The Virtual System Model: A Scalable Approach to Organizing Large Systems

A Thesis Proposal

B. Clifford Neuman

Department of Computer Science and Engineering
University of Washington, FR-35
Seattle, Washington 98195

bcn@cs.washington.edu

Technical Report 90-05-01
May, 1990

1 Motivation

In recent years, many researchers have explored mechanisms that allow systems to span large networks. Few, however, have looked at the problems that will confront the users of these highly distributed systems. There will be a huge amount of information available within such a system, and that information will be scattered about. Even if users can access it all, there will be so much that is irrelevant that it will be difficult to identify the pieces that are of interest.

This problem has two aspects. First, users must be able to keep track of objects about which they already know and in which they have indicated an interest. Second, users must be able to identify objects of potential interest about which they do not already know. The mechanisms that solve the two sub-problems should work together; users are often willing to spend time organizing information for their own use, and it should be possible for the meta-information, so specified, to be shared with others.

In recent years, several approaches have been devised to address these problems, but these approaches break down in environments that scale both geographically and administratively. Among the approaches that have been proposed are user-centered namespaces, databases, attribute-based naming, resource discovery agents, and a uniform global namespace. I will describe these approaches and the problems with each. I will then describe my own approach, and a preliminary implementation testing the concepts of that approach. I will end by succinctly stating my "thesis" and describing what I propose to cover in the dissertation.

This research was supported in part by the National Science Foundation (Grants No. DCR-8420945 and CCR-8611390), US WEST Advanced Technologies, and Digital Equipment Corporation.

2 Existing Approaches

A variety of approaches have been devised in an attempt to help users navigate through large systems. As noted earlier, there are two major goals: to allow each user to organize objects that are known, and to identify objects of potential interest.

The user-centered namespace[1, 2] is the predominant approach used to address the first problem. A user-centered namespace allows each user's view of the system-wide namespace to be organized according to the user's preferences. Objects in which the user has expressed an interest will have short names, while those in which there is little interest will have longer ones (if they can be named at all). One problem with the user-centered namespace is that the same name may refer to different objects when used by different users. This has the potential to cause confusion, and it makes sharing difficult. Another problem is that, while this approach allows users to organize the objects that are already known, it doesn't help them identify objects of potential and unrealized interest.

The approaches described in the remainder of this section are of little help in solving the first problem: organizing objects once they have been discovered. Instead, they are of more use in solving the second problem: how to identify objects of unrealized interest.

One approach is to build a database containing information on all objects. Users search this database using the information that they already know. This is the approach used by commercial database services such as Lexis, or those used in libraries. A key point to observe about these databases is that they are relatively static, and that the indexing information is usually precomputed. The task of indexing is further simplified because all the data is under the control of a single organization (the database service), is distributed over relatively few sites, and can be redistributed as necessary.

Such databases are inappropriate for the problem at hand because of the dynamic nature of large systems. Information is constantly changing, and there isn't a single entity that controls it. As for using a general database system, there are presently none that scale to the number of hosts that are needed, or that allow use by large numbers of users that are not necessarily cooperative. It should be pointed out that the other approaches that I will describe are all, in some sense, databases, but that they are specialized to the task at hand.

An approach related to the database approach is attribute-based[8] naming. In attribute-based naming, objects are named by a collection of attributes. In specifying an object, attributes may be omitted, and only enough attributes to uniquely identify the object must be provided. One problem with attribute-based naming is that the attributes must be registered ahead of time. Further, the information needed to identify an object might not be readily apparent to the user (though it would be recognized if it were seen).

Like databases, attribute-based naming, in its pure form, does not scale well. It is difficult to distribute the database of attributes across a large number of machines. Attribute-based systems require little *a priori* knowledge of what the queries will be, and without this knowledge, it becomes difficult to build the cross references that are required to direct queries to just the right system.

As a result, queries must be sent to all systems. One attribute-based naming system, Profile[8], restricts the set of name servers that are queried, and relies on cross-references to direct queries to servers that were not included in the original set. This approach requires that the right cross-references be in place before the query is made, negating one of the advantages of attribute-based systems, i.e., that the links corresponding to a particular query need not be in place ahead of time.

