly for uploading multiple photos, but also for moving, renaming, and deleting photos. Users may be accustomed to using a tool like Windows Explorer to move, rename, and delete files, but they can't use these skills with a plain Web interface, because the interface features aren't rich enough.
- The service would perform poorly because all the work of collating data and preparing the presentation must be done on the server. The server is the bottleneck for all client access tasks, and because the server does all the heavy lifting, this becomes a very narrow bottleneck indeed.

Figure 14–2 shows a simple Web service architecture that could be used to run a photo album service or many other kinds of data-backed Web service. This architecture is characterized by a completely server-controlled and server-processed presentation layer on the front end and a tight dependency on the data schema on the back end.

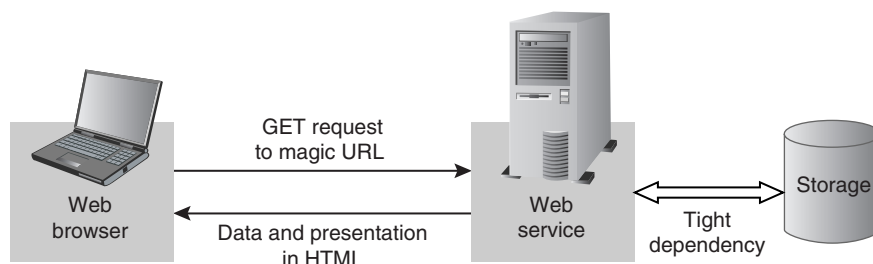


Figure 14–2 Simple Web service architecture.

14.1.3 WebDAV for Photo Authoring

How can WebDAV help in this scenario? What if users could:

- Use a familiar interface, like Explorer, to upload several photos at once?
- Use a familiar interface to move, copy, rename, and delete photos on their collections on the server?

- Use Photoshop or any image-processing application to edit photos from the server and save the photos directly back to the server?

WebDAV offers these features by making photos available as WebDAV resources organized in WebDAV collections. Web Folders and Photoshop natively support the WebDAV protocol and can store files directly on a WebDAV repository. Editing documents directly on a Web repository (without having to download, edit, and upload as three separate steps) is a unique feature of WebDAV and is extremely difficult to provide through HTTP alone.

A photo album service could easily be based on WebDAV from the ground up. This architecture has additional benefits that end users aren't aware of, but these benefits would be important to an Internet startup on a shoestring budget:

- New WebDAV tools are automatically supported. For example, if camera software had WebDAV functionality added, users could save to the Web service right from their camera. Less development is required.
- The server-side presentation layer can be easily separated from the data layer. More data is served directly from the data layer, with less presentation processing required by the server. Fewer servers are required.
- The data layer automatically handles a rich permission model. The Web service implementors can use as much or as little of this model as desired. Permissions are guaranteed to be consistent.
- It is trivial to make the service richer by adding features based on properties. For example, a "style" property (baby album, wedding album, trip album) could be added to each collection simply by using a new WebDAV property. Costly data re-architecture (such as database schema modification) is unnecessary; the programmers can focus on the features.
- It is trivial to extend the service by allowing multiple people to share authoring responsibility for a photo album. With WebDAV locks and permissions, this feature works properly.
- Performance improves because the most frequent type of request (a simple download of a single photo through a GET request) is handled by commercial software that has been tuned to do this quickly.

How Properties Are Exposed

When a Web application uses WebDAV properties, standard Web browser software can't automatically view those properties. Some code is still required to pull the property values into a Web page, format, and display them. Today this is most commonly server-side code such as JSP or ASP pages. However, there are higher-performance alternatives in which the processing and presentation is offloaded to the client, as discussed in Section 14.2.4.

This model assumes that the application is designed with WebDAV in mind, such that WebDAV can be the primary storage model for all the information stored through the Web service. This requires some familiarity with the WebDAV data model to choose how to represent application data most effectively (see Figure 14–3).

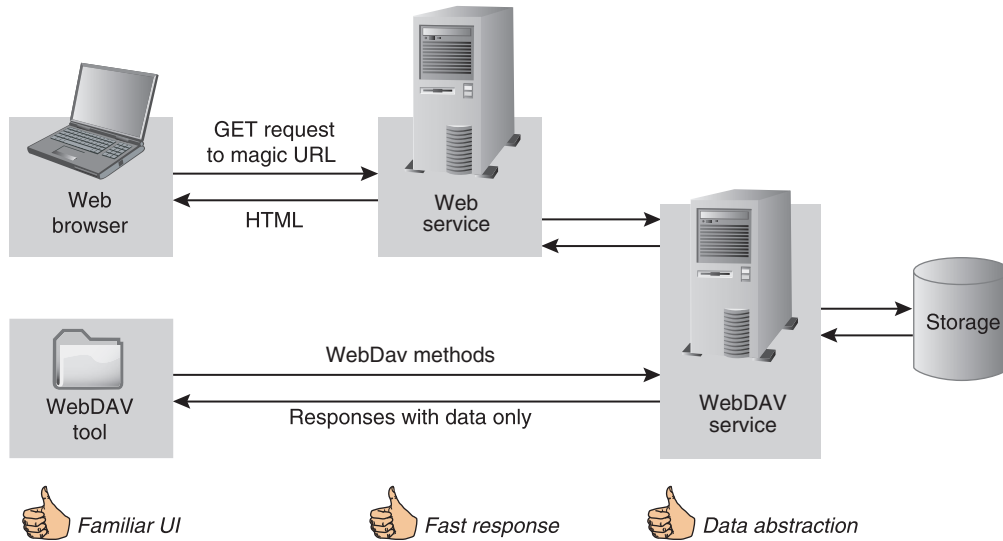


Figure 14–3 WebDAV as primary storage model.

WebDAV is an appropriate technology to use for this purpose, because photos can be simple WebDAV resources and photo albums can be WebDAV collections. Any additional information can be associated with either an individual photo or a collection, using WebDAV properties.

14.1.4 Gateway Model

If the photo album service has already been deployed without WebDAV support, is it worthwhile transitioning to WebDAV? A full transition can be difficult and time consuming. Another model is to expose the existing service through a WebDAV gateway. This could provide the end user benefits of WebDAV access to photos and albums without having to rewrite core architecture.

A WebDAV gateway would not likely be an out-of-the-box solution in this case, but that doesn't mean that the Web service implementors would have to do all the work. For example, the `mod_dav` module can plug into a custom data layer using a simple data access interface (see Figure 14–4).

