

An Investigation of Federated Access Management for WebDav

(WSTIERIA Project Technical Note 2)

08/06/2010, Version 0

1 Background

A previous document (WSTIERIA technical note 1¹) described the difficulties of applying commonly used federated access management software to web services, due both to the lack of support in many web service clients for some required HTTP features (redirection, cookies, SSL) and to the presumption in the commonly deployed browser-based profiles that the user is directly present at a browser (to select an identity provider, enter credentials, etc.) The document also showed how an Apache web server can be configured to act as an authorising proxy or “façade” for a web service, forwarding (non-browser) client requests only if the URL contains a session key obtained separately via traditional browser-based federated access management.

The example web service used both in technical note 1 and in previous work was an Open Geospatial Consortium (OGC) Web Map Service (WMS). A WMS enables a client to obtain map images of geographic areas requested by the client, which is typically an application installed locally on the user’s workstation. While these geospatial web services and their corresponding clients are well known in their specialised application domain, they will be less familiar to many of those interested in federated access to web services in general. We were therefore motivated to consider other examples of web services that might be more familiar and accessible to a general audience.

One web service client that is pre-installed on many desktop operating systems is Web Distributed Authoring and Versioning, usually referred to as WebDav, or simply DAV. An XML-based web service application protocol allows a web server such as Apache or IIS to make a directory tree on its host system available for read and write file transfers by remote clients over HTTP. Clients have been built in to common desktop operating systems which present a directory tree on the web through the same user interface as for a local directory on the client system. For example, on Linux, the Nautilus file browser in the GNOME desktop GUI supports WebDav; on Windows, equivalent “web folder” support is built in to the Windows Explorer shell. In both cases, files in a directory exported from a web server can be opened, copied, moved or deleted by the user in just the same way as local files. Other WebDav clients, such as *cadaver*, provide an alternative user interface similar to a traditional command-line *ftp* program.

A shared, writable directory tree, with access restricted to a particular work group or other set of users by means of federated access management, might well be a generally useful facility. It could allow for the exchange of data sets that are inconveniently large to send as e-mail attachments, the maintenance of up-to-date central copies of shared data and so on, similarly to sharing file space in the cloud (Google Docs, Microsoft Skydrive etc.) but accessible using federated access credentials already possessed by users rather than requiring the provider of the service to distribute and maintain yet another set of credentials to every authorised user. The ubiquity of pre-installed WebDav clients (discussed above) sidesteps the handicap which a solution based on local client software would otherwise face relative to a cloud solution accessed via a browser.

These properties of being a web service whose function is easy to understand (remote file access), where servers are easy to set up (being built in to both Apache and IIS, though requiring some configuration) and easy to use desktop clients are already present in Linux, Windows and Mac OSes, indicated the possibility of using WebDav as an example of the façade approach to federating web services which would be accessible to a general audience without specific prior domain knowledge and with a minimum of the kind of up-front friction of

¹ <http://edina.ac.uk/projects/wstieria/files/TN01-facade.pdf>

installing and configuring pre-requisite software which often militates against taking up and experimenting with new methods. We therefore decided it would be worthwhile to investigate whether the façade approach could in practice be applied to WebDav. This note describes the course of the investigation. It assumes that the reader has at least a general familiarity with the façade technique described in technical note 1.

2 Configuring the DAV Server

The first step was of course to set up a WebDav server. Since an Apache server had already been set up for previous work, it was reconfigured to add the additional functionality. The first step was to ensure that the required support modules were loaded by un-commenting the following statements in the `httpd.conf` configuration file:

```
LoadModule dav_module modules/mod_dav.so
LoadModule dav_fs_module modules/mod_dav_fs.so
LoadModule dav_lock_module modules/mod_dav_lock.so
```

In this version of Apache, the DAV configuration itself is included from a separate file:

```
Include conf/extra/httpd-dav.conf
```

The basic outline of the configuration there is unchanged:

```
DavLockDB "D:/Program Files/Apache Software Foundation/Apache2.2/var/DavLock"
Alias /uploads "D:/Program Files/Apache Software Foundation/Apache2.2/uploads"
<Directory "D:/Program Files/Apache Software Foundation/Apache2.2/uploads">
    Dav On

    Order Allow,Deny
    Allow from all

    . . .
</Directory>
```

The first departure from this standard skeleton was that, rather than accepting connections from any client (Allow from all), we wanted to accept connections only from the address of the façade system that would be checking for client authorisation. Initially, it was hoped that the façade would be hosted on the same machine as the DAV server, giving this:

```
Allow from 127.0.0.1
```