Several recent systems support a uniform global namespace[3, 7, 11]. A uniform namespace facilitates sharing since each user uses the same name to refer to an object. While this helps users identify and find the objects about which they have learned (e.g., from other users), it does not directly help them discover objects of potential interest. Instead, it is the way that objects are organized in these systems that helps users discover objects.

Objects are usually organized hierarchically. Links are named, and typically specify additional information about the objects in the sub-hierarchy reached through the link. As one moves deeper in the hierarchy, progressively more information has been specified about the objects that are to be found.

A problem with the typical organization of such hierarchies is that the information at the top level of the hierarchy pertains to the administrative structure of the system. The information in which one is interested is often scattered across the hierarchy, and because the information about the administrative structure of the system is not inherent in the data one is looking for, it is often necessary to search for information across the entire namespace.

In an alternative organization, everything is arranged by topic. With this approach one is able to quickly find information that is relevant, but this approach suffers from the problems that the administratively structured hierarchy solved: it is difficult to control access to the shared directories; there is bound to be disagreement on what should appear in various directories; each directory is stored in a central location instead of being close to the principals that typically use it, thus preventing one from exploiting locality; and many directories will be too big.

Another problem with uniform hierarchical namespaces as presently supported is that information can often be classified in more than one way. When this happens, it is desirable that an object appear with more than one name. This can be accomplished by making more than one link to an object, but existing systems support only symbolic links.

Symbolic links suffer from several problems. When using symbolic links, one name for an object is primary, and all others are secondary. This means that if the primary name for an object changes (or is removed), the other links cease to be valid. Supporting multiple primary links in a distributed environment makes problems such as garbage collection and object mobility more difficult, but the difference in semantics is well worth it. A second problem with symbolic links, when used to attach a hierarchy of objects, is that an additional name for the hierarchy can only be added at the point of attachment. Below that level, all names remain the same as in the primary hierarchy.

There are other approaches that have been proposed for identifying objects of potential interest. Schwartz proposes the use of resource discovery agents[10] which 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. Techniques have been suggested for making the agents capable in this regard, but the scalability of these techniques is in doubt.

The approaches that have been described so far fall short because they fail to provide a scalable mechanism to adequately direct a search to the correct repository. The next section describes a model for organizing large systems that supplies the needed direction.

3 The Virtual System Model

My approach to the problems of object organization and discovery is embodied in the Virtual System Model[5, 6]. In the Virtual System Model, each user creates a set of virtual systems. Each virtual system defines a different, customized view of the underlying system. This allows users to organize the objects and information about which they already 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 namespaces, can be combined with other information and made available for use by others.

3.1 The Namespace

Within the Virtual System Model, the global namespace forms a generalized directed graph. Each directory within the graph presents a way of viewing the objects that it contains. Users (or agents for users) choose the appropriate view based on the information they are using to locate an object. The Virtual System Model presents a user-centered[1] namespace. Individual users can customize their namespace by selecting the node that is to be their root. In fact, the user's namespace is usually created by establishing a new directory and making links to the parts of the graph in which a user has an interest.

As indicated earlier, the fact that the same name may refer to different objects at different times can make a user-centered namespace confusing, and can hinder sharing. To address this problem, the Virtual System Model includes the concept of closure[4, 9]. Each object has an associated namespace. In this way, the context within which a name is to be resolved is automatically passed along with the object specifying the name. Although the same name may refer to different objects within different contexts, the correct context is always known.

The user-centered namespace identifies more than just files. It specifies the services, the users, the processors, and the other objects that are treated as part of a user's *virtual system*. A virtual system defines a view of the world. It specifies which files, services, users, processors, and other

[1] Technically, it is better described as object-centered since a namespace is associated with each object in the system.

system components are to be visible. A virtual system can be thought of as a computer system constructed from the selected pieces. The pieces can physically reside anywhere in the global system as long as they are accessible. In practice, the most frequently accessed pieces are likely to exist on the local system or at the local site.

Discovering objects of potential interest in such a system requires a mechanism that can narrow the search space. It must direct the user appropriately, bringing the user closer to objects that may be of interest. There are a number of features that a directory mechanism should have if it is to scale while still allowing objects to be organized in ways beneficial to the user: a directory mechanism should allow a single directory to be distributed across multiple systems; it should support multiple primary names for objects; and it should support multiple views of directories. By supporting multiple views, different users can impose their own policy on what they see. Different views also provide greater support for locality since one can choose to see only those parts of a directory that are nearby.