14.1.5 Lessons Learned

Why does WebDAV provide such convenience and benefits for a photo album service? To generalize this example, it's useful to see what characteristics made the photo album application so

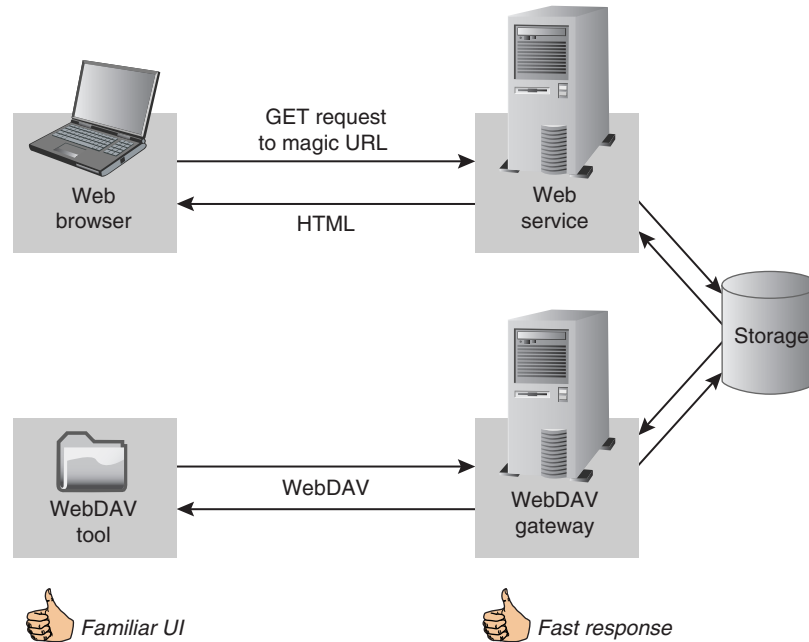


Figure 14–4 WebDAV gateway with existing data storage.

easy to map to WebDAV. Conversely, some of these lessons point out the WebDAV characteristics that make it so useful in this situation.

A Web-Based Service The Photo Album service was conceived as a Web service from the start. WebDAV extends HTTP, so it fits well into this model. Rather than route every JPEG and GIF through the custom code of the Web service, browser requests for photos can be handled directly by the WebDAV repository. The custom code in the Web service can focus on the presentation of thumbnails, browsing albums, and showing banner advertisements—all the value-added functionality, that is, rather than the basics.

Authoring Applications Natively Supported Photos are authored as well as simply uploaded and viewed. End users benefit from being able to use authoring applications (Photoshop, PowerPoint, Illustrator) to directly access files on the server. A photo stored on the server can be touched up with Photoshop quite simply: The user opens the photo, makes changes, and saves it back to the server.

Flexible Metadata Support Photos and albums require metadata to be coordinated into a rich Web presentation. Since a WebDAV repository must already support metadata, the photo album service implementors don't have to do a lot of work to add properties to photos. Even better, there's no fixed schema for properties, so the implementors can add functionality as they go along. That's much easier than knowing the schema, field names, and data types at the

beginning of the project. These properties can be static (defined when the photo is updated), calculated live by the server, or even created by the end user.

Document-Oriented Storage The Photo Album service naturally conforms to storing documents in folders. It works much better when using WebDAV to handle this document storage than when using a database, where storage of large “blobs” is not always easy (photos can be several megabytes in size). Even if the Photo Album service were architected to use regular Web server file storage for the photos and use a database only for storing metadata, the database is inferior for this purpose because the metadata has to be associated with each photo using custom code. Using a WebDAV server means that metadata is already associated with each photo, so that whenever a photo is moved or copied, its metadata goes with it.

Native Support for Document Management For the online Photo Album service, the simple document management features WebDAV provides (moving, locking, synchronizing) are quite useful. WebDAV provides a way for users to organize and upload their photos without requiring custom code on the part of the service providers. Savvy users can use a tool like sitecopy to keep their online photo album synchronized with a local copy of all the photos.

14.2 Email and Calendaring

Exchange Server, Microsoft’s email and collaboration server product, stores and handles users’ email messages, calendar appointments, contacts, and news postings.

Before the Web and the explosion of Internet protocols, Exchange Server had only a few standards to support (RFC822 for message format and SMTP for message exchange). These standards were only supported for communicating with the outside world. To allow client access to server-stored email and calendaring information, Exchange Server instead supported Messaging API (MAPI).

Exchange MAPI

MAPI is an API that both clients and server extensions use to work with Exchange Server. It’s a proprietary interface exposing functions used to create, manipulate, transfer, and store email messages. Confusingly, MAPI is also a protocol used over the wire between Exchange clients and servers or applications designed to be compatible with Exchange (see Figure 14–5).

In addition to Exchange and Outlook, Microsoft has had several other email clients, servers, and services—including MSN mail and Hotmail. To achieve greater interoperability among its own products and services, Microsoft adopted a standard protocol (WebDAV) with a custom schema and proprietary extensions. This section focuses on how and why WebDAV was integrated into Exchange Server 2000.

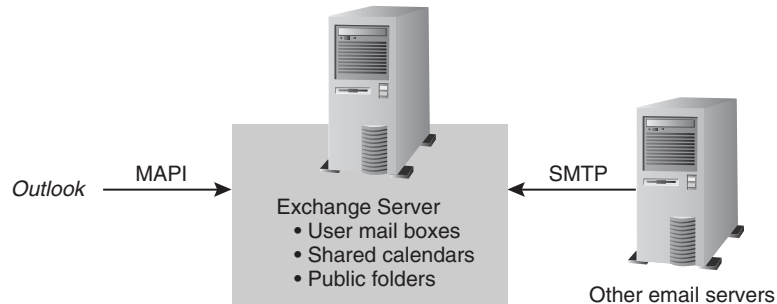


Figure 14–5 Exchange Server communication in early releases.

14.2.1 Web and Internet Standards in Exchange

By the time Exchange Server 5.5 was released, Exchange supported quite a few Internet standards:

- RFC822 for message formatting
- SMTP for message exchange
- POP3 (Post Office Protocol) for client retrieval of email
- IMAP (Internet Mail Access Protocol) for client access to email stored on server
- NNTP (Network News Transport Protocol) for client access to news posts shared among users
- HTTP for browser access to a Web UI, for users who do not have client software installed (e.g., kiosk users)

Exchange Server 2000 (the release after 5.5) needed to do a better job of both Web support and Office 2000 support. According to the Exchange file-sharing vision, every user ought to have Office 2000 (with Outlook) and an email account on an Exchange server. Therefore, Exchange Public Folders were recommended as a central location for sharing documents among Office users. To achieve this vision, Office 2000 users needed to be able to save Office documents directly to an Exchange Public Folder.

Public folders have some advantages over using a simple file system folder to share documents. Public folders can contain news postings, contacts, and emails, as well as Office documents, text files, and so on. Items in the folders can have metadata. Resources in public folders can be locked by a remote user. In other words, a public folder is a lot like a WebDAV collection (see Figure 14–6).

How was Office 2000 going to do remote authoring in public folders? POP3 and IMAP were out of the question because those protocols are limited to email usage. MAPI was rejected because the Office 2000 development schedule was tight and MAPI support would have been a lot of work for not enough benefit. HTTP was a better option, but as we know, HTTP alone doesn't solve remote authoring issues. The development schedule for Office 2000 already

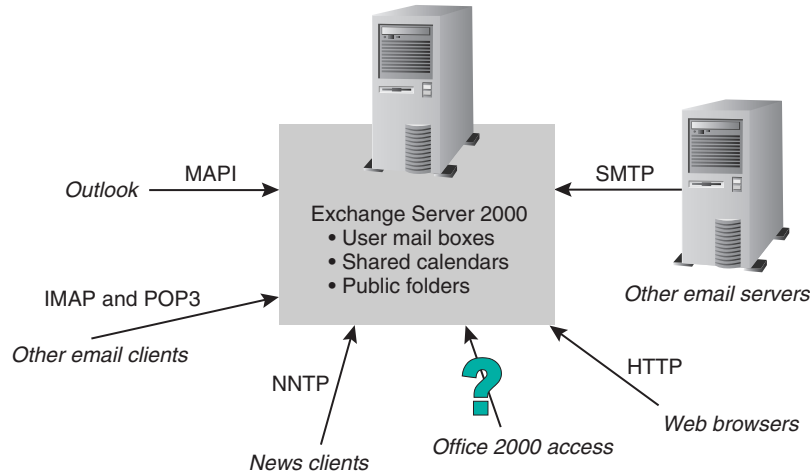


Figure 14–6 Exchange server supporting how many protocols?

included plans for WebDAV support, so client implementors encouraged Exchange Server planners to add support for WebDAV as well.

