The Prospero File System
A Global File System Based on the Virtual System Model

B. Clifford Neuman

Information Sciences Institute
University of Southern California

Distributed file systems have come into widespread use in recent years. Many allow files to be accessed over large geographic areas and across organizational boundaries. However, few systems to date have given much thought to how information should be organized in such a global environment.

This paper describes the Prospero File System, a file system based on the Virtual System Model, a model for building large systems within which users construct their own virtual systems by selecting and organizing the objects and services of interest. This customized view of a global file system makes it easier for users to keep track of files that they have identified as being of interest. The use of multiple name spaces can cause confusion. Such confusion is eliminated by support for closure: every object has an associated name space, and names specified by the object are resolved in that name space.

Tools are provided to allow views to be kept up-to-date, and to allow views to be defined as functions of other (possibly changing) views. These tools promote sharing and enable the organization of files in ways that make it easier to identify information of interest than it is in existing systems.

The prototype implementation has been used to organize information available from Internet archive sites; its directory service has been used from more than 7,500 systems in 29 countries. This paper discusses the goals of the Prospero File System, describes the prototype implementation, and discusses experience with the use of the system to date.

1 Introduction

Much attention has been paid to distributed file systems in recent years. Many of these systems allow files to be accessed over large geographic areas and across organizational boundaries. To date, however, most of the work on such systems has concentrated on access mechanisms, and less attention has been paid to the problems such environments present for the organization of information.

The Internet contains a massive amount of information, but it is hard to use that information. There are several barriers to usability: it is difficult to identify the information of interest; it is difficult to keep track of this information once found; it is difficult to share information about what is available, or to collaboratively maintain such meta-information; and the information is often scattered across multiple file systems of different types, meaning that different mechanisms are needed to access different information.

This research began as the author's dissertation at the University of Washington. It has been supported in part by the National Science Foundation (Grant No. CCR-8619663), the Washington Technology Center, Digital Equipment Corporation, and the Defense Advance Research Projects Agency under NASA Cooperative Agreement NCC-2-539. The views and conclusions contained in this article are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any of the funding agencies. The author may be reached at USC/ISI, 4676 Admiralty Way, Marina del Rey, CA 90292-6695, USA. Telephone +1 (310) 822-1511, email bcn@isi.edu.

To Appear: Workshop on File Systems, May 1992.

This paper describes the Prospero[1] File System. Prospero is based on the Virtual System Model, a model for building large systems that allows users to organize the information and services available to them. Prospero supports customized views of a global file system, making it easier for users to keep track of files that they have identified as being of interest. In traditional systems, a customized name space would cause confusion since the same name might refer to different objects at different times. Prospero avoids this problem by supporting closure: every object has an associated name space, and names specified by the object are resolved in that name space.

Tools are provided to allow users to keep their name spaces up-to-date. These tools allow views to be defined as functions of other, possibly changing, views and improve the user's ability to organize information, making it easier to identify information of interest than it is in existing file systems.

The Prospero File System is heterogeneous; instead of providing its own methods for storing and accessing files, it relies in existing file systems for storage and supports multiple underlying access methods. Prospero is implemented as a distributed directory service that names individual files, plus a file system interface that calls the appropriate access method once a name has been resolved. The prototype supports Sun's Network File System, the Andrew File System, and the File Transfer Protocol (FTP). For FTP, the file is automatically retrieved and the locally-cached copy is then opened.

A prototype is running and has been used from more than 7,500 systems in 29 countries on six continents. Experience with this prototype has shown that the organizational flexibility provided by the Virtual System Model is useful. Initial observations have provided insight into the way that users organize and look for information when they are not constrained to use a single, monolithic name space.

Our discussion begins by describing existing distributed file system approaches, highlighting the advantages and disadvantages of each. Section 3 describes the Virtual System Model, a model for organizing large systems that allows users to organize available information and services as they see fit. Prospero is an operating system based on that model. The file system prototype is discussed in Section 4 and performance figures are provided. Section 5 discusses experience with Prospero and describes some of the ways in which it has been used to organize information on the Internet. Related work is presented in Section 6 and future plans in Section 7. Section 8 summarizes the material presented in this paper and draws conclusions.

2 Existing Systems

The naming of files in existing distributed systems falls into four categories: host-based naming, global naming, user-centered naming, and query-based naming. This section will describe systems that fall into the first three categories, and will discuss the advantages and disadvantages of each. Systems falling into the fourth category are recent and are described as related work in Section 6.

[1] From the *Tempest* by William Shakespeare. Prospero was the rightful Duke of Milan who escaped to a desert island. When his enemies were shipwrecked on the island, Prospero used his power of illusion to separate the party into groups, each of which thought they were the only survivors. Thus, he caused each group to see a different view of the world. As time went on, the shipwrecked parties slowly learned about the others, and thus, the pieces of the other views were added to their own.