Unfortunately (for this purpose), Allow “looks through” the façade or proxy and sees the IP address of the client itself. Listing all end-user IP addresses is obviously impractical, so this approach was discarded in favour of using the system firewall to permit access to the DAV server only from the façade system. Because the firewall operates at the IP packet layer rather than the HTTP layer, it doesn’t have access to, and is therefore not distracted by, proxying at that level. Of course, there is then the substantial drawback that firewall-level blocking will apply to all accesses to port 80, not just accesses to this WebDav directory. This would preclude use of the same web server for multiple purposes in production. For initial experimentation though, it will normally be quite acceptable to use Allow instead of the firewall and to list test client IP addresses explicitly:

```
Allow from 127.0.0.1 192.168.1.42 192.168.1.27
```

The DAV `<Directory>` block above does not contain the following statements which are part of the default configuration. We show these, commented out, below:

```
# AuthType Digest
# AuthName DAV-upload
# AuthUserFile "D:/Program Files/Apache Software Foundation/Apache2.2/
# user.passwd"
# AuthDigestProvider file
```

In normal use, a DAV directory should be configured to require user authentication (digest authentication is shown). Since the directory is writable, it is undesirable to allow unauthenticated users. However, in our case, because of the firewall discussed above, only clients authenticated to the façade can access the DAV server at

all. To avoid presenting the user with a digest authentication challenge in addition to federated authentication, the digest configuration is removed.

Similarly, the default configuration permits only a subset of users (here, a user called `admin`) to perform actions other than a GET or an OPTIONS enquiry:

```
# <LimitExcept GET OPTIONS>
#     require user admin
# </LimitExcept>
```

It might be useful to be able to do the same thing using a federated identifier, perhaps a principal name (e.g., `require user foo@example.ac.uk`). However, in the basic façade model, authentication and authorisation are the responsibility of the façade and are not dealt with at all by the underlying web service. The user identifier is therefore not conveyed to the web service, here the DAV server. Our interest at this stage was simply to see whether the overall approach was feasible, so we did not attempt to go against the grain here and simply left the statements above commented out, meaning that any user authorised to access the DAV directory at all can perform any action on it, without limitation, including writing or deleting files and so on. Note that, even if it were easy to use a federated identifier within the DAV configuration, the reluctance of most identity providers within the UK federation to make non-core attributes such as the principal name available to external service providers would still be an issue.

3 Adding the Authorisation Façade

The basic action of the façade is to proxy requests from clients to the `/uploads` directory of the DAV web server discussed previously, using Apache's `mod_rewrite`:

```
RewriteRule ^/pup(.*) http://davhost/uploads$1 [P]
```

Clients must present a valid session key (`/authNN...NNN`) at the start of the URL used to access the façade. If valid, the key is captured in an environmental variable (`SESSION`) for later use. Otherwise, a new path is substituted (`/goaway`) that results in a “Forbidden” HTTP status (`[F]` flag):

```
RewriteRule ^/auth([0-9]*) (.*) ${uploads:$1|/goaway}$2 [E=SESSION:$1]
RewriteRule ^/goaway [F]
```

The validity of the session key is determined by looking it up in the `RewriteMap` called `uploads` referenced in the snippet above (`${uploads:...}`). Keys not found in the map are treated as invalid. The `RewriteMap` points to a lookup table, in this case a simple text file:

```
RewriteMap uploads txt:/path/to/uploads.txt
```