The biggest question was how all the objects in an Exchange server (Office documents, email, calendars, appointments, contacts, and news postings) would be represented with WebDAV. Calendars and all folders were exposed as WebDAV collections. Emails, appointments, contacts, news postings, and Office documents were exposed as regular WebDAV resources, but each had special properties. For example, an email object needs properties such as “from” address, “to” address, date sent, date received, and subject. Thousands of custom WebDAV properties were defined to hold all the metadata that can be found in an Exchange repository.

14.2.2 “Webifying” Exchange 2000

One of the planned new features for Exchange 2000 was direct Web access to its folders and resources. Although at first blush it doesn’t sound exciting to do a GET to an email, this functionality was useful for a number of reasons and, in fact, consistently requested by Exchange customers.

First, URLs, particularly HTTP URLs, have become universal addresses. A URL is extremely portable because it includes not just the location of the resource within a repository but also the server name and even the protocol to use to access the resource. Exchange resources didn’t have addresses of any kind, so HTTP URLs would be particularly useful, providing universal and interoperable addressing. HTTP URLs allowed information in Exchange to be linked into the World Wide Web. Customers were demanding this feature, with the obvious implication that if Exchange did not provide URLs, customers would have to remove important information from their Exchange servers and put that information somewhere on the Web.

The Exchange designers thought even bigger than this. They aspired to make Exchange the first place users and application writers would think of storing all shared information. Often, shared information is in the format of Web pages (HTML files) and images, and these are now expected to have URLs. HTTP support allows application software developers to easily write code to add and retrieve information from a server (it's much easier than supporting MAPI). HTTP dramatically lowers barriers to custom software. For example, a cell phone that already had some HTTP support would find it relatively easy to download data from a user's calendar and automatically remind the user of appointments.

Finally, customers were also demanding a Web-based GUI front end for Exchange so that users at kiosks and on the road could have full access to their inbox, navigate mail folders, send and reply to email, and delete messages. In response, Exchange developed **Outlook Web Access (OWA)**.

14.2.3 OWA

OWA was the first Microsoft client application to access Exchange 2000 data over WebDAV. (In fact, it shipped with Exchange 2000 [Microsoft00].) OWA is a Web server application that can run either directly on the Exchange server or on a separate server. It allows users at kiosks, or any computer that doesn't have a mail client installed (including Unix and Linux clients), to use a Web browser to get to their Exchange mailbox content. It's also useful when I use a computer configured to view somebody else's mail and I don't want to change the configuration. For example, when I'm at a remote site and borrow a friend's Internet-connected workstation to check my mail, I can use my friend's Web browser to do so without changing the configuration or settings of her machine.

To present its Web-based GUI, OWA must of course dynamically generate Web pages that contain some of the data from a user's mailbox—just enough to fill a screen, so the download is quick and the interface feels responsive to the user. This can be a lot of work for a Web application server, because as soon as the user pages down to view more email or deletes an email, the whole page must be regenerated. At least, that's how it works with IE 5.0 and earlier, as well as with Netscape clients. With IE 5.5, OWA can make the client use WebDAV.

14.2.4 How OWA Uses WebDAV

When OWA builds a dynamic Web page for IE 5.5, it doesn't put the data in the page. Instead, it includes scripted routines for doing WebDAV PROPFIND requests to the user's mailbox. The script uses a component called XMLHTTP. As soon as IE 5.5 receives the page, it executes the script and gets the PROPFIND response. The script also includes instructions for using an XML stylesheet to convert the PROPFIND response body in XML to nicely formatted HTML. Appendix A has a much-simplified example of this kind of code.

Why PROPFIND?

OWA could have used GET requests to dynamic resources, to ask for property sets and get answers formatted in XML. Instead, PROPFIND was used because it was already well defined and specified and included almost all the functionality required (the ability to request a specific range of responses was added). A dynamic GET solution would have had to redesign some of the same solutions.

Furthermore, Exchange Server already had to support PROPFIND in order to support WebDAV for Office 2000 authoring, and PROPFIND already had to be optimized to respond quickly. The OWA designers wanted to take advantage of the functionality already implemented and optimized.

To get the first view of a page such as a mailbox listing, the client has to make three requests instead of one:

- A GET request for the page framework and script
- A PROPFIND request for the listing data
- A GET request for the stylesheet to format the data

However, once these pieces are in place, every time users scroll down in their mailbox or delete an email, the script in the page tells the client how to do another PROPFIND request to fill in the new data. With the same page framework and the same stylesheet, the new data is inserted in the page. Since all of this client processing happens faster than the full HTML could be downloaded, users see their data faster.

Although OWA supports browsers that do not have XMLHTTP, it is much more efficient when XMLHTTP is available to separate data from presentation. Stylesheets can do the presentation part of the job, but there must be a way to download the data only, and that's where XMLHTTP comes in. The server running OWA sends client script code invoking the XMLHTTP object. This script causes the browser to make WebDAV requests (usually PROPFIND) to get the data in compact form in XML. Since fancy Web pages are 90 percent presentation, this can result in much more efficient browsing when the presentation is reused.

A good example to illustrate this concept is the way OWA displays the user's inbox. The screen shows about 20 messages at a time. To scroll down or resort the listing of emails, Internet Explorer 4.0 has to ask the server for a whole new Web page, complete with all the presentation information, even though only a few data fields change. In contrast, Internet Explorer 5.0 is instructed to use the XMLHTTP component to make a PROPFIND request to get the first 20 emails in the inbox and then download a stylesheet to present the PROPFIND response body. When IE 5.0 users scroll down, a new PROPFIND request is made that only retrieves the new data; since the stylesheet is cached locally, it is reused.

OWA servers do less processing when handling WebDAV requests, compared to integrating data into a HTML presentation. That means that OWA servers can handle many more IE 5.5

clients (and later IE versions, too, of course) than other browser versions. A data sheet on OWA performance states that before IE 5.0 features were available, “the effective number of users per server was limited by the overhead needed to support interpreted scripts in ASP and to run MAPI sessions within ASP.” With IE 5.0, WebDAV, XML, and DHTML are used to increase performance. With IE 5.5, stylesheets improve performance even more (see Figure 14–7).

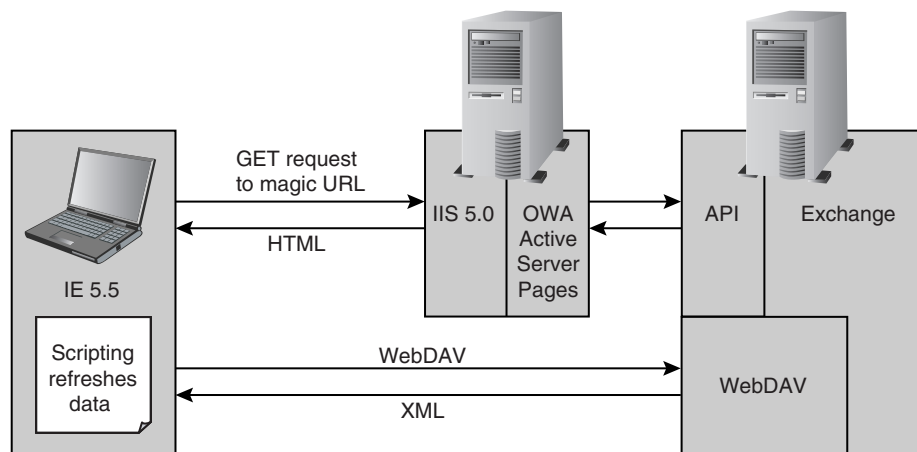


Figure 14–7 OWA architecture.