2.1 Host-Based naming

Early distributed file systems employed host-based naming to identify objects (files or directories). Examples of host-based naming include FTP [17] and Sun's Network File System[2] [20]. In host-based naming, the user must know the name of the host on which an object resides in order to access it. While relatively simple to implement, host-based naming makes it difficult to organize and locate information: the first part of a file name (the host) usually has little or no relation to the topic; logically related information stored on different systems is scattered across the name space; and as these systems are implemented, it is difficult to add links that cross system boundaries (a link is a reference from a directory to an object).

A more recent system, Alex [2], addresses the latter problem by allowing files available by FTP to be named and manipulated using the syntax of local file names. This allows users to create symbolic links to files on remote systems. Unfortunately, if the name of the target of such a link changes, the link will no longer work.

Because of these problems, many users make local copies of information that they have found on the Internet out of fear that it might move, or that they might forget where it is. Often the information is not used locally, it is copied "just in case" it is later needed. Others maintain huge lists of the information available and periodically distribute the lists through electronic mail.

2.2 Global Naming

An alternative to host-based naming employed in a number of recent systems is global naming. The Andrew File System [7], Locus [24], and Sprite [13] are among the systems that employ this approach. In global naming, all names are part of a single name space, and the name of the system on which an object resides is not explicitly part of the object's name. As these systems are implemented, however, objects with similar names must usually be stored on the same system.

A global name space solves some of the problems encountered by the host-based approach. In particular, the name of the storage site is no longer part of an object's name. It is also possible to add links to objects on other systems, though as implemented, these links are symbolic: they return a new name that must be further resolved. This means that if the name of the target of such a link changes, the link will no longer work.

Unfortunately, a global name space runs into problems as a system scales, especially once the system spans administrative boundaries. Organizations, and even users, want control over a piece of the name space. This results in a name space whose top levels are often the names of organizations, and whose second levels are the names of users. Such an organization often results in logically related information being scattered across the name space.

The alternative is to organize information by topic, rather than according to the administrative structure of the system. The difficulty with this approach is that, in a large system, there will be disagreement on what topics should appear near the top of the tree, and once topics are agreed upon, there will be disagreement on which files should be included under each topic. This problem is apparent on Usenet, a worldwide distributed message service for disseminating messages on many topics. A significant share of the messages sent on Usenet

[2] In NFS, the user must specify the host on which the file resides when mounting the file system.

discuss what messages are appropriate for particular newsgroups, whether new newsgroups should be created, and what they should be called. This clearly demonstrates the problem of reaching consensus on globally shared names.

2.3 User-Centered Naming

One of the problems with large systems is that there is a huge amount of information, and much of that information is not of interest to a particular user. User-centered naming attempts to address this problem by allowing each user to choose what is to be included in his or her name space. User-centered naming is employed by Tilde [3], QuickSilver [1], Plan 9 [18], Prospero [11], Jade [15], and some object-based systems such as Amoeba [23].

The customization supported by these system is important for a number of reasons: it reduces clutter in the user's name space; it allows users to define shorter names for frequently referenced objects; and it allows users to replace entire portions of the name space with alternative views that are more appropriate for the particular user, for example, with references to nearby instances of files instead of remote ones.

The systems with global name spaces (Section 2.2) support customization by allowing users to add symbolic links that assign short names to the files that they frequently access, but those systems allow users to assign only names that appear in their own sub-hierarchies of the file system. This is not a sufficient level of customization. Users should be able to customize all parts of their name spaces. Without this ability, users must remember when they have created customized views of parts of the name space that they are not allowed to customize directly.

User-centered naming presents several problems of its own. The lack of name transparency has the potential to make sharing difficult and to cause confusion. The problem is that the same name might refer to different objects when used in different name spaces.

Another problem is that in many user-centered systems, an object, or collection of objects, must first be added to the name space before it can be accessed[3]. Adding an object often requires that the user specify a globally unique name for the object, reintroducing all of the problems associated with the global naming approach.

A final problem is that, with the exception of Prospero, systems supporting user-centered naming do not provide adequate tools for constructing derivative views as functions of existing views. In Tilde and Plan 9 part of the problem is that views are not persistent. Instead, they are constructed by a process (often using a configuration file) and they only live as long as the processes that use them.

The problems just described are addressed by Prospero and the Virtual System Model.

3 The Virtual System Model

The Virtual System Model [12] provides a framework for organizing large systems within which users construct their own *virtual systems* by selecting objects and services that are available over the network; users then treat the selected resources as a single system, ignoring those resources that were not selected. By supporting a customized view of the system, information of interest to a user is prominently located near the center of the user's

[3]Naming in these systems might be better described as user-exclusive since objects that have not been explicitly included in the name space can not be accessed.