This file will contain lines like the following:

```
3992103 /pup
```

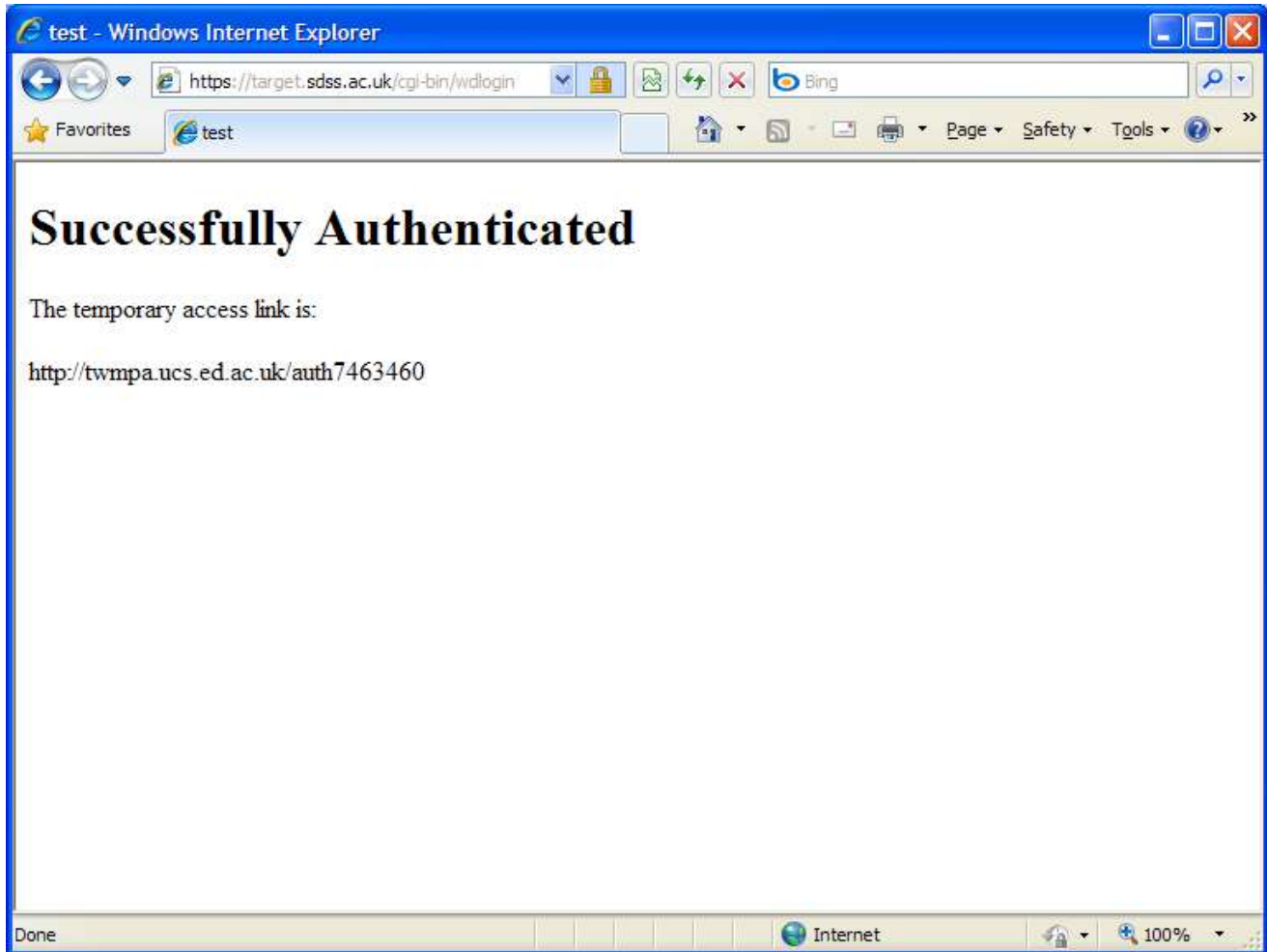
The first field is a session key, the second is always `/pup`, which will be substituted for `/authNN...NNN` by the second `RewriteRule` above and then proxied to `http://davhost/uploads` by the first.

4 Obtaining a Session Key

The user obtains a valid session key for their client to use with the façade by invoking a new-session script from a browser. This script is at a URL protected by standard Shibboleth Service Provider (SP) software, so Identity Provider (IdP) discovery, either via a Where Are You From (WAYF) page or by other means, is performed normally, as is user login at the IdP. Any IdP recognised by the SP can be used; normally this will be an IdP in the same federation as the SP, such as the UK federation.

Once the user has been authenticated in this way, the script can apply any desired authorisation rules to determine who should be able to access the façade, and through it the DAV server, based on user attribute data supplied by the IdP. In our tests, only users having an `eduPersonScopedAffiliation` of `member@edina.ac.uk` were authorised.

Having determined the user's authorisation status, the script generates an HTML page to present to the user. Unauthorised users are rejected with an error page. Authorised users are presented with a page containing a URL that can be copied and used with the façade, as shown below.