Because IE 5.0/5.5 users consume fewer server resources, not only is their experience faster, but the server can also handle more of them.

Apple iCal

Microsoft is not the only company to use WebDAV for calendaring. In 2002, Apple introduced iCal using WebDAV to publish calendaring information to the Web. However, its schema is likely quite different.

14.2.5 Hotmail and Web Infrastructure

At the time when Exchange developers were considering all these Web features, Microsoft had recently acquired Hotmail, and the Exchange and Hotmail teams were planning how to move closer together in terms of technology and platform.

Hotmail was entirely dependent on the Web from day one. The Web allowed an email service to potentially make money by showing ads to its users, which IMAP and POP3 didn’t do. The Web also allowed a service like Hotmail to traverse firewalls. If a public email service were configured to use a proprietary protocol, it would never get through corporate firewalls and onto employee computer desktops. Most computer users are not savvy enough to telnet out of their corporate firewalls to a Unix telnet host and run a command-line email client to

read their non-work-related email, but that's how I had to view email that arrived at my student account during work terms. The Web changed that (see Figure 14–8).

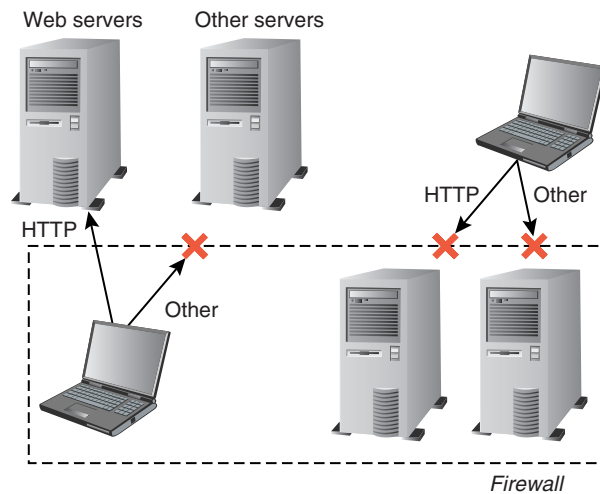


Figure 14–8 Firewalls allow outgoing HTTP traffic but almost nothing else.

Many firewalls simply block traffic through almost every port, but very few firewalls block port 80 entirely. Because the Web is so key to communication, firewall administrators are forced to allow HTTP requests from inside the firewall to reach Web servers outside the firewall. HTTP responses to return through the firewall must return over the same connection.

The Hotmail acquisition thus provided even more reasons to add HTTP support to Exchange server. Users became accustomed to access their email even if they were behind a firewall, and HTTP support via Outlook Web Access allowed them to do that.

14.2.6 Outlook Express

Exchange Server supported another client in addition to OWA and Outlook, and that was Outlook Express. Outlook was designed primarily for corporate use and it had to be purchased as part of the Office suite. Outlook Express was designed to be a free consumer email client, so it had much simpler functionality than Outlook. However, Outlook Express was still a real client-side application, which the user could download, install and configure, and use offline. To make it even easier to obtain, Microsoft included Outlook Express as part of Internet Explorer. In fact, since Internet Explorer shipped with Windows, Outlook Express came preinstalled for many users buying Windows computers.

Outlook Express was frequently used to access MSN and Hotmail email accounts. Because Outlook Express is not a thin client, like OWA, it needs to use a data-oriented protocol so that it can populate its own GUI with data, rather than accept the formatting the

Handling Intermediaries That Block HTTP Requests

WebDAV extends HTTP, so WebDAV benefits from nearly the same treatment at the firewall. WebDAV does not attempt to subvert security, however—in theory, it can be blocked by a slightly more sophisticated firewall that filters HTTP requests based on method.

However, there are two ways to work around firewalls that block methods. First, a protocol can always tunnel through POST. A custom client or gateway can send POST requests, with a custom header that names a WebDAV method. (This technique is also useful to bypass Web application servers that ignore non-HTTP methods.) Second, the server can simply provide SSL/TLS access. Some firewalls are configured to allow any opaque SSL/TLS connection on port 443 [RFC2818] through the firewall, but WebDAV methods are blocked on port 80. Even using port 80, it may be possible to use the CONNECT method [RFC2817], which essentially tunnels an opaque channel through the firewall.

Firewalls and applications using tunneling protocols are sometimes seen to be engaged in an arms race. The firewall can always look deeper into the guts of the message, but the tunneling application can always embed its information more deeply in seemingly innocuous messages. Eventually, the tunneling application can encrypt its messages in a stream, and the firewall's ability to filter on syntax is completely forestalled.

server provides. MSN used IMAP as a data protocol, but Hotmail never did. Instead, to support Outlook Express, Hotmail added very basic WebDAV support. Since WebDAV used the same port, protocol, and infrastructure as HTTP, it was much easier to add and served the purpose at least as well as IMAP.

Some users who had not purchased Office or who simply preferred the simplicity of Outlook Express wanted to use Outlook Express to access their Exchange-hosted email as well as email from MSN, Hotmail, or other services. WebDAV support was needed in Exchange to handle Outlook Express, too.

14.2.7 Scalability

A final pressure on Exchange Server developers was continual performance and scalability improvements. Customers found that in order to support 2000 to 3000 users, they had to buy two Exchange Server licenses. The real cost wasn't so much in the licensing fees but in the system administration costs to link two Exchange servers, share user addresses, manage delivery of incoming mail to the correct server, and share Public Folder content across both.

HTTP/WebDAV offered the promise of handling more users than IMAP and MAPI. Both IMAP and MAPI are connection-oriented protocols, so while a user is online and has his or her email client running, the email client tries to maintain a connection to a server. Even when data isn't being exchanged, the connection is still open and the server has to remember the context for each open connection. In contrast, HTTP allows the server to respond to a request for data and

close the connection. The server can quickly move on to the next request. HTTP makes it easier to load-balance client requests across a number of identical application servers (a model that was integral to the scalability of Hotmail).

However, OWA requires the server to do all the presentation as well as data hosting. OWA servers could handle half as many simultaneous users as a regular IMAP/MAPI Exchange server because of the high cost of producing and transmitting presentation information. Any time the server can delegate presentation to the client and serve only data, the server should be able to move more quickly to the next client and handle more users. With WebDAV requests scripted from the OWA pages when viewed in IE and formatted by the client using stylesheets, OWA was able to delegate some of the presentation to the client and handle more IE users.

14.2.8 Integration Challenges

One problem with implementing email and calendaring in the request/response model is that in some cases it would be very convenient for the server to be able to send the client a notification: “You have new mail” or “You have an appointment in 15 minutes.” IMAP already does this perfectly well, but the approach chosen by the Exchange team was to do notification over HTTP/WebDAV and over port 80 instead so that the client didn’t have to maintain a constant IMAP connection to the server. In WebDAV, frequently updated information may be retrieved through frequent PROPFIND requests, but this polling technique is prohibitively expensive in terms of server performance costs [Fielding98].