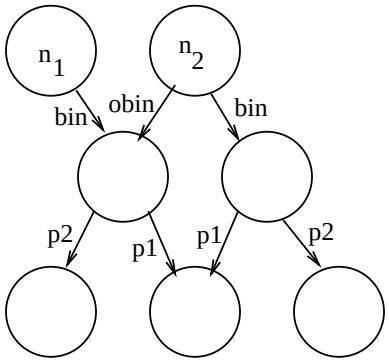

Figure 1: Two name spaces

name space, while information that is not of interest is kept out of the way. Different views also provide support for locality since users can customize what they see, replacing remote references with local ones. The Virtual System Model allows users to define views of information as functions of one or more other views. The derivative view automatically reflects any changes that occur in the views upon which it is based.

Users are able to organize the objects and information about which they know in multiple ways. The process of object discovery is facilitated by these multiple organizations, and by the fact that the information specified by users, when customizing their own name space, can be combined with other information and made available for use by others.

As indicated earlier, the fact that the same name may refer to different objects at different times can make a user-centered name space confusing and can hinder sharing. In the Virtual System Model, every object has an associated name space, forming a closure [10, 19]. In this way, the context within which a name is to be resolved can be automatically determined based on the object specifying the name. Although the same name may refer to different objects within different contexts, the correct context is always known.

3.1 Multiple Views of a Global Name Space

Within the Virtual System Model, the global naming network forms a generalized directed graph. Internal nodes in the graph represent physical directories and leaves correspond to files. The value of a directory is a collection of links, each mapping a single component of a name to a file or a directory. Each link in a physical directory is represented by a labeled edge in the graph from the node representing the physical directory to another node. Each link may have an associated function, called a filter, which when applied to the value of a directory yields a virtual directory (which defines a new view of a directory). Like a physical directory, the value of a virtual directory is a collection of links.

The Virtual System Model supports a user-centered, or more precisely, an object-centered name space. Each name space is a view of the global naming network, selected by choosing a starting node from the graph. We call the starting node the *root* of a virtual system.

Figure 1 shows a simple naming network where *n1* is the root of a name space that names two programs, */bin/p1* and */bin/p2*. A second name space, rooted at *n2*, has its own *bin* directory in which *p2* refers to a different file; the directory */bin* from *n1* has been renamed to *obin*.

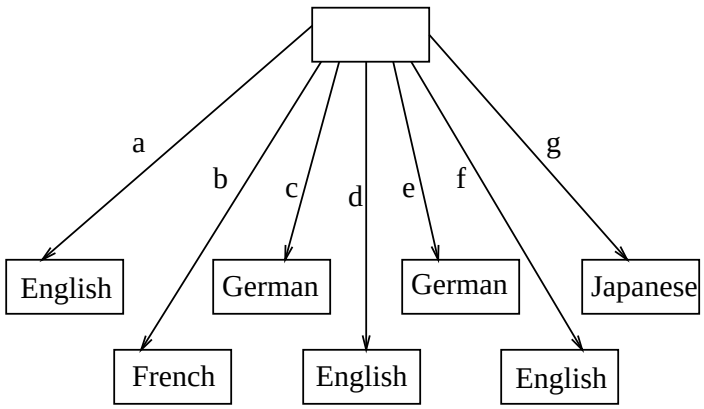

Figure 2: Directory before application of a filter

Most objects start as part of a user's name space, but with long names. As users identify information of interest, that information is moved closer to the center of the name space by adding additional links. These links are added either to a physical directory, in which case other users with views of the physical directory will see the change, or they are added to a view of a physical directory, in which case the change is visible only to other users sharing the view.

3.2 Filters and Union Links

The Virtual System Model supports customization by allowing a virtual directory to be specified as a function of other directories. This is made possible by the filter. A filter is a program, attached to a link, that allows the view of the target directory to be altered. For example, in Figure 2 files are named with the labels *a* though *g*. Associated with each file is an attribute list, one attribute of which is the language in which the text was written. The value of the language attribute is shown in the box representing the file. By attaching the **distribute()** filter to the directory link, a derived view is created within which the files appear to be distributed across subdirectories according to the value of the language attribute. The derived view is shown in Figure 3.

A filter takes the value of a directory as an argument and returns a new directory. By composing a filter with another filter already associated with a virtual directory a view can be specified as a function of another view.

Because filters are associated with links, and because the result of applying a filter is a list of links (the value of a directory) a filter can attach additional filters to the links it returns. This allows a filter to modify more than one level of the hierarchy to which it is attached. It also allows the creation of *ghost* hierarchies, parts of the name space which are specified entirely within the filter.

