Web File System: File-like Access to the Web

Atul Adya Joseph Bank Jim Napier Jordan Slott H.B. Weinberg *

1 Introduction

The Web File System (WFS) is a file system interface to the World Wide Web This interface helps leverage existing investment in file system based tools by allowing them to access the Web as if it were a file system. Because it is implemented as an NFS server, WFS works without modification to standard file system clients or Web servers. We do not propose the file system interface as an alternative to the existing graphical interfaces; these will remain the preferred method of access for humans. Instead, WFS speaks to a disenfranchised community — our old friends like `find`, `grep`, `csh` and `perl` that until now were largely barred from the Web. We also believe that WFS will make it easier to develop new Web savvy applications by shielding programmers from the details of Web protocols.

2 Interface

What is a file system interface to the Web? At a high level we view it as a mapping from Web objects to file system objects and vice-versa. The Web objects we consider are URLs, documents, and hypertext links. These are mapped to the following file system objects: directories, files, and symbolic links, respectively. The mapping is best illustrated by an example.

Consider the following hypothetical web objects: the URL `http://mit.edu/wfs.html` which refers to an HTML document which contains hypertext links to `http://mit.edu`, and `picture.gif`. The corresponding file system objects are summarized in Figure 1 below. First, note that we can safely use the "%%" sequence because Web standards dictate that no URL is permitted to contain it. Second, notice that the symbolic links point to directories. These directories will have a similar structure to `wfs.html`. For example, `/wfs/http:/mit.edu/picture.gif` is a directory which contains one entry, a file called `%%GIF`.

We discuss only two of the many factors that influenced the design of this mapping. The first issue is the different nature of Web objects and file system objects. Many Web objects are rich — HTML contains both text and links. By contrast, the traditional UNIX file system abstraction provides pure objects: flat data (files), containers (directories), and pointers (symbolic links). Early on, we decided that we could best represent HTML documents by a container object with data and pointers inside — i.e. a directory containing a file and symbolic links. (For largely aesthetic reasons, we chose to embellish this design by grouping the symbolic links in a subdirectory of their own; hence the directory "`%%Links`".)

Another influence on our design was the different naming systems. URLs name Web objects; pathnames denote file system objects. The semantics of URLs is weak. Each Web server can define an arbitrary mapping between well-formed URLs and the data it chooses to return. We call this the "blindness" of the Web. This has forced us to model even the *non-HTML* files as directories. For example the URL `http://mit.edu/a` might return a Postscript file. Meanwhile some *extension* of that URL, say `http://mit.edu/a/b`, might return an HTML document. By contrast, in a file system, a pathname has a meaningful extension if and only if it names a directory.

3 Implementation

Our aim is to provide a transparent system — one that looks like a file server to its clients and like a Web client to the Web servers. Our approach is to implement WFS as a user-level network file server using the NFS protocol [4]. This implementation strategy is similar to that used in other systems that provide file-like access to systems that are not exactly file systems, *e.g.,* Alex [1](ftp access), OdeFs [2] (access to an object-oriented database) and Semantic File System [3]. The advantages of using the NFS protocol are that many machines already use it, no modifications to the client's kernel are required, and multiple client machines can connect to one NFS server.

`/wfs/http:/mit.edu/wfs.html/`	a directory — abbreviated `.../wfs.html` below
`.../wfs.html/%%HTML`	the raw HTML file
`.../wfs.html/%%Links/`	a directory containing the links from `wfs.html`
`.../wfs.html/%%Links/The_MIT_Page`	a symbolic link to `/wfs/http:/mit.edu` "The MIT Page" is the text of the hypertext link
`.../wfs.html/%%Links/A_Picture`	a symbolic link to `../../picture.gif` "A Picture" is the text of the hypertext link

Figure 1: Example WFS objects

*Supported by an NSF Graduate Fellowship.