A true notification model would be much better, if the client has an address where it can be contacted with notifications. It’s also possible to design a cheaper polling technique than PROPFIND, in which the client tells the server the kinds of changes it’s interested in and polls to see if changes have been flagged. Clearly, WebDAV doesn’t solve the notification problem adequately.

Other problems were more or less easily addressed. Some examples:

- A search framework had already been proposed within the WebDAV Working Group. Exchange used this framework with SQL syntax searches to provide fully equivalent search functionality.
- Some email folders (Sent Items and Deleted Items for many users) are extremely large. The client doesn’t need to display everything in the folder, however. There ought to be some way to request a range of resources from a large WebDAV collection and to get simple properties to quickly show how many resources are in the collection altogether. This functionality was added rather simply with a range header definition for PROPFIND along with custom properties.
- Calendaring properties weren’t always easy to map to the WebDAV flat property list model. For example, the recurrence pattern of a recurring appointment is hard enough already, and on top of that the server and client must be able to express exceptions to the recurrence pattern.

14.2.9 Lessons Learned

Once again, the characteristics that made Exchange Server and WebDAV suited for each other teach us lessons for similar applications.

Web Addresses Since every object in the Exchange repository became an HTTP resource, every resource had a Web address. Web addresses are useful as universal links, easy to pass around in email and insert into files. This linked information in Exchange into the World Wide Web.

Lower Application Development Costs Exchange found that OWA and other applications were cheaper to develop once the Exchange repository supported HTTP.

Web Infrastructure HTTP and WebDAV support allowed the Exchange server to take advantage of existing Web infrastructure. Users can use Outlook Web Access or a WebDAV client to access their Exchange Server information despite firewalls.

Another benefit of the Web infrastructure comes from caching proxies. Since OWA includes images (e.g., in the toolbar), a client could download images from a cache without even hitting the OWA server.

Not all resources are cachable, though. When a resource is subject to access control, the server must prevent caching to avoid breaches of confidentiality.

Document-Oriented Storage Like the Photo Album service, email and calendaring already have a natural document-oriented storage model. It wasn't hard to adopt WebDAV as a standards-based mechanism to move, copy, and delete objects and manipulate properties that already fit the data model.

Flexible Metadata Support Even more than the photo album example, WebDAV's flexible metadata support was important in Exchange 2000. Several thousand properties were defined for contacts, appointments, emails, postings, forms, events, tasks, and notes, and more were defined for special folders like mailboxes, calendars, and address books. Although the underlying store already existed (and the implementation already had similar metadata support), WebDAV was a useful tool to access metadata. Using the XMLHTTP component to run PROPFIND requests, Exchange administrators and developers of custom applications can easily script Web pages to display any metadata from the store. Chapter 15 has more information on this component.

XML Stylesheets Work with WebDAV Exchange OWA uses XML data from WebDAV responses together with stylesheets to delegate presentation to the Web browser. Exchange isn't the only application to use this trick. Tamino XML server and Tamino WebDAV server provide a platform based on this model.

Support for Remote Document Management To copy, move, delete, or rename Exchange resources or create folders or resources, previously a custom application had only a couple of options. It could run on the Exchange server and use server-side APIs, but that requires

administrator approval, which is difficult at sites where the servers are carefully managed or already nearing their top capacity just handling ordinary usage. The application could run remotely and use MAPI, but MAPI is complicated. Support for WebDAV greatly simplified development of remote applications, which could now put together simple HTTP requests to do many operations on resources on an Exchange 2000 server.

Support for Multiple Authors Email and calendaring applications are not typically designed from the ground up for multiple authors, but in fact there are quite a few scenarios in which multiple author support is necessary. Personal email folders and calendars are often shared between executives and their assistants. Group calendars may need to allow events to be authored by a number of users. WebDAV supports multiple authors, so it was able to handle even these scenarios.

Stateless vs. Connection-Oriented HTTP's stateless design is a contributor to a Web server's ability to scale to handle a great number of users. Every HTTP request contains all the context needed to be able to handle the request. Every HTTP request can be dealt with by sending a single response. Once the response is sent (and potentially some data is saved), the Web server can "forget about" the event that just happened. With a connection-oriented protocol such as IMAP, the server has to maintain a connection for every active user, remember what the user is doing, and use previous messages as context for new messages. On similar machines, an HTTP server can typically handle more active users than an IMAP server can.

14.3 Presence and Instant Messaging

Now let's look at an application that doesn't benefit as clearly or directly from WebDAV. **Instant messaging**, together with presence information, is a relatively new Internet application. Instant messaging is the ability to send a short message to another person while he or she is online and have that person respond immediately, forming a conversation more than a correspondence. An instant message doesn't get stored for delivery later. Presence information is deeply embedded in instant-messaging applications because the instant message isn't composed, sent, or delivered unless the recipient is online. "Buddy lists" has become a common phrase to describe the lists where people can quickly see which of their contacts are online and decide whether to send an instant message to any of them.

The difference between instant messaging and email isn't very clear and may blur more because email has become nearly instant, and instant messages may be stored like email. Part of the difference seems to be the user interface, which in instant messaging is lightweight and sometimes playful.

Instant messaging has quickly grown to include participation from a large proportion of the Internet population. The application appeared and grew suddenly, and the slow protocol standardization process is only catching up now. Part of the problem in developing standard protocols for this application is that it is complicated. The application involves many different facets, complicated security models, and high-performance requirements.

I'll outline the different pieces of the puzzle first, paying particular attention to how the application is used, because in this case the details of how it is used are especially important in choosing a model for a solution. I'll compare and contrast to existing protocols because in five years of designing instant-messaging protocols, I've heard dozens of people with prior protocol experience say, "Why not just use *X*?" where *X* is the existing protocol with which they have experience (including WebDAV but also SMTP, IRC, and others). It's not that simple.

14.3.1 Instant-Messaging Requirements

These are some of the problems and requirements for sending instant messages.

Sending Instant Messages Users send instant messages to only a small group of "buddies." Contrast this to email, where my contacts list contains 104 entries, and there are easily 100 more people I might send email to in a given month. My buddy list contains only 25 buddies and I only send messages frequently to about 10 of them. To have a satisfactory experience sending an instant message, it must be trivial for me to choose a user and send a quick, informal, and small message. It shouldn't be necessary to go to another application and find an address. Thus, all the instant-messaging clients I've used allow me to choose a user and send an instant message with two mouse clicks (no scrolling if the target is in the top of my buddy list).

Instant messages must be received pretty much instantly. Because it's text (not speech), a delay of a couple of seconds (a fairly large Internet latency for a short message) is acceptable. However, a delay of five seconds or more is unacceptable because it noticeably impedes a back-and-forth conversation. Compare this to SMTP store-and-forward design, where a one- or two-minute email delivery time (over the Internet) is considered fast.

File Sharing When users discuss and share files, the easiest way to transfer the file (or a link to a file) is with the same application used to discuss the file. That's why email attachments are ubiquitous. However, sending files directly over an instant-messaging protocol can be problematic. The most worrying concern is viruses, but there are also problems with denial-of-service (DoS) attacks, against the server or directly against instant-messaging clients. The current AOL Instant Messaging (IM) client allows users to transfer files directly to one another, but it warns users that before allowing a buddy to send a file—even before deciding whether to open the file—they should have trust in that buddy. That's because the file transfer reveals the recipient's IP address and opens that system up for an attack. However, the AOL system surely doesn't want the file transfers to pass through the servers, because that would put a large load on servers that have plenty of fast traffic to handle. MSN Messenger also handles direct file transfer to buddies. File sharing is difficult yet necessary. At home, IM users want to share images, sounds, and movies. At work, IM users want to collaborate on and discuss documents.