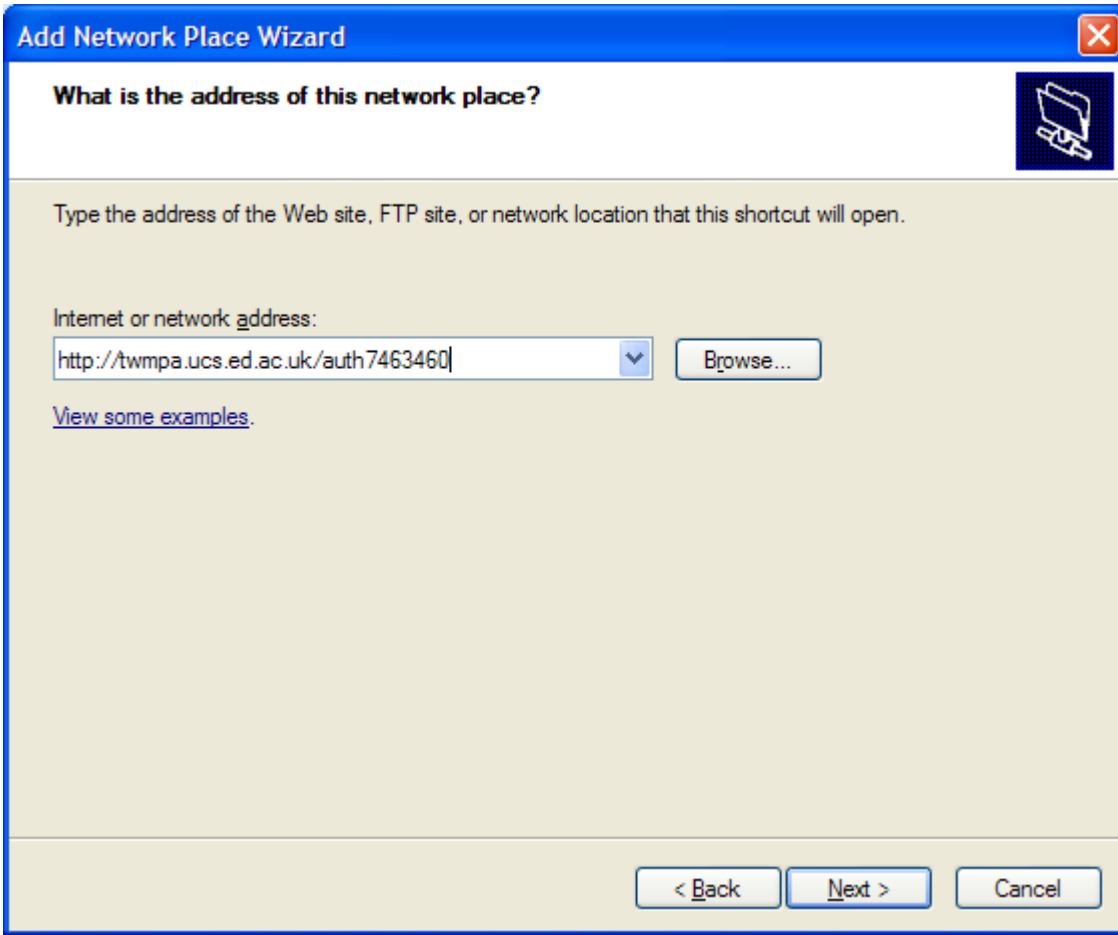


The authorisation script also modifies the RewriteMap file, `uploads.txt`, by adding a line for the newly created session key:

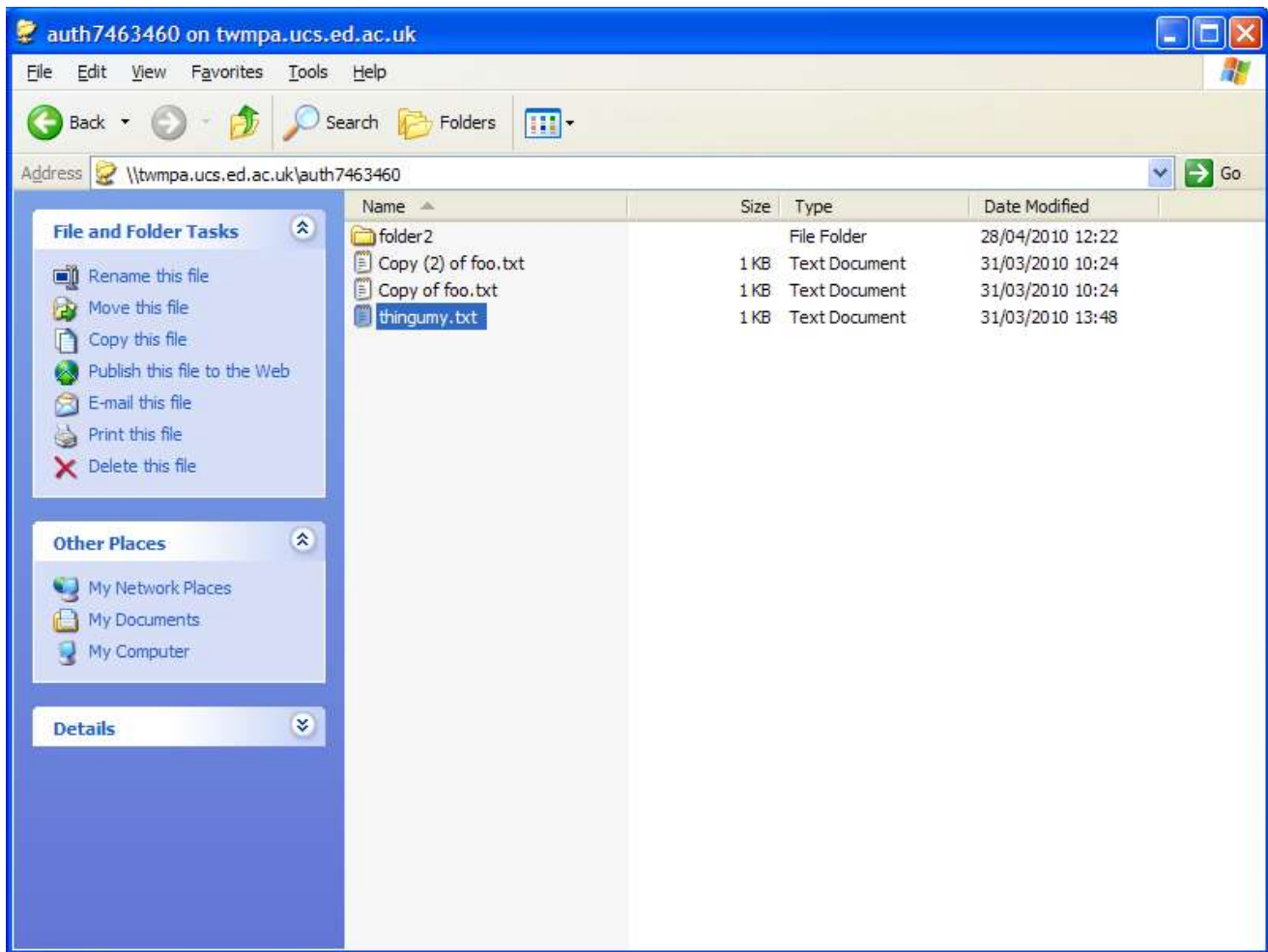
```
7463460 /pup
```

5 Connecting a Windows XP Client

Armed with the link created by the Shibboleth-protected authorisation script, the user can now connect their client to the DAV server through the façade. The first client we tried was Windows XP. Its *My Network Places* window offers the ability to *Add a network place* (in the left-hand *Network Tasks* pane, assuming the *Folders* view is not turned on). Clicking on this invokes the *Add Network Place* wizard. Clicking *Next* brings up a screen asking *Where do you want to create this network place?* One of the choices, or the only choice if no commercial web storage providers are already configured on the system, is *Choose another network location. Specify the address of a web site, network location or FTP site.* Selecting this and clicking *Next* proceeds to a dialogue allowing the user to enter an *Internet or network address*, in our case the URL copied from the authorisation script:



Clicking *Next* will, if everything succeeds, move on to a dialogue allowing the user to *Type a name for this network place*. Accepting the default name derived from the URL is sufficient. The final dialogue should then appear, offering the option (selected by default) to *Open this network place when I click Finish*. Doing so and clicking on *Finish* then brings up an Explorer window showing the files in the DAV directory:



These files can be opened, copied, deleted and so on, just as if they were local files.

6 Other Clients

This initial success with Windows XP was sufficiently promising for a colleague to be persuaded to attempt the same exercise using Windows Vista. There the results were not so encouraging though.

The first hurdle was some superficial GUI differences. The web folders feature is now accessed by right-clicking on *Computer* and selecting *Add network location* rather than through *My Network Places*². From there, the procedure is much the same as for XP. Some additional confusion was caused by discovering that there is now also an alternative way to access a WebDAV folder in Vista, by mapping a network drive.³

Having overcome these minor initial obstacles, hopes were high but disappointment then ensued. On attempting to connect to the test DAV server via the façade, the Vista client refused to connect, with an error message that gave little away as to the cause of the problem. Changing firewall settings, folder locations and so on made no difference.