3.2 Filters and Union Links

One way that a directory mechanism can support distribution and multiple views is by allowing directories to include the contents of other directories. In the Virtual System Model, this functionality is provided by the union link. A directory, as seen by a user, may include the contents of many smaller directories that are scattered across multiple systems. Other directories might include the contents of different sets of directories, possibly overlapping with those in the first set. By changing the set of underlying directories that are included, different views of a directory can be supported.

The Virtual System Model further supports alternative views of a directory by allowing programs called *filters* to be associated with links. A filter is a program, attached to a link, that allows the view of the target directory to be altered. In this way, a view can be specified as a function of another view.

Filters and union links provide a powerful mechanism for supporting customization and the manipulation of namespaces. 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 (Normally, only the directories referenced through the filter are affected.)

It is important that the functions that specify the manipulation of namespaces be associated with links instead of directories. This is because the result of resolving a name is itself a link (a reference to an object). By associating filters with links, a filter can return another filter that is to be applied when the links it returns are further resolved. This allows filters 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 namespace which are specified entirely within the filter.[2]

[2]If the functions describing a directory were associated with each directory, then to affect more than the first level of a hierarchy would require these functions to create new directories on the fly. These directories would either have to be destroyed as soon as the query was done, or their consistency with the data on which they were based would need to be maintained.

4 Prospero

Prospero[3] is a distributed operating system presently under construction, the design of which is based on the Virtual System Model. Because of the large number of files on even a small collection of systems, the initial focus for Prospero has been the file system. A prototype is running on several systems at the University of Washington and elsewhere on the Internet.

The goal of Prospero is to provide a testbed for the features of the Virtual System Model. I hope to have several servers running on sites scattered across the Internet, providing access to the publicly available information stored at these sites. Many sites presently make information available by anonymous FTP and other means. While my ultimate goal is to make all of this information available within the Prospero file system, my short-term goal is to provide a directory service within which this information may be found, and to allow the information to be accessed externally. This approach allows users to experiment with Prospero as an enhancement to the systems that they are currently using.

4.1 Description

The primary component of the Prospero file system is the directory service. The directory service maps a name within a virtual system to the name of the host on which the corresponding file physically resides and the name of the file on that host. Once the physical location of the file has been determined, existing file access techniques (presently NFS) are used to access the file.

The directory service is implemented in a distributed fashion. Each system that stores files runs a directory server. The directory server takes a pointer to a directory, and the name of a component in that directory. It returns the contents of the directory matching the specified component. Each directory entry consists of the name of the object, the host on which the referenced object resides, and the identifier of the object on that host. A directory can contain references to objects that are stored remotely.

Prospero includes tools for constructing and modifying virtual systems. These tools create and destroy virtual systems, create new directories, and add, delete and modify links in existing directories. Directories in Prospero can contain filtered links and union links. When querying directories, these active components are usually returned to the client and processed there. It is possible to request that the processing of these components occur on the remote directory server, but the server is not required to comply with such requests.

[3] 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 (magic) 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.

4.2 Status

As noted, the Prospero directory service maps names within a virtual system to the physical location of objects, and NFS is used to access these object. An application library that allows applications to transparently use the Prospero file system is available, and several applications have been linked with it, including cat, cc, cp, dd, ed, finger, grep, ld, ls, more, strings, tail and wc. Union links are supported, closure is not. Filters have been implemented and are in the process of being integrated with the file system.

4.3 Achieving the goals of Prospero

Among the work to be accomplished: filters must be integrated with Prospero, and support for closure must be added. A mechanism is needed to allow an application to use closure; in particular, the mechanism must allow the application to specify which context is to be used when resolving names (i.e., the user's, the application's, or that of an open file).

The application library is large, and is duplicated in each executable with which it is linked. I would like to implement Prospero in (through) the kernel. This would solve several problems, and would result in automatic support for Prospero by all applications. A kernel implementation would be best handled by incorporating the upcall mechanism from AFS[3], and placing the Prospero resolver in an application process. The hard part has already been done by others.