As described so far, views are not shared, but physical directories are. Each view of a physical directory is distinct and if a change is made to a filter that maintains a view, that change will not be visible through other views. There are cases, however, when it is desirable to share a view. For a view to be shared, the filter that implements it must appear on a link leading out of a physical directory. Unfortunately, the label on the link also names the view. If we want to share a view, but not the name of the view, we must support links that are not named.

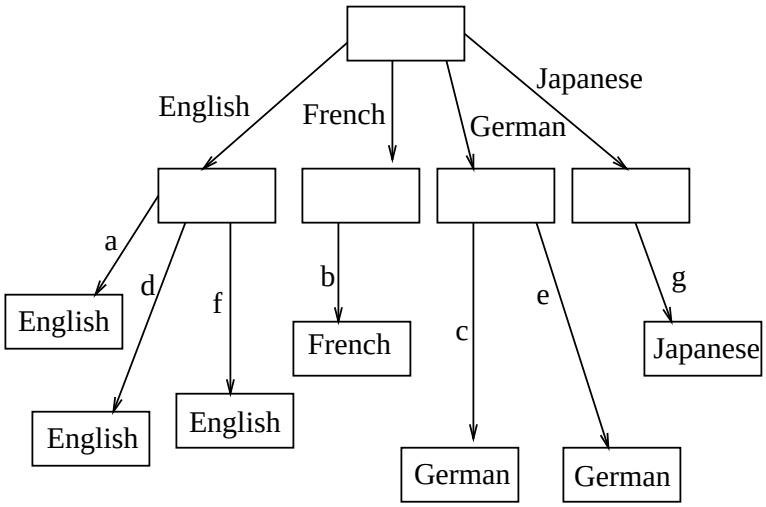

Figure 3: Directory with distribute() applied

The union link is such a link. The target of a union link is another directory. After any filters associated with a union link are applied, the result is merged with the contents of the physical directory containing the union link. Conceptually, the union link is an epsilon transition in the global naming network. Such a link is called a union link because the resulting directory appears to contain mappings that are the union of the normal links and the links from each of the directories included through union links.

The use of a union link on a physical directory containing any other links might result in more than one mapping for the same name. When resolving a name, each of the mappings must be tried[4]. As implemented, however, the union link is combined with a filter that will only pass a single mapping for each name. If an included directory contains a mapping that conflicts with one that exists in the originating directory, or from an earlier union link, then the conflicting mapping is returned separately or ignored.

Filters and union links provide a powerful mechanism for supporting customization and the manipulation of name spaces. Filters are written in standard programming languages and can take any action that can be specified in such languages. The union link allows the manipulation performed by a filter to affect the directory containing the filter.

4 The Prospero File System

The Prospero file system applies the Virtual System Model to the design of a global file system. In Prospero, files that are logically related can be grouped together, even if scattered across multiple systems. Prospero supports multiple views of the global file system and views may be defined as functions of one or more other views. A prototype of the file system has been constructed and is in use across the Internet.

4.1 Implementation

The names of files in Prospero are resolved by contacting directory servers on the hosts that store Prospero directories. The server accepts the system level name of a directory and optionally the name of the link to be returned. The server returns the links in the directory that match the specified name, or all links if the name was not specified. Attributes are

[4]This is similar to the way search paths work.

associated with objects and the directory server responds to requests for specific attributes associated with an object.

A Prospero link specifies the name of the host that stores an object, and the system level name of the object on that host. The link also specifies other information including whether the link is a union-link, and any filters associated with the link. If the target of the link is a directory, the link provides the information needed to resolve a name in that directory by contacting the directory server on the host specified by the link. To access an object that is not a directory, a request is made to the directory server for the value of the ACCESS-METHOD attribute. The response includes a list of acceptable access methods, and the information needed to access the object using each. Prospero presently supports the NFS, AFS, FTP, and local access methods.

The structure of the Prospero directory server is similar to that in capability based systems, such as Amoeba [23] in that the directory server has no idea how its directories fit into the name space. Each directory is a separate object that may be referenced by many other directories. Cycles are even allowed. The Prospero directory service is not capability-based in that the links (object handles) do not grant authorization to access the object. Additionally, links in Prospero contain information about the storage site for the object whereas in most capability based systems they provide a unique ID which must be located using a broadcast mechanism.

The Prospero client (application) remembers the system-level name (the host plus the name of the directory on that host) for the current working directory and for the root of the active virtual system. When the user or an application wishes to resolve a name, the first component is resolved relative to the appropriate directory by sending a query to the corresponding directory server. The next component is resolved by sending a query to the directory server named in the link returned by the first query. This process is repeated until all components of the name have been resolved[5]. If the named file is to be accessed, an additional query is made to obtain the access method.

Communication with the directory server is accomplished using a reliable datagram protocol implemented on top of UDP [16]. This reduces the overhead that would otherwise be incurred when establishing connections to multiple directory servers.