Multiple Clients Users want to have IM functionality available at home, at work, at school, and sometimes even on the road. Some cell phones and PDAs interact with instant-messaging systems. Laptops with wireless connectivity frequently have IM clients installed. This leads to a couple of requirements:

- The user needs a server to help synchronize or manage IM addresses, rather than re-enter addresses on many client devices. Many email servers do this today for email clients.
- In addition, the server must help synchronize or enforce permissions. Email doesn't typically involve "white lists" and "black lists" where certain users are forbidden from sending mail—but modern IM applications always have this feature.

14.3.2 Presence

Sending an instant message is only worthwhile if the recipient is there to see it. Otherwise, people choose email, which gets stored until the user sees it. Thus, successful instant-messaging applications have been coupled with a way to find out who is online. It's most convenient to see at a glance who is online, rather than query one by one for different buddies. It's also most convenient if I can see immediately when somebody comes online (many client applications notify you with a sound like a chime ringing or a door opening) in case I've been waiting to send somebody a message. The entity getting presence information updates is called a **watcher**.

Sensitivity of Presence Information The information about whether I'm online is sensitive information, not for everybody to know (think Internet stalkers). Presence functionality requires a way for the user to decide which watchers can receive their presence information. When a watcher wants to see my presence information for the first time, the system withholds that information until I approve that watcher. If I don't approve of a watcher, the system pretends that I'm always offline (to that watcher). This decision is typically made for the long term: Once I approve a watcher, the system continues to feed that watcher my presence information until I change my mind.

Notice that the ability to hide my presence information from some watchers implies the need for a server. Peer-to-peer instant-messaging applications have been designed and deployed but haven't met with the same success. One reason is that when my client machine is responsible for being a peer in the communication net, publishing my online status directly, forbidden watchers can find out some information based on the reachability of my client machine. If the client machine responds "she's offline" and doesn't respond at all other times, that's too much information. A presence server can do a much better job of pretending, to a watcher I don't like, that I'm completely unreachable all the time. This also implies the need for a server location to help synchronize or manage presence access control information for multiple clients.

Buddy Lists Now that we've looked at the need to send messages quickly to a small group of people and the need to find out conveniently if those friends are online, the "buddy list" concept seems obvious. My buddy list always shows up in my instant-messaging application, with only a glance required to see who is online and only two mouse clicks required to send an online buddy a message.

Recall the requirement that users can do instant messaging on any device. This implies three important requirements for buddy lists:

- The buddy list should be stored somewhere centrally available, so all my devices can use the same buddy list.
- As a user, I should have a consistent address, so all my buddies can still see my presence even when I change devices.
- It should not be obvious that I've switched devices. If I leave for lunch and my PDA takes over from my PC to receive instant messages, that should not affect how my buddies see my presence or how they send messages (unless I choose to provide that information).

14.3.3 Could WebDAV Do Instant Messaging?

In theory, WebDAV could be used as the basic protocol to provide an instant-messaging service (and this idea has been seriously suggested to me several times in the past). However, instant messaging would have to be painfully shoe-horned into the WebDAV model for this to work. I'll lay out a straw man proposal:

- A special collection on a WebDAV server is designated to be an instant-messaging inbox (as Exchange does with email inboxes). The address of this collection is the IM address for both presence and message delivery.
- A set of properties on that collection could advertise my presence status, which other users would query with PROPFIND.
- Messages can be delivered to this inbox with PUT.
- Properties such as sender and recipient can be added to messages with PROPPATCH.

In theory this sounds reasonable: A WebDAV server is likely to be always online, so it can always advertise the user's presence. However, there are a number of problems:

- How are Web proxies and caches expected to handle these messages? Could they cache instant messages, or does every instant message have to be tagged to prevent caching? There's also a risk that firewalls would block incoming HTTP PUT requests, assuming them to be attacks on internal Web sites.
- How do instant messages get from the user's server to a client application? Would the client application have to be a WebDAV server too, in order to receive PUT requests from their server? WebDAV isn't meant to be lightweight on the server side, so this could be a serious burden for a PDA to be a WebDAV server.
- Using PROPFIND to get a user's presence information works fine the first time, but users want to be notified every time a buddy's status changes. WebDAV has no provision for a live constant connection, for notifications from server to client, or for subscriptions to changing information.
- Once an instant message is sent, the sender should have no ability to change the message. Therefore, the sender can't use PROPPATCH to put the appropriate metadata on the message.

- How are MOVE and COPY useful? There is an expectation that any WebDAV server must be a full WebDAV server, supporting at least MKCOL, MOVE, COPY, PROPFIND, and PROPPATCH, in addition to HTTP methods GET, PUT, POST, and so forth.

Surely another protocol, designed more for lightweight messaging and a trusted connection between a presence/messaging client and server, would have many better characteristics. However, while designing a new protocol for presence and instant messaging, there still seems to be a role for file sharing. WebDAV can usefully be a small piece of a much larger picture.

14.3.4 Appropriate Use of WebDAV

Here are some problems related to instant messaging where WebDAV may be appropriate but only to solve these specific problems and not as a general transport. The benefit to this reuse is that the instant-messaging protocol can specialize in the tasks that are truly special and important to the whole character of instant messaging, without solving all of the file-sharing problems that WebDAV has already solved.

Storing Buddy Lists and Access Control WebDAV might be a useful way for a number of client devices to share and synchronize the same buddy list and access control lists. The buddy list can be one resource or several resources, as can the ACLs. This model could be used even if the server enforces access control restrictions.

WebDAV locking is surprisingly useful here, to avoid having one client device overwrite the buddy list changes of another client device. Other configuration information, in addition to buddy lists, could be stored centrally in the same collection.

File Sharing Should files be sent between users by storing them temporarily on a WebDAV repository? With an intermediary like that, some of the security problems of direct file transfer can be mitigated, and it certainly makes the instant-messaging protocol simpler.

Storing Offline Messages Some instant-messaging servers can store messages for the user while the user is offline. When the client reconnects, the offline messages are all delivered. WebDAV could be used as a storage location for these offline messages so that the instant-messaging client can retrieve the offline messages at leisure. However, offline messages really start to resemble email, so probably an email protocol would be even better for solving this problem.

14.3.5 Lessons Learned

Only Clients Make Requests in WebDAV WebDAV is more useful for applications where the client is making the requests. There is no standard way for the instant-messaging server to initiate communication to deliver messages to the client instantly. The WebDAV model would require it to wait until the client asks for instant messages. WebDAV is best used for those applications, or those components of applications, where it's most appropriate for the client to initiate requests and receive responses.

While one could imagine turning every WebDAV client into a WebDAV server so that it could also receive incoming messages, this is a complicated approach and probably not worth it for the application of instant messages. First, WebDAV has more functionality than an instant-messaging receiver needs to receive messages, so a simpler protocol might be better. Second, if the instant-messaging sender and receiver can both make WebDAV requests and handle WebDAV responses, then WebDAV has no way of managing or correlating this parallel traffic.