Finally, Prospero must be made robust enough so that it can be used by others. Most of the advantages of the Virtual System Model come from the way it allows information to be organized. To evaluate its effectiveness, at least a limited amount of real-world use is required.

To aid in this evaluation, Prospero must be distributed to a number of sites. It is unlikely that others will immediately buy into a kernel implementation, or even add Prospero support to key applications. For this reason, the implementation to be distributed will take the form of a shell that can be used to navigate the generalized directed graph that forms the Prospero namespace. In this way, users will be able to use Prospero to find resources available on the Internet without having to completely buy into the model. Once a resource has been found, the user will be able to execute a command that will transfer the resource to the local system using FTP or another file access mechanism.

Prospero servers will need to run on key sites that make information available to the rest of the Internet. There are several collections of information that presently exist. Among them are archive sites for Usenet, and a set of storage sites organized under the *Online Book Initiative*. Both collections keep track of the what is stored at each site through a centrally maintained list. It should be fairly straightforward to allow a distributed directory to be maintained in real time using Prospero. Once set up, Prospero would be the first system tying together these resources, and as such it should be relatively easy to get people to use it. Once receiving regular use, the success of the Prospero prototype may be evaluated by looking at the views of data that are developed by users.

5 The Thesis

My thesis is that scalable mechanisms are needed to find objects of potential, but unrealized, interest, and to organize objects once they have been found. Customization plays an important role in the solution to this problem. It eases the task of users in keeping track of the objects that they use on a regular basis. Further, it is possible to make use of the information specified by users in customizing their own namespaces when constructing views of information to be shared with others. A problem with customized namespaces is the lack of name transparency: the same name may refer to different objects when used in different namespaces. This problem can be addressed by supporting closure.

The ability to identify objects of potential interest is greatly enhanced when objects are organized in more than one way. Large systems should allow information to be organized by multiple entities. Among them are individuals, organizations (e.g., libraries, professional societies), and commercial services (e.g., clipping services). Users should be able to choose among the alternatives, and it should be possible to define one's own view of the information as a function of one or more other views.

The Virtual System Model brings this all together. It supports a user-centered namespace and closure. Objects may be organized in multiple ways, and each name for an object is primary. Finally, tools are provided to allow new views of objects to be constructed as a function of existing views.

Prospero is an operating system based on the Virtual System Model. It provides a testbed upon which the usefulness of the Virtual System Model can be evaluated.

5.1 Structure of the Dissertation

The dissertation will start by describing the problems that arise for the user as systems grow. Next, it will briefly describe the features that are needed to address these problems, and why they are important. It will describe existing systems, and show that they do not adequately address the problem.

A new model for organizing large systems, the Virtual System Model, will be presented, and I will show how it supports the features that I described earlier. I will show how these features apply to the file system, to security, and to the selection of servers, services, and processors. By presenting examples of its use, I will show that the Virtual System Model provides a powerful and flexible model for organizing large systems. In particular, I will show that many of the research tools used in the real world can be implemented within the model, and I will show how it is easier to keep information up-to-date when these tools are implemented in such a manner.

I will make things more concrete by describing Prospero, and I will discuss the ways that it will have been used to organize information that is available over the Internet. Finally, I will assess the success of the Prospero prototype in addressing the problems that arise for the user of large systems. I will show what has been accomplished, where more work is needed, and where additional evaluation is required. I will end by discussing the usefulness of the Virtual System Model.

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.

[2] Douglas Comer and Thomas P. Murtagh. The Tilde file naming scheme. In *Proceedings of the 6th International Conference on Distributed Computing Systems*, pages 509–514, May 1986.

[3] 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.

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

[5] B. Clifford Neuman. The Virtual System Model for large distributed operating systems. Technical Report 89-01-07, Department of Computer Science, University of Washington, April 1989.

[6] B. Clifford Neuman. Workstations and the Virtual System Model. In *Proceeding 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.

[7] 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.

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

[9] Jerome H. Saltzer. *Operating Systems: an advanced course*, volume 60 of *Lecture Notes in Computer Science*. Springer-Verlag, 1978. Chapter on naming.

[10] M. F. Schwartz. The networked resource discovery project. In *Proceedings of the IFIP XI World Congress*, pages 827–832, August 1989. San Francisco.

[11] 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.