We decided at this point to see if the same issue would affect other clients that might be more forthcoming about the cause of the problem. The next client tried was the *cadaver* command-line WebDAV client mentioned previously. It also refused to connect, reporting that the supplied URL was not a collection resource (a DAV term analogous to folder or directory):

² <http://www.webdavsystem.com/server/access/windows>

³ http://www.webdavsystem.com/server/access/map_drive

```
[fc@localhost ~]$ cadaver http://facade/auth7463460
Could not access /auth7463460/ (not WebDAV-enabled?):
Did not find a collection resource.
Connection to `facade' closed.
dav: !> quit
```

Some perusal of man pages revealed the presence of a useful-sounding command-line option, `-t` (tolerant), for use “if the server or proxy server has WebDAV compliance problems”. This sounded ideal and did indeed side-step our problem:

```
[fc@localhost ~]$ cadaver -t http://facade/auth7463460
Ignored error: /auth7463460/ not WebDAV-enabled:
Did not find a collection resource.
dav:/auth7463460/? ls
Listing collection `/auth7463460/': succeeded.
Coll:  folder2                0  Apr 28 12:22
      Copy (2) of foo.txt      21  Mar 31 11:04
      Copy of foo.txt         21  Mar 31 10:28
      thingumy.txt            49  Mar 31 13:48
dav:/auth7463460/? close
Connection to `facade' closed.
dav: !> quit
```

The Windows XP client is clearly sufficiently tolerant by default not to complain about this problem.

To investigate the cause of the error, we made use of some of `cadaver`'s debugging features, capturing the protocol exchanges by redirecting file descriptor 2 (the standard error stream) to a file:

```
[fc@localhost ~]$ cadaver 2>proxy.txt
dav: !> set debug http,httpbody
dav: !> open http://facade/auth7463460
. . . same error message as above . . .
dav: !> quit
[fc@localhost ~]$
```

This exercise was repeated with the client connecting directly to the underlying DAV server, side-stepping the façade (after suitable firewall reconfiguration). When the two debug logs were then manually compared side by side, the likely cause of the problem was quickly apparent: an XML message returned by the DAV server contained an explicit reference to the name of the collection as known to the server (`uploads`). This would of course differ from the name known to the client (`auth7463460`):

```
Read block (687 bytes):
[<?xml version="1.0" encoding="utf-8"?>
<D:multistatus xmlns:D="DAV:" xmlns:ns1="http://apache.org/dav/props/"
  xmlns:ns0="DAV:">
<D:response xmlns:lp1="DAV:" xmlns:lp2="http://apache.org/dav/props/"
  xmlns:g0="DAV:" xmlns:g1="http://apache.org/dav/props/">
<D:href>/uploads</D:href>
. . .
```

The underlying issue here is the same as that discussed in technical note 1, which affects OGC web services' `GetCapabilities` requests: if the actual XML messages exchanged with a web service contain explicit references to URLs (or here, partial paths) on the underlying server then, to maintain the client's illusion that it is communicating directly with the underlying service, the façade must rewrite those URLs or paths to refer to the façade server. The more places in the application protocol where such fragilities appear, the more the messages passing through the façade will require application-specific modification rather than simple, application-independent HTTP proxying.

At the start of this investigation, the author was not familiar with the WebDav protocol (otherwise this particular problem would have been apparent at the outset) but it seemed worthwhile to persevere and attempt to apply the same solution as in the OGC case, by inserting a text filter into the output stream of the proxy part of the façade. That requires some additional Apache configuration at the façade's web server:

Federated Access to an HTTP Web Service Using Apache

```
ExtFilterDefine munge2 cmd="/usr/bin/perl /etc/httpd/conf/davRewrite.pl"  
<Proxy http://davhost>  
    SetOutputFilter munge2  
</Proxy>
```