At any point in the resolution of a name, the directory server might return one or more union links. Such a response indicates that the directory has not been completely searched, and that the current component of the name should be resolved in the directories named in the union links[6].

If a filter is associated with a link, the filter is applied to the result of the directory query before the current component of the name is looked up[7]. A filter can remove links from or add links to a directory, change the names of links, or even change the way a directory hierarchy appears to be organized (e.g., creating subdirectories). Filters are written in C and are dynamically linked with the name resolver when they are applied.

[5] An optimization allows a directory server to resolve more than one component of the name at a time as long as all intervening directories are stored on the same server.

[6] If the directory is being listed, then the results of querying the union linked directories are merged with the rest of response from the directory that returned the union links.

[7] For this to work, the directory server must be instructed to return all links in the directory, not just those matching the component.

```
VDIR filter(dir,ip,argc,argv)
  {
    vdir_init(nd);
    sd = vlcopy(dirlink,0);
    avf = rd_vlink("/lib/filters/avalue.o");

    /* Step through attribute values creating subdirs */
    cl = dir->links;
    while(cl) {
      attributes = pget_at(cl,argv[0]);
      for(ca = attributes;ca;ca = ca->next) {
        /* If not first link, then make copy */
        if(nd->links) sd = vlcopy(nd->links,0);

        /* Set name of new subdir and insert it */
        sd->name = stcopyr(ca->value,sd->name);
        if(vl_insert(sd,nd) == PSUCCESS) {
          /* If successful, then set filter arguments */
          /* Find last filter on current subdir       */
          for(avf=sd->filters;avf->next;avf=avf->next);
          sprintf(farg,"%s %s",ca->aname,ca->value);
          avf->args = stcopyr(farg,avf->args);
        }
      }
      atlfree(attributes);
      cl = cl->next;
    }
    /* Return the result in the original directory*/
    vdir_copy(nd,dir);
    return(dir);
  }
```

Figure 4: Simplified code for the distribute filter

Figure 4 shows code implementing the **distribute()** filter. The **distribute()** filter works by reading the value of the specified attribute for each file in the target directory, and creating a new link to the target directory for each distinct value. The name of the new link is the value of the attribute, and a filter is attached that selects only those files whose attribute matches that value.

Although users can write their own filters, most users can get by using the predefined ones. Among these are: **flatten()** take a directory hierarchy and make it appear like a single level name space, **match()** pass links matching a specified list of names, **matchhost()** pass links whose target is stored on the matched hosts, **distribute()** create subdirectories for each value of the specified attribute and distribute files among those directories according to that attribute, and **attribute()** pass only links for objects with attributes matching those specified.

Protection of objects in Prospero is based on the protection mechanisms associated with the underlying access method. The ability to resolve the name of an object does not grant permission to access the object. Access control lists may be associated with directories or individual links in directories. Such authorization attributes apply to the ability to resolve names in or to modify the directory. They do not apply to the referenced object itself[8].

[8]Though the attributes of an individual object may include the access control information for the under-

Storage site	Time to resolve a name in Prospero Number of components 1	2	3	4	5	Negotiate Access Method	Open Access Method	Time
Remote	38ms	76ms	115ms	153ms	191ms	32ms	NFS	125ms
Local	21ms	43ms	63ms	86ms	107ms	-	Local	27ms

Table 1: Approximate time to resolve a name and open a file

4.2 Performance

Table 1 shows the performance of the Prospero client on a DECstation 5000. The remote Prospero server is running on a second DECstation 5000 on the same Ethernet. The numbered columns represent the time required to resolve a name with the specified number of components. The second to last column is the time required to negotiate the access method and the final column is the time it takes to open the file. Since Prospero uses the existing access methods of the underlying system, the last column is also the time it takes to open the file without Prospero.

In compiling these figures, the optimization that resolves multiple components at the same time has been disabled. Thus, the time to resolve a name with consecutive components stored on the same server would be less; if all components are stored on the same server the time would be close to that in the first column.

The time required to open a file with a one component name using Prospero (name resolution + negotiation + open) is less than twice that required to open the file directly. This is very good when one considers that at least one extra pair of network messages is involved. Once a file is open, no additional overhead is incurred beyond the access times of the underlying file system. While the performance is quite good for a prototype, it is even better when considered in light of the real contribution of this work: that it enables users to better organize information.

5 Experience

Prospero has been available since December 1990. The prototype implementation allows users to construct virtual systems and to navigate through them. Programs linked with the Prospero compatibility library are able to specify file names relative to the active virtual system when opening files. In addition to the basic release, there are several standalone applications that rely on Prospero to retrieve directory information from indexing services. The prototype has been used to organize information on Internet sites world-wide and Prospero-based applications are used on more than 7,500 systems in 29 countries on six continents.