The architecture for the WFS daemon is shown in Figure 2. WFS consists of two modules: the Virtual File Module (VFM) and the Web Protocol Module (WPM). The VFM receives client requests via the RPC-based NFS protocol. Our current approach is to allow read-only access to WFS, so only 8 of the 18 calls in the NFS protocol are implemented. Within the VFM, meta-data (the directory structure and links) is managed separately from the underlying data (the file contents) because the two require different caching strategies. The other module, WPM, handles the interface to the Web. Submodules of WPM either communicate with servers or decompose rich documents. The current implementation of WFS communicates only with HTTP servers and parses only HTML documents (non-HTML documents are passed through as simple files.)

Access to Web servers is substantially slower than access to traditional local file servers. If WFS accessed the Web on every NFS request, clients would experience intolerable delays. We addressed this problem in two ways: WFS procrastinates and it caches. We discuss each of these in turn. The prelude to any NFS operation on an object is a `lookup`. In a `lookup` request, the client supplies a name and the server returns a *handle* and type information for the requested object. Clients tend to perform `lookup` liberally. We said that WFS procrastinates because it does not access the Web during most `lookup` requests. When a `lookup` occurs on a name that would correspond to a valid URL, WFS blindly returns a handle to a directory *without accessing the Web.* Of course, certain subsequent operations on that handle may force an access to the Web. However, very often such an operation never occurs and our procrastination pays off.

As for caching — once forced to access the Web, WFS doesn't discard the results. Meta-data is cached in memory and content data is cached on disk. We discuss only one of the many issues involved in caching: the consistency of the content data. WFS borrows a cache consistency heuristic from Alex [1]: if a file appears to a client to have been last modified at time x and the current time is y, then the file contains data that was current at least $(y-x)/10$ time ago. For example, a file that appears to have been last modified 12 hours ago contains stale data only if it was modified within the last 1.2 hours. This policy is based on the observation that older files are less likely to be modified.

4 Conclusions

There is much work still to be done on WFS: Gopher and other Web protocols, non-read-only NFS calls, improved caching strategies, multi-threading, and security. Currently, we are focused on broadening our user community so that their feedback can guide us. Our experience so far is that WFS succeeds at our initial goal of bringing the Web to the disenfranchised, namely the familiar UNIX tools. Furthermore, we believe that WFS will ease the burden of developing new automated Web applications.

In developing WFS, we encountered numerous problems with the semantics of the Web, only some of which we have mentioned here. In growing and improving the Web, the Web community is working toward finding the correct design for a large information sharing system. We believe that WFS is an important reference point in this design space.

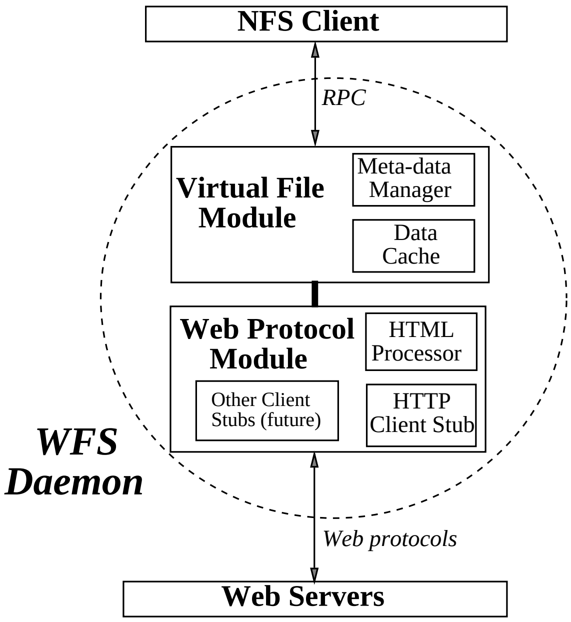

Figure 2: WFS Architecture

References

[1] V. Cate *Alex - a Global File System.* Usenix Technical Conference, 1992.

[2] N. Gehani, H. Jagadish, W. Roome *OdeFs: A File System Interface to an Object-Oriented Database.* 20th VLDB, 1994.

[3] D. Gifford, P. Jouevlot, M. Sheldon, J. O'Toole *Semantic File Systems.* In SOSP 1991.

[4] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, B. Lyon *Design and Implementation of the Sun Network File System.* Usenix Technical Conference, 1986.