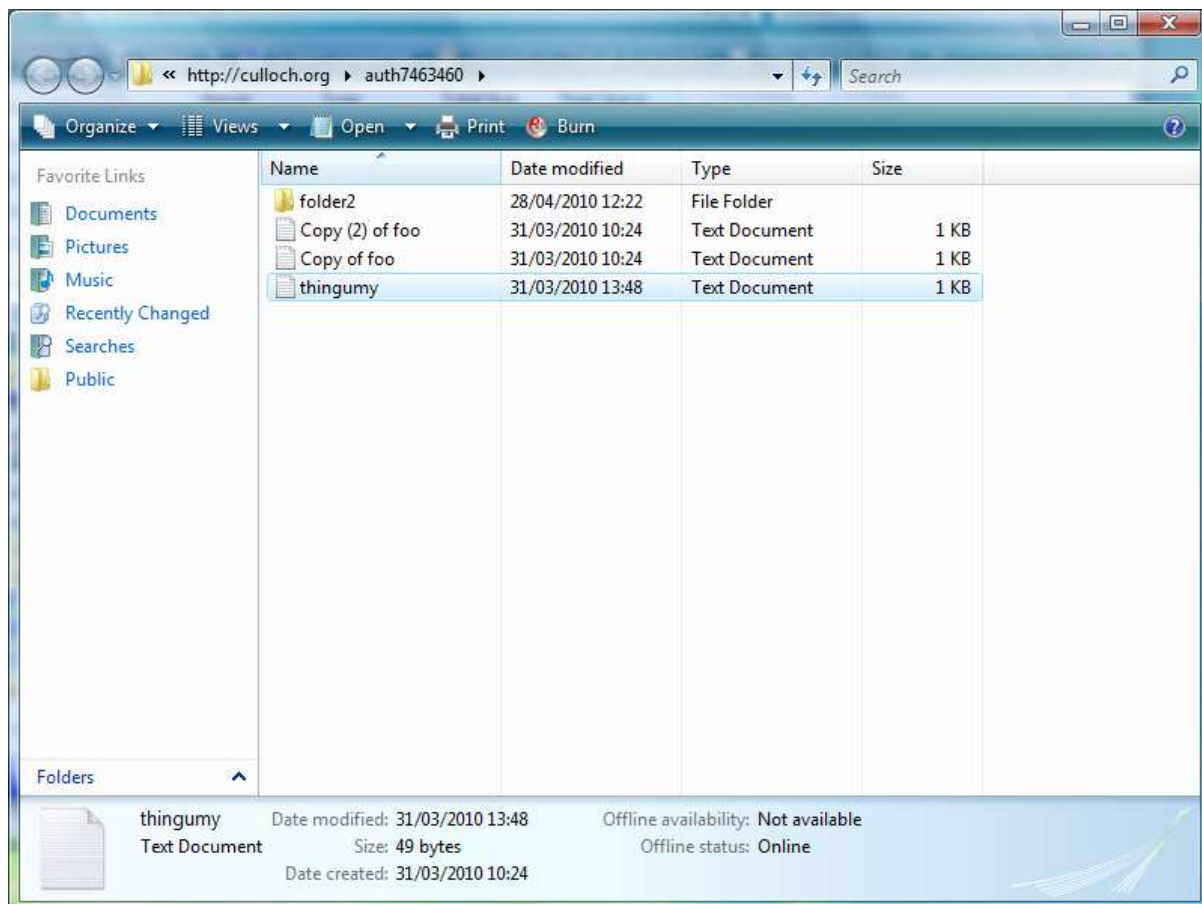
The `<Proxy>` block applies the external command defined by `munge2` as a text filter on output from the underlying DAV server (`davhost`) before the proxy copies it back to the requesting client. The command used invokes a `perl` interpreter on a simple script, `davRewrite.pl`:

```
$firsttime = 1;  
while (<>) {  
    if (/uploads/ && $firsttime) {  
        s|uploads|auth$ENV{SESSION}|g;  
        $firsttime = 0;  
    }  
    print;  
}
```

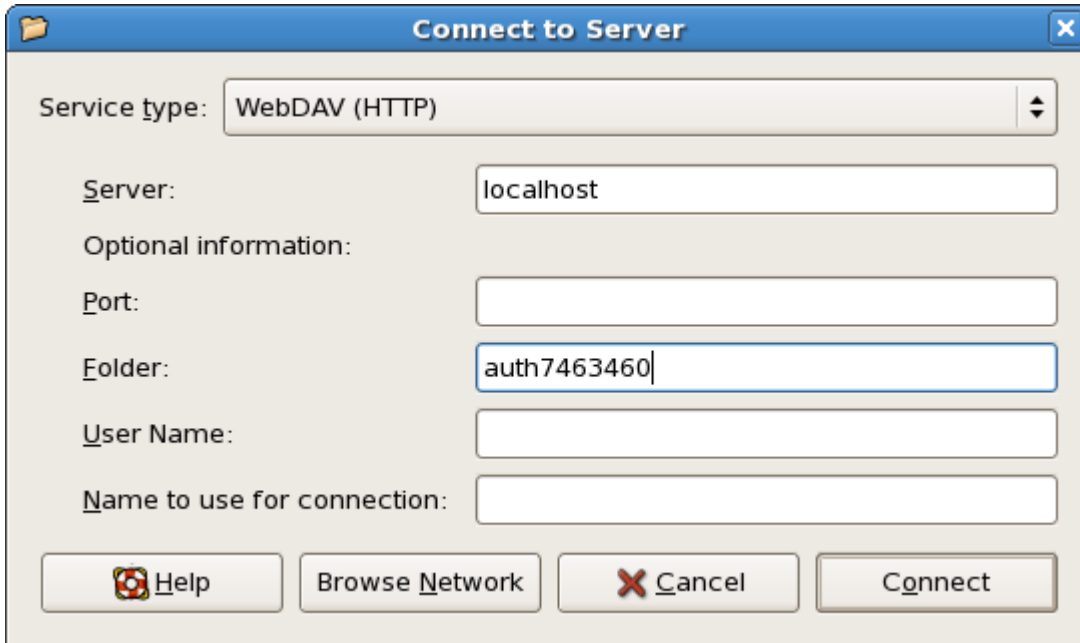
This substitutes the name by which the client knows the collection (`authNNNN`) for the name known by the DAV server itself (`uploads`), but cautiously, at the first instance only.