As distributed, a user's virtual system starts out with links to directories organizing information of various kinds in several ways. When a Prospero file name is mentioned in a mail or news message, the name space that was active when the message was sent appears in the header of the message. Recipients are thus able to properly resolve the name, as well as add links to the object from their own name spaces. This mechanism makes it easy for

lying method by which the object will be accessed.

a user to keep track of files of interest without having to retrieve the file right away. If the file moves and the storage site supports forwarding pointers (which will be the case if the file is moved using Prospero), the link to the file is updated when next referenced.

Users find information by moving from directory to directory in much the same manner as they would in a traditional file system. Figure 5 shows a sample session with Prospero. Users do not need to know where the information is physically stored. In fact, the files and directories shown in the example are scattered across the Internet. At any point, a user can access files in a virtual system as if they were stored on his or her local system.

In the example, the user connects to the root directory and lists it using the `ls` command. The result shows the categories of information included in the virtual system. The information includes online copies of papers (in the papers directory), archives of Internet and Usenet mailing lists (in the mailing-list and newsgroups directories), releases of software packages (in the releases directory), and the contents of prominent Internet archive sites (in the sites directory). Files of interest can appear under more than one directory. For example, a paper that is available from a prominent archive site might also be listed under the papers directory.

Next, the user connects to the papers directory, lists it, and finds the available papers further categorized as conference papers, journal papers, or technical reports. The technical report directory is broken down by organization, and by department within the organization. The journals directory is organized by the journal in which a paper appears, and the two journals that are shown are further organized by issue. Use of the `vls` command shows where a file or directory is physically stored, demonstrating the fact that the files are scattered across the Internet (IEEE TC/OS Newsletter on FTP.CSE.UCSC.EDU and Computer Communications Review on NNSC.NSF.NET.) Though not shown in the example, papers are also organized by author and subject in other directories from the same virtual system.

It is important to note that the example shows only part of the information available through Prospero, and that it shows a typical way that the information is organized. Individuals can organize their own virtual systems differently.

One of the most frequently used directories in Prospero is that representing the archie database, developed at McGill University [5]. That directory includes subdirectories organizing files according to the last components of their file names. For example, the subdirectory *prosp* contains references to the files available by Anonymous FTP whose names include the string "prosp". Among the matches would be files related to Prospero. The contents of each subdirectory are equivalent to what would result from running the Unix `find` command with appropriate arguments over all the major archive sites on the Internet (if it were even possible to do so). The subdirectories do not exist individually but are instead created when referenced by querying the archie database. The use of archie through Prospero has been so successful that the archie group has adopted Prospero as the preferred method for remote access to the archie database.

To provide the benefits of Prospero to users who have not installed it on their systems, Steve Cliffe of the Australian Academic and Research Network (AARNet) Archive Working Group has added Prospero support to one of their FTP servers. As well as making files available from the physical file system, the modified FTP server makes files available from a virtual file system. When a retrieval request is received, the FTP server locates the file

```
Script started on Wed Jan 29 21:02:50 1992
% cd /
% ls
afs              info             papers
databases        lib              projects
documents        mailing-lists    releases
guest            newsgroups       sites
% cd papers
% ls
authors           conferences       subjects
bibliographies    journals          technical-reports
% cd technical-reports
% ls
Berkeley    IAState       OregonSt     UCalgary   UWashington
BostonU     MIT           Purdue       UColorado  Virginia
Chorus      NYU           Rochester    UFlorida   WashingtonU
Columbia    NatInstHealth Toronto      UKentucky
Digital     OregonGrad    UCSantaCruz  UMichigan
% ls UCSantaCruz
crl
% ls UCSantaCruz/crl
ABSTRACTS.1988-89           ucsc-crl-91-01.ps.Z
ABSTRACTS.1990              ucsc-crl-91-02.part1.ps.Z
ABSTRACTS.1991              ucsc-crl-91-02.part2.ps.Z
ABSTRACTS.1992              ucsc-crl-91-02.ps.Z
INDEX                       ucsc-crl-91-03.ps.Z
ucsc-crl-88-28.ps.Z         ucsc-crl-91-06.ps.Z
...
% ls UWashington
cs   cse
%
% ls UWashington/cs
1991           INDEX          PRE-1991
1992           OVERALL-INDEX  README
% cd /papers
% ls
authors           conferences       subjects
bibliographies    journals          technical-reports
% ls journals
acm-sigcomm-ccr  ieee-tcos-nl
% ls journals/ieee-tcos-nl
app-form.ps.Z  v5n1           v5n3
cfp            v5n2           v5n4
% ls journals/acm-sigcomm-ccr
application.ps  jan89           jul90           sigcomm90-reg.ps
apr89           jan90           oct88
apr90           jan91           oct89
apr91           jul89           sigcomm90-prog.ps
% vls journals
   acm-sigcomm-ccr       NNSC.NSF.NET     /usr/ftp/CCR
   ieee-tcos-nl          FTP.CSE.UCSC.EDU /home/ftp/pub/tcos
%
script done on Wed Jan 29 21:06:53 1992
```

Figure 5: Sample session

using Prospero and checks to see if a copy of the file is available locally. Using Prospero to check the last modified time of the authoritative copy, the FTP server checks that the local copy is current. If a current copy does not exist locally, the server retrieves and caches a copy of the file. The local copy is then returned to the client.