WebDAV Can Be a Component of a Solution WebDAV has functionality that can solve some parts of the instant-messaging problem. For example, we can choose to store buddy lists on a WebDAV server and transfer files via intermediate storage on a WebDAV server. In contrast, a lightweight connection-oriented messaging protocol can be optimized to handle the delivery and reception of instant messages. The WebDAV component can behave independently of the other parts of the solution—there is no need for the messaging protocol to be aware of or interact with the WebDAV component. Breaking down problems like this, reusing existing solutions where most appropriate, is an excellent way to simplify.

Similarly, other protocols can integrate with WebDAV to provide benefits in WebDAV-based applications. For example, Jabber (now being standardized through the IETF as XMPP) is an instant-messaging protocol that can be extended to provide all kinds of subscription/notification services to a custom application [Shigeoka02].

The Storage Model Is a Necessary Feature of WebDAV Although an instant-message delivery protocol that looked a lot like WebDAV could be written, it would be difficult to make it interoperable. WebDAV applications that PUT WebDAV resources expect them to be there, and be readable, and be the same content that was just placed there. A store-and-forward or drop-box model, where the messages are PUT into a black hole and can't then be retrieved by the sender, may not be a good use of PUT.

Compliance Means All Methods To be a standards-compliant and interoperable WebDAV server, a server must implement the following functionality:

- All of HTTP, including GET, PUT, POST, HEAD, DELETE, OPTIONS. Also remember HTTP requires ETags (entity tags), ranged requests, and conditional headers.
- All of WebDAV level 1 (everything except locks).

If the server doesn't support all of these, then many existing WebDAV clients won't be able to interoperate. The client may attempt a method to connect to the server and then give up if the server doesn't seem to support WebDAV properly.

Clients can get away with a smaller set of functionality because clients can always choose not to initiate a request that they don't want to have to deal with, but there is still a heavy implementation burden to bear.

14.4 Pacific National Laboratories

The last example is a real case study like the Exchange 2000 discussion, not a hypothetical case. This case study shows how WebDAV can be used to design a custom application and make it extensible and scalable.

14.4.1 Overview

Pacific National Laboratories (PNL) designs problem-solving environments (PSEs) for government research groups. PSEs allow researchers to share and update documents. Researchers author many papers together, even forming teams that cross national boundaries. The final papers are made very widely available, perhaps publicly available on the Web. In contrast to normal multiple-author scenarios, researchers annotate documents more often, adding properties to make searches work better. The information in this chapter is from PNL papers [Schuchardt02a], [Schuchardt02b].

In addition to authoring and publishing papers, researchers now need to share their raw data and semantic data on the Web. For example, a researcher might construct an XML file representing the structure of a specific molecule and publish that semantic data together with raw data from a number of tests involving that molecule. Other researchers can interact with the data and use it more easily than when study results are distributed in paper journals. There are already standard file formats to make data sets or molecule structure information transferable, but there's no standard to share these files.

PSEs have been developed to try to unify these data-sharing functions into one flexible application. The traditional approach to storing information in a PSE has been to store documents in database **Binary Large Object** (BLOB) tables and to store the metadata in related tables. Then the PSE presents a custom view of those documents and metadata, possibly by generating dynamic Web pages. However, this approach leads to overly rigid schemas because it's difficult to plan in advance precisely what metadata will be used. Fixed table schemas for database storage make it difficult to update the system when researchers need to change the way they refer to documents.

PNL researchers designed a new approach to PSEs that involved a number of separate repositories and a flexible schema. Each repository can extend the base metadata schema in different ways. In their design, no one data store or component needs to know the entire schema. Thus, it's much easier for each repository to evolve to serve the needs of its main users (the researchers directly contributing to that repository) while still allowing access to a large readership.

WebDAV is an ideal protocol for this problem. It defines the syntax for this extensible metadata schema (property names and XML namespaces) as well as the access protocol.

14.4.2 Example Problem-Solving Environment

An example of one of the specific problem-solving domains is molecular science and complex chemical systems. PNL has a Molecular Science Software Suite, including software with advanced computational chemistry techniques, project management assistance, and calculation

engines. It allows scientists to construct models of complex molecules, enter research results and analyze them, and launch distributed server-side execution of their computational models for speedy completion.

One component of this suite is called “Ecce,” and among other things, Ecce needs to store its data and modify the data at later stages. Originally (1994–2002), Ecce used an **Object-Oriented Database** (OODB) to store data represented internally as objects. In 2001, Ecce had 70 different kinds of objects it could store in the database, such as Molecule, Task, Experiment, Calculation, and File. Although this approach was successful for years, eventually it began to reach its limitations.

- Any database schema used by multiple parties requires agreement on all aspects of the schema, and the implementors of the different modules in Ecce found the agreement process arduous.
- The choice to use OODBs unexpectedly created a dependency on a specific vendor. Although there are some standards in the area, each OODB may have a proprietary binary document storage format and different ties to programming languages.
- OODBs are not easily and tightly linked to the Web for wide publishing.
- OODB clients are “fat,” not “thin.” OODB clients must know the specific data schema in order to consume data in OODBs. Compare this to the Web, where thin clients accept whatever data the Web server is capable of formatting and simply display the data for the user.

Now that Ecce is adding support for molecular dynamics, the problems of the underlying store may be magnified. Database schemas become more and more complex as the system grows to handle more and more kinds of data, and eventually the system becomes quite unmanageable. Although PNL could have addressed some of these problems by spending more money on OODB software and related software, Ecce’s designers chose to investigate other technology, such as WebDAV/database interface software.

14.4.3 Solution Requirements

PNL required a solution to meet the following requirements:

- Allow direct access to raw data. Data should be accessible in its raw format, not just through the libraries that impose an object model on the stored data.
- Metadata schemas should be discoverable and extensible, rather than fixed in advanced.
- The storage layer should not have to be aware of the nature of each application object (each document or object that has to be stored). This layer separation allows the metadata schemas to evolve independently of the features of the storage system.
- A standard protocol should be used to do data management operations, yet this standard protocol should still not have to be aware of individual schemas.

The solution also had to be deployable in a widely distributed manner, with data stores in many locations and managed independently by different groups.

14.4.4 Solution Choices

PNL chose HTTP, XML, and WebDAV as a solution. HTTP provides extremely broad access. XML is an ad hoc and extensible way to marshal object data. WebDAV combines the two, making Web servers capable of storing XML properties on arbitrary documents. The documents themselves are not restricted to the XML format but can be images or raw data sets. New document types and properties can be added at any time, and each application can use whatever set of properties happens to be needed and understood by that application.

PNL selected `mod_dav` 1.1 as its WebDAV server and data storage implementation. The Apache module was free and easy to extend, and Apache 1.3.11 provided the required security features. PNL used `mod_dav`'s ability to plug in a new repository layer to replace the existing property storage capabilities with a vastly more scalable property storage solution, allowing very large property values (and large numbers of properties per document) to be reliably stored and retrieved. PNL's replacement property storage layer used a hash table in a database manager formatted file. The Gnu DataBase Manager version 1.8 was used to handle the property storage files (one property storage file exists for each WebDAV resource). The Apache Xerces 1.3 XML engine was used to parse and generate XML.

The system was used to create properties as large as 100MB and documents as large as 200MB without problems, sizes that exceeded the expected typical usage.

14.4.5 Mapping Database Schemas to WebDAV Storage

The designers of the new system decided that each data piece that a domain scientist would recognize as a separate object would be stored as a separate WebDAV resource. The designers also considered combining a number of objects inside one WebDAV resource, but this would force clients to download large files and then parse the document body to extract each object. The more granular approach also means that each different object can be annotated individually with metadata of its own. For example, a Molecule object would be stored as a single WebDAV resource with its own properties.