Originally the façade was hosted on the same system as the DAV server, so the `<Proxy>` block referred to `http://localhost`. This configuration did not behave as expected; the filter appeared to be ignored. Rather than following this side track further, we simply moved the façade configuration to an Apache setup on a different system. The filter configuration then behaved as expected.

With these extra modifications the Vista web folder client did then finally connect to our test DAV server via the façade:



On Linux, we now retried the GNOME desktop's Nautilus file browser running under Red Hat (CentOS), which had been equally unwilling to access the DAV server via the façade up to this point. The connection shown below is to localhost because the Apache hosting the façade was on this CentOS system:



The same changes that had been made to accommodate Vista also satisfied the Linux setup:



7 Creating Directories

At this point, a further flaw became apparent. Viewing directories and opening files worked well but any attempt to create new directories failed. This behaviour was initially observed under Vista but affected the other platforms too. On Vista, the system would create the new directory as usual with the placeholder name of “New Directory”. The user interface selects and highlights this name, inviting the user to replace it with the directory name actually desired. On doing this and hitting Enter though, an error was provoked. Investigation, again using *cadaver*'s debug logging features, led to the realisation that during file moves, including this implicit move to rename the directory from “New Directory”, the WebDAV protocol relies on a special HTTP header

("Destination:") containing the name of the collection (directory). Yet again, this name was being given from the client's point of view (auth7463460 in the example above) while the underlying DAV server was expecting to see the name by which it knew the collection (uploads).

Because the incorrect name was being supplied in a header rather than in the HTTP body, it could not be modified by the Apache output filter already used previously. As it happens, Apache also provides a way of modifying request headers that match a given pattern, and this was attempted:

```
RequestHeader edit Destination \/auth[0-9]*\/ /uploads/
```

This had the required effect but unfortunately did not completely resolve the problem. Further investigation revealed a similar issue in the other direction, in a response header ("Location:") sent back to the client. At this point we ran out of options that would obviously enable Apache to deal with the situation (such options may exist; they were simply not obvious to us). Response headers can be matched against a pattern but the substitution string, at least in the version we were looking at (2.2.14), is just a simple string and does not allow for expansions of environmental variables, specifically the session key (SESSION). This limited facility was applied as an experiment only, using an actual session key instead of a general environmental variable, to see whether this was indeed the final hurdle:

```
Header edit Location /uploads/ /auth7463460/  
Header edit Location http://davhost/ /
```

By this point the reader will probably not be surprised to learn that although this allowed *cadaver* to progress further, it did not solve the problem for Vista.

8 Conclusion

At this point, rather than pressing forward on an increasingly unpromising-looking road, with apparently little prospect of producing something that would be applicable to a general user base, it was decided instead to step back and simply write up the experience gained (as this document). Although users in general would not accept the limitations of the technique as it stands (it would be very hard to explain why some perfectly normal operations like creating a directory did not work as expected when others did), it might still be useful in more constrained circumstances. For example, the easiest way to get the Vista screenshot shown earlier off the test machine turned out to be to paste it into a document there and copy that document into the test folder!

The first point that should be apparent from this story is (*mea culpa*) that when considering interposing an authorisation façade between a web service and its clients, it would be wise to have a full understanding of the web service application protocol in question. This seems obvious in hindsight but at the outset it simply did not occur to us that an XML-based protocol would need to resort to planting additional information in special headers.

The second point is that, although there should be no compelling reason for a web service application protocol to transport information about the actual URL at which the service is located as part of the protocol, both of the concrete examples we have looked at in detail do in fact do this:

- the OGC web services discussed in technical note 1 send service endpoint URLs as part of the `GetCapabilities` response message;
- the path information sent at various places in the WebDAV protocol includes the root of the path, which makes it dependent on the absolute location (authNNNN vs. uploads) even before the special headers are considered.

This calls into question, although it does not of itself rule out, the general applicability of the façade approach, at least without some customisation for new web services.