6 Related Work

Allowing users to construct a view of a system by selecting components that are available on the network is a goal that is shared by Plan 9. One of the key differences is that Plan 9 addresses the problems of combining the components, not of finding them. The system components in Plan 9 have global names. Plan 9 does not address the problem of how users identify the components that they want to include in their system view. Prospero is concerned primarily with the mechanisms needed to organize and identify the components of interest and relies on system provided access methods to actually use them.

The functionality of filters in Prospero is similar to the domain-switching portal mechanism found in the Universal Directory Service [9]. A portal is a call to a separate name server that may have a non-standard implementation, enabling it to resolve names in a manner different than that in a standard name server. A portal is implemented as a separate server, while a filter is executed by the name resolver. Though the result of resolving a name through a portal is a function of the remaining components in the name, the result is not affected by the point at which the portal is attached. This means that a new portal (and hence new name server) must be run for each point of attachment. The portal mechanism is closer to that used to integrate Prospero and archie than it is to most filters.

Attribute-based naming, supported by Profile [14] and the Semantic File System[6], provides an alternative mechanism for finding information of interest. In attribute-based naming a database is maintained of object attributes, and the user specifies the known attributes of an object instead of its name. In the Semantic File System, the result is a directory listing those objects matching the specified attributes. In Profile, if enough attributes have been specified to uniquely identify the object, the result is a reference to the object itself.

For attribute-based naming to scale, directory information must be distributed across multiple servers. Without a way to direct a query to the right server, queries must be sent to all servers, an operation that doesn't scale. Profile restricts the set of servers that are queried and relies on cross-references to direct queries to servers that were not included in the original set, but doing so negates one of the advantages of attribute based systems; the necessary cross-references must be in place before a query is made.

When used together, attribute-based naming and Prospero could be very powerful. The databases maintained by such systems could be accessed through filters that could perform any desired pre- or post-processing. Other features of the Virtual System Model could be used to impose a structure that directs queries to the appropriate servers. Such a combination of attribute- and link-based naming is similar to recent work on multi-structured naming [22].

In an alternative approach to finding objects in large systems, Schwartz proposes the use of resource discovery agents [21] that accept queries from users and use the information provided by the user to find objects in which the user is interested. In Schwartz's design, the information needed to direct a query to the appropriate agent evolves over time. A query

is directed to the nearest agent, and agents learn how to direct queries based on the results of previous queries. The problem with this approach is that it gives the agents too much of the responsibility for building the resource discovery graph. However, a combined approach where agents make use of the organization imposed by individuals (e.g., as encoded in the Prospero naming network) might yield better results.

7 The Future of Prospero

Prospero is an evolving system. We continue to collaborate with other groups to extend it. We are in the process of adding support for the retrieval of documents maintained by the Wide Area Information Service (WAIS) [8]. We plan to add filters that access directory information maintained by by semantic file systems [6] and distributed indices [4] when those systems are available.

We plan to implement a new application interface for Prospero in order to allow use by existing applications without relinking. This will be accomplished by adding Prospero support to an NFS server [20], the same approach taken by semantic file systems [6] and Alex [2]. We hope to benefit from changes already made in those systems.

The Prospero protocol provides a lightweight protocol for querying directories and obtaining file attributes. We encourage its use as a base upon which other systems can be built. Archie and AARNet are two examples. It is being considered for use by Alex [2] to improve the performance queries to directories on hosts that run Prospero servers.

8 Conclusions

This paper discussed several problems that arise in the organization of a global file system. It demonstrated the importance of customization and presented two mechanisms, the filter and the union link, that allow views of the global name space to be defined in terms of other views. The lack of name transparency across customized name spaces has the potential to cause confusion, but this problem is addressed by supporting closure.

The prototype file system and directory service described in this paper is used from more than 7,500 systems worldwide. The use of the prototype to solve real problems was discussed; its acceptance demonstrates the benefits of the organizational flexibility provided by the Virtual System Model.

Availability

To find out more about Prospero, or for directions on retrieving the latest distribution, please send a message to info-prospero@isi.edu.