The WebDAV hierarchical storage model was useful in the new system. For example, although each task object in a calculation is a separate WebDAV resource, all the tasks involved in a single calculation can be stored in the same WebDAV collection. This is convenient when using a regular WebDAV browser to look at the repository. However, since each task is tied to each other task in a specified order, there are dependencies or relationships between resources. The implementors planned to represent such relationships between resources through additional property values.

Where data format standards existed (like Molecule representations in Protein Data Bank format), the implementors chose to comply with the data format standard by keeping that data together in one resource body or property value. Otherwise, the implementors broke down metadata into small chunks to maximize flexibility. Many other researchers are working on standard

data formats and MIME types for chemistry information, and these standards are easily integrated into the WebDAV data model.

A namespace was defined for all Ecce properties, and the same namespace was used throughout. However, the implementors envisioned that once the basic system was established, extension work would begin in multiple areas independently, and these extensions would define and use their own namespaces.

14.4.6 Results

The Ecce implementors decided that their new design did alleviate the schema sclerosis that was affecting the old system. With a more flexible XML-based schema, changes in one area of the PSE did not affect other areas as much. The entire system involved fewer up-front costs (in paying for commercial software), as well as allowing for more rapid feature development.

Since WebDAV, HTTP, and XML are such open standards, much more powerful and cheaper middleware software and infrastructure can now be used to deploy a PSE. The system benefits from additional independent layers (storage, transport, data format, data manipulation, and semantics) because most layers can now be constructed out of standard and replaceable software components.

The researchers were happy to report that after Ecce was redesigned to use cheaper middleware, deployment costs became low enough that Ecce was able to reach a wider user base.

14.4.7 Lessons Learned

Higher Performance The implementors of this solution did performance tests to compare the new solution to the network performance of the original OODB-based solution, which had used FTP to transfer files after pulling them out of the database. The file upload performance on the server tested was as good as FTP. The implementors were concerned that the Ecce software modules might perform worse after conversion, and their goal was modest: to avoid a significant performance decrease in these objects. However, overall performance improved, exceeding expectations.

Additional performance optimizations, including HTTP pipelining, multiple TCP connections, and bundling requests, could still be implemented.

Greater Disk Space Required The basic cost of the flexibility in the new system appears to be storage size. Disk requirements increased by up to 25 percent when files and database property files replaced blobs and tables. Part of the bloat, however, was due to the Gnu DataBase Manager software used to store properties: Each property database file has a minimum size of 25KB, even if most of this space is empty. PNL could replace or reconfigure the property database software to improve disk usage. Since the disk storage tests were run with small average resource sizes, the property storage requirements were a significant fraction of the total storage. In a usage scenario with larger documents, the property storage would involve a smaller relative increase in storage space required.

The PNL system could also be optimized to use less storage if the server software used techniques like compression or avoiding file duplication. These kinds of optimizations can be added without disrupting the way the WebDAV repository is used.

Lower Cost Than OODB The implementors also found significant cost benefits resulting from the new architecture. Apache and `mod_dav` were not only free but also cheaper to maintain than OODB systems, even if each department had to have its own storage server. With WebDAV, departments can share storage servers without having to manually harmonize their schemas, so costs are reduced even further. The departments and laboratories could even contract the WebDAV repository hosting to a third-party generic WebDAV hosting service.

Support for Web Access Since all the Ecce data is on the Web, it is now possible for researchers to access their data directly through a Web browser or WebDAV explorer, indirectly through Web server extensions that manipulate the data (results viewed in a Web browser), or through specialized tools.

Note that basic support for Web access means that the system is naturally compatible with MIME and XML standards. Many researchers use text or XML data formats, and there are specific MIME types for many kinds of science-related data files. Chemical Markup Language and Chemical Structure Markup Language, the Math Markup Language, and the Extensible Scientific Interchange Language all use XML. Since they all use XML, they can easily be stored as property values, as well as file bodies. Any format that has a MIME type is automatically labeled and handled by a WebDAV repository.

WebDAV Is Easy to Work With The Ecce developers found that the new architecture was much more conducive to debugging. Web browsers and WebDAV explorer tools became debugging tools, enabling the developers to independently verify the sizes, locations, and names of resources stored and accessed by Ecce. Raw XML data can be inspected in text editors or in XML editors.

Security Was Maintained HTTP security is mature, flexible, and scalable. In addition to meeting the minimum security requirements, the Apache-based system supports many alternative security features. New security features are frequently implemented for Apache, allowing the system to keep pace with recent developments in security functionality.

No Transaction Support The PNL implementors felt that the lack of transaction support in WebDAV was a drawback. Although it did not block deployment in this case, other custom applications could find otherwise.

Continual Improvement Possible A database-backed solution has a certain lack of flexibility. It's a slight pain to add columns to existing tables, and it's more difficult to change columns when data must be upgraded. When functionality changes are large, it's extremely hard to reengineer a set of interrelated tables to handle the new functionality. In comparison, WebDAV makes it easy to change what properties exist on which resources and where those resources are stored.

WebDAV also makes it possible for clients and servers to be improved and upgraded independently. The core WebDAV engine can be upgraded to a new version that will provide backward compatibility for client requests. The custom rules and logic on the server can be updated. New properties can be added to certain types of files, and because of the way PROPFIND is designed, client software that is already deployed will still be able to view those files and retrieve the known properties.

14.5 Other Application Ideas

Finally, here are other ideas for applications or services that could conveniently and usefully incorporate WebDAV:

- Résumé and cover letter database with comments from résumé reviewers
- Special-purpose image libraries (e.g., medical imaging) with rich image metadata
- Class or course resources (syllabus, books, articles, links)
- User and group directories, such as student “e-lockers”
- Online storage services
- Group calendars, event calendars
- WebLog or “blog” server, needing interoperable authoring support
- Virtual safety-deposit boxes
- Publishing and document-related workflow servers
- Web publishing process incorporating contributions, review, approval
- Contract or invoice approval and archiving with simple workflow based on properties
- Collaborative FAQ writing (submit questions, add answers, add further detail)

14.6 Summary

WebDAV provides a number of interesting benefits to different vertical or horizontal client-server applications.

A data-backed Web service can use WebDAV to offer more usable authoring to its content contributors. This kind of application was illustrated with the online photo album example, where users can edit and submit their photos using WebDAV-enabled applications like Adobe Photoshop. The Web service retains control over presentation to users browsing photos over the Web.

A server offering a set of application features like Exchange 2000 can use WebDAV to unify access to application data. WebDAV can be used to model many different kinds of documents and data structures. This also makes the application data available securely over the Internet, reusing Internet infrastructure such as firewall support, SSL/TLS, and HTTP authentication.

WebDAV can be used as a component, even when it's not suitable as the core communication protocol. The instant-messaging and presence example showed that while WebDAV is not a suitable protocol basis for instant messaging and presence subscriptions and notifications, it can still provide associated pieces of functionality.

A vertical or custom data-oriented application can use WebDAV to improve flexibility and lower costs. WebDAV typically introduces a greater level of flexibility into schema management because it is much easier to define custom properties in WebDAV than in OODB systems. Deployment flexibility increases because WebDAV separates the storage layer from the storage management layer and from the server custom logic layer, allowing each layer to be swapped out to improve system functionality.