Acknowledgments

Ed Lazowska provided valuable guidance throughout this work. Discussions with John Zahorjan, Hank Levy, and Alfred Spector helped me refine the ideas that ultimately led to the development of Prospero. Celeste Anderson, Ben Britt, Vincent Cate, Steve Cliffe, Peter Danzig, Peter Deutsch, Alan Emtage, Deborah Estrin, and Dennis Hollingworth provided comments on earlier drafts of this paper.

References

[1] Luis-Felipe Cabrera and Jim Wyllie. QuickSilver distributed file services: An architecture for horizontal growth. In *Proceedings of the 2nd IEEE Conference on Computer Workstations*, pages 23–27, March 1988. Also IBM Research Report RJ 5578, April 1987.

[2] Vincent Cate. Alex: A global file system. In *Proceedings of the Workshop on File Systems*, May 1992.

[3] Douglas Comer, Ralph E. Droms, and Thomas P. Murtagh. An experimental implementation of the Tilde naming system. *Computing Systems*, 4(3):487–515, Fall 1990.

[4] Peter B. Danzig, Jongsuk Ahn, John Noll, and Katia Obraczka. Distributed indexing: A scalable mechanism for distributed information retrieval. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, October 1991.

[5] Alan Emtage and Peter Deutsch. archie: An electronic directory service for the Internet. In *Proceedings of the Winter 1992 Usenix Conference*, pages 93–110, January 1992.

[6] David K. Gifford, Pierre Jouvelot, Mark A. Sheldon, and James W. O'Toole Jr. Semantic file systems. In *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, pages 16–25, October 1991.

[7] John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, and Michael J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, February 1988.

[8] Brewster Kahle and Art Medlar. An information system for corporate users: Wide area information systems. Technical Report TMC-199, Thinking Machines Corporation, April 1991.

[9] Keith A. Lantz, Judy L. Edighoffer, and Bruce L. Hitson. Towards a universal directory service. In *Proceedings of the 4th ACM Symposium on Principles of Distributed Computing*, August 1985.

[10] B. Clifford Neuman. The need for closure in large distributed systems. *Operating Systems Review*, 23(4):28–30, October 1989.

[11] B. Clifford Neuman. Workstations and the Virtual System Model. In *Proceedings of the 2nd IEEE Workshop on Workstation Operating Systems*, pages 91–95, September 1989. Also appears in the *Newsletter of the IEEE Technical Committee on Operating Systems*, Volume 3, Number 3, Fall 1989.

[12] B. Clifford Neuman. *The Virtual System Model: A Scalable Approach to Organizing Large Systems*. PhD thesis, University of Washington, 1992. Department of Computer Science and Engineering (in preparation).

[13] John K. Ousterhout, Andrew R. Cherenson, Frederick Douglis, Michael N. Nelson, and Brent B. Welch. The Sprite network operating system. *Computer*, 21(2):23–35, February 1988.

[14] Larry L. Peterson. The Profile naming service. *ACM Transactions on Computer Systems*, 6(4):341–364, November 1988.

[15] Larry L. Peterson and Herman C. Rao. Accessing files in an internet: The Jade file system. Technical Report TR 90-30, University of Arizona, 1990.

[16] Jon B. Postel. User datagram protocol. DARPA Internet RFC 768, August 1980.

[17] Jon B. Postel and J. K. Reynolds. File transfer protocol. DARPA Internet RFC 959, October 1985.

[18] D. Presotto, R. Pike, K. Thompson, and H. Trickey. Plan 9: A distributed system. In *Proceedings of Spring 1991 EurOpen*, May 1991.

[19] Jerome H. Saltzer. *Operating Systems: an advanced course*, volume 60 of *Lecture Notes in Computer Science*, chapter 3, pages 99–208. Springer-Verlag, 1978.

[20] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and implementation of the Sun Network File System. In *Proceedings of the Summer 1985 Usenix Conference*, pages 119–130, June 1985.

[21] M. F. Schwartz. Resource discovery and related research at the University of Colorado. Technical Report CU-CS-508-91, Department of Computer Science University of Colorado, Boulder, January 1991.

[22] Stuart Sechrest and Michael McClennen. Blending hierarchical and attribute-based file naming. In *Proceedings of the 12th International Conference on Distributed Computer Systems*, June 1992.

[23] Andrew S. Tanenbaum, Robbert van Renesse, Hans van Staveren, Gregory J. Sharp, Sape J. Mullender, Jack Jansen, and Guido van Rossum. Experience with the Amoeba distributed operating system. *Communications of the ACM*, 33(12):47–63, December 1990.

[24] B. Walker, G. Popek, R. English, C. Kline, and G. Thiel. The Locus distributed operating system. In *Proceedings of the 9th ACM Symposium on Operating Systems Principles*, pages 49–70, October 1983.